COMP90020: Distributed Algorithms

7. Consensus in DS with Byzantine Failures

Related Problems and Unfeasibility

Miquel Ramirez

THE UNIVERSITY OF MELBOURNE

Semester 1, 2019

Agenda

1. Revision
2. BG & IC
3. Impossibility Results
4. Biblio & Reading

Agenda

1. Revision
2. BG & IC
3. Impossibility Results
4. Biblio & Reading

Models of Non-Determinism

Both processes and comms channels can fail to show expected behaviour

- Omission – failing to do something (Crash Failures)
- Timing – failing to do something in a timely fashion
- Byzantine – procs and channels show arbitrary behaviour (Most General Case)

Models of Non-Determinism

Both processes and comms channels can fail to show expected behaviour

- Omission – failing to do something (Crash Failures)
- Timing – failing to do something in a timely fashion
- Byzantine – procs and channels show arbitrary behaviour (Most General Case)

Failure Models are useful to design robust algorithms for DS

→ Identify special cases which are easier to handle

→ Apply divide & conquer to design problem: see next slide

DS + DA = Transition Systems

Transition system $\mathcal{T} = \langle \mathcal{C}, \delta, \mathcal{I}, F \rangle$ abstracts DS under DA control

- $\mathcal{C}$ is set of configurations (*global states*) γ of DS,
- a transition function $\delta : \mathcal{C} \mapsto \mathcal{C}$, and
- a *set* initial configurations $\mathcal{I} \subseteq \mathcal{C}$,
- and terminal configurations $F \subset \mathcal{C}$, such that $\delta(f) = f$, $f \in F$.

An execution of DA over DS is a sequence

$$h = (\gamma_0, \gamma_1, \gamma_2, \ldots),\ \gamma_0 \in \mathcal{I},\ \gamma_{i+1} = \delta(\gamma_i)$$

Configs γ^* reachable if exists $h = (\gamma_0, \ldots, \gamma_k)$, $\gamma_k = \gamma^*$, where k is finite.

Transition System + Condition = Problem

To sum up:

- DA's control the evolution through time of DS
- Transition systems $\mathcal{T}$ describe behaviour of DS under DA control
- Requirements on behaviour formalised as logical conditions
  - → Safety: "something bad will never happen" (*Termination*)
  - → Liveness: "something good will eventually happen" (*Agreement*)
  - → Invariant: "safety from every beginning to every end" (*Validity*)

Point to Take Home

We *formulate* the problems DA's solve as the combination of transition systems and conditions .

Consensus from RTO-Multicast (Chandra & Toueg)

Consensus *equivalent to* reliable, totally ordered *multicast*.

Consensus from RTO-Multicast (Chandra & Toueg)

Consensus *equivalent to* reliable, totally ordered *multicast*.

How it works?

- All processes p_i form up a group g
- Every p_i makes a call to **RTO-multicast**(v_i,g)
- p_i sets d_i to m_i, first value coming via **RTO-delivers**$()$

Consensus from RTO-Multicast (Chandra & Toueg)

Consensus *equivalent to* reliable, totally ordered *multicast*.

How it works?

- All processes p_i form up a group g
- Every p_i makes a call to **RTO-multicast**(v_i, g)
- p_i sets d_i to m_i, first value coming via **RTO-delivers**$()$

Why it works?

- Termination guaranteed by *reliability* of **RTO-multicast**
- Agreement and Validity guaranteed by **RTO-deliver**
  - Delivery is totally ordered and reliable

Chandra & Toueg (1996) showed how to obtain RTO multicast from consensus

Dolev–Strong–Attiya–Welch Algorithm for Consensus

DSAW Consensus for process p_i

Initialization

$$V_i^1 \leftarrow \{v_i\},\ V_i^0 \leftarrow \emptyset$$

In round $1 \leq r \leq |\mathcal{F}| + 1$

1. **B-multicast**$(V_i^r \setminus V_i^{r-1}, g)$
2. $V_i^{r+1} \leftarrow V_i^r$

* On **B-deliver**(V_j) from some p_j

a. $V_i^{r+1} \leftarrow V_i^{r+1} \cup V_j$

After $|\mathcal{F}| + 1$ rounds

$$d_i \leftarrow \min V_i^{|\mathcal{F}|+1}$$

Assumptions:

- comms are synchronous,
- $\mathcal{F} \subset \mathcal{P}$ set of faulty procs,
- $f = |\mathcal{F}|$
- failures are crashes

Notes:

- Reentrant
- Round duration based on timer
- We use min because proposed values v_i do not change
- procs can crash but not generate arbitrary outputs

Correctness of DSAW

Termination

- Guaranteed by synchronous communication

Correctness of DSAW

Termination

- Guaranteed by synchronous communication

Agreement & Integrity (Proof Sketch)

- Let γ_l, $l = f + 1$, be cfg with $d_i \neq d_j$ for procs p_i, p_j,
- this can happen iff in γ_{l-1}, a proc p_k sent v to p_i and *crashed*, before being able to send v to p_j,
- if p_k had v, but p_j did not receive it, then in γ_{l-2} some other proc p_m sent v to p_k and crashed,
- easy to see path from γ_0 to γ_l requires $f + 1$ crashes,
- which violates assumption that at most f procs crash.

Correctness of DSAW

Termination

- Guaranteed by synchronous communication

Agreement & Integrity (Proof Sketch)

- Let γ_l, $l = f + 1$, be cfg with $d_i \neq d_j$ for procs p_i, p_j,
- this can happen iff in γ_{l-1}, a proc p_k sent v to p_i and *crashed*, before being able to send v to p_j,
- if p_k had v, but p_j did not receive it, then in γ_{l-2} some other proc p_m sent v to p_k and crashed,
- easy to see path from γ_0 to γ_l requires $f + 1$ crashes,
- which violates assumption that at most f procs crash.

Lower bound for Synchronous Systems

Consensus will require $f + 1$ rounds of message exchanges for any kind of Byzantine failure.

Agenda

1. Revision
2. BG & IC
3. Impossibility Results
4. Biblio & Reading

Why Consensus Matters?

Leading truck wants to go straight

Consensus DA guarantee trucks working correctly will follow leading truck

The Byzantine Generals Problem

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

The Byzantine Generals Problem

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 $v(p^*) \in D$, (v^* for short), v_i^j received values
- *Decision* variable $d(p_i) \in D \cup \{\perp\}$, $p_i \neq p^*$, (d_i for short)
- v^* is constant, d_i initially set to $\perp$

The Byzantine Generals Problem

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 $v(p^*) \in D$, (v^* for short), v_i^j received values
- *Decision* variable $d(p_i) \in D \cup \{\perp\}$, $p_i \neq p^*$, (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, 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^*.

Lamport-Shostak-Pease's Algorithm for $N \geq 4,\ f < N/3$

Process p^*

In round 1

B-multicast(v^*)

In round 2

Do Nothing

Process p_i

Initialization

$v_i \leftarrow \perp$

In round 1

* On **B-deliver**(v^*) from p^*

$v_i \leftarrow v^*$

In round 2

1. **send**(v_i, p_j) for $p_j \neq p^*$
* On **receive**(v^j) from p_j

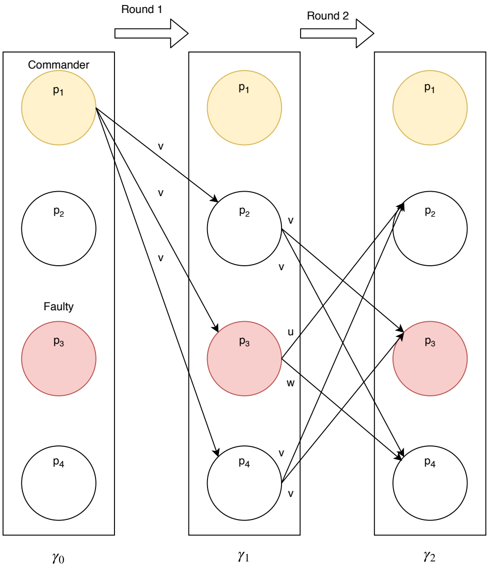
$v_i^j \leftarrow v^j$

2. $d_i = \text{majority}(v_i^1, \ldots, v_i^N)$

$\rightarrow \text{majority}(v_1, v_2, ..., v_n) = \text{argmax}_{v_i} \sum_{v_j} I_{v_j = v_i}$

Example: $\text{majority}(1, 1, 3, 4, 4, 3, 5, 1, \perp) = 1$, $\text{majority}(1, 2, 1, 2, 1, 2) = \perp$.

