On the limitation of the Global Time Assumption in Distributed Systems

(Extended Abstract)

Uri Abraham*
Dept. of Mathematics and Computer Science
Ben-Gurion University
Beer-Sheva, Israel

Shai Ben-David †
Dept. of Computer Science
Technion
Haifa, Israel

Shlomo Moran ‡
Dept. of Computer Science
Technion
Haifa, Israel

Abstract

An ongoing debate among theoreticians of distributed systems concerns the global time issue. The basic question seems to be to what extent does a model with global time reflect the 'real' behavior of a distributed system. The assumption of the existence of global time simplifies the analysis of distributed algorithms to an extent that makes it almost irresistible. On the other hand it should be clear that when the operations discussed are of time duration that is comparable to that of the time needed for a signal to pass between different components of the system, then such an assumption is unrealistic. The debate is of a somewhat philosophical nature mainly because, so far, there were no known examples of faulty conclusions caused by excessive use of the global time model.

In this note we demonstrate, for the first time, a protocol that is guaranteed to perform well as long as it is run in a system that enjoys the existence of global time, yet it may fail in some other conceivable circumstances. The protocol is very simple and is often used as a basic step in protocols for mutual exclusion.

By the results of [5], such a phenomenon could not have occurred had we been running our protocol on one of the three common types of shared registers (safe, regular or atomic). Another contribution of this work is the introduction of a new class of shared memory registers - 'weakly regular registers'. Our weak regularity is a natural variant on Lamport's definition of regularity of registers.

*email: abraham@bungus.ac.il
†email: shai@cs.technion.ac.il
‡email: moran@cs.technion.ac.il

1 Introduction

The global time axiom is very appealing as it simplifies the analysis of distributed protocols. The use of the global time assumption (or some equivalent assumption) for the analysis of distributed protocols is so common that one can hardly find any example of work in this area that does not take this route either explicitly or implicitly (the outstanding counter example being the work of Leslie Lamport). The problem is that researchers seem to neglect the need to justify the use of this convenient simplification.

As shown in [5], the use of this axiom is equivalent to viewing all the events in a system as being executed during intervals on a mutual (time) axis. Somewhat intuitively speaking, allowing the use of a global time in the analysis of a system amounts to assuming that an interleaving model can faithfully reflect all that we care about in a run of that system. We wish to focus on the possible costs of relying on this axiom.

The difficulty with the global time assumption is that it clearly fails in some distributed environments. This failure becomes noticeable as one considers operation executions that are quick relative to the communication lags in the system. To demonstrate such a failure Lamport [10] evokes the theory of Special Relativity. Let us offer a different perspective.

When one discusses 'time' in the context of shared memory in a distributed system, the real issue is usually not so much minutes and seconds as it is the causality relations between events. We apply time mostly as a tool, or even just a metaphor, to help analyze causality. By saying " a read execution R precedes a write W" we usually wish to say "R cannot be affected by the value written by W" rather than "If R terminates at 07:00 p.m. then W should not commence before 07:01 p.m.".

Under such an interpretation the arrow relations '$A \longrightarrow B$' and '$A \dashrightarrow B$' (see section 2) should be read as "The outcome of A is available to the process executing B when the execution begins" and "A can causally affect B" (respectively). The global time axiom should now be interpreted as stating the existence of a causal relation, at least in one direction, between any pair of operations in the system.

Viewing the arrow relations this way it is not hard to imagine a distributed system in which the global time axiom fails, especially when one bears in mind that in reality any shared memory register is implemented with the aid of message passing devices (such an example is explicitly described in section 5).

It should therefore come as a surprise that such an unrealistically permissive framework has not led to faulty conclusions when applied to truly distributed systems. A partial explanation to this phenomenon is given by Theorem 10 of [5]. The theorem states that, for a wide class of protocols and specifications, if the protocols meet their specifications in every system that enjoys the existence of a global time then they are guaranteed to do so in any system complying with Lamport's general axioms (as listed in section 2 below).

