Consensus

Maruth Goyal

UT Austin

Spring 2021

Table of Contents

Consensus

- Suppose you run a large online company, and have a database which your application depends on.

Consensus

- Suppose you run a large online company, and have a database which your application depends on.
- If this database fails, your entire company will come down crashing and burn into flames.

Consensus

- Suppose you run a large online company, and have a database which your application depends on.
- If this database fails, your entire company will come down crashing and burn into flames.
- One way to prevent this is **redundancy**.

Consensus

- Suppose you run a large online company, and have a database which your application depends on.
- If this database fails, your entire company will come down crashing and burn into flames.
- One way to prevent this is **redundancy**.
- Instead of having just one database, we maintain a cluster of several machines, and **replicate** the database on each of them.

Consensus

- Suppose you run a large online company, and have a database which your application depends on.
- If this database fails, your entire company will come down crashing and burn into flames.
- One way to prevent this is **redundancy**.
- Instead of having just one database, we maintain a cluster of several machines, and **replicate** the database on each of them.
- Problem solved? Of course not.

Consensus

- Suppose user U sends a write request $W(k, v)$ to your current main node M.
  - M applies the write to its database.

Consensus

- Suppose user U sends a write request $W(k, v)$ to your current main node M.
  - M applies the write to its database.
  - To maintain redundancy M tries to communicate this write to the other nodes $N_1, \ldots, N_k$.

Consensus

- Suppose user U sends a write request $W(k, v)$ to your current main node M.
  - M applies the write to its database.
  - To maintain redundancy M tries to communicate this write to the other nodes $N_1, \ldots, N_k$.
  - Suppose each node on receiving the write immedeately applies it to its database.

Consensus

- Suppose user U sends a write request $W(k, v)$ to your current main node M.
  - M applies the write to its database.
  - To maintain redundancy M tries to communicate this write to the other nodes $N_1, \ldots, N_k$.
  - Suppose each node on receiving the write immedeately applies it to its database.
  - What if M fails before it can communicate to $N_3, \ldots, N_k$, and N_3 becomes the new main node?

Consensus

- Suppose user U sends a write request $W(k, v)$ to your current main node M.
  - M applies the write to its database.
  - To maintain redundancy M tries to communicate this write to the other nodes $N_1, \ldots, N_k$.
  - Suppose each node on receiving the write immedeately applies it to its database.
  - What if M fails before it can communicate to $N_3, \ldots, N_k$, and N_3 becomes the new main node?
- Thus, we need some mechanism that allows for all the nodes $M, N_1, \ldots, N_k$ to first achieve **consensus** on whether a transaction is to be committed. Moreover, this mechanism must be **fault-tolerant**.

Consensus

- This problem of fault-tolerant consensus seems rather fundamental, right?

Consensus

- This problem of fault-tolerant consensus seems rather fundamental, right?

Theorem ([Fischer et al., 1985])

It is impossible for an algorithm A operating with read/write shared memory to achieve consensus under even a single failure.

Consensus

- This problem of fault-tolerant consensus seems rather fundamental, right?

Theorem ([Fischer et al., 1985])

It is impossible for an algorithm A operating with read/write shared memory to achieve consensus under even a single failure.

- Nonetheless, there exist algorithms that use information such as timing to provide consensus.
- Recent examples include Raft [Ongaro and Ousterhout, 2014], and the much older Paxos [Lamport et al., 2001]

Consensus

- Today we will look at the following results:
  - Impossibility of wait-free consensus (non-topological proof).

Consensus

- Today we will look at the following results:
    - Impossibility of wait-free consensus (non-topological proof).
    - Impossibility of consensus under even a single failure with read/write shared memroy.

Table of Contents

Trying to Solve Consensus

- Before we prove that things are impossible, let's get some hope by solving it under some different settings.

Consensus with Atomic memory

Trying to Solve Consensus

- Before we prove that things are impossible, let's get some hope by solving it under some different settings.
- In particular, if we have access to **Read-Modify-Write** shared memroy, the problem becomes trivial! (we can even do wait-free!)

Consensus with Atomic memory

Trying to Solve Consensus

- Before we prove that things are impossible, let's get some hope by solving it under some different settings.
- In particular, if we have access to **Read-Modify-Write** shared memroy, the problem becomes trivial! (we can even do wait-free!)

Consensus with Atomic memory

- Maintain a single shared atomic variable v initialized to $\perp$.

Trying to Solve Consensus

- Before we prove that things are impossible, let's get some hope by solving it under some different settings.
- In particular, if we have access to **Read-Modify-Write** shared memroy, the problem becomes trivial! (we can even do wait-free!)

Consensus with Atomic memory

- Maintain a single shared atomic variable v initialized to $\perp$.
- Assume each processor already has some input value v_i.

Trying to Solve Consensus

- Before we prove that things are impossible, let's get some hope by solving it under some different settings.
- In particular, if we have access to **Read-Modify-Write** shared memroy, the problem becomes trivial! (we can even do wait-free!)

