Sparse Time versus Dense Time in Distributed Real-Time Systems

H. Kopetz
Technical University of Vienna
Vienna, Austria
email:hk@vmars.tuwien.ac.at

Abstract

This paper proposes the restriction of the significant event occurrences, i.e., the sending and receiving of messages in a distributed real-time system, to the lattice points of a globally synchronized space/time lattice. One dimension of this space/time lattice represents the progression of time, the other dimension denotes the computational processes in the system. This additional constraint simplifies the solutions to the agreement problems. After an analysis of the interdependence between temporal order, causal order, receive order, and the limits to time measurement in a distributed real-time system, criteria for the selection of the lattice points of this space/time lattice are presented.

1. INTRODUCTION

The performance of a distributed real-time system must be predictable. Accepted variations of the hardware components (e.g. specified deviations from the nominal frequency of a quartz crystal) or the occurrence of hypothesized faults should not have any effect on the quality and timeliness of the system service. The slightly varying timebases in the different nodes of a distributed system or the occurrence of faults can lead to major disagreements in the states of the nodes if proper agreement protocols are not provided.

Agreement on time, order, membership, and data are fundamental problems in any distributed system. In a real time system these agreement problems have to be solved within the given time constraints. Many solutions to agreement problems, e.g., the establishment of a consistent order [Lam78], have been investigated in an asynchronous environment without a global timebase. These solutions are complex and expensive. The introduction of a synchronized global time [Lam84] can speed up the solution of agreement problems. [Cri88] has shown that the membership problem has a simpler solution if a common notion of time and a synchronous architecture can be assumed. In this paper we go one step further. We restrict the occurrence of significant events (e.g., the sending and receiving of messages) to the lattice points of a globally synchronized action lattice, i.e., a sparse timebase. If the lattice points of this action lattice are properly chosen, temporal order and simultaneity of actions can be realized without considerable protocol overhead.

This paper is organized as follows. In the next section we describe our computer system model and introduce three desired properties of distributed real-time systems, the consistent order property, the simultaneity property, and the temporal order property. After a discussion of the relationship between temporal order, causal order and receive order we discuss the limits of global time measurement. In order to avoid complex agreement protocols the introduction of a sparse timebase is proposed in Chapter 5. The problems occurring at the interface to the environment are elaborated in the final chapter.

2. DISTRIBUTED COMPUTER SYSTEM

We assume a distributed computer system consisting of a set of fail-silent node computers that are attached to a real-time bus. The nodes communicate by the exchange of *messages* only. Some nodes, the *interface nodes*, support a connection to the intelligent instrumentation, i.e., the sensors and actuators in the environment. We assume that every node has its own real-time clock. All clocks of the nodes are synchronized such that an *approximate global timebase* is available to all client tasks in the different nodes. We will discuss the properties of this global timebase in more detail in chapter 4.

In a typical real-time application, the computer system has to perform a multitude of different functions in parallel, e.g., the display of sensor readings (we call a sensor reading at a particular point in time an *observation*) to the operator, the monitoring of observations (both their value and rate of change) to detect alarm conditions, the processing of observations by process models in order to find new setpoints, etc.. In distributed computer systems, these different functions are normally allocated to different nodes. Additionally, replicated nodes are introduced to provide fault tolerance by active redundancy.

To guarantee a consistent behavior of the distributed computer system as a whole and to maintain replica determinism [Bar90] between replicated nodes, it should be assured that all nodes act

(1) on different observations in the same order (*consistent order property*).

(2) on the same observation at about the same time (*simultaneity property*).

(3) on different observations in the temporal order of their occurrence (*temporal order property*).

If the consistent order property is violated, the internal state changes of replicated nodes can be different and the nodes might enter diverging states. If the simultaneity property is violated, the temporal coordination of the actions of the nodes is impaired and a seemingly unsynchronized behavior of the system may result. If the temporal order property is violated, an event analysis which tries to reestablish the possible causal order of event chains is obstructed.

If the temporal order property can be guaranteed then consistent ordering is implied. However, we will see that there are fundamental limits to the fulfillment of the temporal order property. To find these limits, we need a better understanding of the relationships between causal order, temporal order, and time-measurement.

