RELIABILITY OF DISTRIBUTED SYSTEMS

Mike Greenbaum and David Ponarovsky

November 2021

Exercises

1. Show a protocol solving State Machine Replication for any $f < n$ Mobile crash failures that uses a stable leader. Prove safety and liveness.

   (a) For safety, prove that your protocol behaves the same as an ideal State Machine that never fails.

   (b) i. State and prove a worst case bound for the time it may take a non-faulty client to receive a response for its request. This should be a function of Δ and f . Consider two cases separately: $f < 6$ and $f > 15$.

   ii. During a duration in which the leader is non-faulty, state and prove a worst case bound for the time it may take a non-faulty client to receive a response for its request is just a function of Δ.

   iii. Bonus: For the case of $f > 15$ state a prove a bounded on the expected time it may take a non-faulty client to receive a response for its request. This should be just a function of Δ.

2. Show a protocol solving State Machine Replication for any $f < \frac{n}{2}$ Mobile omission failures that uses a stable leader. Prove safety and liveness.

   (a) For safety, prove that your protocol behaves the same as an ideal State Machine that never fails.

   (b) i. State and prove a worst case bound for the time it may take a non-faulty client to receive a response for its request. This should be a function of Δ and f . Consider two cases separately: $f < 6$ and $f > 15$.

   ii. During a duration in which the leader is non-faulty, state and prove a worst case bound for the time it may take a non-faulty client to receive a response for its request is just a function of Δ.

   iii. Bonus: For the case of $f > 15$ state a prove a bounded on the expected time it may take a non-faulty client to receive a response for its request. This should be just a function of Δ.

Note We understood the exercise in different ways due to the exercise not being well defined and therefore we append 2 different solutions to the questions.

Lemma 1 We will first show that there exists a safe and live protocol for the Crush case where $f < n$. We will prove this via induction.

1. Base case: $n = 2$, in class we showed a live and safe algorithm for the case where there are 2 servers and one of them can crush.

2. Induction statement: We will show that there exists a safe and live protocol for $f < n + 1$. We know that there exists a protocol that is live and safe for $f - 1 < n$ via the induction assumption. We will denote this protocol with the letter A. We know that there exists a live

and safe protocol for 2 servers where one can crush. We will denote this protocol with the letter B. We define the following protocol: We run protocol B where one server is the leader and n are the backups. If we detect that the primary server crushed, we are left with n servers and at most $f-1$ faulty servers and we run protocol A on them.

Notice that if the primary server is not faulty, the protocol is safe and live. Else, if the primary crushes, due to the correctness of protocol B, one server will detect that the primary fell and we will switch to protocol A live and safe algorithm via the induction hypothesis because $f-1 < n+1-1 = n$ (f-1 because the primary is one of the faulty).

Lemma 2 We will show that there exists a simple change in the protocol which still makes it safe even for the mobile case.

The simple change to support safety in the mobile setting is to store all previous actions in a log. When someone restarts, they request such a log from all the others. After they get the log, they execute all the instructions there and then continue the previous protocol. The reason the protocol is still safe is that when the server restarts, due to the safety of the original protocol up to this time, there exists a non faulty server with all the history. Therefore, the rebooted server can get all the history from the non faulty one and continue the protocol as if it was never faulty.

If the rebooted server hears messages from the current time before getting the log, it stores them and executes them after the log.

The reason this works is because it takes at most 2Δ to restore a server after a reboot and takes longer to corrupt a new server.

Lemma 3 We will show that in the mobile cases for $(f-1)\cdot t > 12$ there is no liveness if we follow a leader protocol, where t is the amount of time it takes to switch a leader.