Consensus with Atomic memory

- Maintain a single shared atomic variable v initialized to $\perp$.
- Assume each processor already has some input value v_i.
- Each processor reads v. If it is $\perp$, it writes its value v_i to v. It then decides that its value is the final one.

Trying to Solve Consensus

- Before we prove that things are impossible, let's get some hope by solving it under some different settings.
- In particular, if we have access to **Read-Modify-Write** shared memroy, the problem becomes trivial! (we can even do wait-free!)

Consensus with Atomic memory

- Maintain a single shared atomic variable v initialized to $\perp$.
- Assume each processor already has some input value v_i.
- Each processor reads v. If it is $\perp$, it writes its value v_i to v. It then decides that its value is the final one.
- If the value of v is anything but $\perp$, the processor accepts the value of v as the consensus value.

Trying to Solve Consensus

- This algorithm cannot be easily extended to the case of just simple read/write memory using something like mutual exclusion.

Trying to Solve Consensus

- This algorithm cannot be easily extended to the case of just simple read/write memory using something like mutual exclusion.
- **Issue:** What if a processor fails while it is in the critical region?

Trying to Solve Consensus

- This algorithm cannot be easily extended to the case of just simple read/write memory using something like mutual exclusion.
- **Issue:** What if a processor fails while it is in the critical region?
- Potential mitigation:

Trying to Solve Consensus

- This algorithm cannot be easily extended to the case of just simple read/write memory using something like mutual exclusion.
- **Issue:** What if a processor fails while it is in the critical region?
- Potential mitigation:

HeartbeatMutex ™

- Perform mutual exclusion as before (for instance Dijkstra's algorithm).
- However, now the consensus mechanism A is aware of the process P_i which is currently in the critical region, and moreover is allowed bidirectional communication with P_i.
- Set some timeouts t_{wait} and t_{respond}.
- Every t_{wait} seconds, A sends P_i a $\textsc{Ping}$ message. It then waits for t_{respond} seconds for P_i to respond with $\textsc{Pong}$. If it does not receive a reply, the process is considered to have died and exitted the critical region.

Trying to Solve Consensus

- HeartbeatMutex $^{\text{TM}}$naturally has some problems.
- For instance, since actions are still not atomic the shared memory may be left in a corrupted state when a process fails.

Trying to Solve Consensus

- HeartbeatMutex $^{\text{TM}}$naturally has some problems.
- For instance, since actions are still not atomic the shared memory may be left in a corrupted state when a process fails.
- Moreover, the timeouts must be sufficiently large to not mistake communication delays for process failure, but not so large to keep everyone else waiting etc.

Trying to Solve Consensus

- HeartbeatMutex $^{\text{TM}}$naturally has some problems.
- For instance, since actions are still not atomic the shared memory may be left in a corrupted state when a process fails.
- Moreover, the timeouts must be sufficiently large to not mistake communication delays for process failure, but not so large to keep everyone else waiting etc.
- Raft [Ongaro and Ousterhout, 2014] uses a similar mechanism to ensure the current leader is alive. A lack of response triggers a leader election process etc. It uses some additional metadata to ensure that an out-of-date node is not elected as leader.

CLOUDFLARE The Cloudflare Blog

Thanks for being here, come back soon. Get notified of new posts:

Email Address Subscribe

Product News | Speed & Reliability | Security | Serverless | Cloudflare Network | Developers | Deep Dive | Life @Cloudflare

A Byzantine failure in the real world

11/27/2020

Tom Lianza Chris Snook

An analysis of the Cloudflare API availability incident on 2020-11-02

When we review design documents at Cloudflare, we are always on the lookout for Single Points of Failure (SPOFs). Eliminating these is a necessary step in architecting a system you can be confident in. Ironically, when you're designing a system with built-in redundancy, you spend most of your time thinking about how well it functions when that redundancy is lost.

On November 2, 2020, Cloudflare had an incident that impacted the availability of the API and dashboard for six hours and 33 minutes. During this incident, the success rate for queries to our API periodically dipped as low as 75%, and the dashboard experience was as much as 80 times slower than normal. While Cloudflare's edge is massively

Table of Contents

Impossibility of Wait-Free Consensus

- We will now move to proving some impossibility results for consensus under read/write memory.

Impossibility of Wait-Free Consensus

- We will now move to proving some impossibility results for consensus under read/write memory.
- The proof will be significantly different from [Herlihy and Shavit, 1999]. In particular will look at properties of explicit execution traces.

Impossibility of Wait-Free Consensus

- We will now move to proving some impossibility results for consensus under read/write memory.
- The proof will be significantly different from [Herlihy and Shavit, 1999]. In particular will look at properties of explicit execution traces.
- First, we will formalize the problem, and introduce some definitions we will use in the proof.

