Exclusive Access to Resources in Distributed Shared Memory Architecture

Ludwik Czaja¹,²

¹ Institute of Informatics, The University of Warsaw
² University of Economics and Computer Science Vistula in Warsaw
lczaja@mimuw.edu.pl

Abstract. A protocol of mutual exclusion with FIFO discipline is devised for distributed systems with Distributed Shared Memory (DSM) and without any central server. To this end, replication of data - a principal feature of DSM is exploited. Some data consistency is discussed.

1 Introduction

The primary goals of distributed system designers is to make it possibly handy and possibly efficient. The goals, however, are usually in mutual contradiction to be achieved. Examples are communication between autonomous computers (often with different architecture) and synchronization of access to shared data where, because of absence of a global clock and global memory, users are compelled to struggle with technical details entirely unrelated to their task. Such are, for instance, exclusive locks, timestamps, forward and backward validation in transactions, time of local clocks compensation for establishing event precedence, etc. - on the synchronization side. Also many devices and mechanisms on the communication side, like data marshalling, before their dispatch and after reception, reliability measures of message delivery, layer protocols, etc. Various language and operating system constructs with their implementation mechanisms, both hardware and software, have been devised to alleviate this burden. For the common users (perhaps not software implementors), the Remote Procedure Call (RPC) [2] and Distributed Shared Memory systems (DSM) [12] for instance, are aimed at giving impression to work with their computer in isolation, but with some outside data as if localised in the private computer. An attempt to apply the DSM arrangement to a certain solution of mutual exclusion task is undertaken in this paper.

2 Logical Time and Global Timestamps

An essential feature of distributed systems is absence of common clock for all computers, the global time provided by e.g. an external time service. What is needed is an information about a precedence of some events over some others: their partial ordering only. Assuming that computers work sequentially, the total ordering between events in each of them is determined by their local clocks.
To determine a partial ordering of events in the whole system, it is necessary to specify what kind of events in different computers should be ordered and to establish their precedence (we say "computers", not "processes", neglecting a single computer’s capability to run many processes in parallel). Since in distributed systems message passing is the essential activity, we take sending and reception of messages as events to be ordered, so that sending precedes reception of a single message. Following [9] and [14] (with a slight modification) let us assign rational numbers to the events by a mapping $C : E(S) \rightarrow \text{RAT}$ (called a logical clock) so that $x \rightsquigarrow y \implies C(x) \leq C(y)$. The value $C(x)$ is a logical time of $x$. For $x, y \in E(S)$ let us take two auxiliary binary relations in $E(S)$: $\text{process} \xrightarrow{x} \text{message}$ and $\text{message} \xrightarrow{y} \text{process}$ as primary notions with interpretation:

- If $x$ and $y$ occur in the same process performed by a computer and $x$ precedes $y$ in time or $x = y$ then $x \text{ process } y$.
- If $x$ is sending message by a computer and $y$ - a reception of the same message by another computer then $x \text{ message } y$.

A (weak) precedence $\rightsquigarrow \subseteq E(S) \times E(S)$ is the least relation with:

(i) $(x \text{ process } y \lor x \text{ message } y) \implies x \rightsquigarrow y$;
(ii) $(x \rightsquigarrow y \land y \rightsquigarrow z) \implies x \rightsquigarrow z$.

This is a partial order (one may assume reflexivity and antisymmetry of $\text{process}$).