3. TEMPORAL AND CAUSAL ORDER

Temporal order

Let us assume that the continuum of real time is modelled by a *directed timeline* consisting of an infinite set of instants {T} with the following properties [Whi90,p.208]:

(1) {T} is an ordered set, i.e., if p and q are any two instants, than either p is simultaneous with q or p precedes q or q precedes p and these relations are mutually exclusive. We call the order of instants on the timeline the *temporal order*.

(2) {T} is a dense set. This means that, if p precedes r, there is at least one q which is between p and r.

Let us further assume that there exists an omniscient external observer who can observe all events that are of interest in a given context. (Note that we disregard relativistic effects.) This observer possesses a single reference clock z with a frequency f_Z which is in perfect agreement with the international standard of time TAI. We call $1/f_Z$ the *granularity* g_Z of clock z. Whenever this observer perceives the occurrence of an event e, he will instantaneously record the current state of his reference clock as the time of occurrence of this event e and thus generate a *timestamp* of e. We denote with *clock(event)* the timestamp generated by the use of a given *clock* to timestamp an *event*. Since z is the single reference clock in our system, we call z(e) the *absolute timestamp* of an event e. We will measure the *duration* between two events by counting the ticks of the reference clock which occur between these two events. The temporal order of events which occur between any two consecutive ticks of the reference clock, i.e., within the granularity g_Z, cannot be reestablished from their absolute time-stamps. This is a fundamental limit in time measurement.

We call a set of events {E} *d-precedent* if the events in this set occur either simultaneously or precede each other by more than d time units [Ver89]. Given a single timebase with a granularity g we can only reconstruct the temporal order of a set of time-stamped events if this set is g-precedent.

Causal order

In many real time applications we are interested in the causal dependencies between events. For example, in an alarm analysis we have to identify the event which was the cause of a set of consequent alarms, i.e., an alarm shower.

Reichenbach [Rei57,p.145] suggested to define causality by his known 'mark' method without reference to the time dimension: "If event e1 is a cause of event e2, then a small variation (a mark) in e1 is associated with small variation in e2, whereas small variations in e2 are not necessarily associated with small variations in e1."

Causal order is transitive, i.e., if e1 is the cause of e2, and e2 is the cause of e3, e1 is also the cause of e3.

Temporal order is a prerequisite for causal order. If and only if the occurrence of an event e1 has preceded the occurrence of an event e2 in the domain of real time, it is possible that e1 has an effect on e2. On the other hand, if it can be established that e2 has occurred after e1, then e2 cannot be the cause of e1. It is important to note that the time of event occurrence, not the time of event recognition (the *receive order*) by some subsystem observing the world, establishes potential causality.

To illustrate this point, consider the following simple practical example: A distributed system consisting of three nodes k, l and m monitors a plant. The interface node k observes the rupture of a pipe, OP, and the interface node l observes the consequent fall in pressure OF. As soon as the event OF is recognized by node m it performs an emergency shutdown OS. Because of the variable delays in the communication system the events are reordered during transmission. An omniscient outside observer sees the events in the following temporal order:

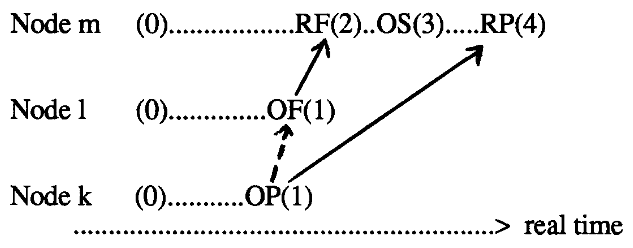
OP OF RF OS RP

OP	Occurrence of pipe rupture
OF	Occurrence of pressure fall
RF	Recognition of pressure fall by node m
OS	Occurrence of shut down action
RP	Recognition of pipe rupture by node m

Although the event "recognition of pipe rupture" was perceived by node m *after* the event "shut down" action, the "occurrence of pipe rupture" was the originating cause for this "shut down" action. The temporal order of the event recognition (the receive order) may not be taken as the basis for a causal analysis, if clandestine communication channels cannot be ruled out. Such clandestine communication channels exist in many environments of real-time systems.