Impossibility of Wait-Free Consensus

- Computational model identical to previous 2 presentations.

Consensus

- **Well-Formedness**: For any i, the interaction between U_i and P_i is a prefix of init$(v)_i$ and decide$(w)_i$.

Impossibility of Wait-Free Consensus

- Computational model identical to previous 2 presentations.
- Users U_i can send init$(v)_i$ messages to processor P_i.

Consensus

- **Well-Formedness**: For any i, the interaction between U_i and P_i is a prefix of init$(v)_i$ and decide$(w)_i$.
- **Agreement**: All decision values are identical.

Impossibility of Wait-Free Consensus

- Computational model identical to previous 2 presentations.
- Users U_i can send init$(v)_i$ messages to processor P_i.
- The processors P_i will then finally send decide$(w)_i$ messages back, indicating that w is the consensus value.

Consensus

- **Well-Formedness**: For any i, the interaction between U_i and P_i is a prefix of init$(v)_i$ and decide$(w)_i$.
- **Agreement**: All decision values are identical.
- **Validity**: If all init messages have the same value v, then v is the only possible decision value.

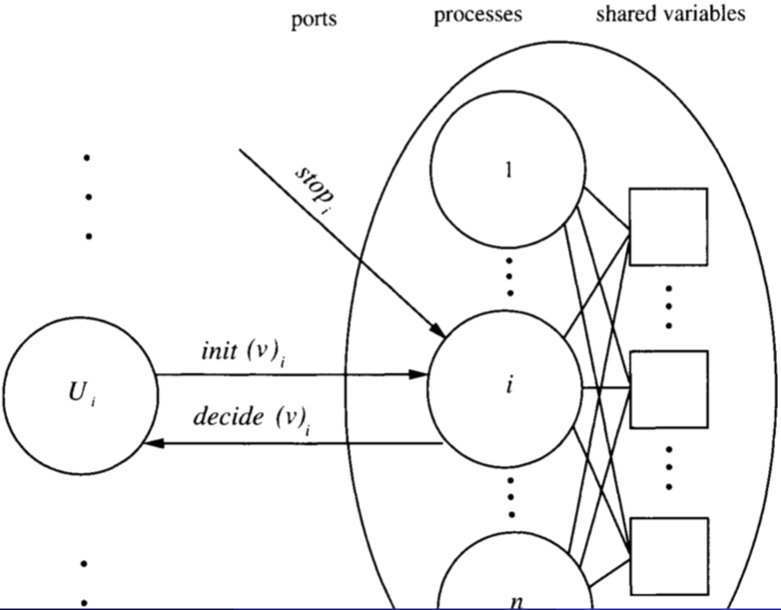
Impossibility of Wait-Free Consensus

- Computational model identical to previous 2 presentations.
- Users U_i can send init$(v)_i$ messages to processor P_i.
- The processors P_i will then finally send decide$(w)_i$ messages back, indicating that w is the consensus value.
- A processor may receive a $stop_i$ message from the environment at any time to model failure.

Consensus

- **Well-Formedness**: For any i, the interaction between U_i and P_i is a prefix of init$(v)_i$ and decide$(w)_i$.
- **Agreement**: All decision values are identical.
- **Validity**: If all init messages have the same value v, then v is the only possible decision value.

Impossibility of Wait-Free Consensus



Impossibility of Wait-Free Consensus

- In this section we will consider **Wait-Free Termination**

Wait-Free Termination

In any fair execution in which init events occur on all ports, a decide event occurs on every non-failing port.

- We will also consider the following restrictions WLOG:
  - The value set is just $V = \{0, 1\}$.

Impossibility of Wait-Free Consensus

- In this section we will consider **Wait-Free Termination**

Wait-Free Termination

In any fair execution in which init events occur on all ports, a decide event occurs on every non-failing port.

- We will also consider the following restrictions WLOG:
  - The value set is just $V = \{0, 1\}$.
  - A is deterministic.

Impossibility of Wait-Free Consensus

- In this section we will consider **Wait-Free Termination**

Wait-Free Termination

In any fair execution in which init events occur on all ports, a decide event occurs on every non-failing port.

- We will also consider the following restrictions WLOG:
  - The value set is just $V = \{0, 1\}$.
  - A is deterministic.
  - Each user simply generates an arbitrary init event, and nothing else.

Impossibility of Wait-Free Consensus

- In this section we will consider **Wait-Free Termination**

Wait-Free Termination

In any fair execution in which init events occur on all ports, a decide event occurs on every non-failing port.

- We will also consider the following restrictions WLOG:
  - The value set is just $V = \{0, 1\}$.
  - A is deterministic.
  - Each user simply generates an arbitrary init event, and nothing else.
  - Every non-failed process always has a locally constrolled step enabled, even after it decides.

