Exercise 1

Let C1 and C2 be two consistent cuts. Show that the intersection of C1 and C2 is consistent.

Solution

$$e \rightarrow e' \ \& \ e' \in C1 \rightarrow e \in C1$$
$$e \rightarrow e' \ \& \ e' \in C2 \rightarrow e \in C2$$
$$e' \in C1 \cap C2 \rightarrow e \in C1 \cap C2$$

Exercise 2

Let C1 and C2 be two consistent cuts. Show that the union of C1 and C2 is consistent.

Solution

$$e \rightarrow e' \ \& \ e' \in C1 \rightarrow e \in C1$$
$$e \rightarrow e' \ \& \ e' \in C2 \rightarrow e \in C2$$
$$e' \in C1 \cup C2 \rightarrow e \in C1 \cup C2$$

Exercise 3

Show that every consistent global state can be reached by some run.

Solution (deprecated)

RUN := $[e_1, e_2, ..., e_n, e_{n+1}]$
Switching all the events that are not into the CUT will deal to a consistent run.

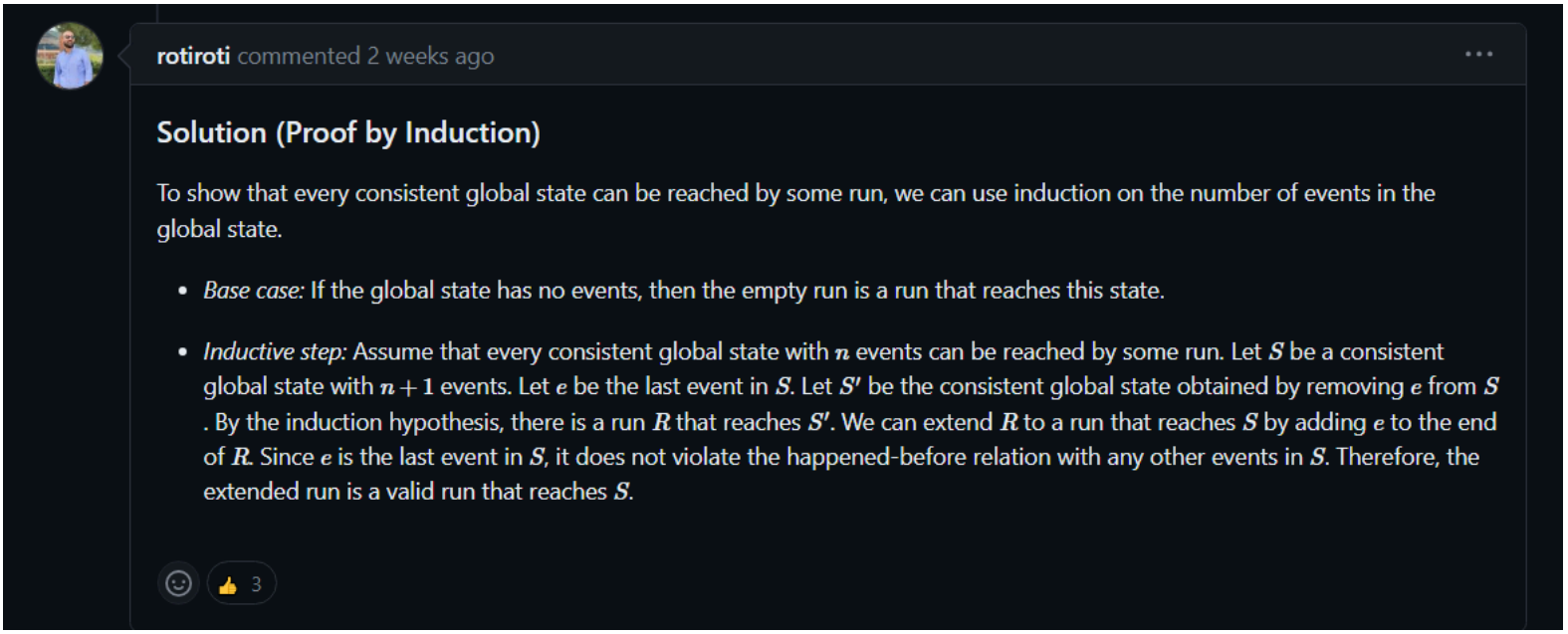
IF $e_i \notin$ CUT and $e_j \in$ CUT and $i > j$:
we can swap those events

So all events $\in$ CUT will be ordered in their causal order on the left of the RUN and all others will be on the right side of the RUN. So the consistent cut will be reached by a consistent run.

Note: this solution was provided by professor Mei.

Solution from Github

(https://github.com/sapienzastudentsnetwork/distributed-system-2023-2024/issues/3#issuecomment-1781303184)



rotiroti commented 2 weeks ago

Solution (Proof by Induction)

To show that every consistent global state can be reached by some run, we can use induction on the number of events in the global state.

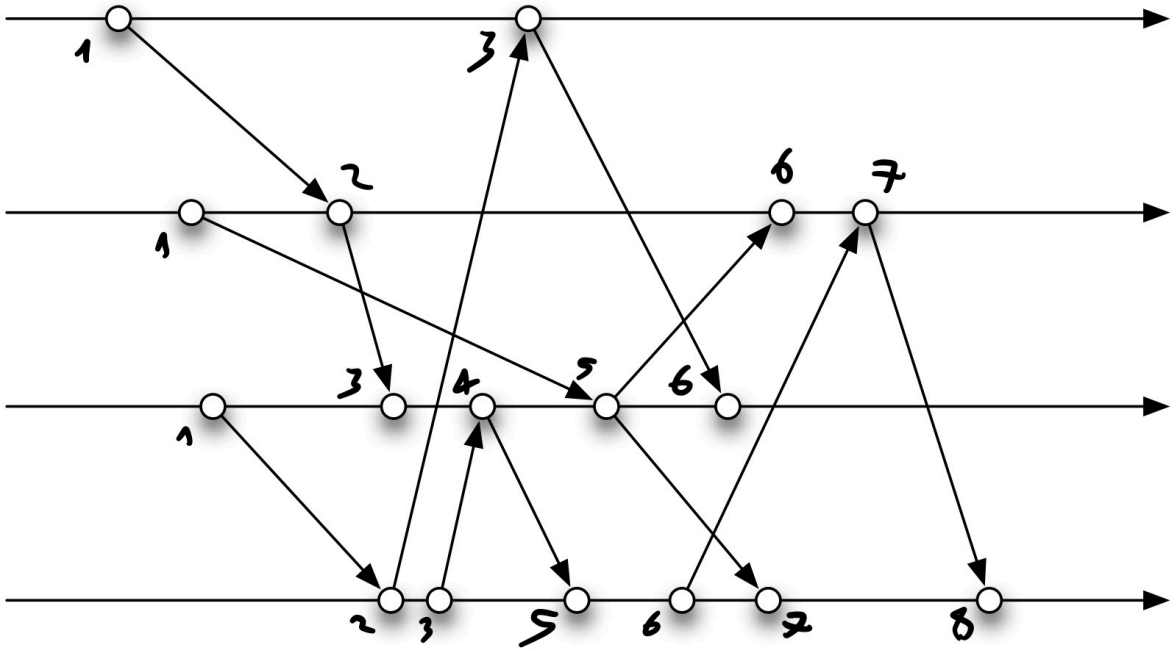
- *Base case:* If the global state has no events, then the empty run is a run that reaches this state.
- *Inductive step:* Assume that every consistent global state with n events can be reached by some run. Let S be a consistent global state with $n+1$ events. Let e be the last event in S. Let S' be the consistent global state obtained by removing e from S. By the induction hypothesis, there is a run R that reaches S'. We can extend R to a run that reaches S by adding e to the end of R. Since e is the last event in S, it does not violate the happened-before relation with any other events in S. Therefore, the extended run is a valid run that reaches S.

3

Exercise 4

Label with proper logical clock all the events of the distributed computation in image vector.pdf. (You can consider events that receive a message and immediately send it as single events.)

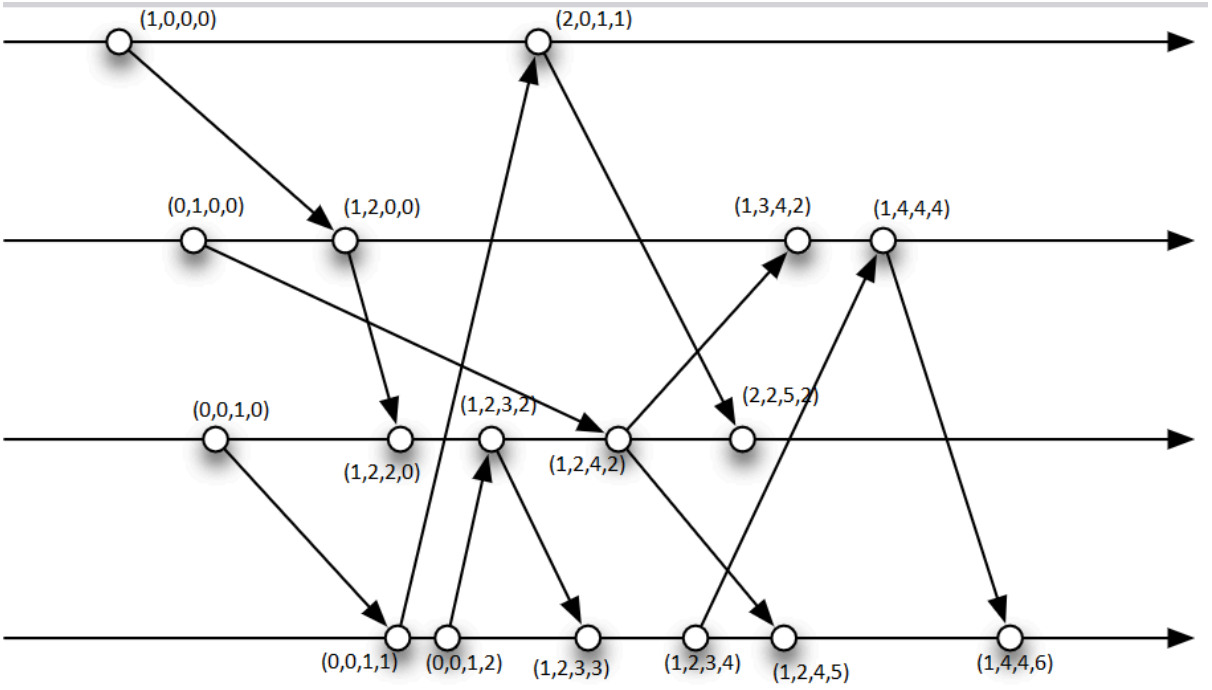
Solution



Exercise 5

Label with proper vector clock all the events of the distributed computation in image vector.pdf.

Solution



Exercise 6

Show that the Chandy-Lamport Snapshot Protocol builds a consistent global state.

Solution 1

Assuming the channels are FIFO (messages from same sender are delivered
in the same order in which they were sent), the network reliable, the channels start recording incoming events when they receive a “take-snapshot” event and they stop recording when they receive from the last process the “take-snapshot” event.
If we take two events e, e_0 for which e -> e_0 $\wedge$ $e_0 \in S \Rightarrow e \in S$ so the Chandy-Lamport algorithm builds a consistent global state. In fact when we receive the last take-snapshot event from a process p_i we are guaranteed that no more events are coming from all the ot loher processes (so there won't be any events e_k -> e_j where $e_k \notin S$ and $e_j \in S$).