In other words, although the global time axiom itself may be false, certain types of conclusions derived from it are valid. The class of protocols to which that theorem applies includes all protocols for implementing one type of Lamport's registers by another, as well as all protocols for implementing mutual exclusion that are based on registers in one of these classes.

At this point one could wonder whether all the fuss about the use of the global time assumption is of a purely philosophical nature. Maybe for all questions that we may care about, the conclusions implied by adding this axiom are already consequences of the basic properties of distributed systems.

This paper provides the first clear indication that this is not the case. We present a very simple and natural algorithm that , on one hand, is guaranteed to meet its specification whenever it is run in a system with a global time, and on the other hand, fails in some other setting. It follows, using the results of [5], that a correctness proof for that algorithm exists when the global time axiom is available but not from the set of Lamport axioms without it.

We begin our exposition by a brief presentation of Lamport's axiomatic framework. In section 3 we present a new class of registers which is the basic tool for our main example. Section 4 describes our protocols and provides the proofs of its claimed properties. We conclude by a 'concrete example' of a communication scenario where our protocol fails to achieve what it would succeed in any interleaving run.

2 A short review of Lamport's formal theory

This is a minimal outline of Lamport's theory, the reader is encouraged to consult [9, 10], for elaborate presentation and discussion.

Lamport bases his formal theory on two abstract relations over operation executions. For operation executions A, B, "$A \longrightarrow B$" stands for "A *precedes* B" and "$A \dashrightarrow B$" for "A *can causally affect* B".

A *System Execution* is a triple $< E, \longrightarrow, \dashrightarrow >$ where E is a set of *operation executions* and $\longrightarrow$, $\dashrightarrow$, are binary relations over E. Lamport offers the following axioms:

A_1: $\longrightarrow$ is an irreflexive transitive relation.

A_2: If $A \longrightarrow B$ then $A \dashrightarrow B$ and $\neg(B \dashrightarrow A)$.

A_3: ($A \longrightarrow B$ and $B \dashrightarrow C$) or ($A \dashrightarrow B$ and $B \longrightarrow C$) implies $A \dashrightarrow C$.

A_4: If $A \longrightarrow B \dashrightarrow C \longrightarrow D$ then $A \longrightarrow D$.

A_5: For every A, "$\{B : \neg(A \longrightarrow B)\}$" is finite. (This is assuming all events are terminating.)

An intuition for these axioms can be gained by considering the following model for it:

Let S be a partially ordered set and let E be a collection of non-empty subsets of S.

For $A, B \epsilon E$, define

- $A \longrightarrow B$ iff $\forall x \epsilon A \; \forall y \epsilon B \; (x < y)$ (in the sense of S).
- $A \dashrightarrow B$ iff $\exists x \epsilon A \; \exists y \epsilon B \; (x \leq y)$.

A straightforward checking shows that such models satisfy axioms A_1, A_2, A_3, A_4, and also the following axiom:

A_4^*: If $A \dashrightarrow B \longrightarrow C \dashrightarrow D$ then $A \dashrightarrow D$.

This last axiom was suggested by Anger[4] and by Abraham and Ben-David [2] where a completeness theorem is proved for the above mentioned class of models with respect to $\{A_1, A_2, A_3, A_4, A_4^*\}$.

An important class of models is obtained when S is a linear (=total) ordering. In such a case the system satisfies an additional axiom:

Global Time Axiom : For all A, B, ($A \dashrightarrow B$ or $B \longrightarrow A$).

2.1 The register axioms

On top of the above axioms there are the register axioms that specify the communication mechanism of a system.

Any register is assumed to satisfy the following basic demands:

$B0$: The set of all write operations to a given register is linearly ordered by the $\longrightarrow$ relation.

$B1$: For any read R and any write W to the same register, $W \dashrightarrow R$ or $R \dashrightarrow W$ (or both).

