Consensus

Via the information theoretic lens (Part 2)

Ittai Abraham, VMware Research

Group blog: Decentralized Thoughts

Consensus [Lamport etal 78]

Parties have initial input

Can send messages via point-to-point channels

Termination (Liveness): In the end of the protocol each party must *decide* on a value

Safety: No two non-malicious parties decide on different values

Trivial: Always decide a default value

Make the problem not trivial:

- *Validity:* If all the non-faulty have the same input, then this must be the decision value
- *Fair Validity*: With constant probability an input of a non-faulty party is decided upon

Lamport Fischer 82: tolerating t crashes requires t+1 rounds

Using the Augilera Tueg 99 proof

A *configuration* of a system is the state of all the parties and the set of all pending, undelivered messages.

C **is a *deciding* configuration**: if all non-faulty parties have decided in C. We say that C is *1-deciding* if the common decision value is 1, and similarly that C is *0-deciding* if the decision is 0

C **is an *uncommitted* configuration**: if it has a future 0-deciding configuration and a future 1-deciding configuration. There exists $C \rightsquigarrow D_0$ and $C \rightsquigarrow D_1$ such that D_0 is 0-deciding and D_1 is 1-deciding

C **is a *committed* configuration**: if every future deciding configuration D (such that $C \rightsquigarrow D$) is deciding on the *same* value. We say that C is *1-committed* if every future ends in a 1-deciding configuration, and similarly that C is *0-committed* if every future ends in a 0-deciding configuration.

Not all beginnings are easy

Existance of an initial uncommitted configuration

Lemma: Every protocol solving consensus must have an initial configuration that is uncommitted

By contradiction, assume all are committed

Hybrid argument (1,1,1),(0,1,1),(0,0,1),(0,0,0): must be two adjacent committed configurations for 1 and for 0

But a CRASH of one party will cause both execution to be indistinguishable!

Why one round is not enough?

Existance of an initial uncommitted configuration

Proof by example for n=3: Consider the 4 initial configurations (1,1,1),(0,1,1),(0,0,1),(0,0,0)

By validity, configuration (1,1,1) must be 1-committed and configuration (0,0,0) must be 0-committed

Seeking a contradiction, lets assume none of the 4 initial configurations is uncommitted. So both (0,1,1) and (0,0,1) are committed

Since all 4 initial configurations are committed there must be two adjacent configurations that are committed to different values. W.l.o.g. assume that (0,1,1) is 1-committed and (0,0,1) is 0-committed

In both configurations, party 2 crashes right at the start of the protocol: Clearly both configurations look like (1,CRASH,0)(1,CRASH,0)

Both worlds must decide the same, but this is a contradiction because one is 1-committed and the other is 0-committed

Tolerating t crashes requires t+1 rounds

[AT99] proof

The general case:

Lemma 1: exits an uncommitted configuration

Lemma 2: with t-1 crashes, there exists a round t-1 execution that leads to an uncommitted configuration

Lemma 3: you cannot always decide in round x if in round x-1 you may be uncommitted and there may be a crash

Theorem: cannot always terminate in t rounds -> need t+1 rounds in the worst case

Learning by Doing

4 parties, each with input in {0,1}

Adversary controls one party (malicious)

Write a protocol for consensus:

- Safety: no two non-faulty decide different vlaues
- Liveness: All non-fualty parties dedcide
- Validity: If all the non-faulty have the same input x, then x is the decision value

Learning by Doing: solution

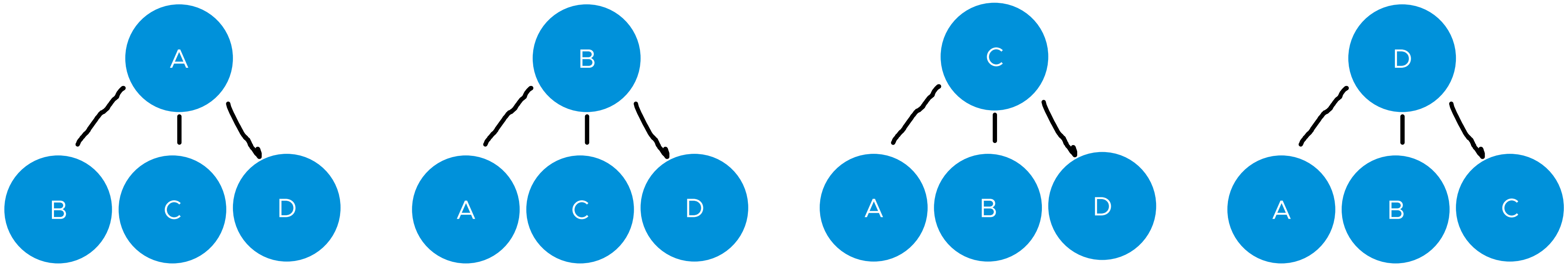
In two rounds, just think about full information