Sample Execution



$v = majority(v, u, v)$

$v = majority(v, v, w)$

Notes on LSP algorithm

Implication of synchronous comms:

- if **send**(v_i, p_j) fails (times out), p_j will set v_j^i to $\perp$,

Notes on LSP algorithm

Implication of synchronous comms:

- if **send**(v_i, p_j) fails (times out), p_j will set v_j^i to $\perp$,

When less than $N/3$ processes are faulty,

- every correct process p_i receives $(2N/3) - 1$ replicas of v^*,
- $\mathrm{majority}$ will filter out messages from faulty procs

Notes on LSP algorithm

Implication of synchronous comms:

- if **send**(v_i, p_j) fails (times out), p_j will set v_j^i to $\perp$,

When less than $N/3$ processes are faulty,

- every correct process p_i receives $(2N/3) - 1$ replicas of v^*,
- $\mathrm{majority}$ will filter out messages from faulty procs

When commander proc p^* fails and all procs correct,

- correct procs p_i will reach consensus (to something),

Notes on LSP algorithm

Implication of synchronous comms:

- if **send**(v_i, p_j) fails (times out), p_j will set v_j^i to $\perp$,

When less than $N/3$ processes are faulty,

- every correct process p_i receives $(2N/3) - 1$ replicas of v^*,
- $\mathrm{majority}$ will filter out messages from faulty procs

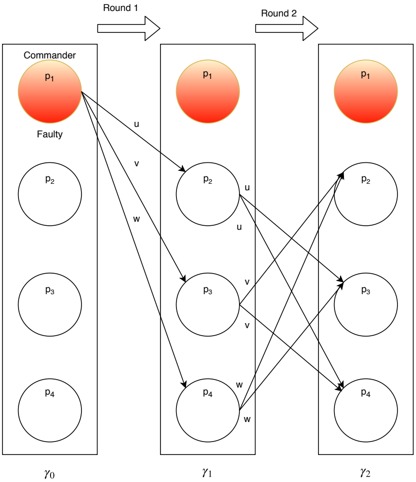
When commander proc p^* fails and all procs correct,

- correct procs p_i will reach consensus (to something),

If p^* failures are fair, sends values equally often

- if all correct, procs p_i will set d_i to $\perp$

Self–Diagnosing Commander is Faulty

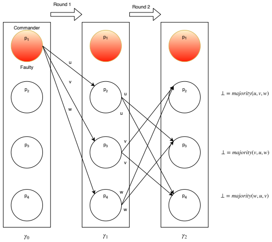


$\perp = majority(u, v, w)$

$\perp = majority(v, u, w)$

$\perp = majority(w, u, v)$

Question: “Unfair” Byzantine failures



Question!

Commander faulty, but sends v to $p4$ rather than w. What are the values of d_i for p_2, p_3 and p_4?

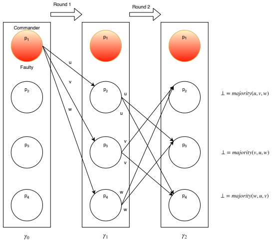
(A): $d_2 = d_3 = d_4 = \perp$

(B): $d_2 = u, d_3 = v, d_4 = w$

(C): $d_2 = v,\ d_3 = u,\ d_4 = v$

(D): $d_2 = d_3 = d_4 = v$

Question: "Unfair" Byzantine failures



Question!

Commander faulty, but sends v to $p4$ rather than w. What are the values of d_i for p_2, p_3 and p_4?

(A): $d_2 = d_3 = d_4 = \perp$

(B): $d_2 = u, d_3 = v, d_4 = w$

(C): $d_2 = v,\ d_3 = u,\ d_4 = v$

(D): $d_2 = d_3 = d_4 = v$

→ (D): Note that it is quite easy for a hacker taking over p_1 to "poison the well" for the subordinate processes.

Interactive Consistency

DS **Specification**:

- $\mathcal{P} = \{p_1, \ldots, p_N\}$, $E = \{(p_i, p_j), (p_j, p_i) \mid i \neq j\}$
- Comms reliable, procs subject to Byzantine (anything goes) failures

Local variables for each p_i:

- *Proposed* value $v(p_i) \in D$, (v_i for short)
- *Decision* vector $d(p_i) \in D^{N-1} \cup \{\perp\}^{N-1}$, ($\vec{d}_i$ for short)
- v_i is constant, d_i^j 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^j to $x \neq \perp$.
2. Agreement: for every correct (p_i, p_j), eventually $\vec{d}_i = \vec{d}_j$.
3. Validity: if $v_i = x$ for every correct p_j then $d_j^i = x$.