Solution 2

The protocol requires that each process, after receiving an initial take snapshot message, sends another take snapshot message to all other processes. This, assuming FIFO communication channels, ensures a consistent snapshot.

Since after the local snapshot is taken, the messages that immediately follow are take snapshot messages, no event that occurs after the snapshot can cause an event that occurred before the snapshot on another process, because the take snapshot message arrives before any subsequent event message due to the channel being FIFO.

Solution 3

Assume communication's channels are FIFO and network reliable.
We list 3 properties of the snapshot protocol that can show the build of a consistent global state:

1. A process Pi before sending the "take-snapshot" message must deliver its outgoing message, if there is one (FIFO assumption).
2. After receiving the first "take-snapshot" from process Pj message, a process Pi starts recording incoming messages from all channels excluding Pj 's one.
3. After receiving another "take-snapshot" message (we have received a "take-snapshot" message yet) from process Pk, we stop recording incoming messages from channel Pk and we can add those to the local history.

If we receive n-1 "take-snapshot" messages we can send our local history.
Being S the global state as the merge of processes local histories got from (4), for each event we have e -> e' $\in$ S. e $\in$ S is ensured by (1) and e' -> S is ensured by (2) and (3).

e -> e' $\in$ S. e $\in$ S is ensured by (1) because it assures that an outgoing event will be delivered before the process that receives the event stops recording events from the sender. e' -> S is ensured by (2) and (3) because receiving the first “take-snapshot” event means that the process will receive it from other n-2 processes, so because channels are FIFO, we record events e and e’ before taking the snapshot.

Solution 4 (from GitHub)

Let e and e' two events. We aim to prove that if e' is part of the snapshot S and $e \rightarrow e'$, then e should also be in S.

Let's demonstrate it by contradiction:

- Assume the contrary:
  $e' \in S \land e \notin S$
- This contradiction is immediately obvious when e and e' are in the same process, because the state of a process is captured atomically.
- If e is a send event and e' is a corresponding receive event on another process, then the marker message causing e' to be in the snapshot S must have been after e, since the Chandy-Lamport Snapshot Protocol assumes that channels are FIFO. Thus, it means the process of event e must have recorded its state (including e) before sending the marker.
  Therefore, e should be in S, which contradicts our assumption.

We conclude from the contradiction that if e' is in the snapshot S and $e \rightarrow e'$, then e must also be in S
.

Exercise 7

Show that the Chandy-Lamport Snapshot Protocol can build a global state that never happened.

Solution 1

Define $\sum^a$ global state of the protocol initialized.
Define $\sum^f$ global state of the protocol end.
Define $\sum^S$ global state constructed by the protocol.

exist a run that goes $\sum^a$ -> $\sum^S$ -> $\sum^f$.

We can label every event as “pre-recording”, if it happened before the first take-snapshot event, and “post-recording”, if it happened after the take-snapshot event.
Now we build $\sum^S$ as the causally ordered set of the recorded snapshot events. So from the run we swap every near events <e, e’> if e is post-recording and e’ is pre-recording, that’s because channels are FIFO and a pre-recording event cannot causally precede a post-recording event, until all post-recording events follow the pre-recorded ones.

That creates a consistent global state which could not happen during the run, because unrelated post-snapshot and pre-snapshot events could be swapped even if the first happened before the pre during the run.

That means that the protocol won’t include any post-recorded events even if they happened into the run yet. So the snapshot of the system will be consistent but never happened.

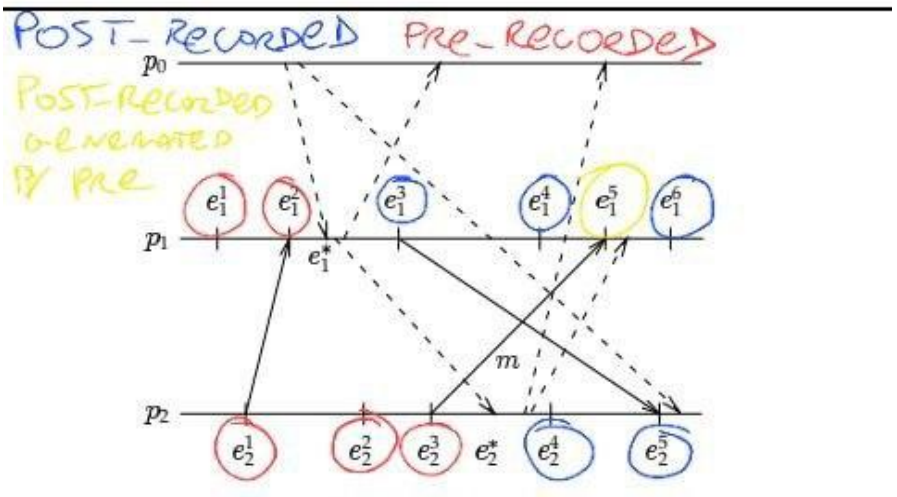
followed by processes p_1 and p_2 while executing the protocol be

$$r = e_2^1\, e_1^1\, e_1^2\, e_1^3\, e_2^2\, e_1^4\, e_2^3\, e_2^4\, e_1^5\, e_2^5\, e_1^6$$

or in terms of global states,

$$r = \Sigma^{00}\, \Sigma^{01}\, \Sigma^{11}\, \Sigma^{21}\, \Sigma^{31}\, \Sigma^{32}\, \Sigma^{42}\, \Sigma^{43}\, \Sigma^{44}\, \Sigma^{54}\, \Sigma^{55}\, \Sigma^{65}.$$

Let the global state of this run in which the protocol is initiated be Σ^{21} and the global state in which it terminates be Σ^{55}. Note that run r does not pass through the constructed global state Σ^{23}. As can be verified by the lattice of Figure 3, however, $\Sigma^{21} \rightsquigarrow \Sigma^{23} \rightsquigarrow \Sigma^{55}$ in this example. We now show that this relationship holds in general.



In this example blue events before the yellow one won't be included in the snapshot, but they actually happen in the global state.

Exercise 8

What good is a distributed snapshot when the system was never in the state represented by the distributed snapshot? Give an application of distributed snapshots.

Solution

It's not possible to have a global clock among a distributed system, in fact the most important thing in a distributed system is to have the cause-effect relationship between events, because if an event may influence another we must not lose this information. The distributed snapshot, otherwise, helps us to solve Global Predicate Evaluation. The goal of GPE is to determine whether the global state of the system satisfies some predicate. Global predicates are constructed so as to encode system properties of interest in terms of state variables. Examples of distributed system problems where the relevant properties can be encoded as global predicates include deadlock detection, termination detection, token loss detection, unreachable storage (garbage) collection, checkpointing and restarting, debugging, and in general, monitoring and reconfiguration. In this sense, a solution to GPE can be seen as the core of a generic solution for all these problems; what remains to be done is the formulation of the appropriate predicate and the construction of reactions or notifications to be executed when the predicate is satisfied.
In fact, without a snapshot that builds a consistent global state we could occur in some problems like ghost deadlock, in which a deadlock is detected even if it will never happen in a consistent run. Also keeping the consistent global history will allow the system to recover until the snapshot checkpoint.

Exercise 9

Consider a distributed system where every node has its physical clock and all physical clocks are perfectly synchronized. Give an algorithm to record global state assuming the communication network is reliable. (Note that your algorithm should be simpler than the Chandy–Lamport algorithm.)

Solution

This means we have access to a real-time global clock. Assuming message delays are bounded to δ, the monitor process starts the protocol by broadcasting a "take a snapshot" message at time t0 for time $t \geq t0 + \delta$. At time t, each process sends its snapshot to the monitor, which can reorder the events in increasing timestamp order. Number of messages goes from $O(n^2)$, required by the Chandy-Lamport algorithm, to O(n).

Exercise 10

What modifications should be done to the Chandy–Lamport snapshot protocol so that it records a strongly consistent snapshot (i.e., all channel states are recorded empty).

Solution (<u>demonstrated</u> that doesn't work)

All the assumptions for the Chandy-Lamport snapshot protocol are the same, the channels must be FIFO and the network reliable. The protocol can be modified in this way:

1. The P_0 process send a "clear" message in broadcast to all processes
2. When a process P_i receive the "clear" message it broadcasts it to all the others (excluded the P_0 process
3. Once a process has received all "clear" messages from all other processes it sends a "take-snapshot" message in broadcast.
4. Once a process has received all "take-snapshot" it send it's local state to P_0

Simple demonstration of the protocol: the FIFO assumption assures us that the clear message will be the last message that will be sent by a process, in this way when we will send a snapshot message we won't have any message incoming.

Solution 2

Simply doesn't record the channels (why? see the notes for solution 3).

Solution 3

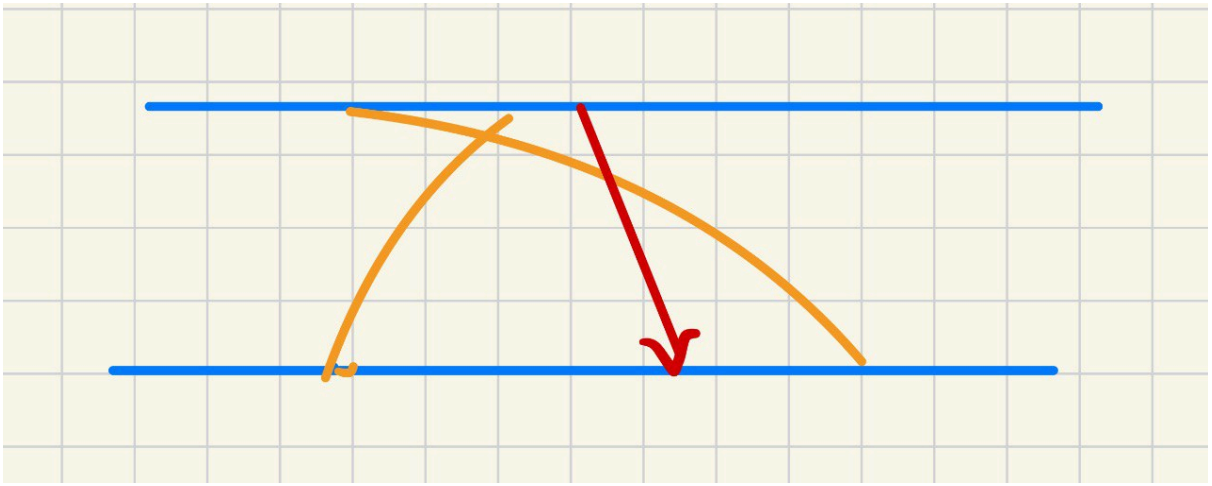
Repeat when a process has received all the clear messages from others, it sends a ACK message which contains 'true' if it has channel's empty, otherwise it will send 'false'. All processes wait for all ACK messages and if at least one of them is 'false' then it restarts the protocol from the new point.

Note on solution 3: of course it can go infinitely, but if you want your channels empty this is the only real working solution found; then I asked the professor what in the original protocol the channels are used for and he told me 'nothing' since they are not included in the snapshot. If you want to waste your time to solve this problem go ahead, but it's required a high comprehension of the Snapshot protocol.