Impossibility of Wait-Free Consensus

Definition

An execution α is said to be 0-valent (resp. 1-valent) if the only value that appears in a *decide* event in α or any execution that extends α is 0 (resp. 1). Moreover, 0 (resp 1) must actually occur in such a decide event.

Impossibility of Wait-Free Consensus

Definition

An execution α is said to be 0-valent (resp. 1-valent) if the only value that appears in a *decide* event in α or any execution that extends α is 0 (resp. 1). Moreover, 0 (resp 1) must actually occur in such a decide event.

Definition

An execution α is said to be univalent if it's either 0-valent or 1-valent.

Impossibility of Wait-Free Consensus

Definition

An execution α is said to be 0-valent (resp. 1-valent) if the only value that appears in a *decide* event in α or any execution that extends α is 0 (resp. 1). Moreover, 0 (resp 1) must actually occur in such a decide event.

Definition

An execution α is said to be univalent if it's either 0-valent or 1-valent.

Definition

An execution α is said to be bivalent if there are possible extensions in which 0 shows up, and those in which 1 shows up.

Impossibility of Wait-Free Consensus

Definition

An execution α is said to be 0-valent (resp. 1-valent) if the only value that appears in a *decide* event in α or any execution that extends α is 0 (resp. 1). Moreover, 0 (resp 1) must actually occur in such a decide event.

Definition

An execution α is said to be univalent if it's either 0-valent or 1-valent.

Definition

An execution α is said to be bivalent if there are possible extensions in which 0 shows up, and those in which 1 shows up.

Lemma

Any execution is either univalent or bivalent.

Impossibility of Wait-Free Consensus

Definition

Given an execution α define $\mathrm{ext}(\alpha, i)$ to be the execution α extended by the next action of processor i.

Impossibility of Wait-Free Consensus

The proof follows by the following sequence of lemmas

Theorem

A cannot exist.

Impossibility of Wait-Free Consensus

The proof follows by the following sequence of lemmas

Theorem

A cannot exist.

Lemma

A has an execution which is (i) bivalent, and (ii) for every i, $\mathrm{ext}(\alpha, i)$ is univalent.

Impossibility of Wait-Free Consensus

The proof follows by the following sequence of lemmas

Theorem

A cannot exist.

Lemma

A has an execution which is (i) bivalent, and (ii) for every i, $\mathrm{ext}(\alpha, i)$ is univalent.

Lemma

A has a bivalent initialization.

Impossibility of Wait-Free Consensus

We first look at the proof of the main theorem assuming the first lemma.

Theorem

No wait-free read/write shared memroy algorithm A can solve consensus.

Impossibility of Wait-Free Consensus

We first look at the proof of the main theorem assuming the first lemma.

Theorem

No wait-free read/write shared memroy algorithm A can solve consensus.

Proof.

We will proceed by contradiction. We will consider 3 cases, and derive contradiction in each case.

- By first lemma, fix some execution α satisfying the properties of the lemma.
- Since α bivalent, exist processors i, j such that $\mathrm{ext}(\alpha, i)$ is 0-valent, and $\mathrm{ext}(\alpha, j)$ is 1-valent.

Impossibility of Wait-Free Consensus

Proof.

Case $\#1$: WLOG Process i's next step is a read.

- Consider an extension **of** $\mathrm{ext}(\alpha, \mathbf{j})$ such that i takes no steps, and every other process takes infinitely many steps.

Impossibility of Wait-Free Consensus

Proof.

Case $\#1$: WLOG Process i's next step is a read.

- Consider an extension **of** $\mathrm{ext}(\alpha, \mathbf{j})$ such that i takes no steps, and every other process takes infinitely many steps.
- Since $\mathrm{ext}(\alpha, j)$ is 1-valent, all other processors will decide 1.

Impossibility of Wait-Free Consensus

Proof.

Case $\#1$: WLOG Process i's next step is a read.

- Consider an extension **of** $\mathrm{ext}(\alpha, \mathbf{j})$ such that i takes no steps, and every other process takes infinitely many steps.
- Since $\mathrm{ext}(\alpha, j)$ is 1-valent, all other processors will decide 1.
- Now, since i's step is a read, **only** i's state is affected.

Impossibility of Wait-Free Consensus

Proof.

Case $\#1$: WLOG Process i's next step is a read.

- Consider an extension **of** $\mathrm{ext}(\alpha, \mathbf{j})$ such that i takes no steps, and every other process takes infinitely many steps.
- Since $\mathrm{ext}(\alpha, j)$ is 1-valent, all other processors will decide 1.
- Now, since i's step is a read, **only** i's state is affected.
- Thus if we extend $\mathrm{ext}(\alpha, i)$ with a step of j and the same extension as before, the state is indistinguishable to all the other processes.

Impossibility of Wait-Free Consensus

Proof.

Case $\#1$: WLOG Process i's next step is a read.