Events $x, y$ are independent (concurrent) iff $\lnot(x \rightsquigarrow y \lor y \rightsquigarrow x)$. In order to reasonably link the $\rightsquigarrow$ relation with time precedence, it is required that $x \rightsquigarrow y \implies C(x) \leq C(y)$. So, to make it physically possible, compensations of local (physical) clocks are necessary: if a sender sends a message accompanied by its local time of dispatch and a receiver receives it earlier according to its local time, then the latter must put forward its clock to the time a little bit later than time received from the sender. This would make the last implication implementable and we assume that the logical clock $C$ measures the compensated time. Obviously, the reverse implication is not true (for concurrent events $C(x) \leq C(y)$ does not imply $x \rightsquigarrow y$) and the set of all such values, called timestamps of events, is totally (linearly) ordered. Moreover, it may happen that $C(x) = C(y)$ but $x \neq y$ if $x, y$ are concurrent. Hence, there is no total order $\sqsubseteq \subseteq E(S) \times E(S)$ such that $x \sqsubseteq y \iff C(x) \leq C(y)$ - $\sqsubseteq$ cannot not be antisymmetric. To find such total order and retain implication $x \rightsquigarrow y \implies C(x) \leq C(y)$, the notion of the timestamp is enriched by a number of computer in which the timestamped event occurs. So, the computers are numbered $1, 2, ..., n$ and let $p_x$ denote the number of computer in which the event $x$ occurs (any event identifies a computer in which it occurs). A pair $(C(x), p_x)$ is called a global timestamp of event $x$ and let $\preceq$ denote a relation between global timestamps defined as $(C(x), p_x) \preceq (C(y), p_y)$ iff $C(x) < C(y)$ or if $C(x) = C(y)$ then $p_x \leq p_y$. Obviously $\preceq$ is a total order and $x \rightsquigarrow y \implies (C(x), p_x) \preceq (C(y), p_y)$. Finally, let $\sqsubseteq \subseteq E(S) \times E(S)$ be defined as $x \sqsubseteq y \iff (C(x), p_x) \preceq (C(y), p_y)$, so, we have the global timestamps and events linearly ordered, what will be used in the next
section. For some notational facilitation let us represent global timestamps by rational numbers, taking arbitrary 1-1 mapping \( f : RAT \times N \to RAT \) (e.g. by scaling logical times \( C(x) \) to the open sector \((0,1)\) of rational numbers and represent global timestamps as decimal fractions with \( p_x \) and \( C(x) \) as integer and fractional parts respectively). Since we use the global timestamps only in what follows, we say "timestamps" instead of "global timestamps" and for simpler notation (examples in Fig. 3 and 4) we use natural numbers for them.

3 Mutual Exclusion in DSM Systems

Another essential feature of a distributed (in contrast to a centralised) computer system is absence of the physical memory common to each CPU. This makes data transmission through network lines between computers necessary. But this imposes some burden on a user to code/decode the data to a suitable form and to employ some send/receive program constructs by means of communication protocols. To alleviate this burden was the objective of two known ideas: Remote Procedure Call [2] and Distributed Shared Memory [12]. Adaptation of data to be transmitted in these solutions is delegated to some software modules, possibly assisted by hardware facilities. In the DSM implementation, copies of physical memory fragments of one computer into another are made, somewhat like in the paging or segmentation mechanism of data movement between internal and external storage. But here we abstract from such technicalities and treat DSM as a sort of a virtual memory - the address space - accessible to all computers. However, the replication of memory fragments creates problems of data consistency. Indeed, in a period of time between writing a value to a cell in one physical memory and reading it from its replica in other physical memories, storing a different value may occur in the cell without its replication during the period. If this never happens, we say that the DSM enjoys the strict consistency: every reading fetches a "recently" written value. But strict consistency is unrealistic because in distributed systems data transmission time is unpredictable and not all events are seen by all computers equally time-ordered: the adverb "recently" is here meaningless. That is why a number of relaxed models of consistency have been introduced (cf. [8, 15, 4, 13, 3, 10, 5, 7, 11, 6, 1]). A view of a distributed system with DSM on some abstraction level sufficient for the purpose of this paper is in Fig.1.