$B2$: A read of a register always obtains one of the values that may be written in the register.

Lamport [10] defines three classes of registers. The classes vary in the behavior they can guarantee when a read operation from a register is executed concurrently with a write operation to the same register.

We shall quote here Lamport's informal description of the types of registers. As this work focuses on one of these types - the regular register - we shall present (in section 3) a detailed formal definition for that class.

Following is Lamport's description of the register classes [10]:

- "The weakest possibility is a *safe* register. In which it is assumed that only a read not concurrent with a write obtains the correct value - that is, the most recently written one. No assumption is made about the value obtained by a read that overlaps a write, except that it must obtain one of the possible values of the register."
- "The next stronger possibility is a *regular* register, which is safe (a read not concurrent with a write obtains the correct value) and in which a read that overlaps a write obtains either the old or the new value."
- "The final possibility is an *atomic* register, which is safe and in which reads and writes behave as if they occur in some definite order."

3 A new class of registers

We define our notion of *a weakly regular register* in terms of the two arrow relations.

Definition: *A register is* weakly regular *if any read operation from it, R, obtains a value written to it by a write operation, W, satisfying the following two conditions:*

1. $W \dashrightarrow R$.
2. *There exist no W' such that $W \longrightarrow W' \longrightarrow R$.*

Intuitively, this is just what one could understand from Lamport's informal definition of regularity. In fact, if one checks the formal definition of such registers in [10], it is easy to see that his definition amounts to demanding that the write W whose value is obtained by R satisfies:

1. $W \dashrightarrow R$.
2. There exist no W' such that $W \longrightarrow W'$ and $\neg(R \dashrightarrow W')$.

Note that the global time axiom implies the equivalence of '$A \longrightarrow B$' with '$\neg(B \dashrightarrow A)$' and therefore, assuming the existence of global time, weak regularity coincides with regularity.

The focal point of Lamport's definition is the notion of concurrency. If a write operation is concurrent with a given read then there may be a difficulty in obtaining the value of either that write or the write preceding it.

The point where we vary from his idea is the definition of such concurrency. Lamport considers '$(A \dashrightarrow B) and (B \dashrightarrow A)$' while our definition is justified by claiming that '$(A \dashrightarrow B) and \neg(A \longrightarrow B)$' suffices to indicate some concurrency. Consequently, it is conceivable to consider a register for which, as long as there is no full precedence of a read by a write, this read may still obtain the value that was in the register prior to that last write operation.

The above weaker form of concurrency has similar implications to the definitions of safe and atomic registers as well. We postpone a fuller discussion of this issue to the full version of this paper.

4 The protocol

The following protocol is for a system with two processors, each *owning* a register into which only the owner can write, we allow both processors reading access to both of the registers.

We assume both processors hold an initial value 0. The protocol for each process is as follows:

1. *Write* 1 *to your register.*
2. *Read the other process register.*

Consider the following statement:

$*$ "If both the above registers are weakly regular then, upon termination of the protocol by both processes, at least one process have read the value 1"

This statement is a simple correctness claim. We shall see that it is provable on the basis of the global time axiom and yet it may fail in some system execution (in the sense of section 2).

Lemma 1 *: If the global time axiom holds then the claim $*$ holds.*

Proof: : Let us denote by W_i^0, W_i, R_i the operations of processor i (i.e. the initial 'write 0' the 'write 1' and the 'read' respectively). For both $i = 0$ and $i = 1$ we have $W_i^0 \longrightarrow W_i \longrightarrow R_i$. Let us consider two cases:

If $R_0 \dashrightarrow W_1$ then by axiom A_4 we have $W_0 \longrightarrow R_1$ and then by the weak regularity R_1 is bound to obtain the value written by W_0 - namely 1.

