Atomic Memory in BAMP systems

Yosef Goren

On **'Atomic Read/Write Memory in Signature-Free Byzantine Asynchronous Message-Passing Systems'**.
A paper by:
Achour Mostefaoui, Matoula Petrolia, Michel Raynal, Claude Jard

Table of Contents

Why care about implementing registers?

What we get

This implementation provides a reduction from Message Passing models to Atomic Memory models.

What it can be used for

Many distributed algorithms are based on atomic memory; this reduction provides instant implementations of these algorithms in message passing systems.

Examples

- Atomic, multi-writer multi-reader registers
- Concurrent time-stamp systems
- Atomic snapshot scan

Previous Atomic Register Algorithms

BAMP: Byzantine Asynchronous Message Passing

A distributed system of n processes $p_1, p_2, ...p_n$.

Byzantine

A byzantine process is one that acts arbitrarily, it may crash or even send 'malicious' messages to correct processes.
Let t be the number of byzantine processes, we assume $t < \frac{n}{3}$.

Asynchronous

A message sent from p_i to p_j may take any amount of time to arrive.

Signature Free

Many algorithms cope with byzantine processes by requiring them to sign messages, thus requiring assuming cryptographic primitives to be correct, it is not the case here.

Rliable Broadcast Abstraction

Based on a seperate paper, we can use a reliable broadcast algorithm as a 'black box' (in *BAMP* systems).

Guarantees

A reliable broadcast has the syntax '*r_brodcast(m)*', and guarantees that message the *m* arrives at all correct processes eventually.

Atomic Concistency

Atomic Concistency is how we intuitively expect memory to be, it is also known as Linearizability.

A Simple Definition

'for any execution of the system, there is some way of totally ordering the reads and writes so that the values returned by the reads are the same as if the operations had been performed in that order, with no overlapping.'

- 'On Interprocess Communication' (1985), Leslie Lamport.

Single Writer Multiple Reader Registers (*SWMR*)

A single process can write; everyone can read.

Single Writer & Byzantine Processes

If all shared memory can be written by all processes - a single Byzantine process can destroy it.

Local Copies

Each process p_i has Reg_i, but can only write to $Reg_i[i]$.

p_1	p_2	p_3
$Reg_1[1]$	$Reg_2[1]$	$Reg_3[1]$
$Reg_1[2]$	$Reg_2[2]$	$Reg_3[2]$
$Reg_1[3]$	$Reg_2[3]$	$Reg_3[3]$

Local variables

Each process has local variables *reg*, *rsn* - arrays of n, and an additional counter *wsn*.

Initialization

$reg \leftarrow [(init_0, 0), ..., (init_n, 0)]$.

$rsn \leftarrow [0, ..., 0]$.

$wsn \leftarrow 0$.

Write

```
Reg[i].write(v):
    wsn := wsn + 1
    r_brodcast WRITE(v, wsn)
    wait WRITE_DONE(wsn) recived from (n-t) processes
    return
```

Read

```
Reg[j].read():
    rsn[j] := rsn[j] + 1
    r_brodcast READ(v, rsn[j])
    wait untill reg[j].sn is greater than the arguments of
        n-t messages STATE(rsn[j], _) from different processes.
    let (w, wsn) := reg[j]
    r_brodcast CATCH_UP(w, wsn)
    wait CATCH_UP_DONE(j, wsn) recived from (n-t) different processes.
    return w.
```

Message Handling

```
on reciving WRITE(v, wsn) from process j:
    wait (wsn = reg[j].sn + 1)
    reg[j] := (v, wsn)
    send WRITE_DONE(wsn) to process k

on reciving READ(v, rsn) from process k:
    send STATE(rsn, reg[j].sn) to process k

on reciving CATCH_UP(j, wsn) from process k:
    wait (reg[j].sn >= wsn)
    send CATCH_UP_DONE(j, wsn) to process k
```

Lemma 1

Lemma

If a correct process p_i recives a message m from a $r\_brodcast(m)$ by another correct process - any other correct process will recive m.

Proof.

Immidiate from the guarantees of the broadcast algorithm. $\square$

Lemma 2

Lemma

Any two sets of processes of size $(n - t)$ mast have at least one correct process in common.

Lemma 2 - Proof

Proof.

Denote the set of processes with P, and the set of faulty ones F.
Let $Q_1, Q_2 \subseteq P$ s.t. $|Q_1| = |Q_2| = n - t$.

$$|\overline{Q_1} \cup \overline{Q_2}| \leq |\overline{Q_1}| + |\overline{Q_2}| \Rightarrow n - |\overline{Q_1} \cup \overline{Q_2}| \geq n - |\overline{Q_1}| - |\overline{Q_2}|$$

$$\Rightarrow |\overline{\overline{Q_1} \cup \overline{Q_2}}| \geq n - t - t$$

$$\Rightarrow |Q_1 \cap Q_2| \geq n - 2t > 3t - 2t = t = |F|$$

$$\Rightarrow \exists p \in Q_1 \cap Q_2 \notin F$$

$\square$

Lemma 3

Lemma

Let p_i be a correct process. Any invocation of $Reg[i].write()$ terminates.

Proof.

When p_i invokes $Reg[i].write()$ it sends $WRITE$ to all others, due to reliable brodcast - $n - t$ correct processes recive and handle this message eventually, thus send back $WRITE\_DONE$.
At some point these arrive and $Reg[i].write()$ temrminates. □

Lemma 4

Lemma

Lemma 5

Lemma 6

Lemma 7

Lemma 8

Atomic Concistency too much?

Requiring a system to implement Atomic Concistency is a very strong requirement and often comes at a steep runtime cost.
Is it possible to implement a (faster) Sequential Consistency algorithm, implementing a 'fence' operation?

Exploding Serial Numbers

Number of messages sent is unbounded, memory complexity is logarithmic with number of messages sent (due to counters).

Reset Serial Numbers

Is it possible to add a mechanism to reset the serial numbers?

Mallicious Serial Numbers

Is it possible for byzantine processes to cause the serial numbers (within correct processes) to explode?
If so, is it possible to prevent this?

Thanks for listening!