It is assumed that:

- the computers work in parallel and are numbered 1, 2, ..., \( n \);
- reading and writing is governed by the memory manager of each computer; it does this by send/receive message mechanism hidden from the user who envisions usage of the data as if it were stored in the physical memory of the private computer;
- each message is accompanied by a timestamp, a number representing logical time compensated as described in Section 2 and the sender’s number; no two different messages are accompanied by the same timestamp and no two different timestamps accompany the same message;
there is one critical section assuring mutual exclusive usage of a resource by the computers;
• no hardware or software failure happens during performing the mutual exclusion mechanism described here;
• the lapse of time in the system is measured by progress of the timestamp, thus, phrases "before", "after", "later", etc. are used in the sense of this measure;
• computer number \( i \) keeps a vector \( r_i = [r_{i1}, r_{i2}, ..., r_{in}] \) of variables \( r_{ij} \) allocated in its physical memory; it stores a timestamp in the component \( r_{ii} \) when requesting for the critical section; the same timestamp cannot be stored in more than one \( r_{ii} \); in the component \( r_{ij} \) for \( j \neq i \), computer number \( i \) stores a copy of a timestamp stored by the computer number \( j \) in \( r_{jj} \) when it requested for the critical section; in Figures 1, 3, 4 variables \( r_{ii} \) are typed in black bold and \( r_{ij} \) for \( j \neq i \) - gray;
• initially all the variables \( r_{ij} \) \( (i, j = 1, 2, ..., n) \), called request-time registers, contain value \( \infty \) greater than any number;
• by \( \text{min}(r_i) \) is denoted the minimal (the least in fact, since the timestamps are totally ordered) value of all the components of the vector \( r_i \).

The mutually exclusive usage of the protected resource by the computers is accomplished by the protocol depicted as a transition graph in Fig.2 with the following descriptions:

• colours of states:
  \( W \) - white: execution of local (not critical) section
  \( B \) - blue: request for critical section
  \( Y \) - yellow: refusal of critical section (waiting state)
R - red: execution of critical section
P - pink: release of critical section;

• set of local states of the computer number i: $S_i = \{W_i, B_i, Y_i, R_i, P_i\}$;
• state of the computer number i: $Q_i \in S_i$;
• state transit from $Q_i$ to $Q'_i$ (written: $Q_i \rightarrow Q'_i$) is given by the transition graph:

Fig. 2.

• set of global states of the whole system: $S \subseteq S_1 \times S_2 \times ... \times S_n$ satisfying the mutual exclusion condition: if $[Q_1, Q_2, ..., Q_n] \in S$ then $\neg \exists i, j :(i \neq j \land Q_i = R_i \land Q_j = R_j)$;
• state of the whole system: vector $Q = [Q_1, Q_2, ..., Q_n] \in S$ where $Q_i \in S_i$;
• initial state of the whole system: $[W_1, W_2, ..., W_n]$ with all $r_{ij}$ containing $\infty$ in each local state $W_i$;
• state transit from $Q = [Q_1, Q_2, ..., Q_n] \in S$ to $Q' = [Q'_1, Q'_2, ..., Q'_n] \in S$ (written $Q \Rightarrow Q'$): there exist $Q_i \in S_i$ and $Q'_i \in S_i$ satisfying $Q_i \rightarrow Q'_i$ and if $\neg Q_j \rightarrow Q'_j$ then $Q_j = Q'_j$.

Remark 1. In the states $B$ and $P$ transmission of data between the computers take place, so, the computer-receiver, say computer number $i$, when filling its timestamp’s vector $r_i$ passes through $n$ sub-states, each for one component of $r$. 
In order not to enlarge the protocol in Fig.2 we contracted all these sub-states to these two. In Fig.3 all transmissions are completed when the computer is in B or P, while in Fig.4 transmissions are spread over several states: it shows the possible reason of DSM’s inconsistency when the original \( r_{ii} \) (in computer number \( i \)) and its copy \( r_{jj} \) (in the computer number \( j \)) have different values.

It follows from the transition graph in Fig. 2 and our assumptions that for any computer number \( i \):

1. Not all states \( Q \in S_1 \times S_2 \times ... \times S_n \) are reachable from the initial state: unreachable are those with more than one red component (see also Corollary 1).