Exercise 11

Show that, if channels are not FIFO, then Chandy–Lamport snapshot algorithm does not work.

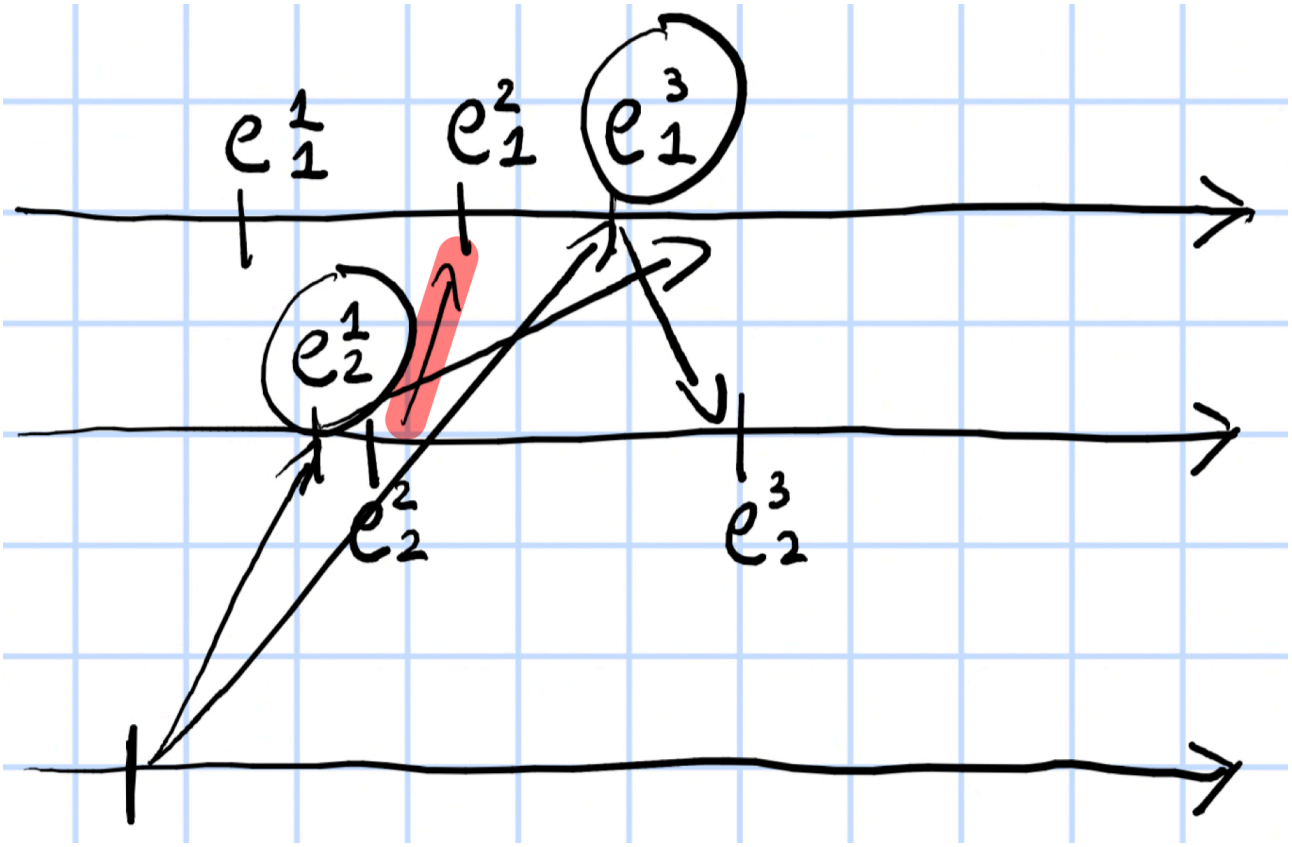
Solution



Yellow arrows are the take snapshot messages. The red one is a message that could not exist if the channels are FIFO, in this case the snapshot's global state is inconsistent.

Formally: $e \rightarrow e' \wedge e' \in S \nRightarrow e \in S$ with e the *send* event in the red arrow and e' the *receive* event.

Another example:



Exercise 12

Let S0 be the global state when the Chandy-Lamport snapshot protocol starts, S be the global state built by the protocol, and S1 be the global state when the protocol ends. Show that S is reachable from S0 and that S1 is reachable from S. Remember that S might not have happened.

Matteo's Solution

Taking the actual run r that the system followed, such run will pass through S_0 and S_1.

If we mark all the events in r that happened before the snapshot (and are therefore included in S) with a $*$, we would have some instances in the run where an event e appears before an event e^*. Since the event e^* is in the snapshot while the event e isn't and the snapshot is consistent, it is not possible that $e \rightarrow e^*$.

So it's safe to swap any pair of (e, e^*) events in order to have a run with all e^* events appearing before all e events. Such run would then go from S_0 to S and eventually to S_1.

Solution

In the context of the Chandy-Lamport snapshot protocol, let's address the question of reachability between the global states S0, S, and S1.

1. **S is reachable from S0:**

   When the Chandy-Lamport snapshot protocol starts (S0), it initiates the process of capturing a snapshot. The protocol works by propagating marker messages through the communication channels, and processes record their local states and channel states as they encounter these markers.

   The global state S represents the snapshot captured during the protocol execution. Since S is built during the protocol's execution, it is, by definition, reachable from S0. The protocol ensures that the global state S is an outcome of the protocol's execution starting from the initial state S0.

2. **S1 is reachable from S:**

   The Chandy-Lamport snapshot protocol eventually ends and reaches a termination state, resulting in a new global state S1. In this termination state (S1), all processes have completed their snapshot-taking activities, and all channel states are recorded as empty.

   Because S1 is the result of the protocol's execution, and it captures the system's state after the protocol has concluded, it is, again by definition, reachable from S. S is an intermediate state during the execution of the protocol, and S1 represents the final state when the protocol ends.

   Now, it's important to note the final part of your question: "Remember that S might not have happened." This is an acknowledgment that while S is an intermediate state during the execution of the protocol, it doesn't necessarily mean that S is a state that the system will always reach. It depends on when and how the protocol is initiated and executed. S1, on the

other hand, represents a consistent snapshot at the end of the protocol execution, where all channel states are guaranteed to be empty. This means that, while S1 is reachable from S, the existence of S is contingent on the initiation and execution of the Chandy-Lamport protocol.

Another possible solution that seems correct in my opinion (I'm Lorenzo) is this one: https://github.com/sapienzastudentsnetwork/distributed-system-2023-2024/issues/12#issuecomment-1784826328

Exercise 13

Give an ACP that also satisfies the converse of condition AC3. That is,
If all processes vote Yes, then the decision must be Commit (AC3 condition). Why is it not a good idea to enforce this condition?

"sometimes is a good idea to abort"

Solution

Suppose every process votes YES except P , for which we do not know its decision (maybe it crashed or its message takes too long to deliver). If the decision is COMMIT anyway and P recovers, it may not have voted YES before and unilaterally decide ABORT. Thus the protocol would not be safe.

Exercise 14

Consider 2PC with the cooperative termination protocol. Describe a scenario (a particular execution) involving site failures only, which causes operational sites to become blocked.

Solution

p is the *initiator* and *q* a *responder* in the termination protocol. There are 3 cases:

1. *q* has already decided Commit (or Abort): *q* simply sends a COMMIT (or ABORT) to *p*, and *p* decides accordingly,
2. *q* has not voted yet: *q* can unilaterally decide Abort. It then sends an ABORT to *p*, and *p* therefore decides Abort.
3. *q* has voted Yes but has not yet reached a decision: *q* is also uncertain and therefore cannot help *p* reach a decision.

With this protocol, if *p* can communicate with some *q* for which either (1) or (2) holds, then *p* can reach a decision without blocking. On the other hand, if (3) holds for all processes with which *p* can communicate, then *p* is blocked. This predicament will persist until enough failures are repaired to enable *p* to communicate with a process *q* for which either (1) or (2) applies. At least one such process exists, namely, the coordinator. Thus this termination protocol satisfies AC5.
Even if the probabilities of having a block are reduced, they aren't eliminated.

Exercise 15

Show that Paxos is not live.

Solution

In Paxos algorithm an acceptor send a learn message only if round *rnd* > *i*, where *rnd* is the round in which the acceptor promised not to vote for any lower round and *i* is the round where the proposer proposed the value. For this reason if at least two proposers continue to send “propose messages” with higher rounds the acceptors won’t be able to choose any value, and might be stuck in this loop. For this reason a coordinator should be elected by nodes; but this won’t fix the problem, in case of coordinator’s failure the problem will persist. In fact, with the FLP theorem we cannot have all of these three properties in the same distributed system: liveness, fault tolerance and agreement.

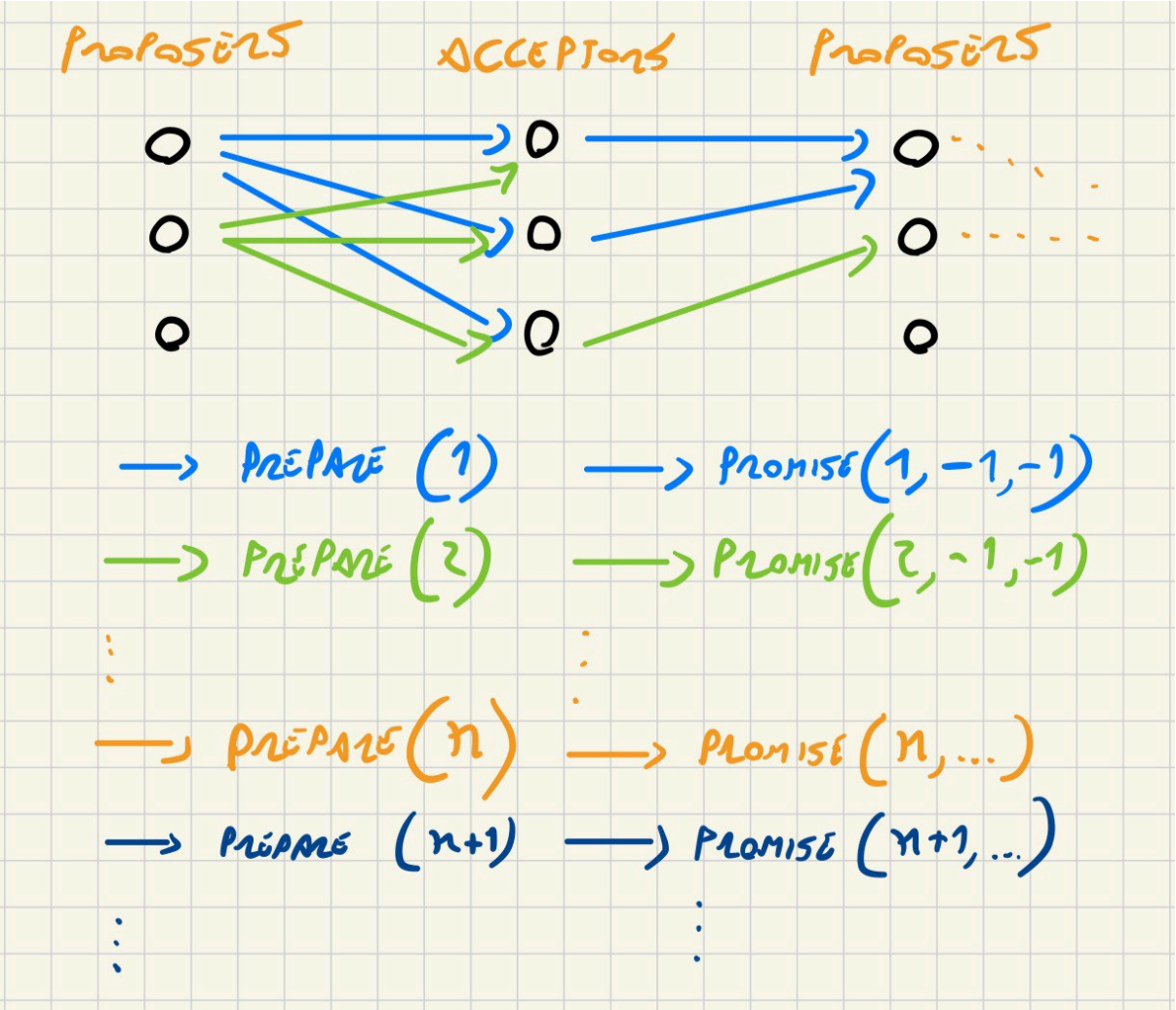
from “A Simpler Proof for Paxos and Fast Paxos”:

~~In the presence of faults, leader election is impossible as well. However, with Paxos we can accept that the election fails and that two or more leaders are chosen. In that case we cannot guarantee liveness but Paxos is there to guarantee safety whatsoever.~~

~~Taking the actual run that the system followed, such run will pass through~~ ~~and~~ ~~.~~

~~If we mark all the events in~~ ~~that happened before the snapshot (and are therefore included in~~ ~~) with a~~ ~~, we would have some instances in the run where an event~~ ~~appears before an event~~ ~~. Since the event~~ ~~is in the snapshot while the event~~ ~~isn't and the snapshot is consistent, it is not possible that~~ ~~.~~

~~So it's safe to swap any pair of~~ ~~events in order to have a run with all~~ ~~events appearing before all~~ ~~events. Such run would then go from~~ ~~to~~ ~~and eventually to~~ ~~.~~



Exercise 16

Assume that acceptors do not change their vote. In other words, if they vote for value v in round i, they will not send learn messages with value different from v in larger rounds. Show that Paxos, with this modification, is safe. Unfortunately, the modification introduces a severe liveness problem (the protocol can reach a livelock).

Solution

For demonstrate the safety we must demonstrate those three properties:

1. Only a proposed value may be chosen
2. Only a single value is chosen
3. Only a chosen value may be learned by a correct learner

1: easy to check, in fact only the proposer's value will be memorized by acceptors.
2: Since our assumption is that the value once proposed is never changed, we can see that the chosen value can be only one. That's because that value will never change in the acceptors memory, so they'll propose their value until one gets enough votes and the learners will choose it.
3: learners will learn only the immutable value that will reach the quorum.

Exercise 17

How many messages are used in Paxos if no message is lost and in the best case? Is it possible to reduce the number of messages without losing tolerance to failures and without changing the number of proposers, acceptors, and learners?

Solution

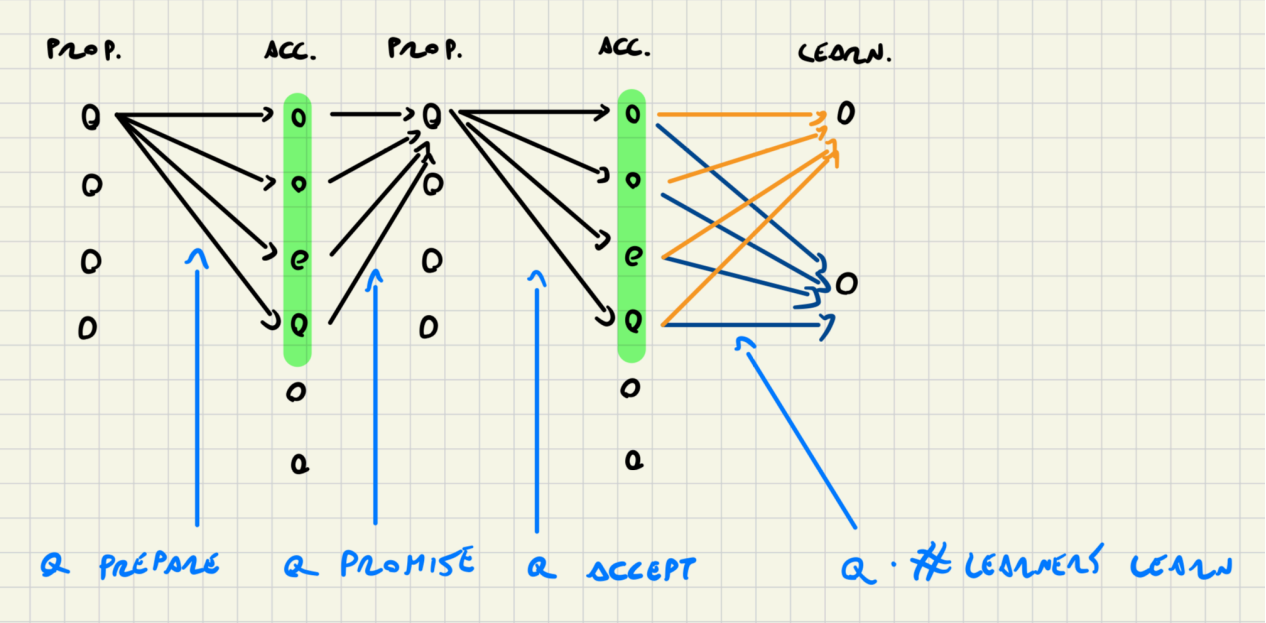
In classic Paxos we need $3n + n \cdot l$ messages:

- n prepare
- n promise
- n accept
- $n \cdot l$ learn (where l is the number of learners)

Consider the quorum Paxos needs to get a majority:

$$Q = n - f \text{ and } f = \left\lfloor \frac{n-1}{2} \right\rfloor$$

If we send the messages only to a majority, the number of messages is reduced at least to $3Q + Q \cdot l$. Fault tolerance is provided by using this number of messages only in one round of Paxos, and then reverting to classic Paxos in case of failures.



Exercise 18

Assume that you remove the property that every round is associate to a unique proposer. After collecting a quorum of n-f promises (where n is the number of acceptors and f is such that n=2f+1), the proposer chooses one of the values voted in max round in the promises (of course it is not unique, the proposer chooses just one in an arbitrary way). Show that Paxos is not safe any more.

Solution 1

If more proposers can send a prepared request with the same round number the acceptors will send promises with the same round number. That will cause the proposers to send different values in the same round (accept message) and the acceptors will send learn messages with different values that break the second rule of the safe's properties (only a single value is accepted).

Solution 2

Suppose we have n = 7 acceptors and f = 3 failures for round i. The proposer collects a quorum of n − f = 4 promises, where some last voted for value 1 in round j and some last voted for value 2 in round j. But we still do not know the last vote of the other three acceptors and these might be enough to form a quorum of n−f = 4 votes in round j either on value 1 or value 2. Then the proposer cannot make a safe choice for round i.

Exercise 19

Assume that all proposers are learners as well. Let even rounds be assigned to proposers with the rules that we know. Moreover, If round 2i is assigned to proposer p, then also round 2 i+1 is assigned to proposer p. Odd rounds are "recovery" rounds. If round 2i is a fast round and if the proposer of round 2i sees a conflict (it is also a learner), then the proposer immediately sends an accept for round 2i+1 with the value that has been most voted in round 2i, without any prepare and any promise. Is safety violated? If yes, show an example. If not, demonstrate safety

Solution

It doesn’t violate safety. In Paxos, the safety theorem states that if an acceptor a has voted for value v in round i, then no value v' != v can be chosen in rounds that are smaller than i. In Fast Paxos, when the coordinator recognizes a conflict in round i, it can always start a round that is greater than i in order to solve it. It doesn’t break the second rule of safety, that’s because the value v will be accepted at a greater round.

Solution by Luca

Safety is guaranteed with the proprieties bla bla bla
Only the second property (only one value is chosen) is problematic

In the recovery round, choosing the value that's most voted in round 2i the proposers act as coordinators with a recovery strategy already well defined, so every conflicting proposer will agree and accept a correct, single, value (the protocol is safe).
If there are failures in receiving the LEARN message back (so one proposer gets a majority of value X and will vote for X in the recovery round, but the other gets a majority of value Y and will vote for Y in the recovery round, creating the same problem of the normal round) or if both proposers receive the same amount of votes for the two values, we can't decide a common value between the two proposers. If we suppose to have only one recovery round, we can choose to simply abort the decision process, but keeping the protocol safe anyway.
With more recovery rounds we will eventually reach a decision using the recovery strategy and, again, keeping the protocol safe.

Exercise 20

You are an optimization freak. You realize that, in some cases, it is not necessary that the proposer collects n-f’ (the Fast Paxos quorum) promises to take a decision. Which is the minimum quorum and under what hypothesis this minimum quorum is enough to take a decision?

Solution

In Fast Paxos, we need at least a quorum of 2/3 acceptor in order to make safe choices. In Fast Paxos we can also distinguish between 2 phases: prepare, promise phase and accept, learn phase. Quorum intersection is required only between phase one quorum and any pair of phase two fast round quorums. We can conclude that a simple majority is sufficient for phase one of a fast round, instead of requiring 2/3 of the acceptors.

Non ho capito… :(

Drive solution

Fast Paxos requires that any three fast round quorums Qf intersect, i.e. $\forall Q, Q0, Q00 \in Qf : Q \cap Q0 \cap Q00 \neq \varnothing$, meaning that a quorum of 2/3 of acceptors is
needed to make safe choices. However, we can differentiate between phase-1 (prepare, promise) quorums Q1 and phase-2 (accept, learn) fast round quorums Q2f . We then observe that quorum intersection is only required between phase-1 quorums

Q1 and any pair of phase-2 fast round quorums Q2f , i.e. ∀Q ∈ Q1, ∀Q0, Q00 ∈ Q2f : Q ∩ Q0 ∩ Q00 6= ∅. The weakened intersection requirements show that a simple majority is sufficient for phase-1 of a fast round, instead of requiring 2/3 of the acceptors.

Exercise 21

Show a possible implementation of a failure detector and discuss its properties.

Solution

There are various possible solutions for implementing a failure detector, all solutions may be transformed into stronger failure detectors. A simple and strong way to do it is to force all nodes to send an “im alive” message to all other nodes. When a node doesn't receive the “im alive” message in a time ΔT + **δ** it suppose that node is dead. After time ΔT + **δ** (the same as timeout) all processes exchange their suspect set and update their. The update must work as follow:

- ➔ if node i is suspected by other nodes but not me: ignore.
- ➔ if node is not suspected by other node but from me: remove node from suspected and increase **δ**.
- ➔ if all nodes suspect a node dead: add it to the suspected list.

