Distributed Systems I
Winterterm 2020/2021
Exercise 1
Team 3

Tim Schubert (3229613)
Denis Moslavac (3227767)
Haoqing Wu (3493407)
Johannes Haberstock (3065727)

Total: 29/36

Exercise 1: Transparency Levels

8/12

(a)

- Replication transparency: There cannot be any replication because there is just one host.
  We do not know how often the file is saved. Only one copy of the object is returned.

2/3

- Concurrency transparency: 1
  We access the file in the same way if only one user is loading the file from the web. The web-service can treat every request in a single thread.
- Scaling transparency: 1
  The web-service can hold more than one mp3-file and expand the number of the stored mp3s without changing the access to the files (via browser and filename)

(b)

- Replication transparency: I think you mean "stored" not "saved". In my understanding "saving" is the actual process of writing the file to a storage medium.
  We do not know how often the file is saved. Only one copy of the object is returned. The web-service can be deployed on more than one server across the world.

0.5

- Concurrency transparency:
  We access the file in the same way if only one user is loading the file from the web. The web-service can treat every request in a single thread.

1

- Migration transparency:
  Since we do not access the server via its ip-address, but via its name, the web-service can migrate to a different server with a different ip-address. If the domain-name system provides our browser with the new ip-address, we do not recognize a migration of the web-service.

1

4.5/5

- Access transparency:
  We don't know to which server the request is made. The domain-name system returns our browser the ip-address of the actual server. This server can be anywhere, even on our own machine. Be carefull not to mix access transparency with location transparency. For access transparency it is only important that local and remote accesses cannot be distinguished.

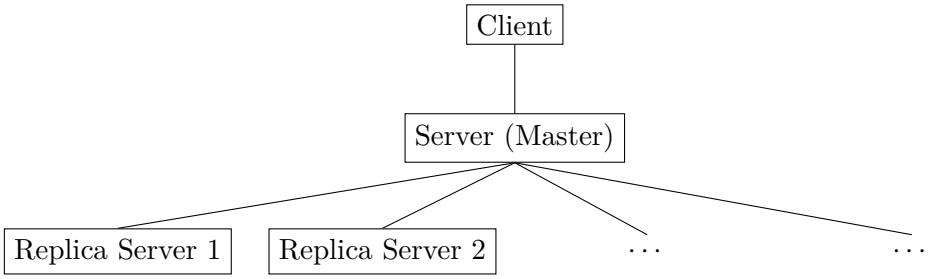
1

- Scaling transparency:
  We can access more than one mp3-file because we are querying a title and not accessing a song directly. The database of the songs may expand, but as a user the structure of requesting one via the browser does not change.

1

Tim Schubert (3229613)
Denis Moslavac (3227767)
Haoqing Wu (3493407)
Johannes Haberstock (3065727)

(c) i. We use a hierachical server topology:



Protocol:

0.5/2
A somewhat more formal description would be nice. Im not sure if I understand your example correctly but it seems to be a write-all-read-one approach. However, this does not ensure that only a majority of the copies is locked.
Also, if the master is coordinating the requests, why doesn't a client directly contact the master?

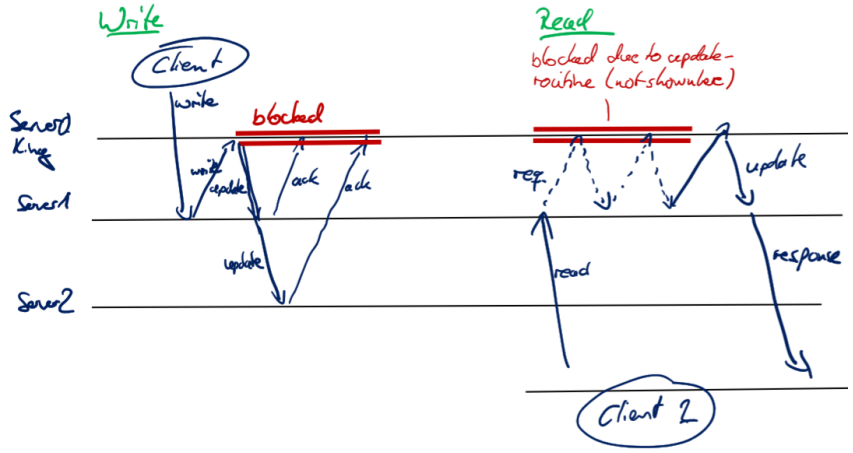


Figure 1. Replication protocol

ii. The concept ist similar to a master/slave conzept: There's a "Master-Server"which provides a block-functionality. When a Client contacts a server (read or write) the server first contacts the master to see, if he's up to date. If so, he can provide his data and know that it's the newest verison. If master is blocked, he restarts the request over and over again until free, get's update and provides update to client.

1/2 This would lead to the newest copy. However, when the master is locked all the servers are locked basically, but your protocol should only have to lock a majority of the replicas which leads to a more complex solution/explanation. Also, the master server introduces a single point of failure which could be avoided.

Exercise 2: System Models

11/12

(a)
- no process failures:
  Since request and reponse are sent at most once, if one process fails the request/response is lost and P_1 will no recieve any resonse from P_2

2/2

- no message is lost:
  Since request and reponse are sent at most once, every message which is send must be eventually recieved from the other process.

