Two Problems with Distributed Systems: Data Access Control and Memory Sharing

Ludwik Czaja

[1] University of Economics and Computer Science Vistula in Warsaw

[2] Institute of Informatics, The University of Warsaw

lczaja@mimuw.edu.pl

Extended abstract

Distributed computer system is a set of autonomous computers connected by a network of transmission channels and equipped with a distributed system software. This is a work environment for a class of tasks of common objective; the network is a communication infrastructure only. Main features of distributed systems dealt with here are:

- absence of common clock for all computers – no global time provided by external service
- absence of physical memory common to each computer - message exchange by the network

A schematic structure of distributed system with distributed shared memory (DSM) is illustrated in Fig. 1 ([Cz 2012]).

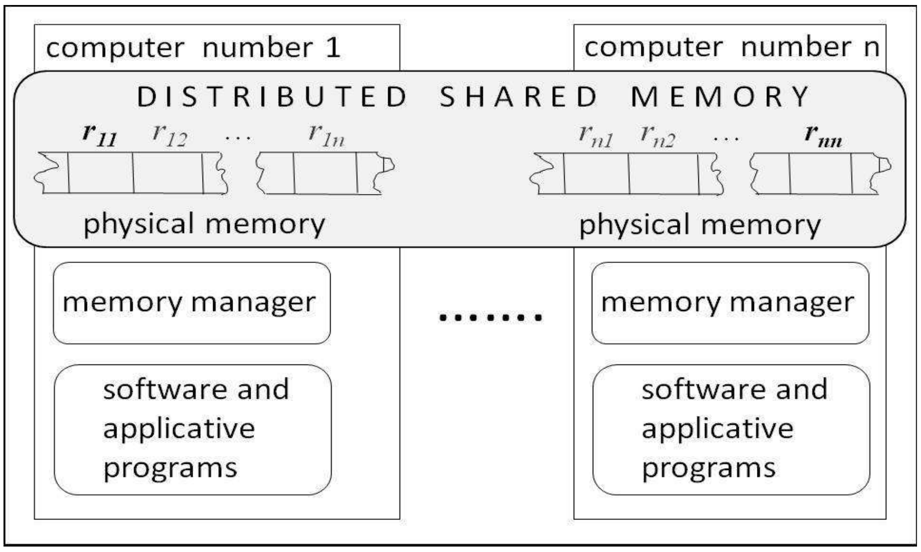

computer number 1
computer number n
DISTRIBUTED SHARED MEMORY
r_{11} r_{12} ... r_{1n}
r_{n1} r_{n2} ... r_{nn}
physical memory
physical memory
memory manager
memory manager
....
software and applicative programs
software and applicative programs

Fig.1

<u>Assumptions and denotations</u>:

- the computers work in parallel and are numbered 1,2,...,*n*
- reading and writing is governed by the memory manager of each computer;

- each message is accompanied by a global timestamp – a pair of computer number and compensated local timestamp;
- there is one critical section assuring mutually exclusive usage of a resource by the computers;
- no hardware or software failure happens during performing the mutual exclusion mechanism;
- computer number i keeps a vector $r_i = [r_{i1}, r_{i2}, \ldots, r_{in}]$ of variables r_{ij} allocated in its physical memory; it stores a global timestamp in the component r_{ii} when requesting for the critical section;
- computer number i stores in r_{ij} a copy of a global timestamp stored by the computer number j in r_{jj} when it requested for the critical section;
- initially all variables r_{ij} contain ∞ ;
- by $min(r_i)$ the least value of the components in r_i is denoted;

Protocol of the mutually exclusive usage of a resource by computer number *i*

Fig. 2 shows a transition graph of the protocol based on global timestamps.