Properties:

- **Strong completeness**, all processes that crash will be detected by all other correct processes: If a process A suspects another process B after time ΔT + **δ** (because it didn't get the live message), after receiving the table of suspects from all other processes, A can assume that B is alive if another process C doesn't suspect it (because C received the live message from B before its timeout).
- **Eventually strong accuracy:** there is a time after which correct processes are not suspected by any other correct process. In the update step when a process receives the other table, if there is a conflict for a process, its entry is updated with the “alive” value. So even the process that suspected the death of a process that’s alive won’t suspect it anymore.

Exercise 22

Show that you can make Paxos live with a failure detector in W.

Solution

Assume that each proposer in the system has a unique ID. We can select the coordinator as the proposer with the lower ID. The failure detector checks if the coordinator is still alive, and in case of failure, the proposer with the lower ID is selected as the coordinator. Starting with a weak failure detector, we can create a strong failure detector using the following algorithm:
Note: a strong failure detector S has strong completeness and weak accuracy, but it's important to have strong completeness in order to make Paxos live.

initially: $\text{output}_p = \varnothing$
when D_P changes:
$\text{suspects}_p \leftarrow D_P$
send($P, \text{suspects}_p$) to everybody
when msg is received (q, S_q): (S_Q contains the suspects from Q)

$$\text{output}_p = (\text{output}_p \cup S_q) - \{q\}$$

note: q is removed because it is the node that sends the message, so it is alive.

Note: this does not violate the FLP theorem because it introduces non-determinism and probabilistic progress, rather than offering a deterministic solution to the consensus problem. (Consensus by randomization)

Solution 2

By definition a weak failure detector has weak accuracy and weak completeness. Weak accuracy means that there exists at least one process p which is never suspected by any correct process. That means that the intersection of correct processes is never void; so we can exchange the correct set with each other and elect as coordinator the smallest node id in the intersection. So we will make Paxos live with ONLY a weak failure detector.

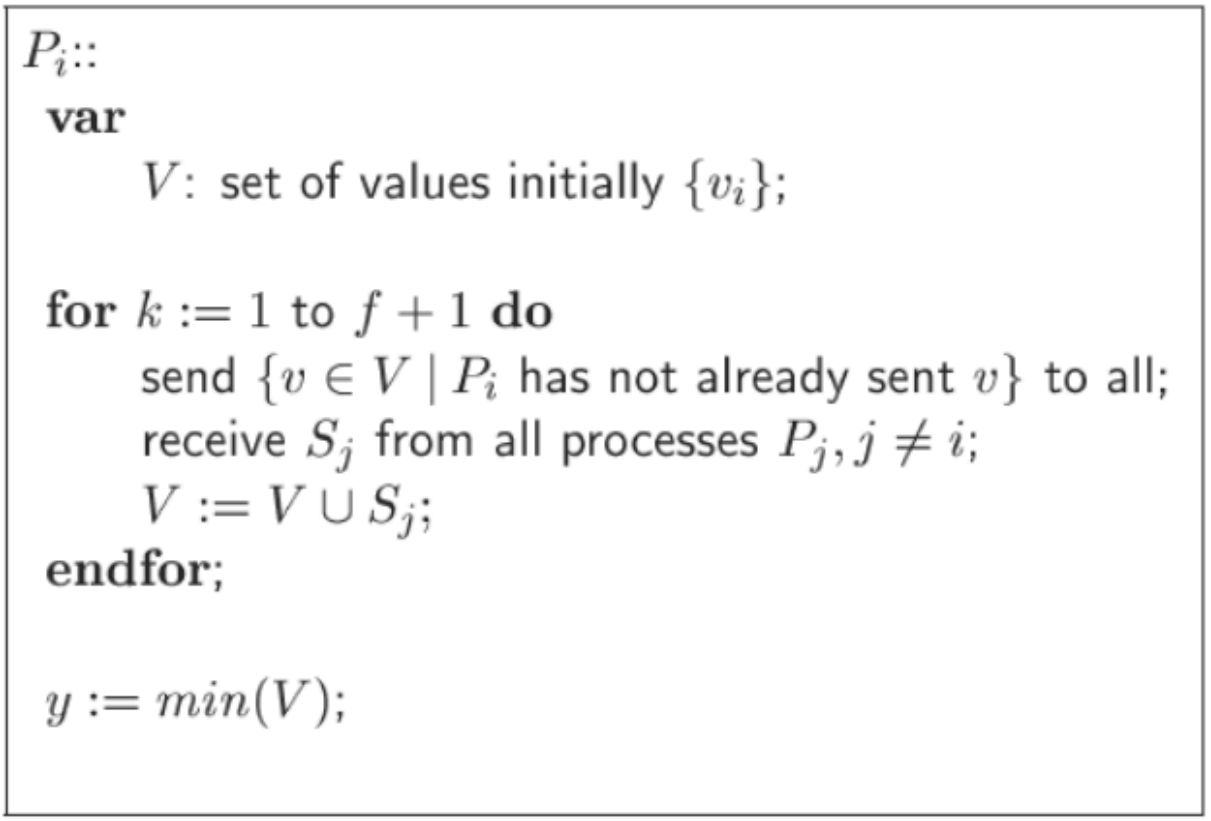
Note: scusate l'orario ma il mio cervello ogni tanto si attiva ~~la sera~~.

Exercise 23

Show how to solve consensus in a synchronous distributed system with at most f crash failures.

Solution

To solve consensus in a synchronous distributed system with at most f crash failures and n nodes, each process maintains a set V of values proposed by other processes. Initially, the set contains only the process's value, but then it's updated.
In each round, a process sends to all other processes the values from V that it has not sent before. After f+1 rounds, each process decides on the minimum value in V.



TAKEN FROM: https://cnitarot.github.io/courses/ds_Fall_2016/505_consensus.pdf page 11
https://cnitarot.github.io/courses/ds_Fall_2016/index.html#lectures

Exercise 24

Build a run of the Ben-Or randomized consensus algorithm that never terminates.

Solution on github

For testing progress for F<N/2, we use the **Progress** property that says *eventually all processes decide something*. If we start with the same p1v values at all nodes, progress is guaranteed for N>2F. In that case all nodes will see that value as the majority value and propose it as the p2v value in round 1, and they all decide in phase2 of that round.

But what about if the initial p1v values have a mixture of 0s and 1s? They may not decide because it may not be possible for any available node to observe a majority 0 or 1 from their sample of N-F values, and the default/skip value $p2v = -1$ will be proposed for the phase 2. When such an ambiguity is possible, the decision can then be postponed forever by choosing the worst possible "random" assignments to the nodes for the next round. This is of course not any more a uniformly random assignment of 0s or 1s to p1v, but a pathological/malicious value assignment by the model checker to show how the Progress can be violated.

For example for N=4 and F=1, INPUT=<<0,1,1,1>> will violate the Progress property. The model checker will hide the "p1v=1" from node, and since the other nodes would not see the value "1" in the majority (which is 3 for N=4), they will all propose p2v=-1. The model checker also will find the "random" assignments that will extend this indecision by not giving majority to any value. (On the other hand, for F=0, in this setup, the majority value of "1" will be seen by all nodes, and the system will decide in one round.)

Exercise 25

Consider an asynchronous system of 5 processes that run the Ben-Or randomized consensus algorithm. The number of failures that the system allows is 2. Show that, if at most 2 failures occur, then the probability that the protocol terminates after x rounds (or more) is smaller than alpha^x, for some alpha.

Solution

Warning: this solution may be wrong, but we'll ask the professor today (hopefully)

The probability of reaching a majority with 5 nodes and 0 crashes is the probability to get the majority (3 nodes) to randomly get the same value. This is a binomial distribution problem, where the probability of success can be calculated as follows:
$P(x \geq 3) = P(x = 3) + P(x = 4) + P(x = 5)$
Where x is a random variable representing the number of successes. Each term can be calculated using the binomial probability formula:
$P(x = k) = C(n, k) \cdot (p^k) \cdot ((1 - p)^{n-k})$ = p^n C(n,k)

Where $C(n, k)$ is the binomial coefficient, p is the probability of success $(1/2)$ in this case and n is the number of trials.

If two nodes crash, the problem becomes the probability that all the 3 nodes alive get the same value, so $(\frac{1}{2})^2 = \frac{1}{4}$ (the first coin flip could be 1 or 0, the important thing is that the remaining two will get the same result).

So the probability will be:

$$P(\text{process terminates after } x \text{ rounds}) = (\frac{1}{4})^x$$

So the probability will be smaller than α^x for any α greater than $\frac{1}{4}$.

Parte 2

Question 1

Fork in blockchain (hash after computation)

A fork is a division in the block chain that it's verified when two different new blocks are accepted because both got the hash right. In this case, the blockchain keeps working in parallel on both, creating two different chains. The longest chain (the chain with the most effort, which is usually the longest) will be preferred because it will have the updated values. Once the best chain is found, the other is discarded.

Question 2

Describe Akamai, how does a CDN work, and how does DNS work in akamai (deprecated)

Akamai is a platform that offers a Content Delivery Network service. CDN are a series of techniques to deliver the content to users as fast as possible. A solution is to use a distributed system (a DNS) to translate addresses from a written form to a numeric one. There is a central server, called DNS root server, which is responsible for storing the top level domains (.com, .gov, .it, ...). These domains have their servers that keep addresses of other domains below them (SLDs - Second Level Domains). A user has to set his DNS server and use it to ask for address mapping. The server first asks the DNS root, then, reading the answer, will ask the server right below, and so on. Usually, all of this information is cached, so we don't have a lot of messages. Caching raises the consistency problem because if someone changes the IP of a website already present in our cache, we have to wait some time before we also get the updated one. Also, the address translation does not depend on where the user is. The Akamai idea is to make it dependant. Akamai changed the traditional DNS system by using DNS translation that depends on where you are. The user will get the address from the nearest server. This is done by akamaizing every heavy file inside a domain (caching it into the akamai server). So the first file (the HTML page) is retrieved from the original website server, while all the other files are retrieved from the Akamai nearest server, called *Local Akamai Storage* (based on the IP address sent along with the initial request). This speeds up the process, but the nearest server may not have the file ready (the file could also be not replicated in that server, because every local storage replicates content that is relevant to local users): in this case, the content is taken from another server and then the file is cached so that next time it will be available. However, we still have consistency problems, since we may read from an old file, and load balancing problems.

Answer 2.0 (Describe Akamai, how does a CDN work, and how does DNS work in akamai)

Akamai is a CDN service that uses IP addresses and their location to provide the fastest response for resource requests. Before going into details let's see what happens when we request a page:

1. the browser makes a GET request to the web server to retrieve the html page.
2. for every resource into the page e.g. images, videos, ecc… it makes another http request.

So where is the problem? When we make a request to a webpage that is very distant on location from us, the browser requesting all the resources will heavily slow down the process.