On the other hand, if $\neg(R_0 \dashrightarrow W_1)$ then we apply to the global time axiom. Assuming the global time axiom weak regularity is as strong as regularity (or, if one wishes, $W_1 \longrightarrow R_0$), and therefore R_0 is bound to obtain the value written by W_1 - namely 1.

Note that we have actually shown that our protocol is correct as long as it is run on regular registers. The only role of the global time was to turn the weakly regular registers into regular ones.

Lemma 2 *: There exists an execution system for which * fails.*

Proof: : We shall use the notation of the previous lemma. Let us define the arrow relations among our operations. We assume as before $W_i^0 \longrightarrow W_i \longrightarrow R_i$ for both $i = 0$ and $i = 1$. We shall also impose a $\dashrightarrow$ from every write to every read and whenever it is implied by a $\longrightarrow$ relation. No further arrow relations hold in our system. It is straightforward to check that what we've just defined complies with the axioms $\{A_1, A_2, A_3, A_4, A_4^*, A_5, B0, B1\}$ and therefore is a legitimate system execution. Still, as we have for both $i's$ $\neg(W_i \longrightarrow R_{1-i})$, both read operations are allowed to return the values written by the W^0's without violating the weak regularity of the registers.

5 Example

We describe below a simple model in which the protocol discussed in the previous section indeed fails to satisfy statement $*$. Rather than settle for a formal definition of a 'System Execution' as above, we wish to demonstrate how such a model can result from some conceivable implementation of shared-memory registers by a message-passing design. This design seems to reflect realistic physical constraints, and hence our example implies that the limitation of the global time assumption demonstrated in Lemma 2 cannot be ignored in practical considerations.

We start by describing the message-passing net, in this level of granularity we do assume the existence of a global time. It may be thought of as the real physical time (we assume that differences in the physical clocks of the different processes are negligible). It is only when we pass to the shared-memory representation of the system that the global time axiom fails. As noted in the introduction, at that level of abstraction, the global time assumption should be viewed as a statement about causality relations rather than about physical time.

The system consists of two processes, P_0 and P_1, and two registers, G_0 and G_1. Register G_i is owned by process P_i. Process P_i and register G_{1-i} are *located* at location L_i. We assume that there is a global time in the system, and that each read or write operation is subject to a non-negative delay, that depends on the locations of the corresponding process and register: an operation of process at location L_i at register at location L_j is subject to delay $\delta_{i,j}$, where $\delta_{i,i} = 0$ (this model reflects a situation where a process can read from its read-only register with no delay, but operations to its read/write register requires communication time).

For simplicity, we assume that a write operation of process P_i (at location L_i) to register G_i (at location L_{1-i}) takes zero time at the location of P_i. Thus, such an operation which occurs at time t in P_1 (meaning that at this time P_1 transmit the value to be written at G_1), occurs at time $t + \delta_{1,0}$ in G_1 (meaning that at this time the value is actually written at G_1).
A read operation of P_1 from G_i ($i = 0, 1$) is subject to similar rules. Such an operation consists of transmitting a request signal from P_1 to G_i, and then transmitting the value in G_i back to P_1. When such a read operations is initiated at time t at P_1, it occurs at G_i and terminates at P_1 at the same time, $t + \delta_{1,i}$. The value read by P_1 is the one written in G_i at time $t + \delta_{1,i}$. The write and read operations of process P_0 are defined in a similar way.
We also assume, as in [10], that the registers hold some initial value, by assuming that the first write operation to each register precedes all read operations to that register. For simplicity, we assume that this initial write operation is subject to no delay.

We now turn to the shared-memory representation of the system. The operations at this level are the reads and the writes of the two participating processes. When we define the arrow relations in the system we wish to reflect the possible causality relationships that are implied by the implementation model described above.