Consider a system without globally synchronized clocks. In such a system the causal relationship which is brought about by a clandestine channel within the environment between nodes k and l will not be necessarily reflected in the consistent order generated by an atomic broadcast protocol [Cri85] nor in the "happened before" relation, the precedence order, established by Lamport [Lam78]. It could even happen that this precedence order, based on the event recognition, is opposed to the temporal order, based on the event occurrence, and contradicts the actual causality generated by a clandestine channel in the environment (Fig 1 shows the possible logical order of the events introduced in the previous paragraph). Ordering protocols [Ray90], which are viable in a *closed* system may not be used in a real-time system which is an *open* system.

A global time, which is used to time-stamp the event occurrences helps to reconstruct the temporal order, which is a prerequisite for causal order. But what are the limits to time measurement in a distributed system?


Node m (0)..................RF(2)..OS(3).....RP(4)
Node l (0).............OF(1)
Node k (0)...........OP(1)
....> real time

---- clandestine channel _____ message transmission

The logical clock-tick number generated by Lamport's algorithm [Lam78] is given in the parenthesis.

Fig.1 Physical and logical order of events

4. GLOBAL TIME

Accuracy and Precision

Let us analyze a real-time clock k with a granularity g_k. Provided the granularity g_k of this clock k is much larger than the granularity of the reference clock g_z (this is normally the case), we can measure the granularity of clock k with the reference clock. We assume that whenever clock k ticks, the state of the clock is incremented by its intended granularity g_k, expressed in the ticks of the reference clocks, and that, on start up the clock is initialized with the state of the reference clock. We denote the state of clock k after its *ith* tick by $k(k_i)$, (we denote the *ith* tick of clock k by k_i). Since the state of the clock is not modified between two consecutive ticks, all events which occur within a granule of time g_k will be time-stamped with the same value. On the basis of the time-stamps of clock k with granularity g_k, the temporal order of a set of events can only be reestablished if they are known to be g_k-precedent.

Ordinary clocks are not perfect. They can drift by up to σ sec/sec, the drift rate of clocks, from the reference clock z. Quartz controlled clocks have a drift rate of the order of 10^{-5} to 10^{-7}. The granularity of such a clock may differ by up to $\sigma * g_k$ seconds from the intended granularity. Therefore a normal clock has to be resynchronized with the reference clock every so often. We call the maximum guaranteed difference [Kop87] between the state of clock k and the state of the reference clock

$$A^k = MAX (\forall i \mid k(k_i) - z(k_i) \mid)$$

the accuracy A^k of the clock k.

Let us now consider an ensemble of n clocks with the same intended granularity g. We call the maximum difference between any two elements of this ensemble

$$\Pi = MAX (\forall i, \forall k, \forall l \mid z(k_i) - z(l_i) \mid)$$

the precision Π of the ensemble. Precision and accuracy are two different (but related) concepts. If all clocks of the ensemble run at the same speed, but faster than the reference clock, then they might have a good precision but a bad accuracy. The opposite is not possible, since

$$\Pi \leq 2A.$$

Granularity of a Global Time

A *global time* is an abstract notion which can be approximated by properly selected ticks from synchronized local real-time clocks of an ensemble. Let us assume that in a given ensemble of { N } nodes the clocks are synchronized to a precision Π . The "global tick" will then happen within an interval Π on the timeline (the *global tick interval*). In order to avoid the overlap of global ticks, the granularity of the global time g_g must be

$$g_g > \Pi.$$

We call this relation between granularity of the global time and the precision of synchronization the *"granularity condition"*.

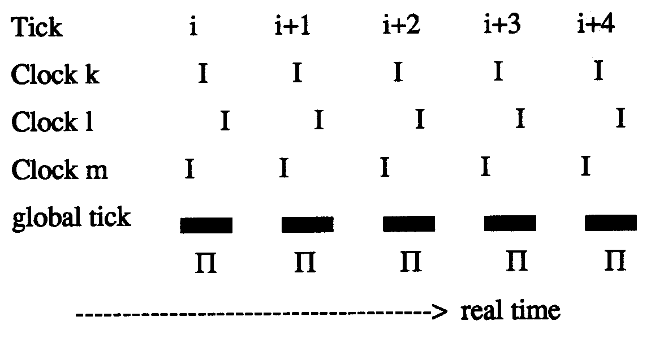


Fig. 2: Precision of global timebase

Dense Timebase versus Sparse Timebase

Assume a set { E } of significant events which are of interest in a particular context. This set { E } could be the ticks of all clocks or the events of sending and receiving messages. If these events are allowed to occur at any instant of the timeline, we call the timebase *dense*. If the occurrence of these events is restricted to some sections of the timeline, we call the timebase *sparse*. If a system is based on a sparse timebase, there are timeintervals where no significant event is allowed to occur.