The Akamai idea was to speed up the process requesting the resources from the nearest Akamai's CDN servers. So the resources doesn't need anymore to cross continents.
To do so, they use a DNS system that returns the IP address of the closest Akamai storage server to your location. How can it return the nearest server's IP? To solve this the DNS replies with a CNAME alias that points to Akamai's GTM servers. GTM replies with the geographically closest IP addresses for the server not overloaded at an optimal data center location (this is a simplified version of the protocol, it should be everything that is needed to know for the written exam/midterm, however if you want to know more). But the nearest server may not have the file ready (the file could also be not replicated in that server, because every local storage replicates content that is relevant to local users): in this case, the content is taken from another server and then the file is cached so that next time it will be available. In the Akamai case and the CAP theorem we have:

- **Availability**: you get something in response.
- **Consistency**: you get the latest version of the requested resource.
- **Partition Tolerance**: if partitions can't communicate anymore, some servers could have old data or have not data for a request.

Question 3

Talk about the Cap Theorem

CAP is short for Consistency-Availability-Partition tolerance. It has been first introduced to describe the compromise between the consistency, availability and partition tolerance:

- **Consistency:** The system answers every client request with the most updated values
- **Availability:** The client always get an answer
- **Partition tolerance:** Communication isn't reliable and so servers can be divided in groups (partitions) that can't communicate between them

When you have an unreliable network you can either

- loose availability
- loose consistency
- loose partitioning

The implication of the CAP theorem is that you can't have all three properties together.

How to choose what property to lose? There are no right or wrong answers, it depends on the situation and what is your system's goal.

Examples:

- If you run a **ticket seller system,** maybe that a lot of time before the event you would prefer to lose consistency over availability if you do not expect a total sold out.
- Always in the **ticket seller system**, as soon as the event is starting it would be preferable to lose availability over consistency, because selling a wrong ticket would be an awful problem.

- **Financial Transactions in a Local Bank Branch:** Consider a local bank's distributed system branch operating in an area with reliable and stable network connectivity. The bank prioritizes consistency in financial transactions and high availability of services for its customers. The branch has a strong focus on ensuring that all transactions are processed accurately and promptly.

Answer 2.0

The CAP theorem, also known as Brewer's theorem, is a fundamental concept in the field of distributed systems. It was formulated by computer scientist Eric Brewer in 2000. The CAP theorem states that it is impossible for a distributed system to simultaneously provide all three of the following guarantees:

1. **Consistency (C):** All nodes in the distributed system have the same data view at the same time, meaning that every read receives the most recent write. In a consistent system, all nodes agree on the current state of the data.
2. **Availability (A):** Every request to the distributed system receives a response without the guarantee that it contains the most recent version of the data. Availability ensures that every request gets a response, even if it might not reflect the most recent write.
3. **Partition Tolerance (P):** The system continues to operate and provide consistent or available services despite network partitions or communication failures between nodes. In other words, even if network connections between nodes are unreliable or temporarily unavailable, the system can still function.

The CAP theorem asserts that a distributed system can achieve at most two out of the three guarantees simultaneously. This means that when a network partition occurs (P), and nodes are isolated from each other, the system must make a trade-off between maintaining consistency (C) or availability (A).

Here are some scenarios based on the CAP theorem:

1. **CA Systems (Consistent and Available):** In scenarios where network partitions are rare or unlikely, a distributed system can prioritize both consistency and availability. However, in the face of network partitions, these systems may become unavailable or inconsistent.
2. **CP Systems (Consistent and Partition Tolerant):** Systems that prioritize consistency and partition tolerance will maintain a consistent data view even in the presence of network partitions. However, this might lead to unavailability during partitions as some nodes might not be reachable.
3. **AP Systems (Available and Partition Tolerant):** Systems that prioritize availability and partition tolerance may sacrifice consistency. In the event of a network partition, the system may continue to operate and provide responses, but these responses might not reflect the most recent state of the data.

It's important to note that the CAP theorem does not prescribe a one-size-fits-all solution for distributed systems. The choice between consistency, availability, and partition tolerance depends

on the specific requirements and goals of the application or system. Different distributed databases and systems make different trade-offs based on their use cases and priorities.

Question 4

Is Blockchain safe? Is it live?

In the Drive, I found this answer:

4.2.2 Consensus protocol

We have seen how miners choose what block is the next one in the chain. It's easy to see that this protocol is nowhere near to safety, because of the chance of a fork. At the same time, we can't be sure that this protocol is live, but it's very unlikely to have multiple forks that stops the protocol. So when we have a fork, one of the two branches must win, probabilistically speaking.
This is an example of a randomized consensus protocol, safe and live with high probability.
This protocol has several problems, the first it's the high power resources needed.

Question 5

In Bitcoin, why does the hash "change" every time it is calculated? How many hashes do we have to compute before finding a new block? What is the value for k? How many hashes in one minute (circa)? Do you know its throughput (accepted transactions per second)? How does it handle double spending? How do you transfer money in Bitcoin?

In Bitcoin, a hash code is used for every block in order to be able to identify it univocally. It is calculated using a hash function (that can be different, producing different hashes for the same inputs), and usually hashes have to be computed for an average of ten minutes to find a new block. Using the Proof of Work the hash of the block must be lower than a given target, this means that the k most significant bits of the hash have to be zero. For this reason in each block, a nonce field is present: the miners have to find the correct nonce for the block to get the hash lower than the target. One of the main problems in Bitcoin is double spending, that is when the same Bitcoin is used in two different transactions. Bitcoin handles it by using the blockchain. Through the blockchain, the entire path of a Bitcoin can be traced, so that it's easy to spot every malicious attempt.
An example of double spending is:
A has a bitcoin and B and C are users.

- A sells its bitcoin to B before committing the sell, in the blockchain B is registered as the new owner
- A wants to use the same bitcoin in a transaction with C, but C can check the blockchain to know that even if the commit hasn't happened yet, the same bitcoin has been virtually used

Bitcoin uses a probabilistic model for consensus, since Paxos couldn't handle the amount of operations required. Each owner transfers the coin to the next by digitally signing a hash of the previous transaction and the public key of the next owner and adding these to the end of the coin.

Question 6

Describe Proof of work (deprecated)

Proof of work is a security system. Usually slows down the system and consumes a lot of energy. For example in bitcoin it takes roughly 10 minutes to find the new special value (meaning to perform the proof of work) that allows the creation of a new block. This is a great mechanism to avoid block manomission since this amount of time to calculate a new block. All the network is in fact searching for the same new block, which makes it very difficult to manomit an entire chain of blocks. This is why fork happens.

Answer 2.0

Proof of Work is part of the system to reach consensus in the Bitcoin blockchain (consensus in Bitcoin is PoW + longest chain rule). Proof of Work is used to keep the Bitcoin network safe. If a miner wants to propose a block, it has to solve a computationally intense puzzle to be able to publish the block: the **hash** (SHA-256) of the block must be lower than a given target, for this reason in each block there is a special field called **nonce** (an unsigned integer); the target value is updated every 14 days, or precisely every 2016 blocks, this is to ensure that the mined blocks are published in a constant rate time, If miners are finding blocks too quickly, the difficulty increases, making it more challenging to find the next block. Conversely, if blocks are taking longer to mine, the difficulty decreases to make mining easier. The miner has to find the correct nonce to make the hash's block lower than the target. Doing so is computationally expensive, but once the solution is found, the other nodes can easily check if the nonce is correct and the block is valid. The difficulty of the work depends on the target and is automatically adjusted to make sure that every ten minutes a block is generated. When two miners find the correct Proof of Work simultaneously for two different blocks we can have a **fork**, in that case, the chain with the most computational effort is chosen (usually the longest one) and the other one is dropped.

Question 7

What is TOR and how does it work? Does it have problems with regard to bandwidth, latency or both? How are Tor nodes selected?

https://tor.stackexchange.com/a/674

TOR is an anonymity system that provides private browsing and therefore it doesn’t expose IP addresses. TOR is composed of n nodes called relays. An user that wants to browse with TOR, selects at least 3 relays: the Guard, the Relay and the Exitpoint. The user talks to the Guard that talks to the relay that talks to the exitpoint which communicates with the website. To guarantee anonymity, the outgoing packet is wrapped, like an onion, with encryption layers and the next destination relay; then every relay decrypts the packet, reads the destination and sends it to the next relay until the endpoint that sends it to destination. More in depth, TOR packets are encrypted using the Authenticated Diffie-Hellman algorithm: both the user and the guard agree on a common value (say g). Then the user chooses a value (say a) and sends g a encrypted with the public key of the guard to the guard. The guard chooses another value (say b) and sends back to the user g b

and the Hash of the concatenation of g a∗b with a string (for example ”Handshake”) to the user. By doing so, they shared a private key in a public space that nobody can guess. This is done between the user and every other relay using the other relays as messengers. The package is then encrypted with those keys, each relay can decrypt the package only to find out which will be the next node. In this way TOR guarantees anonymity but also increases problems to both bandwidth and latency since the package must be transmitted through at least 3 relays. Nodes are selected randomly.

What is the reward for those who run a TOR node?

Volunteers who run nodes are rewarded with publicity and possibly better anonymity.

More nodes means increased scalability and more users can mean more anonymity.

-took from paper section 10.

Do you know why some servers are hidden, not part of the public list?

Location-hidden services allow a user to offer a TCP service, such as a web server, without revealing his IP address. This type of anonymity protects against distributed DoS attacks: attackers are forced to attack the onion routing network because they do not know the user's IP address.

Answer 2.0

Servers that are hidden and not part of the public list are often associated with the dark web and onion routing for privacy and security reasons. The dark web is a part of the internet that is intentionally hidden and can only be accessed using specific software, such as the Tor browser, which utilizes onion routing.

Here are some reasons why servers on the dark web might remain hidden:

- **Anonymity and Privacy**: Dark web users, including both website operators and visitors, often prioritize anonymity and privacy. By hiding servers and not listing them publicly, operators can reduce the risk of identification and potential legal consequences associated with hosting or accessing certain types of content.
- **Censorship Resistance**: Some users operate websites on the dark web to share information or communicate in environments where there may be restrictions or censorship. By keeping server locations private, these operators can resist censorship attempts more effectively.
- **Security Against Attacks**: Servers on the dark web may host content or services that are controversial or illegal in certain jurisdictions. Keeping server locations hidden adds a layer of security, making it more challenging for adversaries to locate and attack the servers.
- **Protection Against Law Enforcement**: Law enforcement agencies often work to identify and shut down illegal activities on the internet. Hidden servers make it more difficult for authorities to locate and take action against operators engaging in illegal activities.

- **Selective Access**: Some websites on the dark web are intentionally exclusive and may require specific credentials or knowledge to access. Keeping the server hidden helps maintain control over who can access the content.

Onion routing, which is a technique used on the Tor network, plays a crucial role in achieving this level of privacy. In onion routing, data is encrypted in layers, and each layer is decrypted at a different node in the network, adding an extra layer of anonymity. This makes it difficult to trace the origin and destination of the data.

Suppose that it controls both starting and ending nodes, is the communication secure?

Into the Onion Routing network, despite the anonymity, some attacks are still possible to achieve. If the entry point is controlled It can see our information just as long as we have that relay as guard. This means that with both ends controlled the traffic can be analyzed and attackers could recognize the path of the communication.But, this opens to a problematic behavior. In fact controlling both ends guard and exitpoint is possible to perform some powerful passive and active attacks. **observing user traffic patterns:** this is not strictly connected to control both ends, but observing user's traffic patterns can reveal the service which is connected. **End to End timing correlation:** Tor minimally hides those correlations, an attacker may observe traffic patterns of the guard and the exitpoint, so it will be possible to find who are the peers and their correlations. In fact the guard node will know where the connection comes from, and the exitpoint will know where the connection is directed; so thanks to traffic patterns the privacy achieved with OR will be compromised. **End to End size correlation:** Storing the size of incoming and outgoing packets, is it possible to find peer correlations. However, a limited protection is constituted by the "leaky pipe" nature of the network topology. It is in fact possible that a packet that came in through a circuit and exit through another, thus making the attack ineffective.