Relating C, BG and IC

Under some conditions we can reuse DA's for C, BG & IC

- Assumption: Faulty procs send only one (wrong) value

Relating C, BG and IC

Under some conditions we can reuse DA's for C, BG & IC

- Assumption: Faulty procs send only one (wrong) value

$C_i(v_1, \ldots, v_N)$ DA for Consensus

- returns d_i of proc p_i solving Consensus from vals $v_1, \ldots, v_N$

$BG_i(j, v)$ DA for Byzantine Generals

- return d_i for proc p_i, commander $p^* = p_j$ proposing v

$IC_i(v_1, \ldots, v_N)$

- returns vector $\vec{d_i}$ for proc p_i,
- $v_1, \ldots, v_N$ are proposed values of processes $\mathcal{P}$,
- and $IC_i(v_1, \ldots, v_N)^j$ is j-th value of $\vec{d_i}$.

Putting it Together

Interactive Consistency from Byzantine Generals

- Run BG_i N times, once with each p_j acting as p^*

$$IC_i(v_1, \ldots, v_N)^j = BG_i(j, v)$$

Putting it Together

Interactive Consistency from Byzantine Generals

- Run BG_i N times, once with each p_j acting as p^*

$$IC_i(v_1, \ldots, v_N)^j = BG_i(j, v)$$

Consensus from Interactive Consistency

- Run IC_i, obtain $\vec{d_i} = IC_i(v_1, \ldots, v_N)$
- Apply suitably chosen function to select d_i

$$C_i(v_1, \ldots, v_N) = \text{majority}(\vec{d_i})$$

Putting it Together

Interactive Consistency from Byzantine Generals

- Run BG_i N times, once with each p_j acting as p^*

$$IC_i(v_1, \ldots, v_N)^j = BG_i(j, v)$$

Consensus from Interactive Consistency

- Run IC_i, obtain $\vec{d_i} = IC_i(v_1, \ldots, v_N)$
- Apply suitably chosen function to select d_i

$$C_i(v_1, \ldots, v_N) = \text{majority}(\vec{d_i})$$

Byzantine Generals from Consensus

- Commander $p^* = p_k$, send v to itself and other procs p_i
- Every proc p_j (including p^*) runs C_j with $v_1, \ldots, v_N$

$$BG_j(k, v) = C_j(v_1, \ldots, v_N)$$

Efficiency in the Byzantine Failure Model

How many rounds necessary for Consensus?

Efficiency in the Byzantine Failure Model

How many rounds necessary for Consensus?

- We have shown $f + 1$ to be a lower bound,
- so we cannot do better than that!

Efficiency in the Byzantine Failure Model

How many rounds necessary for Consensus?

- We have shown $f+1$ to be a lower bound,
- so we cannot do better than that!

How many messages need to be sent?

Efficiency in the Byzantine Failure Model

How many rounds necessary for Consensus?

- We have shown $f + 1$ to be a lower bound,
- so we cannot do better than that!

How many messages need to be sent?

- Pease et al. sends exponential nr. messages $O(N^{f+1})$,
- but that's an upper bound.

Efficiency in the Byzantine Failure Model

How many rounds necessary for Consensus?

- We have shown $f + 1$ to be a lower bound,
- so we cannot do better than that!

How many messages need to be sent?

- Pease et al. sends exponential nr. messages $O(N^{f+1})$,
- but that's an upper bound.

Improvements on $O(N^{f+1})$

Efficiency in the Byzantine Failure Model

How many rounds necessary for Consensus?

- We have shown $f + 1$ to be a lower bound,
- so we cannot do better than that!

How many messages need to be sent?

- Pease et al. sends exponential nr. messages $O(N^{f+1})$,
- but that's an upper bound.