- Consider an extension **of** $\text{ext}(\alpha, \mathbf{j})$ such that i takes no steps, and every other process takes infinitely many steps.
- Since $\text{ext}(\alpha, j)$ is 1-valent, all other processors will decide 1.
- Now, since i's step is a read, **only** i's state is affected.
- Thus if we extend $\text{ext}(\alpha, i)$ with a step of j and the same extension as before, the state is indistinguishable to all the other processes.
- Thus, they will all decide 1.

Impossibility of Wait-Free Consensus

Proof.

Case #1: WLOG Process i's next step is a read.

- Consider an extension **of** $\mathrm{ext}(\alpha, \mathbf{j})$ such that i takes no steps, and every other process takes infinitely many steps.
- Since $\mathrm{ext}(\alpha, j)$ is 1-valent, all other processors will decide 1.
- Now, since i's step is a read, **only** i's state is affected.
- Thus if we extend $\mathrm{ext}(\alpha, i)$ with a step of j and the same extension as before, the state is indistinguishable to all the other processes.
- Thus, they will all decide 1.
- But, $\mathrm{ext}(\alpha, i)$ is 0-valent by assumption! Contradiction!

Impossibility of Wait-Free Consensus

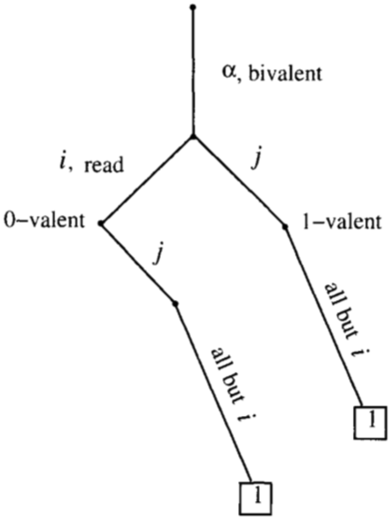


Figure 12.5: Construction for Case 1.

Impossibility of Wait-Free Consensus

Proof.

Case #2: Process i and j write to different variables.

- Consider extensions of α where
  - We first run a step of i, then j
  - We first run a step of j, then i

Impossibility of Wait-Free Consensus

Proof.

Case #2: Process i and j write to different variables.

- Consider extensions of α where
  - We first run a step of i, then j
  - We first run a step of j, then i
- Resulting system state is the same in either case!

Impossibility of Wait-Free Consensus

Proof.

Case #2: Process i and j write to different variables.

- Consider extensions of α where
  - We first run a step of i, then j
  - We first run a step of j, then i
- Resulting system state is the same in either case!
- Since we assumed wait-free if we let all processes run, they must decide on some value.

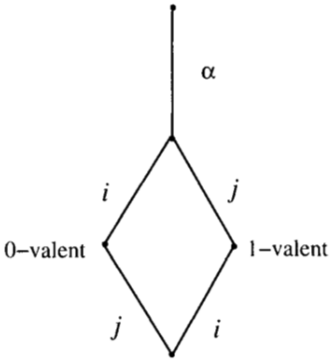
Impossibility of Wait-Free Consensus

Proof.

Case #2: Process i and j write to different variables.

- Consider extensions of α where
  - We first run a step of i, then j
  - We first run a step of j, then i
- Resulting system state is the same in either case!
- Since we assumed wait-free if we let all processes run, they must decide on some value.
- However, any decision value results in contradiction. If they decide 0, contradiction since if we run a step of j first execution should have been 1-valent.

Impossibility of Wait-Free Consensus



Impossibility of Wait-Free Consensus

Proof.

Case #3: Process i and j write to the same variable.

- Once again extend $\text{ext}(\alpha, j)$ so that i doesn't move, and everyone else decides 1.

$\square$

Impossibility of Wait-Free Consensus

Proof.

Case #3: Process i and j write to the same variable.

- Once again extend $\mathrm{ext}(\alpha, j)$ so that i doesn't move, and everyone else decides 1.
- Observe that state of $\mathrm{ext}(\mathrm{ext}(\alpha, i), j)$ is indistinguishable from state of $\mathrm{ext}(\alpha, j)$ to everyone but i.

$\square$

Impossibility of Wait-Free Consensus

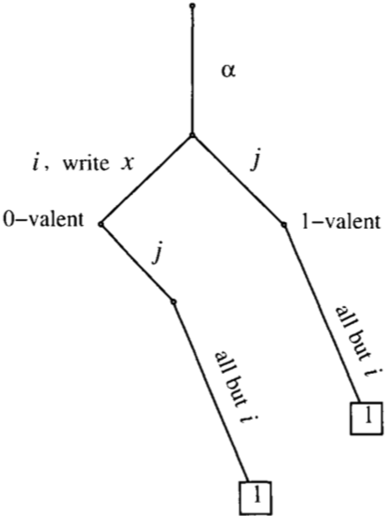
Proof.

Case #3: Process i and j write to the same variable.

