Università della Svizzera italiana

Faculty of Informatics

USI Technical Report Series in Informatics

Impossibility(?)

Eduardo[1], Fernando[1]

[1] Faculty of Informatics, Università della Svizzera italiana, Switzerland

Abstract

-

Report Info

Published
December 2012

Number
USI-INF-TR-2010-4

Institution
Faculty of Informatics
Università della Svizzera italiana
Lugano, Switzerland

Online Access
www.inf.usi.ch/techreports

1 ...

We consider a system consisting of an unbounded set of clients $\mathscr{C} = \{c_1, c_2, ...\}$, a set of n servers $\mathscr{S} = \{s_1, ..., s_n\}$, a set of p partitions (subsets of $\mathscr{S}$) $\mathscr{P} = \{P_1, ..., P_p\}$ and a set of m variables $\mathscr{V} = \{v_1, ..., v_m\}$. Clients send commands to servers, which execute them. Every variable is stored in some server, and every server stores at least one variable. Every server belongs to a partition; servers in the same partition store the same set of variables.

In a given execution $\mathscr{E}$, every command C comprises a set of accessed variables $\mathscr{V}_C$ (which, in turn, consists of the union of a set of read-variables and a set of written-variables), a set of input values, a start time $C.s$ and a finish time $C.f$ (both in real-time). In $\mathscr{E}$, a single matching response is associated with each command executed. There is a sequential specification for the variables: given a sequence of commands σ, the response matching any command x that reads some variable v must be consistent with the most recent command that writes v and precedes x in σ. A permutation π of all commands in execution $\mathscr{E}$ is legal if π respects the sequential specification of every variable. Execution $\mathscr{E}$ is *linearizable* if there is such a permutation that respects the sequential specification of all variables and also respects the real-time ordering of commands, i.e., given commands x and y in $\mathscr{E}$, if $x.f < y.s$ then $x \prec y$ (meaning "x precedes y") in π.

Using atomic multicast to define command precedence ensures that a legal permutation of commands can be found for every execution $\mathscr{E}$, even if amcast does not define precedence for all pairs of commands (Prove.).

1.1 SSMR

In SSMR, each partition is mapped to a multicast group. If there is only one partition in the system, then replication is full and can be solved trivially by atomic-broadcasting commands to all servers; otherwise, replication is partial and each command C is multicast only to the partitions that store at least one variable accessed by C. Each partition that delivers C executes a subcommand s of C upon delivering C. A command that is delivered by only one partition has only one subcommand. Each subcommand s has its own start and finish times $s.s$ and $s.f$, which depend on when it is executed by the partition. The execution of a command is finished when the execution of every one of its subcommands is finished. Command C is said to be *overlapping* if there is an instant when every subcommand of C is being executed. More formally, if C_{sub} is the set of all subcommands of C, then there exists an instant t such that $s.s < t < s.f$ for every $s \in C_{sub}$.

Proposition 1. (LINEARIZABILITY).
If every command in execution $\mathcal{E}$ of SSMR is overlapping, then $\mathcal{E}$ is linearizable.

Proof. Suppose, by means of contradiction, that there exist two commands x and y, where x finishes before y starts, but $y \prec x$ in the execution. There are two possibilities for this:

i) x and y access some variable in common v

   In this case, at least one partition P_v (which contains v) delivers both x and y. As x finishes before y starts, then P_v delivers x, then y. From the properties of atomic multicast, and since each partition is mapped to a multicast group, no partition delivers y, then x. Moreover, atomic multicast ensures acyclic order, so there are no commands $z_0, ..., z_m$ such that their atomic order is $y \prec z_0 \prec \cdots \prec z_m \prec x$. So, we reached a contradiction in this case.

ii) x and y access no variable in common

   In this case, if there were no other command in $\mathcal{E}$, then the execution of x and y could be done in any order, which would contradict the supposition that $y \prec x$. Suppose, then, that there are commands $z_1, ..., z_n$ such that their atomic order is $y \prec z_1 \prec \cdots \prec z_n \prec x$.

   As $y \prec z_1$, then both commands y and z_1 are delivered by some partition P_1, which executes their subcommands s_y and s_{z_1} in the order of atomic delivery. Therefore, $s_y.f < s_{z_1}.s$. Since $z_1 \prec z_2$, some partition P_2 delivers both z_1 and z_2 and executes their subcommands s'_{z_1} and s_{z_2}, with $s'_{z_1}.f < s_{z_2}.s$. Since every command in the execution is overlapping, there is a time t_1, such that $s_{z_1}.s < t_1 < s_{z_1}.f$ and $s'_{z_1}.s < t_1 < s'_{z_1}.f$. Therefore, $s_y.f < t_1 < s_{z_2}.s$.

   Following similar steps, we can conclude that there are times $t_1, ..., t_n$ such that $s_y.f < t_1 < \cdots < t_n < s_x.s$, where s_x is a subcommand of x, meaning that $y.s < x.f$, which is also a contradiction.

□

1.1.1 Commands and subcommands

In SSMR, every command contains a set ς of operations. Each operation can be:

a) *read*(x): retrieve the value of variable x;

b) *write*(x, v): assign the value v to variable x;

c) some computation, which may depend on some previously read variable.

Every partition P, upon delivering a command C, translates it into a subcommand s, which depends on what variables read and written by C are stored in P. This way, the sequence of operations ς of C is translated into ς_s, which is the sequence of operations that compose s. There are different ways to do such mapping. A simple such way is by having every operation in ς_s as the same as in ς, except in the following cases:

i) If the i-th operation of ς is *read*(v), it translates into:

   (a) If v is stored in P, then P sends v to every partition that delivers C and does not store v.

   (b) If v is not stored in P, it consists of waiting until the value of v has arrived at P.

ii) If the i-th operation in ς is *write*(x, v) and P does not contain v, then P does nothing for the i-th operation.

Besides, to ensure linearizability, SSMR forces all subcommands of each command to overlap. This may be done in a straightforward way. Let $\mathscr{P}_C$ be the set of all partitions that deliver C. Every $P_i \in \mathscr{P}_C$ sends a message sig_i^C to every $P_j \in \mathscr{P}_C$. After doing this, and before finishing the execution of C's subcommand, P_i waits for sig_j^C from every $P_j \in \mathscr{P}_C$. Except for the fact that sending obviously must precede the receiving of such signals, they can be sent and waited for at any time during the execution of the subcommand, e.g., sending may be the first thing to be done, and waiting may be the last one, in order to minimize waiting time.

(What about sending the reply? Who sends it?)

Every partition that executed a subcommand of C replies to the client that issued it, piggybacking any information required to assemble the full reply based on the partial replies sent by the different partitions.