Round 1: Send one bit, receive 3 bits

Round 2: send 3 bits, receive 9 bits

Decide as function of [1 bit + 3 bit + 9 bits]

Its not about you or what you say, its about what others say about you



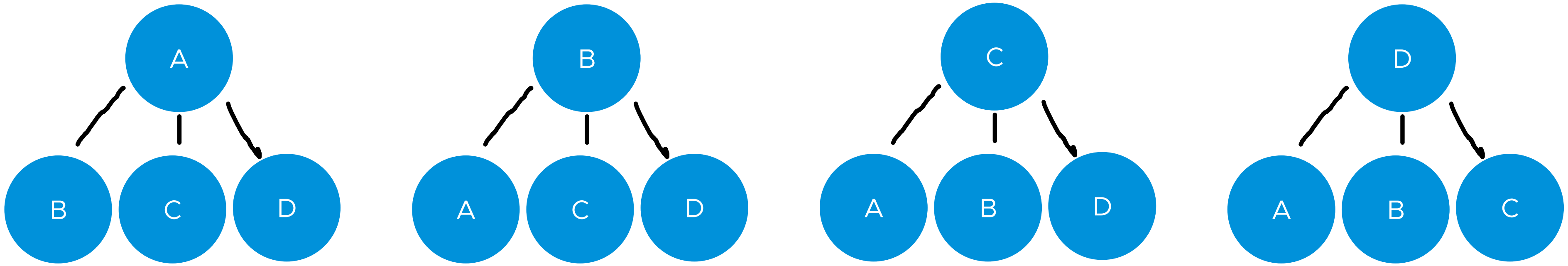
Learning by Doing: solution

Liveness: trivial

Safety:

- Non-faulty: majority gossip will be correct
- Faulty: gossip only by non-faulty - so everyone will agree!

Validity: take majority



Learning by Doing

3 parties, each with input in {0,1}

Adversary controls one party (omission failure)

Write a protocol for consensus:

- (Uniform) Safety: no two decide different vlaues
- Liveness: All non-fualty parties dedcide
- Validity: If all the non-faulty have the same input x, then x is the decision value

Learning by Doing

7 parties, each with input in {0,1}

Adversary controls two partes (malicious)

Write a protocol for consensus:

- Safety: no two non-faulty decide different vlaues
- Liveness: All non-fualty parties dedcide
- Validity: If all the non-faulty have the same input x, then x is the decision value

Byzantine Primary-Backup (at view v):

with Reliable Boradcast and locking

1. Primary sends *<send, (value, v, u)>* to all
2. Replica receives *<send, (value, v, u)>*,
   - If u>=lock, n-f *<echo2, (value, u)>* arrive, and first send from primary in view v, then
     - sends : *<echo1, (value, v)>* to all
3. Replica gathers *n-f <echo1, (value, v)>, then*
   - Sends *<echo2, (value, v)>* to all
4. Replica gathers *n-f <echo2 (value, v)>, then (at view v)*
   - Set lock:=v; lock value:=value
   - Sends *<lock, (value, v)>* to all
5. Replica gathers *n-f <lock, (value, v)>, then*
   - Decide (value)

Replica gathers *f+1 <echo2, (value, v)>, then*

- If did not send echo2
- Sends *<echo2, (value, v)>* to all

View change:

- **Replica**:
- Sends its lock and lock value

- **Primary**:
- accept a lock (value',v') if also n-f *<echo2, (value, v)>* arrive
- Wait for n-f such locks
- Choose the value with the highest lock (view)

Validity

If all non-faulty have input x then x must be the decision value

If a non-faulty is the first primary we are fine

But what if the first primaries are faulty?

Virtual primary!