- Once again extend $\text{ext}(\alpha, j)$ so that i doesn't move, and everyone else decides 1.
- Observe that state of $\text{ext}(\text{ext}(\alpha, i), j)$ is indistinguishable from state of $\text{ext}(\alpha, j)$ to everyone but i.
- Thus, run same same extension as before, resulting in everyone but i deciding 1, contradicting 0-valence.

$\square$

Impossibility of Wait-Free Consensus



Impossibility of Wait-Free Consensus

We will now prove the first two lemmas.

Lemma

A has a bivalent initialization.

Proof.

- Suppose all initializations are univalent.

Impossibility of Wait-Free Consensus

We will now prove the first two lemmas.

Lemma

A has a bivalent initialization.

Proof.

- Suppose all initializations are univalent.
- Consider α_0 the all-0 initialization, and similarly α_1.

Impossibility of Wait-Free Consensus

We will now prove the first two lemmas.

Lemma

A has a bivalent initialization.

Proof.

- Suppose all initializations are univalent.
- Consider α_0 the all-0 initialization, and similarly α_1.
- Define a sequence $\alpha_0, \alpha_{i_1}, \ldots, \alpha_{i_{n-1}}, \alpha_1$ where each α_{i_k} initializes k processors to 1.

Impossibility of Wait-Free Consensus

We will now prove the first two lemmas.

Lemma

A has a bivalent initialization.

Proof.

- Suppose all initializations are univalent.
- Consider α_0 the all-0 initialization, and similarly α_1.
- Define a sequence $\alpha_0, \alpha_{i_1}, \ldots, \alpha_{i_{n-1}}, \alpha_1$ where each α_{i_k} initializes k processors to 1.
- Thus neighboring initializations in this sequence differ in at most one processor.

Impossibility of Wait-Free Consensus

We will now prove the first two lemmas.

Lemma

A has a bivalent initialization.

Proof.

- Suppose all initializations are univalent.
- Consider α_0 the all-0 initialization, and similarly α_1.
- Define a sequence $\alpha_0, \alpha_{i_1}, \ldots, \alpha_{i_{n-1}}, \alpha_1$ where each α_{i_k} initializes k processors to 1.
- Thus neighboring initializations in this sequence differ in at most one processor.
- Since all univalent, some neighboring α, α' such that α 0-valent, and α' 1-valent.

Impossibility of Wait-Free Consensus

Lemma

A has a bivalent initialization.

Proof Cont'd.

- Since α, α' are neighboring suppose they differ on process i.

$\square$

Impossibility of Wait-Free Consensus

Lemma

A has a bivalent initialization.

Proof Cont'd.

- Since α, α' are neighboring suppose they differ on process i.
- Consider extension of α in which processor i fails immedeately. Since α is 0-valent, must decide on 0.

$\square$

Impossibility of Wait-Free Consensus

Lemma

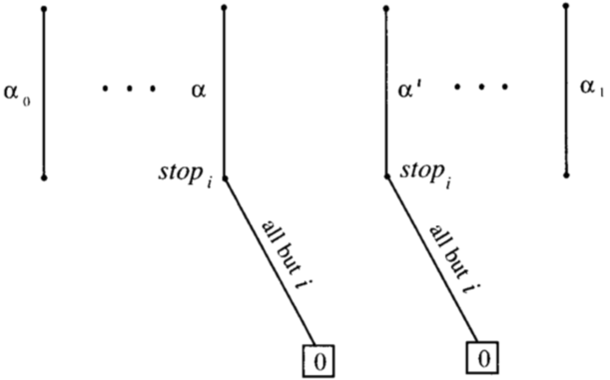
A has a bivalent initialization.

Proof Cont'd.

- Since α, α' are neighboring suppose they differ on process i.
- Consider extension of α in which processor i fails immedeately. Since α is 0-valent, must decide on 0.
- But, we can apply this exact same extension to α' which is 1-valent. However, the system state is indistinguishable to all the other processors from the previous case, and thus will decide 0.

$\square$

Impossibility of Wait-Free Consensus



Impossibility of Wait-Free Consensus

Lemma

A has an execution which is (i) bivalent, and (ii) for every i, $\mathrm{ext}(\alpha, i)$ is univalent.

Proof.

- Suppose every bivalent execution has some i such that $\mathrm{ext}(\alpha, i)$ is bivalent.

□

Impossibility of Wait-Free Consensus

Lemma

A has an execution which is (i) bivalent, and (ii) for every i, $\mathrm{ext}(\alpha, i)$ is univalent.

Proof.

- Suppose every bivalent execution has some i such that $\mathrm{ext}(\alpha, i)$ is bivalent.
- Then there exists an infinte extension, where we keep extending with i resulting in bivalence.

$\square$

Impossibility of Wait-Free Consensus

Lemma

A has an execution which is (i) bivalent, and (ii) for every i, $\mathrm{ext}(\alpha, i)$ is univalent.

Proof.

