COMP90020: Distributed Algorithms

8. Consensus in Asynchronous Systems and Taking Chances

Randomised Algorithms

Miquel Ramirez

THE UNIVERSITY OF MELBOURNE

Semester 1, 2019

Agenda

Agenda

Asynchronous Systems: Basic Facts

No general DA for C when DS asynchronous and limited to crash failures

Asynchronous Systems: Basic Facts

No general DA for C when DS asynchronous and limited to crash failures

- Procs p_i can respond to messages at arbitrary times,
- there is no way to tell a crash apart from slowing down.
- Proof Sketch
  - Executions h can be extended indefinitely by an *adversary* so reaching config γ with $C \models \gamma$ requires infinite number of transitions.

Implications of lack of guarantees in DA's:

Asynchronous Systems: Basic Facts

No general DA for C when DS asynchronous and limited to crash failures

- Procs p_i can respond to messages at arbitrary times,
- there is no way to tell a crash apart from slowing down.
- Proof Sketch
  - Executions h can be extended indefinitely by an *adversary* so reaching config γ with $C \models \gamma$ requires infinite number of transitions.

Implications of lack of guarantees in DA's:

- no solutions for Interactive Consistency and Byzantine Generals,
- neither exists *guaranteed* RTO *multicast*.

Design Relaxation: Weakening System Assumptions

Masking Faults (e.g. *Transactional Systems*)

- Idea: failures appear like intermittent slowdown in message processing
- Ensure process p_i local vars are persistent,
- restart procs, initialize from persistent storage.

Failure Detectors (e.g. TCP/IP)

- Idea: pessimistic stance, assume no response within T = failure.
- "fail–silent": discard any messages after time out,
- so we get de facto *synchronous* comms.
- Issues: latency blow outs, pessimism leads to many false positives for failure.

Algorithm Relaxation: Weakening Algorithm Guarantees

Idea: Turn DA into a probabilistic algorithm

- Procs p_i generates event e with probability ρ_e
- Example: p_i flips a coin and trigger event based on outcome

Executions h are now probabilistic too:

- Conditions P reachable with h some probability ρ,
- sometimes same execution h does not reach γ where P is true!

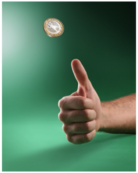
Fairness Assumption

Events e can be delayed *arbitrarily long time* but not forever:

Fair Message Scheduling

For every Send event e_1 generated at γ_k, there is a a Receive event e_2, such that configurations γ_l, $l > k$, generate it with probability $\rho_{e_2} > 0$.

Fairness is not for the Impatient



(c) Wikimedia Foundation

Question!

I have tossed a coin 100 times, and I got 99 heads in a row and 1 tail at the end. Did the fairness assumption hold?

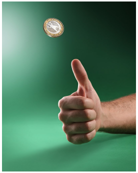
(A): Yes

(B): No

(C): With $p = \frac{1}{42}$

(D): Ask me tomorrow

Fairness is not for the Impatient



(c) Wikimedia Foundation

Question!

I have tossed a coin 100 times, and I got 99 heads in a row and 1 tail at the end. Did the fairness assumption hold?

(A): Yes

(B): No

(C): With $p = \frac{1}{42}$

(D): Ask me tomorrow

→ (Yes): The execution described is very improbable ($0.5^{100} \approx 7.8 \cdot 10^{-31}$, but still fair)

Agenda

Communication via Shared Objects

Internal events at procs p_i read or write to *local var*

- Comm channels can be *abstracted* by having shared objects x
- Procs p_i have all access to x, via atomic *read-modify-write* ops

Atomic Read-Modify-Write Ops

Typical hardware enabled atomic read-modify-write ops:

- test-and-set: *writes* $\top$ to Boolean var, returns previous value
- get-and-increment: *increases* integer var by 1, returns previous value
- get-and-set(new): *writes* new in var, returns previous value
- compare-and-set(old,new): if var $= old$, then set var to new and return $\top$