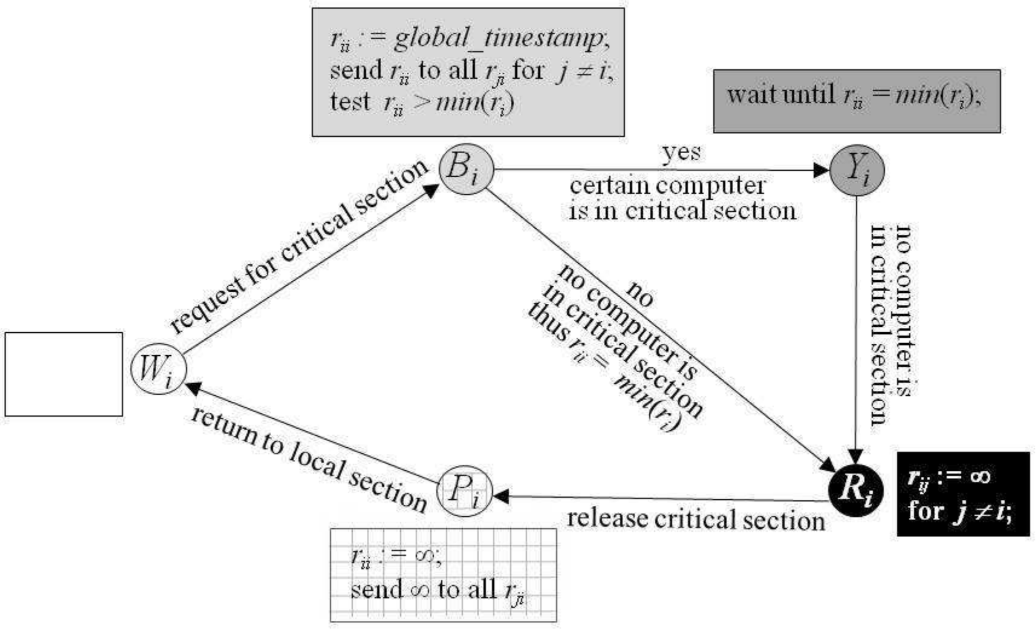

$r_{ii} := global\_timestamp$;
send r_{ii} to all r_{ji} for $j \neq i$;
test $r_{ii} > min(r_i)$
wait until $r_{ii} = min(r_i)$;
B_i
yes
certain computer is in critical section
Y_i
request for critical section
no
no computer is in critical section thus $r_{ii} = min(r_i)$
no computer is in critical section
W_i
return to local section
P_i
release critical section
R_i
$r_{ij} := \infty$ for $j \neq i$;
$r_{ii} := \infty$;
send ∞ to all r_{ji}

Fig.2

- background of states:

W – white: execution of local (not critical) section
B – blue (light gray): request for critical section
Y – yellow (dark gray): refusal of critical section (waiting state)
R – red (black): execution of critical section
P – pink (grid): release critical section

- set of states of computer number *i*: $\boldsymbol{S}_i = \{W_i, B_i, Y_i, R_i, P_i\}$
- state of computer number *i*: $Q_i \in \boldsymbol{S}_i$
- transit $Q_i \longrightarrow Q'_i$: given by the transition graph of the protocol
- set of global states: $\boldsymbol{S} = \boldsymbol{S}_1 \times \boldsymbol{S}_2 \times \ldots \times \boldsymbol{S}_n$ satisfying: if $[Q_1, Q_2, \ldots, Q_n] \in \boldsymbol{S}$ then $\neg\exists i,j: (i \neq j \wedge Q_i = R_i \wedge Q_j = R_j)$
- state of the whole system: $Q = [Q_1, Q_2, \ldots, Q_n] \in \boldsymbol{S}$
- initial state: $[W_1, W_2, \ldots, W_n]$ with $r_{ij} = \infty$ in each local state W_i
- transit $Q \rightrightarrows Q'$ there exist $Q_i \in \boldsymbol{S}_i$ and $Q'_i \in \boldsymbol{S}_i$

  with $Q_i \longrightarrow Q'_i$ and if $\neg Q_j \longrightarrow Q'_j$ then $Q_j = Q'_j$

Example

$[W_1,W_2,W_3,W_4] \rightrightarrows [B_1,B_2,B_3,W_4] \rightrightarrows [Y_1,Y_2,R_3,B_4] \rightrightarrows [Y_1,Y_2,R_3,Y_4] \rightrightarrows [R_1,Y_2,P_3,Y_4] \rightrightarrows$

$[R_1,Y_2,W_3,Y_4] \rightrightarrows [P_1,R_2,W_3,Y_4] \rightrightarrows [W_1,R_2,W_3,Y_4] \rightrightarrows [W_1,P_2,W_3,R_4] \rightrightarrows [W_1,W_2,W_3,P_4] \rightrightarrows$

$[W_1,W_2,W_3,W_4]$

Attendant circumstances of the DSM usage:

- time consumptive transmission with data marshalling/unmarshalling;

- necessity of time compensation (various frequency of computer clocks);

- overlapping (concurrent) memory accesses;

- possible data replication (cache);

They bring on the following consequences and problems to be tackled:

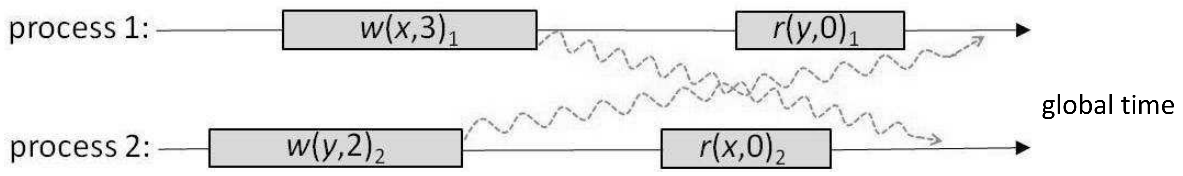
- a message directed to a group of receivers is delivered at different moments; hence memory locations alloted to this message may hold different content at a time;
- reading a memory location may fetch content different from that assigned to this location by its latest (?) (by global time) actualization (writing);
- suspension of the system activity for a period of any memory access would solve the above problems, thus to achieve <u>*data consistency*</u> but is unacceptable by reason of dramaic decrease of system performance; locking of access operations - too restrictive!
- need of a compromise between data consistency and effectiveness;
- weakening of data consistency – when this does not violate requirements of system's main objectives; example: frequent stocked goods actualization (written) of chain-stores

network may be seen (read) different in different countries; data consistency maintenance – unnecessary effort!

- degrees of resignation from consistency to enhance effectiveness – bring on the so-called models of consistency;
- consistency models are usually explained informally – not always univocally; precise understanding them by users of DSM is one of disadvantages of DSM; hence need of their formal description;
- but formal description of consistency models must be based on a formal model of computing thus: (1) on <u>actions</u> and <u>states</u> as primary units of computing (2) on interleaving or non-interleaving („true concurrency”) model of computing;
- common descriptions of consistency models in DSM admit actions <u>read</u> and <u>write</u> from/to memory (fetch and update) as primary units and, informally, non-interleaving computing model;
- but read/write actions are <u>too coarse</u> to formally express memory consistency in the interleaving model along with retaining efficiency provided by true concurrency model;
- partial order of read/write not always is linearizable, i.e. their arrangement in sequence with retaining result of concurrent computation not always possible;

Example of non-linearizability

Notation: $w(z,\alpha)_j$, $r(z,\alpha)_j$ - operations of writing and reading value α to/from variable (memory cell) z by computer j. Initially $x = 0$, $y = 0$. Scenario of not linearizable computing:



position of r/w in processes relative to global time:

$w(x,3)_1 \prec r(y,0)_1$ (1)

$w(y,2)_2 \prec r(x,0)_2$ (2)

position of r/w ensuing from values read by processes:

$r(y,0)_1 \prec w(y,2)_2$ (3)

$r(x,0)_2 \prec w(x,3)_1$ (4)

$\prec$ - global time precedence, $\rightsquigarrow$ - flow of write result

$r(y,0)_1 \prec w(y,2)_2 \prec r(x,0)_2 \prec w(x,3)_1 \prec r(y,0)_1$

$r(x,0)_2 \prec w(x,3)_1 \prec r(y,0)_1 \prec w(y,2)_2 \prec r(x,0)_2$

contradictions!

Remarks, consequences:

- actions $w(x,3)_1$, $r(y,0)_1$, $w(y,2)_2$, $r(x,0)_2$ cannot be interleaved in one sequence so that to retain result of the concurrent computation;
- scheduling protocol for linearization of accesses to DSM not always possible: serial equivalence to each computing scenario impossible;
- difficulty with formal description of consistency issues within the „true concurrency" model; expressions like „seen by processes", etc. are intuitive;
- instead of time-extensive actions read/write, let us take atomic (timeless) events of their beginnings and ends: $\overline{w}(z,\alpha)_j$, $\underline{w}(z,\alpha)_j$, $\overline{r}(z,\alpha)_j$, $\underline{r}(z,\alpha)_j$; finer <u>granularity</u> allows for applying interleaving model.
- assumption: a process completes reading and writing a cell with the same value as commenced; another process may change this value between these events.

Strict consistency

Informally: a value read from any cell (variable) is either the initial value or the one written to it prior to the reading, and between these operations no writing to this cell occured. Note: „prior" is undefined since there is no global system clock. Unrealistic model!

Denotations:

V - set of variables used in programs with DSM

D - set of values the variables may assume

R - set of events of the form $\overline{r}(z,\mathrm{a})_j$ and $\underline{r}(z,\mathrm{a})_j$

W - set of events of the form $\overline{r}(z,\mathrm{a})_j$ and $\underline{w}(z,\mathrm{a})_j$

R* - set of all finite sequences of elements from ***R***

W* - set of all finite sequences of elements from ***W***

$\boldsymbol{IL} = \boldsymbol{R}^* \# \boldsymbol{W}^*$ - union of all interleavings of sequences from ***R**** i ***W****

Let: $Q = q_1 q_2 \ldots q_n \in \boldsymbol{IL}$ and $q_i = \underline{r}(x,a)_f \in \boldsymbol{R}$ $(i = 1,2,\ldots,n)$. Q is strictly consistent iff:

- either q_i is the first such event in the interleaving Q
- or there exists $q_k = \underline{w}(x,a)_g \in \boldsymbol{W}$ where $k < i$ and there is no $q_l = \underline{w}(x,b)_h \in \boldsymbol{W}$ such that $k < l < i$ with $b \neq a$.