- Suppose every bivalent execution has some i such that $\mathrm{ext}(\alpha, i)$ is bivalent.
- Then there exists an infinte extension, where we keep extending with i resulting in bivalence.
- However, in this extension no decision can be made, which contradicts our assumption of wait-free termination.

$\square$

Table of Contents

Impossibility of Consensus under Single-Failure

- Now we consider the much weaker case, where we require being tolerant to just a **single** process failure.

Impossibility of Consensus under Single-Failure

- Now we consider the much weaker case, where we require being tolerant to just a **single** process failure.
- However, as we will see even this case is impossible!

Impossibility of Consensus under Single-Failure

- Now we consider the much weaker case, where we require being tolerant to just a **single** process failure.
- However, as we will see even this case is impossible!
- We will now use the following termination definition:

Definition (1-failure Termination)

In any fair execution in which *init* events occur on all ports, if there is a *stop* event on at most 1 port, then a *decide* even occurs on every non-failing port.

Impossibility of Consensus under Single-Failure

Theorem

For $n \geq 2$ there is no algorithm in the read/write shared memory model that solves the agreement problem and guarantees 1-failure termination.

Impossibility of Consensus under Single-Failure

Theorem

For $n \geq 2$ there is no algorithm in the read/write shared memory model that solves the agreement problem and guarantees 1-failure termination.

We state the following lemma without proof

Lemma

If α is a bivalent failure-free input-first execution of A, and i is any process, then there is a failure-free extension α' of α such that the extension $\mathrm{ext}(\alpha', i)$ is bivalent.

Impossibility of Consensus under Single-Failure

Theorem

For $n \geq 2$ there is no algorithm in the read/write shared memory model that solves the agreement problem and guarantees 1-failure termination.

Proof.

- We will use the preceding lemma to construct an execution in which no process decides, contradicting termination.

□

Impossibility of Consensus under Single-Failure

Theorem

For $n \geq 2$ there is no algorithm in the read/write shared memory model that solves the agreement problem and guarantees 1-failure termination.

Proof.

- We will use the preceding lemma to construct an execution in which no process decides, contradicting termination.
- We start with a bivalent failure-free execution α which is simply a bivalent initialization.

$\square$

Impossibility of Consensus under Single-Failure

Theorem

For $n \geq 2$ there is no algorithm in the read/write shared memory model that solves the agreement problem and guarantees 1-failure termination.

Proof.

- We will use the preceding lemma to construct an execution in which no process decides, contradicting termination.
- We start with a bivalent failure-free execution α which is simply a bivalent initialization.
- Then, we keep extending α using the preceding lemma, and taking a step of processor i in the i'th "round" in round-robin order.

□

Impossibility of Consensus under Single-Failure

Theorem

For $n \geq 2$ there is no algorithm in the read/write shared memory model that solves the agreement problem and guarantees 1-failure termination.

Proof.

- We will use the preceding lemma to construct an execution in which no process decides, contradicting termination.
- We start with a bivalent failure-free execution α which is simply a bivalent initialization.
- Then, we keep extending α using the preceding lemma, and taking a step of processor i in the i'th "round" in round-robin order.
- The round-robin order makes the execution fair. However, clearly no process ever makes a decision.

$\square$

Impossibility of Consensus under Single-Failure

Lemma

If α is a bivalent failure-free input-first execution of A, and i is any process, then there is a failure-free extension α' of α such that the extension $\mathrm{ext}(\alpha', i)$ is bivalent.

Impossibility of Consensus under Single-Failure

Lemma

If α is a bivalent failure-free input-first execution of A, and i is any process, then there is a failure-free extension α' of α such that the extension $\mathrm{ext}(\alpha', i)$ is bivalent.

Proof.

- By contradiction. Suppose that there is a bivalent failure-free execution α, and a process i, such that for every failure-free extension α' of α, $\mathrm{ext}(\alpha', i)$ is univalent.
- Thus, $\mathrm{ext}(\alpha, i)$ is univalent. WLOG let it be 0-valent.

Impossibility of Consensus under Single-Failure

Lemma

If α is a bivalent failure-free input-first execution of A, and i is any process, then there is a failure-free extension α' of α such that the extension $\mathrm{ext}(\alpha', i)$ is bivalent.

Proof.

- By contradiction. Suppose that there is a bivalent failure-free execution α, and a process i, such that for every failure-free extension α' of α, $\mathrm{ext}(\alpha', i)$ is univalent.
- Thus, $\mathrm{ext}(\alpha, i)$ is univalent. WLOG let it be 0-valent.
- Since α bivalent, there's some failure-free extension α'' which is 1-valent.

Impossibility of Consensus under Single-Failure

Lemma

If α is a bivalent failure-free input-first execution of A, and i is any process, then there is a failure-free extension α' of α such that the extension $\mathrm{ext}(\alpha', i)$ is bivalent.

Proof.