It is evident, that the occurrences of events can only be restricted if the given system has the authority to control these events, i.e., these events are in the sphere of control of the computer system [Dav79]. For example, in a distributed computing system the sending of messages can be restricted to some subsections of the timeline and can be forbidden at some other subsections. In general, the occurrence of events outside the sphere of control of the computer system cannot be restricted, i.e., these external events are based on a dense timebase.

Let us assume that

$$g_g = \Pi + K$$

where K is the time interval separating the latest clock of global tick i from the earliest clock of the global tick i+1 (In Fig. 2 these are the intervals between the global tick bars). If the occurrence of significant events within a system is restricted to the intervals denoted by K--a sparse timebase--, then these events will be timestamped with the same tickvalue by every clock of the ensemble. If the events can happen on a dense timebase, such a consistent timestamping of events in a distributed system cannot be guaranteed. It may happen that a single event may be timestamped differently (difference = 1) by two nodes of the ensemble, even if the granularity condition is observed. We call this phenomenon the "*imprecision effect*". The imprecision effect can be circumvented if the event set under observation is at least $2g_g$-precedent.

5. COMMUNICATION WITHIN THE SYSTEM

Let us consider a message exchange between two real-time tasks residing at different nodes. We call the real-time interval between the start of the send statement in the sending task and the termination of the receive statement in the receiving task, measured by their absolute timestamps, *the message transit delay* d. This message transit delay is determined by the time it takes to execute the protocol stack at both ends of the communication and by the transmission delay. We call the maximum message transit delay d_{max} and the minimum message transit delay d_{min}. The difference

$$\varepsilon = d_{max} - d_{min}$$

is called the *temporal uncertainty* ε of the communication protocol.

If the receiver acts on a message as soon as it is delivered, neither the consistent ordering property, nor the simultaneity property, nor the temporal order property is assured. Because of the temporal uncertainty of the communication protocol messages can be reordered, i.e., a message A which has been sent after another message B can arrive before this earlier sent message B.

Therefore a receiver must wait after the receipt of a message until the system is stable. A system is *stable* in relation to a given message A if all outstanding messages that have been sent earlier than this message A are guaranteed to have arrived. We call this temporal interval between the start of sending of a message and the point in time after which the system becomes stable, the *action delay*. The action delay depends on the performance parameters of the given system architecture, such as the bandwidth and the delay characteristics and of the communication channel, the speed of the computer hardware, the structure of the operating system and the communication protocols (e.g. the media access strategy), and the precision of clock synchronization. In a system with a global timebase the action delay is $d_{max} + 2g_g$ after the send time. In a system without a global timebase the action delay is $d_{max} + \varepsilon$. [Kop90].

We can guarantee all three properties, the consistent order property, the simultaneity property and the temporal order property within a distributed system if all significant events, i.e., the sending and receiving of messages, are restricted to the globally synchronized lattice points of a sparse timebase with a granularity larger than the action delay. Since there may be many different processes in the system with different timing requirements, this lattice is two-dimensional. The x-axis represents the progression of time and the y-axis denotes the process identifications as shown in Fig. 3. We call this timebase an action lattice [PDCS90] and the interval between lattice points --the granularity of the action lattice--g_a. In the present implementation of the MARS architecture, which supports a sparse time-base, the granularity of the action lattice g_a is in the order of a millisecond[Kop89]. Such a system has the following advantages:

(1) The consistent order property is guaranteed since all nodes receive all messages before the next message is sent.

(2) The simultaneity property is guaranteed since all nodes act on the message at about the same time, i.e., at the next lattice point, which is globally synchronized with a precision of Π.

(3) The temporal order property is guaranteed, since the system is g_a-precedent. Messages which are sent simultaneously are sent at the same action lattice point with the same timestamp. Messages which are not simultaneous are sent at different lattice points. The imprecision effect is avoided.

(4) Since at the lattice points no messages are in transit, the system is in a *ground state* [Ahu90]. The internal state of a node is well defined at the ground states. This supports the reconfiguration of the system and the reintegration of nodes.