DSM is *<u>strictly consistent</u>* iff every interleaving generated by the DSM protocols is strictly consistent.

Example of not strictly consistent interleaving: $\underline{w}(x,0)_2$ $\underline{w}(x,1)_3$ $\underline{r}(x,0)_1$ $\underline{r}(x,1)_2$

Memory coherence [D-S-B 1988]

$\boldsymbol{S} = \{P_1, P_2,\ldots, P_N\}$ - system of sequential programs running in computers numbered 1,2,…,N with DSM;

$\boldsymbol{V}_j$ – set of variables of program P_j allocated in the local memory of computer number j;

$\boldsymbol{D}_j$ – set of values the variables may assume;

σ_j: $\boldsymbol{V}_j \rightarrow \boldsymbol{D}_j$ - function being a state of the local memory of computer P_j;

$$\boldsymbol{V} = \bigcup_{j=1}^{N} \boldsymbol{V}_j$$ - set of variables of the system $\boldsymbol{S}$;

$$\boldsymbol{D} = \bigcup_{j=1}^{N} \boldsymbol{D}_j$$ - set of values the variables may assume;

DSM is coherent if the union of functions: $\sigma = \bigcup_{j=1}^{N} \sigma_j$ is a function

σ: $\boldsymbol{V} \rightarrow \boldsymbol{D}$ (global state of DSM), that is: if $x = y$ then $\sigma(x) = \sigma(y)$ for $x, y \in \boldsymbol{V}$

Example: $\sigma_1 = \{(a,0), (b,1)\}$ $\sigma_2 = \{(a,1), (b,1)\}$, $\sigma_1 \cup \sigma_2 = \{(a,0), (a,1), (b,1)\}$

Not coherent memory of the system $\boldsymbol{S} = \{P_1, P_2\}$.

Sequential consistency [La 1979]

Informally: (1) partial order of read/write operations always linearizable; (2) order of events in each interleaving is identical with order of their occurrences during execution in every individual program; (3) all locations of the same variable in DSM contain identical value „seen” by every computer after its actualization (memory coherence).

Let Q_j be a subsequence of an interleaving $Q = q_1q_2\ldots q_n \in \boldsymbol{IL}$ resulted from removal of events different from events issued by program P_j.

Q is sequentially consistent if for every j = 1,2,…,N, events in Q_j occur in the order determined by programmer of the program P_j.

DSM is *sequentially consistent* iff every interleaving generated by the DSM protocols is sequentially consistent and DSM is coherent.

Causal consistency [H-A 1990]

Informally: causal consistency requires that every two causally dependent write actions, be „seen" (readout) in every process in the same order. That is, events q_i, $q_j \in \boldsymbol{W}$, where q_i, is a cause of effect q_j, are „seen" by events q_k, $q_l \in \boldsymbol{R}$ in the same process, then q_k must precede q_l.

Let $\boldsymbol{RW} = \boldsymbol{R} \cup \boldsymbol{W}$ and $\rightsquigarrow \subseteq \boldsymbol{RW} \times \boldsymbol{RW}$ denote a causality relation defined by means of the following primary relations:

(1) $p \xrightarrow{process} q$ iff $p = q$ or p occurs before q in the same process;

(2) $p \xrightarrow{reading} q$ iff event $q \in \boldsymbol{R}$ terminates reading a certain value α of a variable x assigned to x by writing operation terminating with event $p \in \boldsymbol{W}$;

(3) $p \xrightarrow{writing} q$ iff event $p \in \boldsymbol{R}$ terminates reading a certain value α of a variable x needed for computing a value $\beta = f(\alpha)$ assigned afterwards to a variable y by writing operation terminating with event $q \in \boldsymbol{W}$;

(4) events p and q in (2) and (3) may occur in the same or different processes; they are called *cause* and *effect* respectively.

Causality relation $\rightsquigarrow$ is the least relation in the set $\boldsymbol{RW}$ satisfying:

(i) if $p \xrightarrow{process} q$ or $p \xrightarrow{reading} q$ or $p \xrightarrow{writing} q$ then $p \rightsquigarrow q$

(ii) If $p \rightsquigarrow q$ and $q \rightsquigarrow r$ then $p \rightsquigarrow r$

Events p, q are in the relation of *cause* and *effect* if $p \rightsquigarrow q$.