To try to avoid this problem it is recommended to choose entry point and exit point from different autonomous systems.

How many servers are there in the world?

approximately 2000 servers (cercato su google :))

Question 8

Dark Web, how does it work? What happens if all the intermediate servers are malicious, are you safe? If you use TOR to visit CNN, is that the Dark Web? How do you find hidden websites?

The user connects to the so-called Directory Service to ask for a hidden service. The DS selects a relay called rendez-vous point that the user and the hidden service will use to communicate. Everything is performed using TOR and the relay doesn't know anything about the user or the hidden service. No, if we visit CNN, it's not on the dark web. If all the intermediate servers are malicious we are safe since they don't know anything about the user; but it is still possible to be cut off from the service we are looking for.

We can address hidden websites in the normal way, but with a .onion domain, only solvable by communicating with the directory service. Websites that use this domain form the dark web.

Question 9

Describe how the Bitcoin network works. Do you know how many miners there are? Why do servers do all that hard work?

The bitcoin protocol is based on proof of work. every block is composed by:

- Hash of the previous block
- Transactions
- Nonce
- Hash of the block

If a miner wants to insert the new transaction block into the blockchain, it must find the right nonce that results in having the least k bit of the Hash set to zero. This is not an easy task that requires at maximum 2^k tries.

Question 10

Show how to make Paxos live with a perfect failure detector.

1. In the first phase every process executes n - 1 rounds. In each round, every process p broadcasts their proposed value. Then each process waits for the proposed value of every process who is not in Dp (who is not crashed). While waiting, a process q can be suspected. In this case q is inserted in the list of suspects and p won't wait anymore for its message. When process p receives an estimate from another process q, it updates its list of estimates accordingly.
2. In the second phase, every process starts broadcasting the list of estimates and waits until it receives the list of estimates from every other process who is not suspected (same process as before). In the end, we will end up with an array where the proposed value of process i is in the i th position (if every other process have agreed on it, $\perp$ otherwise).
3. In the third phase, the process decides on the first non null value in the vector. An algorithm for Perfect Failure Detector can also be used for Strong Failure detector and also for θ and Weak failure detector. Because a Weak can simulate a Strong and a θ can simulate a Perfect.

Question 11

Propose an implementation of a failure detector and discuss its properties in a synchronous system.

There are various possible solutions for implementing a failure detector, all solutions may be transformed into stronger failure detectors. A simple and strong way to do it is to force all nodes to send an “im alive” message to all other nodes. When a node doesn't receive the “im alive” message in a time $\Delta T + \delta$ it suppose that node is dead. After time $\Delta T + \delta$ (the same as timeout) all processes exchange their suspect set and update their. The update must work as follow:

- ➔ if node i is suspected by other nodes but not me: ignore.
- ➔ if node is not suspected by other node but from me: remove node from suspected and increase δ.
- ➔ if all nodes suspect a node dead: add it to the suspected list.

Properties:

- **Strong completeness:** all processes that crash will be detected by all other correct processes: If a process A suspects another process B after time $\Delta T + \delta$ (because it didn't get the live message), after receiving the table of suspects from all other processes, A can assume that B is alive if another process C doesn't suspect it (because C received the live message from B before its timeout).
- **Eventually strong accuracy:** there is a time after which correct processes are not suspected by any other correct process. In the update step when a process receives the other table, if there is a conflict for a process, its entry is updated with the “alive” value. So even the process that suspected the death of a process that’s alive won’t suspect it anymore.

Question 12

Describe briefly the bit-torrent system.

https://blog.libtorrent.org/2020/09/bittorrent-v2/

Generally, a Peer to Peer system is a global scale system where there isn’t a central authority. Examples are Bit-torrent and the old Emule. It’s a very dynamic system where there is a very high number of connections and disconnections from the users. Everything starts from a server that shares a file in the system. This file is then stored somewhere between the peers. Once that is done, we can remove the server. The peer-to-peer system is composed of different types of nodes:

1. **Tracker:** has the list of the seeds who are sharing the file.
2. **Seeder:** have the entire file and are only distributing it.
3. **Leecher:** downloads the file and becomes a seed when the download is over.
4. **Free-riders:** downloads the file without uploading after.

A file (in the Bit-Torrent case) is composed of:

1. Name of the file
2. Size
3. Tracker info
4. Size of each part (usually 256 kb)

5. Hashes of the parts (used to avoid poisoning)

When we want to download a file we should ask the tracker for the list of seeds that contain part of the file. There are 3 main ways to download a file from the seeds: in order, starting from the rarest (overloading a node) or randomly. The best solution is to combine random and rarest: first, choose randomly so that all parts are well distributed, then choose a rare one. Usually, the download at 99% becomes very slow and that's because BitTorrent uses the end game mode. It's an approach that splits the last block of data into multiple sub-parts and requests them to all the available peers. As soon as a piece is obtained all the requests get canceled. Another problem of P2P systems are free-riders: the choking algorithm protocol was invented to stop them. Every peer can unchoke at max 4 other peers, with a higher probability the ones it downloads from or uploads to. When a new peer enters the system, it doesn't have anything to give to the others so the probability that someone unchokes it is very small. To solve this there are multiple approaches (random, depending on the bandwidth or reliability etc), for instance there's one called optimistic unchoking which means that nodes can unchoke every type of node (also free-riders).

Question 13.0

Describe shared memory system.

Shared memory is an abstraction of a distributed system giving the impression of a monolithic memory. Programmers only use read and write primitives to access the network and do not have to deal with send and receive. To provide programmers with the illusion of a single shared address space, a memory mapping management layer is required to manage the shared virtual memory space.

Advantages:

- simplifies task of programmers
- simplifies passing-by–reference
- exploits locality to reduce communication overhead
- fairly cheap
- no bottleneck due to single memory access
- large virtual main memory
- portability

Disadvantages:

- programmers are not shielded from having to know about various replica consistency models
- overhead as high as message passing implementation (programmers lose the ability to program their own solution)

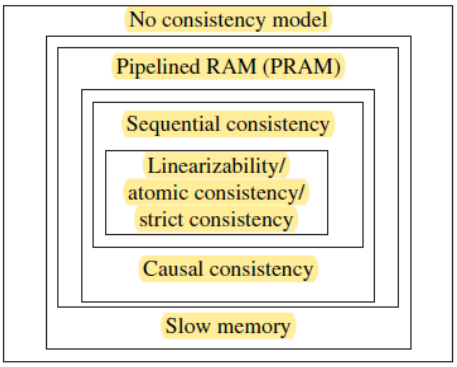
There are four broad dimensions along which DSM systems can be classified and implemented:

- data replicated or cached
- remote access via hardware or software
- caching/replica via hardware or software
- DSM controlled by distributed memory system, operating system , language runtime system

Memory coherence is the ability of the system to execute memory operations correctly. There are various consistency models, the following figure represents their hierarchy.

The order is (those above include those below??):

- No consistency model
- Slow memory
- Pipelined RAM (PRAM)
- Causal consistency
- Sequential consistency
- Linearizability / atomic consistency / strict consistency



Question 13.1

Describe the difference between Sequential consistency and Atomic consistency.

Atomic consistency:

- **<u>Any Read to a location (variable) is required to return the value written by the most recent Write to that location</u>**
- For operations that overlap as per the global time reference, the following further specifications are necessary
    - All operations appear to be executed atomically and sequentially.
    - All processors see the same ordering of events, which is equivalent

An execution sequence in global time is viewed as a sequence Seq <inv, resp> that must satisfy:

1. **liveness:** every invocation must have a response
2. **correctness:** the projection of Seq on any processor i, denoted Seq_i, must be a sequence of alternating invocations and responses if pipelining is disallowed.

A sequence is linearizable if there is a permutation Seq' of adjacent pairs of corresponding events satisfying:

1. For every variable v, the projection of Seq' on v, denoted Seq v' is such that every read returns the most recent write that preceded it. **Every processor sees a common order of events**

2. If the response op1 of operation op1 occurred before the invocation op2 of the operation op2 in Seq, then op1 occurs before op2 in Seq'. **Common order Seq' must satisfy the global time order of events.**

Sequential consistency uses logical time reference instead of global time reference.

- The result of any execution is the same as if all operations of the processors were executed in *some* sequential order
- The operations of each individual processor appear in this sequence in the local program order.

More formally, a sequence of invocation and response events is sequentially consistent if there is a permutation Seq' of adjacent pairs of corresponding events such that:

1. Same condition of linearizability
2. **Common order Seq' must satisfy only the local order of events at each processor, instead of the global order of non-overlapping events.**

Answer 2.0 (estratta dai miei appunti, tom)

Atomic/linearizable: we have atomic consistency if
1. any read to a location is required to return the value written by the most recent write as per global time reference.
2. all operations appear to be executed atomically.
3. all processors see the same sequence of read/write operations.

read and write operations are sequences of <inv, resp>. Sequence Seq of invocations and responses.

Seq is linearizable if ∃ Seq^i of adjacent pair of <inv, resp> such that:
1. ∀ variable v, the projection of Seq^i on v, $Seq^i\ v$, is such that every read returns the most recent write.
2. if the resp of OP1 occurred before the inv of OP2, then OP1 appears before OP2 in Seq^i.

Sequential consistency: it's weaker than atomic. The result of the computation is as if all operations were executed in some order. The internal order in each process must be preserved.

If atomic ⇒ sequentially consistent but is not true the opposite atomic ⇏ sequentially.
Every protocol that does not implement total order broadcast implements sequential consistency.

Question 14

Talk about privacy on the internet: analysis of Wifi Beacons in crowds, VPN IPv6 leak, user IP identification in cellular networks, browser fingerprint, privacy on social networks.

Crowd analysis

According to the 802.11 standard, a WIFI access point can announce its presence by broadcasting *Beacon Frames* containing network configuration parameters such as SSID (a string that identifies a WIFI AP in a human-readable form 'home network'). On the client side, it can scan for WIFI AP in two ways: **Active** send a probe request to discover networks, **Passive** listen for beacons and decide which to connect to.
To further improve this mechanism a PNL (preferred network list) is kept by the OS.
It is possible to collect those beacons to get some information about the device owner.
A possible usage of the collected information is to create a graph-structured "social network", where nodes are smartphones and edges are common Access Points. The key idea is that people connected in the real world are also connected in this network.
Another possible usage of those collected packets is to connect the smartphone owner to the specific location where the owner lives: this is done by searching the home network SSID in the Wigle Database. Since home networks usually have an SSID like "TIM-xxxxxx" the SSID (structured like that) is unique and points to a unique location.