These guidelines imply the following arrow relations: For operations A and B on the same register and which are subject to delays δ_A, δ_B respectively, let us denote by $[t_A, t'_A]$ and $[t_B, t'_B]$ the time intervals along which A and B occur (in the global clock of the massage-passing implementation model). Then $A \longrightarrow B$ if $t'_A + \delta_A < t_B$, and $A \dashrightarrow B$ if $t_A + \delta_A \leq t'_B$. As for operation executions of the same processor, we impose the demand that the $\longrightarrow$ relation induces a total order among the events of any process.

We leave to the reader the straight forward, yet crucial for our claims, checking that the system thus defined is exactly the system described in the proof of lemma 2 of the previous section (and therefore, it satisfies the axioms $\{A_1, A_2, A_3, A_4, A_4^*, A_5, B0, B1\}$ as well as the weak regularity axiom for both registers).

Figure 1 below depicts an execution of the protocol of the previous section in the system described above, in which both processes read 0, and hence statement $*$ does not hold. We assume in this example that $\delta_{1,2} = \delta_{2,1} = 2$. (i.e., a write of P_i that occurs at time t in P_i occurs at time $t + 2$ in G_i).

time	P_0	G_0	P_1	G_1
0	W_0^0	W_0^0	W_1^0	W_1^0
1	W_0		W_1	
2	R_0	R_1	R_1	R_0
3		W_0		W_1

Figure 1

For $i = 1, 2$, W_i^0 is the initial write operations that write 0 to G_i. W_i, R_i stands for write (read) operation of P_i. Each entry indicates the time and location of the operation (thus, W_0 occurs in P_0 at time 1, and in G_0 at time 3). It is easy to see that both read operations return the value 0.

We conclude this section by noting that a protocol which satisfies statement $*$ for the above specific system is obtained by forcing each process P_i to read its own register G_i after it writes to it, and only then to read the other process' register. This additional read operation prevents P_i from reading R_{1-i} before the value of R_i is 1 (in global time!), which implies that $*$ must hold. The protocol for P_i is, therefore:

1. *Write* 1 *to your register.*
2. *Read your own register*
3. *Read the other process register.*

References

[1] Abraham, U., “On interprocess communication and the problem of common atomic registers”, manuscript, 1989.

[2] Abraham, U., and Ben-David, S., “Informal and Formal Correctness Proofs for Programs”, manuscript, November 1987.

[3] Avraham, U., Ben-David, S., and Magidor, M., “On Global Time and Inter-Process Communication”, Semantics for Concurrency, M.Z. Kwiatkowska, M.W. Shield, and R.M. Thomas (Eds.), Springer - Verlag, July 1990, 311-323.

[4] Anger, F. D., “On Lamport’s interprocess communication model”, ACM Transactions on Programming Languages and Systems, Vol. 11 No. 3, July 1989, 404-417.

[5] Ben-David, S., “The global-time assumption and semantics for concurrent systems”, Proceedings of the 7th Annual ACM Symposium on Principles of Distributed Computing, ACM Press, 1988, 223-232.

[6] Ben-David, S., “On Lamport’s Shared Memory Registers”, in preparation.

[7] Van Benthem, J. F. A. K., “ Time, Logic and computation”, in Bakker, Roever and Rozenberg (Eds), Linear Time, Branching Time and partial Order in Logics and Models for Concurrency, pp.1-49, Springer, Berlin, 1989.

[8] Fishburn, P.C., “Interval orders and interval graphs”, Wiley, New-York (Wiley-Interscience series in discrete mathematics), 1987.

[9] Lamport, L., “ The mutual Exclusion Problem: Part I–A Theory of interprocess Communication; Part II–Statements and Solutions”, J. of the A.C.M., Vol 33, No.2(1986), pp. 313-326.

[10] Lamport, L., “On Interprocess Communication, Part I: Basic formalism, Part II: Algorithms”, Distributed Computing, Vol. 1(1986), pp. 77 - 101.

[11] Wiener, N., “A contribution to the theory of relative position”, Proc. Camb. Philos. Soc. 17(1914), pp.441-449.