2. Writing a timestamp to register \( r_{ii} \) proceeds only on transit \( W_i \to B_i \) and \( r_{ii} \) retains this value until the transit \( R_i \to P_i \) takes place.

3. Writing \( \infty \) to \( r_{ii} \) proceeds only on transit \( R_i \to P_i \). Thus, it follows from (2) that \( r_{ii} \) decreases its value only on transit \( W_i \to B_j \).

4. Reading registers \( r_{ij} \) \((j = 1, 2, ..., n)\) to find \( \min(r_i) \) proceeds only when the computer is in the states \( B_i \) or \( P_i \) and takes place only once when passing from \( W_i \) to \( W_i \) in one cycle. Thus, finding that \( r_{ii} > \min(r_i) \) in the state \( B_i \) puts the computer in the waiting state with no need to compute \( \min(r_i) \) again and again: no busy wait. Note that this inequality is tested after transmission of all data to the vector \( r_i \) is over, while storing the timestamp into \( r_{jj} \) is local, thus (we may assume) instantaneous.

5. The order of entering computers into the critical section depends on the value of timestamp when the request is issued but not on the order of reading \( r_{jj} \) registers to find \( \min(r_i) \). Consider a computer number \( i \) requesting for the critical section and reading all \( r_{jj} \) registers to compute \( \min(r_i) \). Obviously \( \min(r_i) \) never decreases. Increase of \( \min(r_i) \) does not take place if transit \( R_j \to P_j \) by a certain computer number \( j \) occurs before reading value of \( r_{jj} \) by the computer number \( i \) since it cannot compute \( \min(r_i) \) yet ("before" in the sense of relation \( \preceq \) between timestamps of these events, but due to the representation of timestamps as numbers, "\( \preceq \)" is isomorphic with "\( \leq \)" between numbers). But if after, then \( \min(r_i) \) remains the same too. In any case, computer number \( i \) does not change its state unless - in the first case - computer number \( j \) finds that \( r_{ji} = \min(r_{jj}) \) and activates computer number \( i \) before it completed filling up its vector \( r_i \).

6. Activation of a kth computer by the ith in the state \( P_i \) starts when the latter completed filling up its vector \( r_i \) with timestamps the remaining computers stored in their cells \( r_{jj} \) (when requesting for the critical section) and it found \( r_{ik} = \min(r_i) \) to hold. The activation occurs either if the kth is in the state \( Y_k \) or in \( B_k \). If in \( Y_k \) - it means that it completed filling up its vector \( r_k \) with timestamps stored by the remaining in their \( r_{jj} \) cells on requests for the critical section. If in \( B_k \) - it means that the kth did not complete this yet, but nevertheless \( r_{kk} < r_{jj} \) \((j \neq k)\) because \( r_{ii} = \infty \) and each computer apart from the ith requested later than the kth (otherwise not the kth but certain lth with \( r_{ll} < r_{kk} \) would be activated before the kth) - see state 4 in Fig.4.
These points are illustrated by a fragmentary run of a distributed system shown in Example 1.

**Example 1.** A one-cycle run of a system of 4 computers with Distributed Shared Memory is presented in Fig.3. This is the sequence of states \[ W_1, W_2, W_3, W_4 \Rightarrow B_1, W_2, B_3, W_4 \Rightarrow Y_1, W_2, R_3, B_4 \Rightarrow \]
\[ Y_1, B_2, R_3, Y_4 \Rightarrow Y_1, Y_2, R_3, Y_4 \Rightarrow Y_1, Y_2, P_3, Y_4 \Rightarrow R_1, Y_2, W_3, Y_4 \Rightarrow \]
\[ P_1, Y_2, W_3, Y_4 \Rightarrow [W_1, Y_2, W_3, R_4] \Rightarrow [W_1, Y_2, W_3, P_4] \Rightarrow [W_1, R_2, W_3, W_4] \Rightarrow \]
\[ W_1, P_2, W_3, W_4] \Rightarrow [W_1, W_2, W_3, W_4]. \] The pictures (numbered in the top-left corners) illustrate successive states the system passes through during the run. The colours of computers represent their (local) states. The directed lines in DSM represent transmission of copies of request-time registers \(r_{jj}\) to \(r_{ij}\) for \(j \neq i\) and the local writing to \(r_{ii}\) (the arrows \(\Leftarrow\) and assignment sign ":=" denote these actions in the programs). The thick short arrows show the current place of control in the computers. The critical sections are between wait and signal statements (the customary names of delimiters of the critical section, not operations on a semaphore, absent in the distributed system).