Improvements on $O(N^{f+1})$

- Use Digital Signatures, messages bound by $O(N^2)$,
- Exploit knowledge on source of failures

Agenda

1. Revision
2. BG & IC
3. Impossibility Results
4. Biblio & Reading

Negative Results

Consensus with Byzantine failures not possible for many DS's

- Proportion of faulty too big for consensus to be well–defined
- Comms asynchronous

Negative Results

Consensus with Byzantine failures not possible for many DS's

- Proportion of faulty too big for consensus to be well–defined
- Comms asynchronous

What does that mean?

Negative Results

Consensus with Byzantine failures not possible for many DS's

- Proportion of faulty too big for consensus to be well–defined
- Comms asynchronous

What does that mean?

- Re-design DS, add redundancy (replication)
- or *simulate* synchronous comms (e.g. Synchronous API mode for TCP/IP),
- or require digital signatures on every message.

Negative Results

Consensus with Byzantine failures not possible for many DS's

- Proportion of faulty too big for consensus to be well–defined
- Comms asynchronous

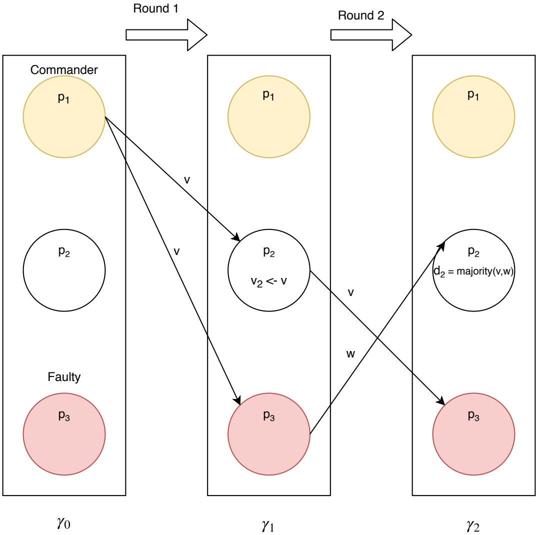
What does that mean?

- Re-design DS, add redundancy (replication)
- or *simulate* synchronous comms (e.g. Synchronous API mode for TCP/IP),
- or require digital signatures on every message.

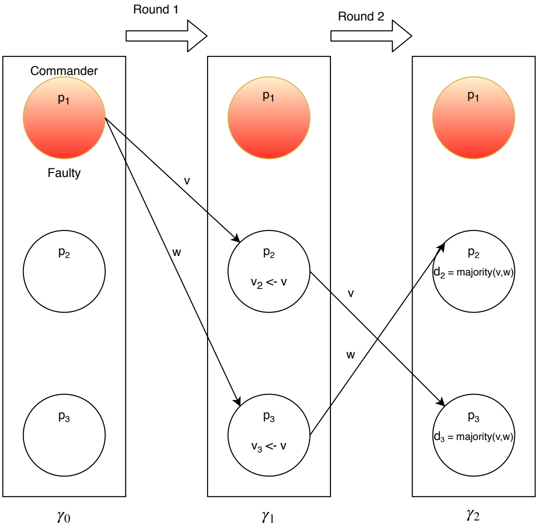
Hope for the best

Relax expectations on DA's, conditions guaranteed with $p > 0$

Counterexample #1 for Lamport-Shostak-Pease Algorithm



Counterexample #2 for Lamport-Shostak-Pease Algorithm



Generalization for $N \leq 3f$ (Lamport-Shostak-Pease)

1. Assume DA exist for $N \leq 3f$.
2. Divide procs $\mathcal{P}$ into disjoint sets S, T and U, $p^* \in S$.
3. Three cases possible when running DA:
   - Case #1: all faulty in U, $S \cup T$ reach Consensus, $p_i \in U$ agree with $p_j \in T$ if not faulty.
   - Case #2: all faulty in T, $S \cup U$ reach Consensus, $p_i \in T$ agree with $p_j \in U$ if not faulty.
   - Case #3: all faulty in S, including p^* too.
     a. S propagates v to T, $d_i = d_j$, $p_i \in S$, $p_j \in T$.
     b. S propagates w to U, $d_i = d_j$, $p_i \in S$, $p_j \in U$.
4. Contradiction: We found scenario where DA incorrect.

Agenda

1. Revision
2. BG & IC
3. Impossibility Results
4. Biblio & Reading

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 12 - Consensus with Crash Failures
- Chapter 13 - Consensus with Byzantine Failures