Safety

Let v^* be the first view that any replica decided (value X, view v^*)

Prove by induction that any *accepted send* of view $v \geq v^*$ must be with value X

- for base case due to non-equivocation

Safety

Induction claim:

1. Any *accepted send* of view $v \geq v^*$ must be with value X
2. Existence of a *core* of *f+1* non-faulty that have a lock on view at least v* with value X
3. Any non-faulty, its maximal view lock is either:
   - On view v* or larger and with value X
   - On a view smaller than v*

Base case at view v*:

- Core is the n-2f out of the n-f that sent a lock to decider
- Any other non-faulty: trivial since v* is the highest view

Assume claim is true for v>=v* and prove for v+1:

- If a primary uses a view that is at least v*, from induction it must be with value X
- If a primary uses a view that is lower than v*: it needs n-f echo1, but the core will block

Liveness

If a non-faulty primary is elected and the system is synchronous

Primary will hear locks from *all* non-faulty and will choose the maximum one

All non-faulty replicas will:

- See the accepted send from the old view that the primary used
- This accepted send is from a view that is at least their lock view
- Hence all non-faulty will echo1 the primary
- The rest of the protocol is unconditional

Responsivness: liveness in asynchrony

In asynchrony, non-faulty primary can wait for just n-f responses during view change

- May miss a lock of a non-fualty
- So non-faulty primary may choose a lock that is smaller than the maximum
- Some non-faulty will block primary and n-f echo1 will not be reached

Solution: add one more round ☺

- After seeing n-f echo2, send ***key***
- After seeing n-f keys, send ***lock***
- If a non-fualty has a lock, then there are at least f+1 non-faulty that have a key
- During view change, ask for keys
- Hearing from n-f means that at least one key holder will be heard

Responsive Byzantine Primary-Backup (at view v):

Information Theoretic HotStuff

1. Primary sends *<send, (value, v, u)>* to all
2. Replica receives *<send, (value, v, u)>*,
   - If u>=lock, n-f *<echo2, (value, u)>* arrive, and first send from primary in view v, then
     - sends : *<echo1, (value, v)>* to all
3. Replica gathers *n-f <echo1, (value, v)>, then*
   - Sends *<echo2, (value, v)>* to all
4. Replica gathers *n-f <echo2 (value, v)>, then (at view v)*
   - Set key:=v; key value:=value
   - Sends *<key, (value, v)>* to all
5. Replica gathers *n-f <key, (value, v)> and n-f <echo2 (value, v)>, then (at view v)*
   - Set lock:=v
   - Sends *<lock, (value, v)>* to all
6. Replica gathers *n-f <lock, (value, v)>, then* Decide (value)

Replica gathers *f+1 <echo2, (value, v)>, then*
- If did not send echo2
- Sends *<echo2, (value, v)>* to all

View change:
- **Replica**:
- Sends its key and key value

- **Primary**:
- accept a key (value',v') if also n-f *<echo2, (value, v)>* arrive
- Wait for n-f such key
- Choose the value with the highest key (view)

Liveness

If a non-faulty primary is elected

~~Primary will hear locks from *all* non-faulty and will choose the maximum one~~

Primary will hear the maximal key from n-f during view change

- If a non-faulty is locked, its because of n-f keys, f+1 of them are non-faulty
- At least one key holder will be in the n-f view change quorum
- So the maximal key will be at least as high as the maximal lock of all non-fualty

All non-faulty replicas will:

- See the accepted send from the old view that the primary used
- This accepted send is from a view that is at least their lock view
- Hence all non-faulty will echo1 the primary
- The rest of the protocol is unconditional

vmware

Byzantine Paxos: adding randomness

Elect a random primary

Revolving coordinator

- After f view changes (O(f) rounds) a non-faulty primary will be elected

Assume we have an *oblivious leader election* functionality

- At least f+1 honest must request the functionality to start
- Each party i outputs a leader L(i)=j
- With probabilty at least ½ (can use any constant) :
  - all non-fualty output the same value j and,
  - j was non-faulty before functionality started

Good for a static adversary

Adaptive adversary will adaptivly corrupt that chosen primary ☹

Byzantine Paxos: adaptive adversaries

Everyone is a Priamry ☺

Adaptive adversary will shoot down the primary

Solution:

- Let everyone be a primary
- Then choose who the real primary is in hindsight (and all other are just decoys)

Liveness: with constant probability a good primary is chosen

Safety:

- In hindsight, looks like a single primary each view
- If a faulty primary or a confusion of primaries is chosen, then this is just like a faulty primary
  - Safety is maintained!