These considerations lead to the following:

**Corollary 1.** In a distributed system organised as above, the mutual exclusion usage of a resource protected by a critical section is assured. The computers use it in the FIFO strategy in accordance with the event order determined by advancement of the value of timestamp.

**Justification**

As shown in Fig.2 if computers number \(i\) and \(h\) request for the critical section, then both enter the waiting state if \(r_{ii} > \min(r_i)\) and \(r_{hh} > \min(r_h)\) and stay there with unchanged values of \(r_{ii}\) and \(r_{hh}\) until a certain computer, which leaves the critical section, decides that, say, \(r_{ii}\) stores a value smaller than all remaining request-time registers \(r_{jj}\), \((j \neq i)\), thus also \(r_{ii} < r_{hh}\). Then it lets enter the \(i\)th one the critical section, while the \(h\)th remains in the waiting state. If computer number \(i\) requests for the critical section then it immediately enters it if \(r_{ii} = \min(r_i)\) (computation of \(\min(r_i)\) requires fetching copies of all registers \(r_{jj}\) \((j \neq i)\) from remaining computers and during this activity the original values of \(r_{jj}\) may increase only, which would not change the value of \(\min(r_i)\)). But recall also point (6) above.

## 4 Consistency Issues

In general, systems with Distributed Shared Memory are exposed to some undesired situation when a memory cell in one computer and its replica in others, store different values. Because to assure simultaneous identity of these values is not possible since it would require timeless transmission of data, such inconsistency between the original and its copies occurs. In order to cope with this phenomenon in distributed computer systems, a number of relaxed consistency models have been proposed ([10,7,11,5,6,1]). Here, we restrict the problem.
to possible impact of DSM’s inconsistency on the mutual exclusion mechanism proposed in Section 3. That is, to inconsistency between the original $r_{ij}$ in the timestamp’s vector of the $i$th computer and the copies $r_{ji}$ of $r_{ij}$ in other computers. Although the mechanism is susceptible to such inconsistencies, we argue that they do not violate its capability to assure mutual exclusion with FIFO discipline. Indeed, a copy is used only locally, i.e. by its owner-computer, to compute minimal timestamp for making decision whether to wait or enter the critical section or to select a computer to enter the critical section and activate it. Then, regardless of the outcome, the computer replaces all its copied values by $\infty$ and some inconsistencies occur (for instance, in Fig. 3, values of the original $r_{33}$ and all its copies $r_{13}$, $r_{23}$, $r_{43}$ differ in the state 6). Inconsistencies happen
also because of data transfer latency between computers, but they do not cause incorrect behaviour if the protocol in Fig. 2 is applied. To justify this note that for any \( i \) and \( j \):

(a) if \( r_{ij} \neq \infty \) then \( r_{jj} \geq r_{ij} \) (original cannot be less than its copy different from \( \infty \));
(b) if \( r_{ii} > r_{ij} \) then \( r_{ii} > \min(r_i) \) (obvious).

**Proposition 1.** A difference between values of an original \( r_{ii} \) and its copy \( r_{ji} \) may occur during the protocol's in Fig. 2 activity, but this does not violate the FIFO discipline of entering the critical section by computers.