Note: single reads and writes are always atomic

Asynchronous 2-Consensus with Crash Failures

DS Specification:

- $\mathcal{P} = \{p_1, p_2\}$, $E = \{(p_1, p_2), (p_2, p_1)\}$
- Comms via asynchronous read & writes to shared objects, procs subject to Crash failures

Asynchronous 2-Consensus with Crash Failures

DS Specification:

- $\mathcal{P} = \{p_1, p_2\}$, $E = \{(p_1, p_2), (p_2, p_1)\}$
- Comms via asynchronous read & writes to shared objects, procs subject to Crash failures

Local variables for each p_i:

- *Proposed* value $v(p_i) \in D$, (v_i for short), shared objects x_1, x_2 and y, local var $s_i \in \{\top, \bot\}$
- *Decision* variable $d(p_i)$, $x_i \in D \cup \{\bot\}$, (d_i for short)
- v_i is constant, d_i, x_i, y initially set to $\bot$

Asynchronous 2-Consensus with Crash Failures

DS Specification:

- $\mathcal{P} = \{p_1, p_2\}$, $E = \{(p_1, p_2), (p_2, p_1)\}$
- Comms via asynchronous read & writes to shared objects, procs subject to Crash failures

Local variables for each p_i:

- *Proposed* value $v(p_i) \in D$, (v_i for short), shared objects x_1, x_2 and y, local var $s_i \in \{\top, \bot\}$
- *Decision* variable $d(p_i)$, $x_i \in D \cup \{\bot\}$, (d_i for short)
- v_i is constant, d_i, x_i, y initially set to $\bot$

DA Design Problem

Find DA that guarantees the following for every execution h

1. Validity: if $v_1 = v_2$ then $d_1 = d_2$
2. Agreement: if p_1 and p_2 do not crash $p_1 = p_2$.
3. Wait Free: eventually every correct p_i sets d_i to $x \neq \bot$.

Algorithm for 2-Consensus with test-and-set

Code for process p_i

1. $s_i \leftarrow \perp$
2. write(x_i, v_i)
3. $s_i \leftarrow$test-and-set(y)
4. if $s_i = \top$
5. then $d_i \leftarrow x_j$
6. else $d_i = v_i$

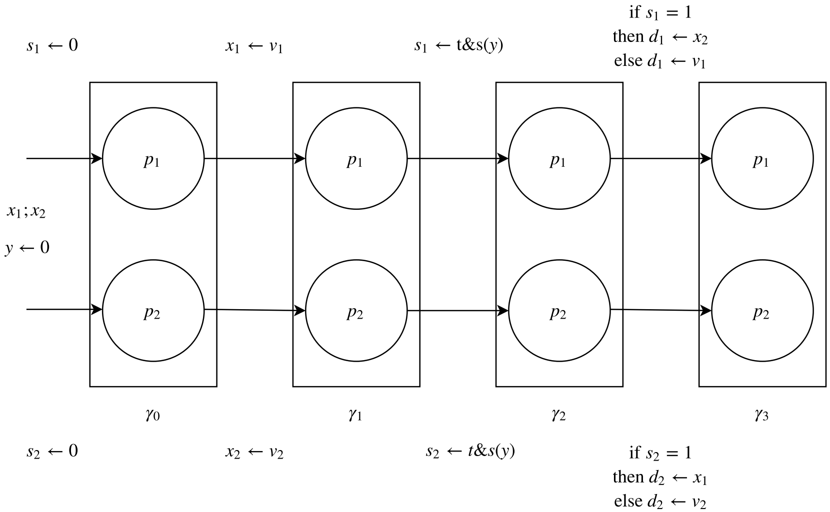
Assumptions:

- Asynchronous execution, time between steps 1 and 2, and 2 and 3 are variable,
- failures are crashes (i.e. p_i never reaches a step)

Notes:

- When proc p_j crashes, p_i guaranteed to reach step 6.
- y resolves contentions,
- atomic ops always terminate,
- if p_i execting step 3, p_j waits,
- if both procs correct, and p_i fastest proc, $d_j = d_i = v_i$.