- By contradiction. Suppose that there is a bivalent failure-free execution α, and a process i, such that for every failure-free extension α' of α, $\mathrm{ext}(\alpha', i)$ is univalent.
- Thus, $\mathrm{ext}(\alpha, i)$ is univalent. WLOG let it be 0-valent.
- Since α bivalent, there's some failure-free extension α'' which is 1-valent.
- Then, $\mathrm{ext}(\alpha'', i)$ must be 1-valent.

Impossibility of Consensus under Single-Failure

Lemma

If α is a bivalent failure-free input-first execution of A, and i is any process, then there is a failure-free extension α' of α such that the extension $\text{ext}(\alpha', i)$ is bivalent.

Proof.

- There must thus be some intermediate extension α', and processor $j \neq i$ such that
  1. $\text{ext}(\alpha', i)$ is 0-valent
  2. $\text{ext}(\text{ext}(\alpha', j), i)$ is 1-valent

Impossibility of Consensus under Single-Failure

Lemma

If α is a bivalent failure-free input-first execution of A, and i is any process, then there is a failure-free extension α' of α such that the extension $\mathrm{ext}(\alpha', i)$ is bivalent.

Proof.

- There must thus be some intermediate extension α', and processor $j \neq i$ such that
  1. $\mathrm{ext}(\alpha', i)$ is 0-valent
  2. $\mathrm{ext}(\mathrm{ext}(\alpha', j), i)$ is 1-valent
- Case analysis similar to previous lemma over the nature of i, j's steps shows contradiction in every case.

Impossibility of Consensus under Single-Failure

Proof.

Case #1: Process i's step is a read

- $\mathrm{ext}(\alpha', ji)$ and $\mathrm{ext}(\alpha', ij)$'s states are indistinguishable to everyone but i.

Impossibility of Consensus under Single-Failure

Proof.

Case $\#1$: Process i's step is a read

- $\text{ext}(\alpha', ji)$ and $\text{ext}(\alpha', ij)$'s states are indistinguishable to everyone but i.
- Extend $\text{ext}(\alpha', ij)$ such that i takes no more steps, while everyone else takes infinitely many.

Impossibility of Consensus under Single-Failure

Proof.

Case #1: Process i's step is a read

- $\text{ext}(\alpha', ji)$ and $\text{ext}(\alpha', ij)$'s states are indistinguishable to everyone but i.
- Extend $\text{ext}(\alpha', ij)$ such that i takes no more steps, while everyone else takes infinitely many.
- By 1-termination and 0-valence, they all decide on 0.

Impossibility of Consensus under Single-Failure

Proof.

Case #1: Process i's step is a read

- $\text{ext}(\alpha', ji)$ and $\text{ext}(\alpha', ij)$'s states are indistinguishable to everyone but i.
- Extend $\text{ext}(\alpha', ij)$ such that i takes no more steps, while everyone else takes infinitely many.
- By 1-termination and 0-valence, they all decide on 0.
- By indistinguishability, run this same extension after $\text{ext}(\alpha', ji)$, and they again decide on 0, contradicting 1-valence.

Impossibility of Consensus under Single-Failure

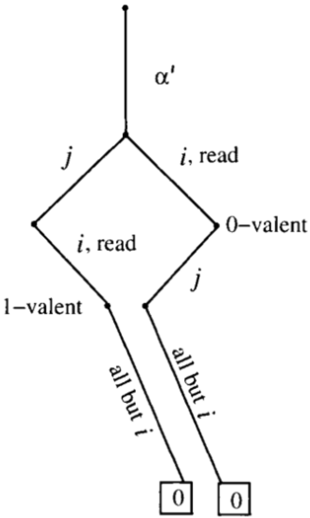
Proof.

Case #1: Process i's step is a read

- $\mathrm{ext}(\alpha', ji)$ and $\mathrm{ext}(\alpha', ij)$'s states are indistinguishable to everyone but i.
- Extend $\mathrm{ext}(\alpha', ij)$ such that i takes no more steps, while everyone else takes infinitely many.
- By 1-termination and 0-valence, they all decide on 0.
- By indistinguishability, run this same extension after $\mathrm{ext}(\alpha', ji)$, and they again decide on 0, contradicting 1-valence.

Other cases identical to previous lemma. $\square$

Impossibility of Consensus under Single-Failure



References I

Fischer, M. J., Lynch, N. A., and Paterson, M. S. (1985).
Impossibility of distributed consensus with one faulty process.
Journal of the ACM (JACM), 32(2):374–382.

Herlihy, M. and Shavit, N. (1999).
The topological structure of asynchronous computability.
Journal of the ACM (JACM), 46(6):858–923.

Lamport, L. et al. (2001).
Paxos made simple.
ACM Sigact News, 32(4):18–25.

Ongaro, D. and Ousterhout, J. (2014).
In search of an understandable consensus algorithm.
In *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*, pages 305–319.

Questions?

Thank You!