(5) The testability of the system is improved. Since events are restricted to the action lattice points, the size of the input space is reduced significantly [Sch91]. The test coverage, i.e., the relation between the number of test cases to the number of points in the input space is increased accordingly.

Whenever an action has to be taken, it has to be delayed until the next lattice point of the action lattice. If we do not want to pay this price, we have to introduce agreement protocols, e.g., an atomic broadcast protocol to guarantee consistent ordering. However, the execution of these agreement protocols might take longer than the delay introduced by the action lattice.

A distributed real-time system which supports such an action lattice provides an ideal execution environment for the implementation of real time programs written in a synchronous language, such as LUSTRE [Cas87] or ESTEREL [Ber85]. These languages separate temporal concerns from data-transformation concerns by assuming that every program step is executed within a granule of the action lattice. Since no significant event can occur within such a granule, the program execution is taking place in a "timeless" environment.
If we are only interested in the temporal order property (and not in the simultaneity property), then it is sufficient to introduce a sparse time lattice with a lattice interval of $2g_g$, where g_g is the granularity of the global timebase. However, we still may have to wait for the full action delay until the system becomes stable.

Process identifier								
A	o	o	o	o	•	o	o	o
B	o	•	o	•	o	•	o	o
C	o	o	•	o	o	o	•	o
D	•	o	o	•	o	o	•	o
E	o	•	o	o	•	o	o	o
F	•	o	o	•	o	o	o	•

---> real-time

• an event occurs at this lattice point
o no event occurs at this lattice point

Fig. 3: Space/time lattice

6. COMMUNICATION WITH THE ENVIRONMENT

Whereas it is possible to restrict the points of event occurrences within the sphere of control of the computer system, such a restriction cannot be enforced on the events happening the environment. These events happen on a dense timebase. Because of the imprecision effect it is not even possible to guarantee that two observations of a single external event will carry the same timestamp, even if the clocks are synchronized. It cannot be ruled out that the external event happens during a global tick interval.

One solution to solve this problem is the execution of an agreement protocol at the interface between the distributed computer system and the environment. All nodes observing a part of the environment must agree on the point in time and the value of an observation before this observation is released to the rest of the system. These agreed values will reflect the temporal order only if the events are at least $2g_g$ apart, but at least the agreed timestamps are consistent within the system. The quality of clock synchronization constitutes a fundamental limit for the recovery of the temporal order and thus for the causal analysis of events.

In most applications these agreement protocols will be application specific. Knowledge about the physical process, the quality of instrumentation, and the past performance of the sensors will be integrated into these application specific algorithms.

7. CONCLUSION

The slightly varying clock rates in the nodes of a distributed system, as well as the requirement to tolerate faults which are covered by the fault hypothesis, can lead to divergent states in replicated nodes. One solution to this problem is the execution of agreement protocols. In the general case of asynchronous architectures and unrestricted fault-models these agreement protocols can be expensive in time and computational resources. The introduction of additional regularity assumptions simplifies the solutions of many agreement problems. Such assumptions relate to the fault model, e.g., fail-silent nodes and fail-silent communication links, and to the points in time when significant events are allowed to occur. If fail-silent nodes are assumed and if the occurrence of a significant event is restricted to the lattice points of a properly chosen globally synchronized action lattice, then the problem of temporal order and simultaneity of actions can be solved without the execution of agreement protocols. Since, in most cases the execution of an agreement protocol takes longer than the the granularity of the action lattice, we may improve the responsiveness of a system by delaying an action until the next lattice point. This result is counter-intuitive. Another advantage of such a system relates to the faster solution to other agreement problems. For example, the membership problem, is easier to solve if the system is based on a sparse time model[Kop91].

The introduction of an action lattice is in some sense orthogonal to the issue of whether a real-time system is time-triggered or event-triggerd. In both designs the regularity provided by the action lattice will reduce the complexity of the solutions to the agreement problems.

Since the events occurring in the environment cannot be restricted to an action lattice--they occur on a dense timebase--, agreement protocols are still needed at the interface between the distributed computer system and the environment.