Sample Execution



Do we Need the Third Shared Object?

Code for process p_i

1. $other_i \leftarrow \perp$
2. write(x_i, v_i)
3. $other_i \leftarrow$read(x_j)
4. if $other_i \neq \perp$
5. then $d_i \leftarrow v_i \vee other_i$
6. else $d_i = v_i$

Notes:

- $D = \{0, 1\}$
- $0 \vee 0 = 0$, $0 \vee 1 = 1$, $1 \vee 0 = 1$, $1 \vee 1 = 1$

Question!

Assume that all the code for p_1 executes before p_2, what are the end values of d_1 and d_2?

(A): $d_1 = 0, d_2 = 0$

(B): $d_1 = 0, d_2 = 1$

(C): $d_1 = 1, d_2 = 0$

(D): $d_1 = 1, d_2 = 1$

Do we Need the Third Shared Object?

Code for process p_i

1. $other_i \leftarrow \perp$
2. write(x_i, v_i)
3. $other_i \leftarrow$read(x_j)
4. if $other_i \neq \perp$
5. then $d_i \leftarrow v_i \vee other_i$
6. else $d_i = v_i$

Notes:

- $D = \{0, 1\}$
- $0 \vee 0 = 0$, $0 \vee 1 = 1$, $1 \vee 0 = 1$, $1 \vee 1 = 1$

Question!

Assume that all the code for p_1 executes before p_2, what are the end values of d_1 and d_2?

(A): $d_1 = 0, d_2 = 0$ (B): $d_1 = 0, d_2 = 1$

(C): $d_1 = 1, d_2 = 0$ (D): $d_1 = 1, d_2 = 1$

$\rightarrow$ ($d_1 = 0, d_2 = 1$, or $d_1 = 1, d_2 = 0$): Conensus not reached, d_i depends on each v_i.

Actually, Yes

Theorem

There is no DA α for 2–consensus such that

1. Procs communicate via only atomic read and write on shared objects,
2. α guarantees agreement, validity and wait–freedom

Proof Sketch

- Config γ is uncommitted if γ_0 s.t. $d_1 = d_2 = 0$ and γ_1 s.t. $d_1 = d_2 = 1$ reachable,
- configs γ_s and γ_t are p_2-indistinguishable if *state* of p_2 and *shared object values* same in both,
- at least one initial config γ_I is uncommitted,
- there exists uncommitted γ_s reachable from γ_I, s.t. all $\gamma_t = \delta(\gamma_s)$ are committed,
- for every γ_s, transitions exist where updates to x_1 (or x_2) are lost, and successors can be committed to different values

Agenda

Las Vegas versus Monte Carlo

Two classes of *probabilistic* DAs

Las Vegas algorithm

A probabilistic DA that guarantees

- Termination: probability of reaching terminal config γ^* is positive
- Validity: for every history $h = (\gamma_0, \ldots, \gamma_m)$, configs $\gamma_m \models P$

Monte Carlo algorithm

A probabilistic DA that guarantees

- Termination: every history h is finite
- Validity: for every history h, $\gamma_m \models P$ with probability $p > 0$.

Las Vegas' Byzantine f-Consensus: Specification

DS Specification:

- $\mathcal{P} = \{p_1, \ldots, p_N\}$, $E = \{(p_i, p_j), (p_j, p_i) \mid i \neq j\}$
- There is a leading process $p^* \in \mathcal{P}$ ("the general")
- Comms reliable, procs subject to Byzantine (anything goes) failures

Las Vegas' Byzantine f-Consensus: Specification

DS Specification:

- $\mathcal{P} = \{p_1, \ldots, p_N\}$, $E = \{(p_i, p_j), (p_j, p_i) \mid i \neq j\}$
- There is a leading process $p^* \in \mathcal{P}$ ("the general")
- Comms reliable, procs subject to Byzantine (anything goes) failures

Local variables for each p_i:

- *Proposed* value $v(p^*) \in \{0, 1\}$, (v^* for short),
- *Decision* variable $d(p_i) \in \{0, 1, \perp\}$, $p_i \neq p^*$, (d_i for short)
- v^* is constant, d_i initially set to $\perp$

Las Vegas' Byzantine f-Consensus: Specification

DS Specification:

- $\mathcal{P} = \{p_1, \ldots, p_N\}$, $E = \{(p_i, p_j), (p_j, p_i) \mid i \neq j\}$
- There is a leading process $p^* \in \mathcal{P}$ ("the general")
- Comms reliable, procs subject to Byzantine (anything goes) failures

Local variables for each p_i:

- *Proposed* value $v(p^*) \in \{0, 1\}$, (v^* for short),
- *Decision* variable $d(p_i) \in \{0, 1, \perp\}$, $p_i \neq p^*$, (d_i for short)
- v^* is constant, d_i initially set to $\perp$

DA Design Problem

Find DA that guarantees the following for every execution h

1. Termination: eventually every correct p_i sets d_i to v^*.
2. Agreement: for every correct p_i and p_j, $p_i \neq p^*$, $p_j \neq p^*$, eventually $d_i = d_j = v^*$.
3. Validity: if p^* correct, then every correct p_i, d_i eventually set to v^*.

Las Vegas' Byzantine f-Consensus: Algorithm

Bracha-Toueg f-Byzantine Consensus for p_i

For every round n, $n \geq 0$

1. Send (**vote**, n, v_i) to every p_j (and p_i)

* On Receive (**vote**, m, b) from p_j

    1. Send (**echo**, j, m, b) to every p_j (and p_i)

2. Count # (**echo**, j, n, b) for each pair (j, b).
3. For each j and b, if count$(j, b) > \frac{N+f}{2}$, $v(j, b) \leftarrow v(j, b) + 1$
4. If number $v(j, b) \neq 0$ greater than $N - f$, then $n \leftarrow n + 1$, and set $v_i^j = \text{argmax}_b\, v(j, b)$
5. $d_i = \text{majority}(v_i^1, \ldots, v_i^n)$, if $d_i \neq \perp$, then broadcast (**decide**, d_i) and terminate.

Remarks

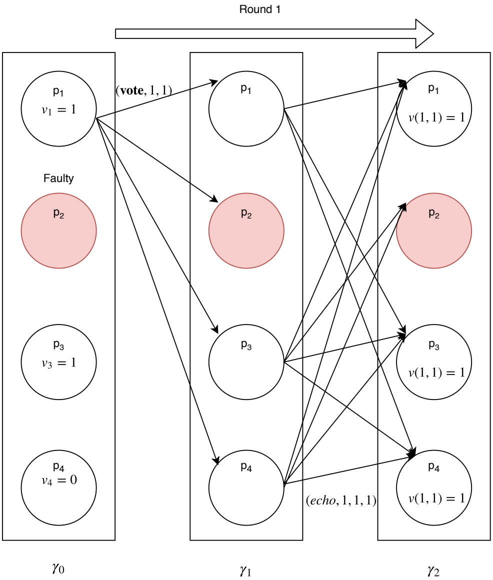
$\rightarrow$ $v(j, b) = 0$, $v_i^j = \perp$ initially

$\rightarrow$ p_i stores (**echo**, j, m b) and (**vote**, m, b) with $m > n$ for future rounds, otherwise discard.