Responsive Byzantine Primary-Backup (at view v):

Determinstic version

1. Primary sends *<send, (value, v, u)>* to all
2. Replica receives *<send, (value, v, u)>,*
   - If u>=lock, n-f *<echo2, (value, u)>* arrive, and first send from primary in view v, then
     - sends : *<echo1, (value, v)>* to all
3. Replica gathers *n-f <echo1, (value, v)>, then*
   - Sends *<echo2, (value, v)>* to all
4. Replica gathers *n-f <echo2 (value, v)>, then (at view v)*
   - Set key:=v; key value:=value
   - Sends *<key, (value, v)>* to all
5. Replica gathers *n-f <key, (value, v)> and n-f <echo2 (value, v)>, then (at view v)*
   - Set lock:=v
   - Sends *<lock, (value, v)>* to all
6. Replica gathers *n-f <lock, (value, v)>, then*
   - Decide (value)

Replica gathers *f+1 <echo2, (value, v)>, then*

- If did not send echo2
- Sends *<echo2, (value, v)>* to all

View change:

- **Replica**:
- Sends its key and key value

- **Primary**:
- accept a key (value',v') if also n-f *<echo2, (value, v)>* arrive
- Wait for n-f such key
- Choose the value with the highest key (view)

Responsive Byzantine Primary-Backup (at view v):

with random leader election

1. Each party as Primary, sends *<send, (value, v, u)>* to all
2. Run oblivious leader election to decide who to listen to
3. Replica receives *<send, (value, v, u)>,*
   - If u>=lock, n-f *<echo2, (value, u)>* arrive, and first send from primary in view v, then
     - sends : *<echo1, (value, v)>* to all
4. Replica gathers *n-f <echo1, (value, v)>, then*
   - Sends *<echo2, (value, v)>* to all
5. Replica gathers *n-f <echo2 (value, v)>, then (at view v)*
   - Set key:=v; key value:=value
   - Sends *<key, (value, v)>* to all
6. Replica gathers *n-f <key, (value, v)> and n-f <echo2 (value, v)>, then (at view v)*
   - Set lock:=v
   - Sends *<lock, (value, v)>* to all
7. Replica gathers *n-f <lock, (value, v)>, then*
   - Decide (value)

Replica gathers *f+1 <echo2, (value, v)>, then*

- If did not send echo2
- Sends *<echo2, (value, v)>* to all

View change:

- **Each Replica**:
- Sends its key and key value ***to everyone***

- **Each Primary**:
- accept a key (value’,v’) if also n-f *<echo2, (value, v)>* arrive
- Wait for n-f such key
- Choose the value with the highest key (view)

Oblivious Leader Election

Choosing a random leader is a simple MPC protocol

But MPC uses VSS, and VSS requires broadcast ☹

Solution:

- a notion that is weaker than VSS but strong enough for OLE
- Moderated VSS (KK06) and Graded VSS (MF88)
- Tailor made MPC (with a constnat error probabilty)

Gradecast -> MVSS ->OLE ->O(1) time expected Byzantine Agreement

Gradecast (MF88, D81)

Dealer P* has inoput m

Each party outputs a vlaue m and a grade in {0,1,2}

If the dealer is no-fualty then all non-fualty output (m,2)

If a non-fualty outputs (m’,2) then all non-fulaty output (m’,g) with g>0

(If two non-faulty have grade 1 then have same value)

Gradecast protocol (MF88)

round 1: Dealer P* <sends m> to all

round 2: Party sends <echo1 m> to the first message it recives from the primary

round 3: If party gathers n-f echo1 it sends <echo2 m>

End of round 3:

- Grade 2: If party gatheres n-f echo2; otherwise
- Grade 1: if party gathers f+1 echo2; otherwise
- Gread 0 (default value)

Gradecast proof (MF88)

round 1: Dealer P* <sends m> to all

round 2: Party sends <echo1 m> to the first message it recives from the primary

round 3: If party gathers n-f echo1 it sends <echo2 m>

End of round 3:

- Grade 2: If party gatheres n-f echo2; otherwise
- Grade 1: if party gathers f+1 echo2; otherwise
- Gread 0 (default value)

Echo1 causes non-equivocation -> any two grade 1 must have same value

Non-fualty dealer -> all non-fulty have (m,2)

Non-fualty has (m’,2) -> all nonfaulty have at lesT f+1 echo2 -> all non-fualty have (m,g) with g>0

Moderated VSS [KK06]

MVSS from VSS

Dealer P*

Moderator P**

Take any VSS that uses broadcast only in share phase

Replace <broadcast m by party j> with:

- Party j runs gradecast (m)
- The moderator P** takes the value m’ of the gradecast and runs gradecast (m’)

Outcome for party i:

- Let (m,g) be the outcome of the first gradecast
- Let (m’,g’) be the outcome of the first gradecast
- If g’<2 or (g’=2 and g=2 and m≠m’) then set OK=false

Proof for Moderated VSS

If OK=true for any non-faulty then VSS properties hold

- Because all see the moderator’s value and the moderator's value is consistent with any non-faulty broadcaster

If the moderator is non-faulty then all non-faulty have OK=true

- From the grade cast properties of an honest sender

Oblivious Leader Eelection

OLE from MVSS

For each i,j, do a MVSS with dealer i and moderator j (say random value in n^4)

The secret ballot for j will be the sum mod n^4 of all the VSS where j is a moderator

Reveal all the secret ballots for all parties

But if for some moderator j you see OK=false in any MVSS then set secret ballot to 0

Choose the leader to be the party with the highest secret ballot

With large probability there are no collisions, and then with constant probability a non-faulty is elected

Its all about the adversary!