8. ACKNOWLEDGEMENTS

This work has been supported, in part, by Austrian Science Foundation Project P8002-TEC and by ESPRIT Project PDCS. Valuable comments on an early version by Mirek Malek from the University of Texas have improved this paper. Many discussions with Kane Kim from the University of California, Irvine, within the PDCS project, in particular with A. Bhattacharyya from the University of York, and within the MARS group at the Technical University of Vienna, in particular with G. Grünsteidl, are warmly acknowledged.

9. REFERENCES

[Ahu90] Ahuja, M., Kshemakalyani, A.D., Carlson, T., A Basic Unit of Computation in Distributed System, Proc. of 10th Distributed Computer System Conference, Paris, 1990, pp.12-19

[Bar90] Barret, P.A., Hilborne, A.M., , Bond, P.G., Verissimo, P., Rodrigues, L., Speirs, N.A., The Delta-4 XPA Extra Performance Architecture, Proc. of FTCS-20, IEEE Press, 1990

[Ber85] Berry, G., Gonthier, G., The synchronous programming language ESTEREL, design, semantics, implementation, Technical Report 327, INRIA, 1985

[Cas87] Caspi, P, Pilaud, D., Halbwachs, N., Plaice, J., LUSTRE, a declarative language for programming synchronous systems, Proc. of the 14th ACM Symposium on Principles of Programming Languages, January 1987

[Cri85] Cristian, F., Aghili, H., Strong R. Dolev,D., Atomic Broadcast: From Simple message diffusion to Byzanthine Agreement, Proc. FTCS 15, Ann Arbor, Mich., June 1985, pp.200-206

[Cri88] Reaching Agreement on Processor Group Membership in Synchronous Distributed Systems, Proc. FTCS 18, Tokyo, japan June 19888, pp. 206-211

[Dav79] Davis, C.T., Data Processing Integrity, in: Computing Systems Reliability, ed. by T. Anderson and B. Randell, Cambridge University Press, 1979, pp.288 - 354

[Kop87] Kopetz, H., Ochsenreiter V., Clock Synchronization in Distributed Realtime Systems, IEEE Transactions on Computers, August 1987, pp.933-940

[Kop89] Kopetz, H., Damm, A., Koza, C., Mulazzani, M., Schwabl, W., Senft, C., Zainlinger, R., Distributed Fault-Tolerant Realtime Systems: The MARS Approach, IEEE Micro, Vol. 9, Nr. 1, pp., 25-40, Febr. 1989

[Kop90] Kopetz, H., Kim, K.,Real-Time Temporal Uncertainties in Interactions among Real-Time Objects, Proc. of the 9th IEEE Symp. on Reliable Distributed Systems, Huntsville, Al, Oct. 1990

[Kop91] Kopetz, H., Gruensteidl, G., Reisinger, J., Fault-Tolerant Membership Service in a Synchronous Distributed Real-Time System, in: Dependable Computing for Critical Applications, ed by A. Avizienis and J.C.Laprie, Springer Verlag, Vienna, pp. 411-430

[Lam78] Lamport, L., Time, Clocks and the Ordering of Events in a Distributed System, Comm. ACM, Vol. 21, Nr. 7, July 1978, pp. 559-565

[Lam84] Lamport, L., Using Time instead of Timeout for Fault-Tolerant Distributed Systems, ACM Trans. on Programming Languages and Systems, Vol.6, Nr. 2, April 1984, pp.254-280

[PDCS90] Specification and Design for Dependability, Esprit Project Nr. 3092 (PDCS: Predictably Dependable Computing Systems), 1st Year Report, LAAS, Toulouse, 1990, Vol. 2, p.2.1-2.17

[Ray90] Raynal, M., Order Notions and Atomic Multicast in Distributed Systems: A short survey, Proc. Workshop on Future Trends in Distributed Computing Systems, IEEE Press, Cairo, Sept. 90

[Rei57] Reichenbach, H., The philosophy of space and time, Dover, New York, 1957,

[Sch91] Schutz, W., On the Testability of Distributed Real-Time Systems, Proc. of the 10th Symposium on Reliable Distributed Systems, Pisa, Italy, Sept. 1991, pp.52-61.

[Ver89] Verissimo, P., Rodrigues, L, Order and Synchronism Properties of Reliable Broadcast Protocols, Technical Report RT/66-89, INESC, Lisboa, Portugal

[Wit90] Withrow, G.J., The Natural Philosophy of Time, Clarendon Press, Oxford, 1990