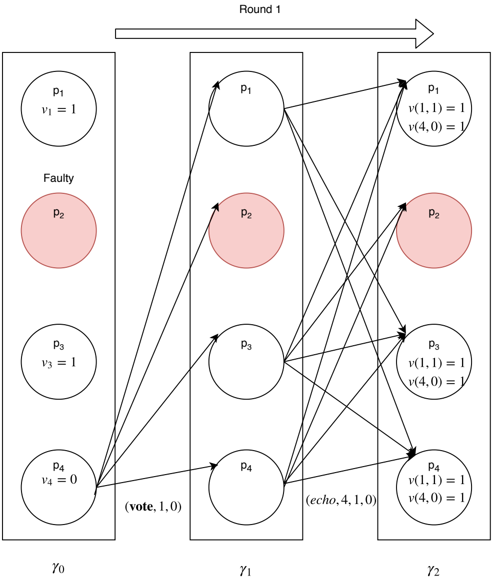
$\rightarrow$ (**decide**, b) interpreted as both a **vote** and **echo** message for b.

$\rightarrow$ Byzantine p_j detected by tracking **vote** and **echo** messages.

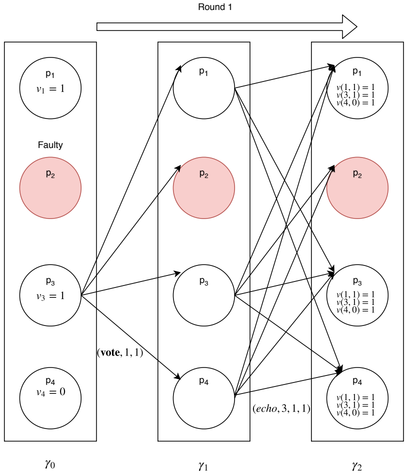
One Round of Message Passing, p_1



One Round of Message Passing, p_4



One Round of Message Passing, p_3



Consensus Reached: $d_1 = 1,\ d_3 = 1,\ d_4 = 1$

Probabilistic Consensus

Theorem: Bracha-Toueg f-Byzantine Termination

If the scheduling of messages is fair, then the Bracha-Toueg f-Byzantine consensus algorithm, for each $f < \frac{N}{3}$ is a Las Vegas algorithm that terminates with probability 1.

Proof Sketch:

- Correct proc p_i does not increment $v(j, b)$ unless confirmed by $N - f$ procs
- If correct proc p_i sets $d_i = b$ after n rounds, then necessarily accepted more than $\frac{N+f}{2}$ votes on b, hence all correct procs received more than $\frac{N+f}{2}$ votes on b too, and have decided for b.
- From fair scheduling follows that correct procs will receive $N - f$ votes from every other correct process, eventually forcing a decision.

Consensus in Adversarial Settings

What is the *adversary* that prevents Consensus

- manipulates network topology to delay messages,
- slows down procs $\mathcal{P}$ so their state is wrong.

Make the *adversary* life difficult with randomisation

- Add random delays to **send**() primitives,
- introduce random, local pauses in procs code.

Consensus may still be unfeasible but

- If *adversary* is Nature, make *unfortunate* coincidences unlikely,
- If *adversary* Intelligent, make *strategy* hard to compute.

Agenda

Wrapping up

Three variants of the problem of bringing $\mathcal{P}$ in DS to agreement

- Consensus
- Byzantine Generals
- Interactive Consistency

Closely related, we can sometimes reuse DAs

Problems are solvable (DA exist) when comms synchronous

... but efficiency is low, unless DS re designed or specific knowledge of failures' nature available.

Consensus is impossible in general in asynchronous systems,

... but often possible to redesign DS or use probabilistic DA.

Agenda

Further Reading

Coulouris et al. *Distributed Systems: Concepts & Design*

- Chapter 2, Section 2.4.2
- Chapter 15, Section 15.5

Wan Fokkink's *Distributed Algorithms: An Intuitive Approach*

- Chapter 2 - Introduction & Preliminaries
- Chapter 13 - Consensus with Byzantine Failures

Rajeev Alur *Principles of Cyber-Physical Systems*

- Chapter 4, pp. 170–177 for proof 2-Consensus in Asynchronous Systems