**Proof.** Suppose that in a certain global state, computers number \( i \) and \( j \) are in states \( B_i \) and \( B_j \) and an inconsistency \( r_{ii} > r_{ji} \) occurs. Consider cases:

- if \( r_{ii} > r_{jj} \) and \( r_{ij} \neq \infty \), that is, the \( j \)th computer should outstrip the \( i \)th in competition for the critical section. Then by (a), (b): \( r_{ii} > r_{ij} \) and \( r_{ii} > \min(r_i) \), which in accordance with the protocol in Fig. 4, the \( i \)th computer enters the waiting state \( Y_i \) thus gives way to the \( j \)th.
- \( r_{ii} < r_{jj} \) and \( r_{ji} \neq \infty \), that is, the \( i \)th computer should outstrip the \( j \)th in competition for the critical section. Then by (a), (b): \( r_{ji} < r_{ii} < r_{jj} \), thus, by (b): \( r_{jj} > \min(r_j) \), hence the \( j \)th computer enters the waiting state \( Y_j \), thus gives way to the \( i \)th. Thus the FIFO discipline is ensured. For example, in Fig. 4 inequalities \( r_{33} > r_{44} \), \( r_{44} > r_{43} \) hold in the state 6. Computer number 3 requested for the critical section later than computer number 4, so the 4th will get the resource before the 3rd. Note, that this will take place in spite of \( r_{43} < r_{44} \) since there is no computer number 3 with \( r_{43} = 0 \) in the state 6.

It may also happen, that the copies are not used even by their owner-computer. This happens when a computer, say number \( i \), is so slow in fetching timestamps from other computers, that before it completes filling up its vector \( r_i \), another computer, on leaving the critical section, would activate the former (for instance, in Fig. 4, the third computer in the state 4 activates the first one before the first completed fetching copies). Such scenarios illustrate that discrepancy between the original and its replica do not affect a correct behaviour of the system.

Let us check how the mutual exclusion mechanism designed in Section 3 for the DSM copes with some models of consistency:

- **Strict consistency**: a value read from a variable should be the same as was written to it "recently" (by a global time, i.e. measured by the timestamp's advancement). That is, no other value was written to this variable between these two operations. In this type of consistency, the read/write operations in each run of the system should be linearizable (totally ordered), i.e. represented as interleaving of instances of these operations. More formally, in every such interleaving, each instance of reading is preceded by an instance of writing with the same value, and no other writing instance is in between. Otherwise the strict
consistency does not take place: this happens when a variable is being read by a process and during the reading another process changes its value (such variable is called "unsafe"). So, if, in the solution to mutual exclusion proposed here, one requires the value of original $r_{ij}$ and its copy $r_{ij}$ to be always one and the same object, then the strict consistency is not observed. This is shown in the sequence of states in Fig. 4: in the state 6, the 4th computer reads (to compute $min(r_4)$) timestamp 10 from the copy $r_{43}$ of $r_{33}$ made (written) in the state 3; but the 3rd computer was fast enough to request again for the critical section with timestamp 30 before the 4th computed $min(r_4)$ with the previous timestamp 10.

Fig. 4.
• **Sequential consistency** [10]: the order of read/write operations (each from its beginning to its end) in every run is such as specified in the programs of the system. Here too, the read/write operations in each run of the system should be linearizable. What is necessary in our solution is that the local operations \( r_{ii} := \text{timestamp}, r_{ij} := \infty \ (j \neq i), r_{ii} := \infty \), in the protocol in Fig. 2 are executed in the order written in the protocol. However the message transmission operations \( r_{ij} \leftarrow r_{jj} \ (j \neq i) \) may occur in each system run in arbitrary order with no violating the FIFO discipline of entering the critical section.

• **Causal consistency** [7]: causally dependent write events occur in the same order in every run. In general, such dependence is specified according to a task or a class of tasks carried out on the distributed system. But it should maintain the ordering defined in Section 2, thus reflected by the global timestamp advancement. Therefore, the mutual exclusion implemented by the protocol in Fig. 2 provides the causal order in the DSM system acting with this protocol.

References