Define the series of leaders of the protocol $l_1, l_2 \ldots,$. At first, the leader will corrupt $l_1, \ldots, l_f$. We will show by induction that there exists an adversary that can make sure each server crushes when in it primary: After t time the protocol switches to l_2

1. Base case: $i = 1$, The first server crushes immediately, After t time the protocol switches to l_2, the adversary uncorrupt l_1 and after 12Δ corrupts l_{f+1}. Then the corrupted ones are $l_2, \ldots, l_{f+1}$. (notice that when we reach l_{f+1} in the induction it is already corrupted due to $(f-1)\cdot t > 12$.
2. Induction statement: We know that all $l_1, \ldots, l_{i-1}$ crushed, next leader is l_i and the corrupted ones are $l_i, \ldots, l_{i+f-1}$. $i = 1$, then we crush l_i, After t time the protocol switches to l_{i+1}, the adversary uncorrupt l_i and after 12Δ corrupts l_{i+f}. Then the corrupted ones are $l_{i+1}, \ldots, l_{i+f}$. (notice that when we reach l_{i+f} in the induction it is already corrupted due to $(f-1)\cdot t > 12$.

Therefore we showed an infinite run for every leader based protocol for the mobile case, therefore a leader based protocol can't be live.

We could have iterated the proof in class and shown that any protocol has infinite run but the idea is rather similar and it was required to prove for our protocol

Solutions 1

1. We will use the helper lemmas to shorten the exercise:
   (a) We showed a safe and live algorithm for the Crush case for $f < n$ in lemma 1. Now via lemma 2, we get a protocol which is safe under the Mobile-Crush case.
      i. We will denote in $c\Delta$ the amount of time it takes a good leader to reply, notice that c is constant. In the case that $f > 15$, then by Lemma3, the protocol described above isn't live. In the case that $f < 6$, notice that the worst case is that the adversary corrupt the first f leaders, in this case it will take $2f\Delta$ to find a good leader and at most $c\Delta$ more to return the result, So the total time will be $(c+2f)\,\Delta$. The reason that it works is that in this case, the adversary doesn't have the time to uncorrupt and corrupt new leaders.

ii. We will denote in $c\Delta$ the amount of time it takes a good leader to reply, notice that c is constant. Notice that the c is a the same constant in the protocol we saw in previous exercise and therefore it will take at most 3Δ to receive a reply if we choose a non-faulty leader.

iii. We will denote in $c\Delta$ the amount of time it takes a good leader to reply, notice that c is constant.
We can use randomness to make the protocol live with high probability. Before switching the leader, each server will choose a random $(rank, leader)$ $[n]^2$ tuple and will send it to all others. The leader will be the one that the one with the highest rank chose.
Note that with probability $1 - \frac{f}{n}$ the server with the highest rank is non faulty and with the same probability the chosen leader non faulty.
Therefore, with probability of at least $\left(1 - \frac{f}{n}\right)^2 \geq \left(\frac{1}{n}\right)^2 = \frac{1}{n^2}$ we will get a non faulty leader. Notice that the extra step takes at most another 2Δ to switch the leader.
Therefore, In expectation it will take us n leaders in order to find a non-faulty one and therefore the expected time to get a response is $(c + 4 \cdot n^2)\Delta$. Note that the protocol can take at most 2Δ to finish if the leader is non-faulty and therefore the adversary can't do anything when we get a non faulty leader.

2. We will use the helper lemmas to shorten the exercise:

(a) We saw in class a safe and live algorithm for the Omission case for $f < \frac{n}{2}$ that only requires the leader for the first 2Δ (the Bayesian protocol that we saw will Gilad Stern). Now via lemma 2, we get a protocol which is safe under the Mobile-Omission case.

(b) i. We will denote in $c\Delta$ the amount of time it takes a good leader to reply, notice that c is constant. In the case that $f > 15$, then by Lemma3, the protocol described above isn't live. In the case that $f < 6$, notice that the worst case is that the adversary corrupt the first f leaders, in this case it will take $2f\Delta$ to find a good leader and at most $c\Delta$ more to return the result, So the total time will be $(c + 2f)\,\Delta$. The reason that it works is that in this case, the adversary doesn't have the time to uncorrupt and corrupt new leaders.

ii. We will denote in $c\Delta$ the amount of time it takes a good leader to reply, notice that c is constant. Notice that the c is a the same constant in the protocol we saw in class and therefore it will take at most 7Δ to receive a reply if we choose a non-faulty leader.