Events p, q are *independent* (concurrent) if neither $p \rightsquigarrow q$ nor $q \rightsquigarrow p$, write then $q \parallel p$

Interleaving $Q = q_1 q_2 \ldots q_n \in \boldsymbol{IL}$ *is causally consistent* wrt. relation $\rightsquigarrow$ if for any two elements q_i, $q_j \in \boldsymbol{W}$ with $q_i \rightsquigarrow q_j$ the following holds:

If there are elements q_k, $q_l \in \boldsymbol{R}$ both belonging to the same process and such that

$p_i \xrightarrow{reading} q_k$ and $p_j \xrightarrow{reading} q_l$ then $p_k \xrightarrow{process} q_l$

But if for some q_i, $q_j \in \boldsymbol{W}$ with $q_i \rightsquigarrow q_j$ reverse succession $p_l \xrightarrow{process} q_k$ holds then Q is *causally inconsistent*.

DSM is <u>*causally consistent*</u> iff every interleaving generated by the DSM protocols is causally consistent.

Causal consistency and inconsistency may be represented by diagrams in Fig. 3:

$$\begin{tikzcd}
q_i \arrow[r, rightsquigarrow] \arrow[d, "reading"'] & q_j \arrow[d, "reading"] \\
q_k \arrow[r, "process"] & q_l
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
q_i \arrow[r, rightsquigarrow] \arrow[d, "reading"'] & q_j \arrow[d, "reading"] \\
q_k & q_l \arrow[l, "process"']
\end{tikzcd}$$

Causal consistency Fig.3 Causal inconsistency

PRAM (Pipelined Random Access Memory) consistency [Lip-Sa 1988]

Informally: weaker than causal consistency model: if some values are written by the same process in a certain order and read (“seen”) by another process, then the reading must take place in the same order as writing

Formal definition is similar to causal consistency: causal dependence $q_i \rightsquigarrow q_j$ should be replaced with $p_i \xrightarrow{process} q_j$ only.

Thus PRAM consistency and inconsistency may be represented by diagrams in Fig. 4.

$$\begin{tikzcd}
q_i \arrow[r, "process"] \arrow[d, "reading"'] & q_j \arrow[d, "reading"] \\
q_k \arrow[r, "process"] & q_l
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
q_i \arrow[r, "process"] \arrow[d, "reading"'] & q_j \arrow[d, "reading"] \\
q_k & q_l \arrow[l, "process"']
\end{tikzcd}$$

PRAM consistency Fig.4 PRAM inconsistency

Weak consistency [D-S-B 1986]

While aforesaid models of data consistency ensure access to DSM by respective protocols supporting a chosen model without user’s intervention, efficiency of the system may sometimes justify such intervention – for the prize of transparency, one of the desirable features of distributed systems. The weak consistency model provides users with the so-called synchronization variables, counterparts of semaphores in centralized systems. Access to them takes place as in the case of sequential consistency model – in any process they occur in the order specified by the program evoking this process. The users are provided with means to create critical sections applying e.g. protocol illustrated in Fig. 2. During execution of such critical section no other process may access data protected by this critical section until their actualization is finished. No data access is permitted until all previous accesses to synchronization variables are completed.

Conclusion

The sample of five mentioned models of data (or memory) consistency is probably the most common in applications. But a formalization and analysis of quite long list of these and other models might propose a challenging research topic. Apart from the discussed above, a list (not exhaustive) of consistency models contain the following: entry, release, scope, process, cache, fork, eventual, session, read-your-write, monotonic-read, monotonic-write. Every one is a sort of a “contract” between the memory management (sub)system and usage of DSM, stating that if the contracted rules are observed then the memory will behave in accordance with the rules accepted by the user.

References

[Cz 2012] Czaja L., *Exclusive Access to Resources in Distributed Shared Memory Architecture*, Fundamenta Informaticae, Volume 119, Numbers 3-4, 2012 s. 265-280

[D-S-B 1986] Dubois M., Scheurich C., Briggs F.A., *Memory Access Buffering in Multiprocessors*, Proc. 13th Ann. Int’l Symp. On Computer Architecture, ACM 1986, pp. 434-442

[D-S-B 1988] Dubois M., Scheurich C., Briggs F.A., *Synchronization, Coherence and Event Ordering in Multiprocessors*, IEEE Trans. Computer, 1988, 21, 2, pp. 9-21

[La 1979] Lamport L., *How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs*, IEEE Trans. On Computers, 1979 C-28, s. 690-691