VPN IPv6 leak

In operative systems, the IPv4 stack and the IPv6 stacks are separated and only the needed one is used. By testing 14 commercial VPNs the researchers discovered that most of them (10/14) were vulnerable to IPv6 leak: when the user is connected to a VPN and sends data through the IPv6 stack, the data are instead sent in clear and not through the VPN, the result is that the user of the VPN is not protected.
Almost all the VPNs tested (13/14) were also found to be vulnerable to DNS Hijacking: when the user is connected to a VPN and makes some DNS queries, they are intercepted and manipulated. This is a big security issue because an attacker can redirect the user to some malicious websites to steal credentials or to make some scams. This means that the user types, for example, "paypal.com" and the DNS server redirects it to a malicious clone of it. The browser shows a warning to the user (the red lock icon), but a lot of users ignore it.

User IP identification in cellular networks

By looking at the Round-Trip-Time it is possible to know if a target smartphone is sleeping, since it'd have a longer RTT than active smartphones. To get the IP address of the target the attacker can send a message to wake up the device using one of the popular messaging apps widely used, and then ping all the smartphones in the network (this means that the attacker will send millions of pings): the smartphone with the lowest RTT is the one the attacker is looking for. By performing this attack several times (within 20 messages) is it possible in 80% of the cases to correctly identify the smartphone's address, which can then be localized.

Question 15

Talk about your favorite seminar.

Seminar 1 (BlockChain and Liquidity Pools)

A **transaction** is composed by: from address, to address, amount + fee, that all compose the transaction hash. Nodes can be proposers, validators or simple (the forward transactions). Full nodes store all the blockchain while not full nodes store only chunks of transactions. Transactions pass through a memory pool before getting accepted and only then transferred into blocks. Transactions can be used to exchange coins or tokens. A **coin** is for example ethereum, the **token** is an asset. A **smart contract** is instead a program that runs on the blockchain and it manages transactions. Smart contracts have been first introduced by ethereum and aren't present in bitcoin for example. An example of smart contract could be ERC-20 (smart contract token) and is composed by:

- Total supply
- Name
- Symbol
- Decimal (function that returns an integer)

Decimal and total supply are indications on the quantity of token and the amount that can be sent in the transaction.

On some blockchains, a **data field** is present and it stores the parameters of the Tx, for example TRANSACTION(ADD1,ADD2,AMOUNT) takes arguments from this field. The transfer event is generated by the smart contract.

Decentralized services have as a main goal to allow anyone to create a market for a token. DEX (dec. exchanges) employ smart contracts. The **liquidity pool** is a type id smart contract. It's composed of two tokens. Traders can interact with them and use them to balance the value of a coin. The more you buy one of the two types of token, the more its value increases to match the collective value of the other type of token. Ex:

Token A = 0.2 Token B = 0.1 and in the liquidity pool there are 3 Token A and 6 Token B

Someone buys 2 Token B and so they have to put in 1 Token A (to match the value). At this point though the value is different cause 4 Token B have the same value of 4 Token A. The **liquidity pool provider** is the one who puts in the liquidity for both of the tokens. They get in exchange LP-Token, a special token that signals that they have invested in the pool.

A particular case of liquidity pool is the Squid Coin gate. Through a liquidity pool the creator made the value of their coin go up and up by making everybody buy it, but they didn't include the sell option into the smart contract of the coin. He got the money and ran basically, leaving everybody with useless coins.

Seminar 2 (Pump & Dumps)

Pump and dump schemes in the Bitcoin era aim to generate cryptocurrency, engage in cryptocurrency trading, and manipulate the crypto market. Before delving into the details, it's beneficial to define what a cryptocurrency coin and token are. A coin serves as the native cryptocurrency of a blockchain, while a token exists on an established blockchain and is managed by smart contracts. How do cryptocurrencies trade to manipulate the market? Two methods are employed: utilizing a centralized exchange or a decentralized one that ensures anonymity. Two types of orders come into play—limit orders (specifying the desire to buy/sell a defined number of tokens at a set price, recorded in an order book) and market orders (expressing the intent to buy/sell a certain number of tokens regardless of the current price). The process involves a group of individuals buying tokens and placing orders in the order book at a predetermined price. Subsequently, fake advertisements inundate the internet to attract external investors. As these investors join in, the price surges, and those who sell do so at the peak profit. The sellers then prompt a rapid price decline, resulting in only a select few making money while others incur losses. The primary beneficiaries are the organizers and the higher-ranked users within these groups. These groups operate similarly to a Ponzi scheme, where users receive information about upcoming pumps based on a subscription fee. Higher fees grant more information to subscribers, positioning them as privileged participants in the scheme.

Seminar 3 (Telegram exploration - TG dataset)

Telegram's channels are virtual rooms where only an administrator can write and broadcast messages to the subscribers. A fake channel, to deceive the users, usually has the exact name with slight variations; it attempts to qualify itself as the official miming the original behavior.

The study has the goal to detect real and fake TG channels, to do so were built 2 datasets. The first one **TGDataset** includes 120'000+ channels. This dataset takes a snapshot of the actual telegram ecosystem instead of focusing on one single topic.

Dataset creation: the dataset was built using a python's wrapper for telegram's official API, they take the last 3000 messages per channel and their lifetime (the time from the channel creation to his last message); then channels were divided in 'verified' and 'not verified' using official statistics, so from the verified ones, the fake ones that was claiming to be officials were extracted.

A further analysis of the dataset has highlighted some patterns: Verified channels generally have more subscribers than fake ones, fake channels usually 10% of subscribers of the original one, the original channels send more messages and post more media than fake ones, fake channels are more prone to forward messages from others channels.

The chones feature were: writing style (messages length, emojis, ecc…), temporal features (number of messages sent in n month) and external interaction (forwarded messages, external links, ecc…).

In the end the best model found was using an MLP (Multilayer perceptron, not my little pony) classifier with an F1 score of ≈85%.

Seminar 4 (BGP)

DISCLAIMER: AI SUMMARY

The document "The parallel lives of Autonomous Systems: ASN Allocations vs. BGP" provides a comprehensive analysis of Autonomous Systems (AS) and their allocation and operation in the Border Gateway Protocol (BGP). It delves into the administrative and operational lives of AS, the policies and practices of Regional Internet Registries (RIRs), and the process of data collection and preparation.

The administrative life of an AS begins when a registry allocates a specific AS number to an organization, removing that number from the available pool. The end of the administrative life occurs when an AS number is either returned by the holder organization or reclaimed by the respective RIR, in accordance with RIR internal resources allocation policies.

The document discusses the different approaches of RIRs in handling ASN allocations, including eligibility criteria, recovery of unused resources, reuse of resources, and special cases. For example, ARIN has been reclaiming number resources from non-compliant organizations since 2010, while other RIRs have different strategies for reclaiming or reusing resources.

It also details the process of data collection and preparation, which involves restoring 17 years of ASN delegations and BGP data. The authors collected all delegation files from the RIRs' FTP sites and processed historical BGP data from all available RIPE RIS and RouteViews collectors, spanning from October 9, 2003, to March 1, 2021.

In summary, the document provides an in-depth analysis of:

- The administrative and operational lives of Autonomous Systems.

- The policies and practices of Regional Internet Registries (RIRs).
- The process of data collection and preparation, including restoring 17 years of ASN delegations and BGP data.

For a more detailed understanding, the document includes insights into global and per-RIR trends, such as the significant gap between administratively allocated ASNs and those that are operationally alive. It also discusses the challenges and methodologies involved in inferring administrative and operational lifetimes of ASNs, as well as the impact of different BGP activity timeout values on the analysis.

The document "The parallel lives of Autonomous Systems: ASN Allocations vs. BGP" addresses several key issues related to Autonomous Systems (AS) and their allocation and operation in the Border Gateway Protocol (BGP). Some of the main issues described in the document include:

RIR Practices and Delegation Files: The document highlights the differences in Regional Internet Registries' (RIRs) practices in updating and handling delegation files. It discusses the variations in the timing of when an allocated AS number appears in the delegation files, as well as challenges faced by RIRs in keeping the files up to date and dealing with corner cases of resource allocations, which can lead to resources disappearing from the files for a few days.

Data Collection and Preparation: The document provides insights into the process of restoring 17 years of ASN delegations and BGP data. It discusses the challenges and methodologies involved in collecting, restoring, and sanitizing the delegated files and BGP data used in the study. This includes filling gaps in missing files, restoring missing records, cleaning invalid duplicate records, and addressing inter-RIR inconsistencies.

Administrative and Operational Lifetimes of ASNs: The document delves into the administrative and operational lives of ASNs, including the start and end of the administrative life when an AS number is allocated or reclaimed by the respective RIR. It also discusses the challenges in inferring administrative and operational lifetimes of ASNs, as well as the impact of different BGP activity timeout values on the analysis.

RIR-Specific ASN Allocation Policies: The document outlines the differences in RIR-specific ASN allocation policies and reporting practices, such as the eligibility criteria, recovery of unused resources, reuse of resources, and special cases. It provides detailed insights into how RIRs handle ASN allocations and the impact of these policies on ASNs' administrative lives.

In summary, the document addresses the complexities and challenges associated with ASNs' administrative and operational lives, RIR practices in handling delegation files, data collection and preparation, and RIR-specific ASN allocation policies. It provides a detailed analysis of these issues, shedding light on the intricacies of ASNs' allocation and operation in the context of BGP.

Miscellaneous (to do not 2023/2024)

Algorand (A quanto pare inutile, ma lo lascio qui)

Describe Algorand? What does the committee do?

Algorand is a blockchain cryptocurrency protocol whose main goal is to be scalable and fast. Its consensus algorithm is based on proof of stake and Byzantine agreement protocol. His native cryptocurrency is called ALGO. Algorand uses a decentralized Byzantine agreement protocol: as long as the majority of the stake is in non-malicious hands, the protocol tolerates malicious users and achieves consensus. Consensus requires 3 steps: propose, soft vote and certify vote.

During the first phase, a committee of users is randomly selected in a weighted way (based on their stakes) to propose a new block. Each user uses a Verifiable random function (VRF) to determine if they are on the committee or not. If the VRF indicates that one user is chosen, it returns a cryptographic proof that can be used to verify if the user is in the committee. The chances that a user is in the committee are directly proportional to the number of ALGO he has (the stake). The selected users build a block and broadcast it through the network for review in the second phase. In the second phase, a Byzantine Agreement protocol is used to vote on the proposed blocks. A new committee is formed again and they have to vote on the blocks. If the committee achieves consensus on a new block, then the new block is broadcasted through the network. In the last phase, a new committee checks the block that was voted and if valid, votes again to certify the block. These votes are collected and certified by each node until a quorum is reached, triggering the end of the round. Then the node will create a certificate for the block and write it into the ledger.

Do fork happen in Algorand?

Proof of stake Proof of stake is built on Byzantine consensus where each user's influence on the choice of a new block is proportional to its stake (the number of its coins). In this way owners of a small fraction of coins can’t harm the entire system while big owners won’t misbehave otherwise the currency’s purchasing power will diminish, affecting their assets. Proof of stake compared to proof of work is faster, requires standard equipment and provides fast transactions. Also, a fork can never happen in Proof of stake since only one block can be 2/3 of the votes. All blocks are final.

Special thanks

This document would never have been possible without the priceless contributions of:

- ★ Tommaso
- ★ Lorenzo
- ★ Elena
- ★ Nunzia
- ★ Luca
- ★ Michele

✨Offer us a coffee☕