Can you solve Byzantine Agreement with sub-quadratic messages?

Static Adversary vs *Adaptive* Adversary

Byzantine agreement with sub quadratic messages is *easy* against a *Static* adversary

- With randomization
- Just use the US jury system: its a scalable consensus mechanism!

1. Choose a random poly-log size committee
2. Since the adversary is static, it controls a small fraction of the committee
3. Run Byzantine Agreement in the committee and then report back to everyone the verdict

Dolev Reischuk [82]

Cannot solve Broadcast against omission adversary with just (f/2)^2 messages

Assume that if a party receives no message it never decides 1

- Either decides 0 or does not decide

Proof approach:

- Create World 1 where all honest decide 1 (with f/2 corrupt called *C*)
- Create World 2 with f/2 more corrupt *X* and one old corrupt *p* becomes honest
  - For all honest (but *p*) in world 2: world 1 and world 2 are indistinguishable
  - Party *p* receives no messages

World 1

Set $|C| = f/2$ are corrupt

All honest decide 1

World 2

$C \setminus \{p\}$ are corrupt, p is honest

Additional set $|X| \leq f/2$ are corrupt

Honest *p* receives no messages

For all honest (but *p*) in world 2:

worlds 1, 2 are indistinguishable

Dolev Reischuk [PODC 1982]

Cannot solve Broadcast against omission adversary with just (f/2)^2 messages

World 1: Corrupt a set *C* of f/2 parties:

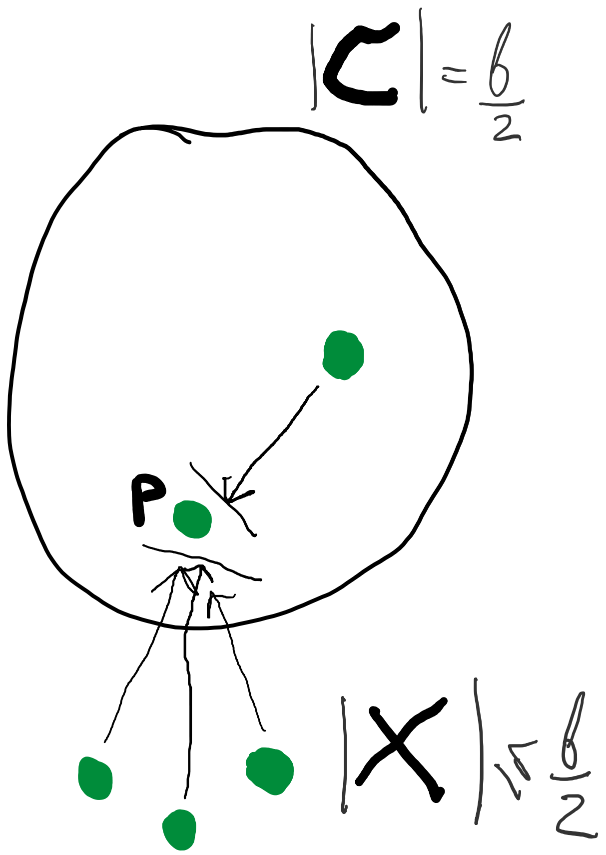
- Run them as honest; except
- For each member of *C*
  - Block all communication from other parties in *C*
  - Block the the first f/2 message from parties not in *C*
- Validity: all honest parties decide 1

Assume protocol sends just (f/2)^2 messages

- So one member, *p* of *C* must get at most f/2 message from a set of parties *X* not in *C*

World 2: un-corrupt *p* and corrupt *X* as follows:

- Run *X* as honest; other than:
  - Block the first f/2 messages to *p* from *X*
- All other honest cannot distinguish - must decide 1
- Honest party *p* hears nothing – cannot decide 1 ☹



Thank you

Moving to asynchrony

Responsivness: we added a key round

MVSS does not work:

- n>4f, costnat time [MF]
- AVSS constnat time, but has non-zero deadlock [CR]
- ShunningAVSS no deadlock but polynomial time [ADH]

Attach f+1 secrets. Honest attach only after the RB works