iii. We will denote in $c\Delta$ the amount of time it takes a good leader to reply, notice that c is constant.
We can use randomness to make the protocol live with high probability. Before switching the leader, each server will choose a random $(rank, leader)$ $[n]^2$ tuple and will send it to all others. The leader will be the one that the one with the highest rank chose.
Note that with probability $1 - \frac{f}{n}$ the server with the highest rank is non faulty and with the same probability the chosen leader non faulty.
Therefore, with probability of at least $\left(1 - \frac{f}{n}\right)^2 \geq \left(\frac{1}{2}\right)^2 = \frac{1}{4}$ we will get a non faulty leader. Notice that the extra step takes at most another 2Δ to switch the leader.
Therefore, In expectation it will take us 4 leaders in order to find a non-faulty one and therefore the expected time to get a response is $(c + 2 \cdot (4 - 1) + 2 \cdot (4 - 1))\Delta$.
Notice that after the leader is chosen, in at most 2Δ the protocol is non dependent on the leader and therefore the mobile can't do anything to break it.

Solutions 2

1 Mobile Crash model.

The general concept is that each party suspects that is himself might wakes up after "restoration". So first he will wait till he get a least $f+1$ messages in which the parties share their state, (the **type** and **Q** are the configuration of the state machine). If we can prove that in the end of each iteration all the non faulty parties (including those which had just restarted) share a correct configuration then we done.

1. Safety. **Proof**: Note that if the enemy doesn't perform any uncommit operation then the protocol is just the generalization of the backup-primary protocol (with n servers). As the proof of the general case (In the non mobile model) is almost identical to safety proof of 2-parties case, we will skip straight to the event where the enemy has choose to uncommit someone, denote him by v. Now v check the configurations which were sent to him.

   Since we have assumed that $f < n$ and that the running of the protocol till the first uncommit operation is identical the running in the non mobile model. Then there is must to be at **least one** non faulty party which send a correct configuration to v. Then, v will update his state, and will function as any other non faulty backup server for the current round.

   Hence, any scenario in which v advances the state machine through wrong transition leads to contradiction to the safety property of the non mobile protocol.

2. (a) Our protocol holds in the case that $f < 5$. Each round require $2 \cdot \Delta$ waiting time. Where the first Δ time slot is due to the requirement to get the configuration and the second one is the time in which the parties are waiting to message from the leader/primary. (Note, that the delay in line (7) is overlaps with the first waiting). Let ξ be the time which needed to accomplish one round, and w be the time that the enemy should wait till is budget increase. Then we could absorb

   $$\overbrace{\omega}^{\text{the charging time}} < \overbrace{f \cdot \xi}^{\text{the time which last till the enemy lost his budget}} + \overbrace{\xi}^{\text{single round}}$$

   In our case, for $f < 5, w = 10\Delta, \xi = 2\Delta$ we get the desired. We sure that by squeezing the parameters we could set a similar protocol which will hold for $f = 5$. In case that $f > 15$, The enemy can promise that immediately after he has crushed a leader i, he will uncommit him in the next round. Hence when the client will try to set $i + f + 1 \geq i + 16 > i + w$ to be a leader, it's guaranteed that the enemy will have a positive budget. (and will able to execute another crash). That it, the time might be infinite ∞.

   (b) In case that the leader is non faulty he will send a response in a single round which cost 2Δ.

   (c) In the randomized version the client will choose the leader randomly. As the enemy can't predict the leader, it's bast strategy is to guess. For setting an upper bound over the expectation time, we can assume that w equals to ξ (meaning that in every round, the enemy can choose to re-disturber it's budget f). Then we get that the probability the enemy will success is lower then $\frac{f}{n} \leq 1 - \frac{1}{n}$. Hence the number of rounds is geometrically distributed with parameter $\frac{1}{n} \Rightarrow \mathbb{E}[\text{time}] = 2\Delta n$.

Algorithm 1: i'th processor

```
1  send all empty configuration
2  while True do
3  |  get configurations ⟨type, Q⟩
4  |  if there is nonempty message then
5  |  |  set type, Q ← ⟨type, Q⟩
6  |  end
7  |  if get ⟨pong, input⟩ then
8  |  |  set input ← ⟨pong, input⟩
9  |  |  type, Q ← makestep(type, Q, input)
10 |  |  send all ⟨type, Q⟩
11 |  |  send ⟨ping, type, Q, i⟩ to client.
12 |  end
13 |
14 |  else
15 |  |  wait Δ
16 |  |  if get ⟨type, Q⟩ then
17 |  |  |  set type, Q ← ⟨input⟩
18 |  |  end
19 |  |  else
20 |  |  |  send ⟨ping, type, Q, i⟩ to client
21 |  |  end
22 |  end
23 |
24 |  send all ⟨type, Q⟩ (including to myself)
25 end
```

Algorithm 2: client

```
1  set leader ← 0
2  send ⟨pong, input⟩ to the leader
3  while True do
4  |  wait Δ
5  |  if get ⟨ping, type, Q, id⟩ then
6  |  |  if not done then
7  |  |  |  set input ← next request.
8  |  |  |  set leader ← id
9  |  |  end
10 |  |  else
11 |  |  |  return
12 |  |  end
13 |  end
14 |  else
15 |  |  set leader =_n leader + 1
16 |  end
17 |  send ⟨pong, input⟩ to the leader
18 end
```

Algorithm 3: random protocol, client

```
1  set leader ← 0
2  send ⟨pong, input⟩ to the leader
3  while True do
4  |  wait Δ
5  |  if get ⟨ping, type, Q, id⟩ then
6  |  |  if not done then
7  |  |  |  set input ← next request.
8  |  |  |  set leader ← id
9  |  |  end
10 |  |  else
11 |  |  |  return
12 |  |  end
13 |  end
14 |  else
15 |  |  draw p ←∼_u [n]
16 |  |  set leader =_n leader + p
17 |  end
18 |  send ⟨pong, input⟩ to the leader
19 end
```

2 Omission Mobile Model.

The idea here is to execute an almost identical protocol to the crush mobile. When the only change is that every communication between the parties is done by using broadcast. The proof will be almost identical to the above case. The main change is that now in each round the parties exchange quadratic numnber of messages via broadcast.

1. The proof of the safety will be the same as in the mobile crush model, here the correction is lay over the correction of the broadcast protocol under omission attack with less then $\frac{1}{2}n$ budget. we will look again on the first party that back from the death. From the safety property of the broadcast protocol all the non empty message that he gets are correct configurations. Then the current round is identical to normal non-mobile protocol against omission attack.

2. (a) Unfortunately the analysis above yields that our protocol is robust only for $f = 1$. We believe that other protocols which can guaranteed robustness against $f = 6$ are exists, but their safety proof will be mach harder. Let's focus on the probabilistic model.

   (b) as explained above, single round takes at least 4Δ.

   (c) Even that in the crush model, we could provide a livens against much greater budgets (5 vs 1), we still get resoult that are the same for the crush model (where the enemy has $f \leq \frac{n}{2}$). Now, the probability of the enemy to guess right the leader will be $\frac{f}{n} \leq \frac{1}{2}$ and therefore $\mathbb{E}[\text{time}] = 8\Delta$.

Algorithm 4: *i*'th processor

```
1  send all empty configuration
2  while True do
3  |  get n − f messages ⟨type, Q⟩
4  |  if there is nonempty message then
5  |  |  set type, Q ← ⟨type, Q⟩
6  |  end
7  |  if get ⟨pong, input⟩ then
8  |  |  set input ← ⟨pong, input⟩
9  |  |  type, Q ← makestep(type, Q, input)
10 |  |  send all ⟨type, Q⟩ via Broadcast
11 |  |  send ⟨ping, type, Q, i⟩ to client.
12 |  end
13 |
14 |  else
15 |  |  listen (take part) to the Broadcast (2Δ required).
16 |  |  if get ⟨type, Q⟩ via Broadcast then
17 |  |  |  set type, Q ← ⟨input⟩
18 |  |  end
19 |  |  else
20 |  |  |  send ⟨ping, type, Q, i⟩ to client
21 |  |  end
22 |  end
23 |
24 |  send all ⟨type, Q⟩ via Broadcast (including to myself)
25 end
```