(b)
- same from a)

3/3

- dependency between t_{max} and message delay + processing time:
  It must hold that: $t_{max} > (2 \cdot \text{message-delay} + \text{processing-time})$

(c)
- same from a)

Tim Schubert (3229613)
Denis Moslavac (3227767)
Haoqing Wu (3493407)
Johannes Haberstock (3065727)

3/3

- message delay (from P_1 to P_2):
  Let Bounded-drift := d
  $t_{P_2}^{recieve}(m_{req}) \geq t_{P_1}^{send}(m_{req}) \cdot (1+d)$
  Then it must hold true, that: $\delta t >$ message-delay $\cdot (1+d)$

(d)

- same from a)
- non blocking send:
  Since $t_{P_1}^{\text{receive}}(m_{res1}) > t_{P_1}^{\text{send}}(m_{req2})$, P_1 must sends its request non-blocking in order to send Req_2 before Res_1 is recieved.

3/4 Correct, but you should mention somewhere that you need FIFO channels and FIFO processing at P2 in order to achieve this ordering

- ordering of messages:
  It has to be garanteed that Req_1 is recieved before Req_2, is processed before Req_2, the result Res_1 is send before Res_2 and the result is recieved before Res_2.
  $t_{P_1}^{\text{send}}(m_{req2}) < t_{P_1}^{\text{receive}}(m_{res1}) = t_{P_1}^{\text{send}}(m_{req1}) + \text{delay}_{mreq1} + \text{processing}_{m1} + \text{delay}_{mres1}$
  $t_{P_1}^{\text{receive}}(m_{res1}) < t_{P_1}^{\text{receive}}(m_{res2}) = t_{P_1}^{\text{send}}(m_{req2}) + \text{delay}_{mreq2} + \text{processing}_{m2} + \text{delay}_{mres2}$

Exercise 3: Three-Army-Problem

5/6

(a)

- Order of attack:
  An asynchronous or synchronous system which has no lost messengers to ensure the agreement of who leads the attrack.

1/2
+ no process failures (death of a leader)

- Time of attack:
  Only synchronous system, where every messengers arrive in time to guarantee the bound of messenger transfer time.(Messengers take at least x minutes and at most y minutes.) So that can ensure the time of attack from each divison.

(b)

1. To ensure the interval between first attack and the last attack minimal should arrange a exact timepoint for leader to start attack, and in this synchronous system is it possible for other two divisions after this time start attack with k and $2k$ minutes delay.

4/4

   - Order of attack:
     As figure 2 shows, we assume B initiates the protocal at t_0.The messages are delivered to A and C, which contains the number of B. After A and C receive the messages, they send the messages to another two division with the number of their amry as well. The transfer time is different, but since this system has no failure, all three divisons will get the information from another two sides, and find which side is the largest or smallest divison. So the guarantee of order of attack is possible.
   - Time of attack:
     The timepoints of three division which they finally get the information from another two divisons are different and need two transfer procedures. But the maximal time of these two transfer is fixed, which is $2d$. To ensure the first attack will start after the last division who receives the message, leader should start at $t_1 = t_0 + 2d$. And the second and third divisions attack at $t_2 = t_0 + 2d + k$ and $t_3 = t_0 + 2d + 2k$. This protocal makes sure that the time between the first attack and the last attack minimal, which is $2k$.

Tim Schubert (3229613)
Denis Moslavac (3227767)
Haoqing Wu (3493407)
Johannes Haberstock (3065727)

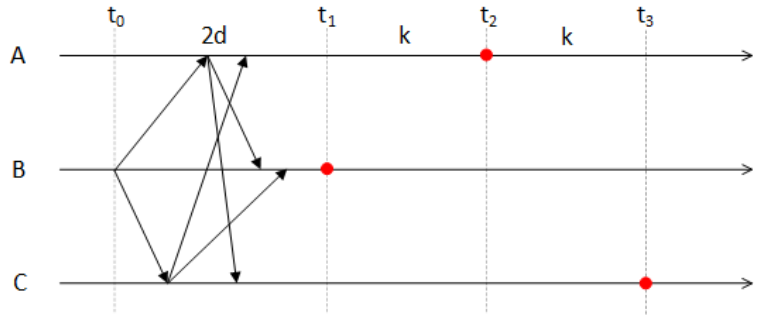


Figure 2. minimal attack interval

2. When the largest division gets the message and immediately lead the first attack without waiting can meet the need of the earliest attack.

   - Order of attack: same from 1.
   - Time of attack:
     In this case we need to make a synchronized timepoint to arrange all the attack, so that the requirement of attack interval in the first condition can be guaranteed. So we assume A is the largest division as the situation in figure 3. At the moment when A receives the message from B and C, starts its attack as leader and at the same time sends two messengers back to B and C to inform the first attack time t_1, the delivary takes up to d but $d < k$. So it's possible for the second and third divisions to arrange the attack, which are $t_2 = t_1 + k$ and $t_3 = t_1 + 2k$.

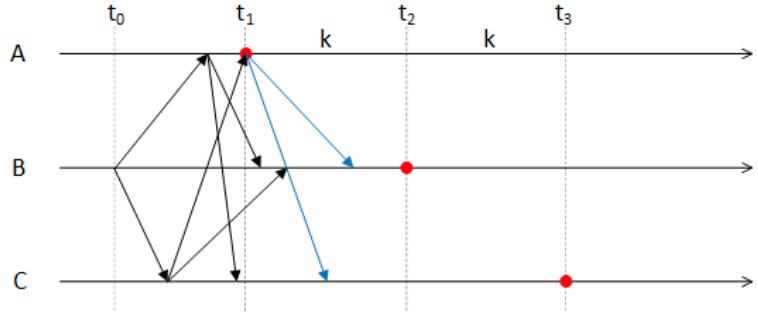


Figure 3. earliest first attack

Exercise 4: System Availability

5/6

a) $A_x = \frac{E[Uptime_x]}{E[Uptime_x]+E[Downtime_x]} = \frac{60}{100} = 0.6$
$A_y = \frac{E[Uptime_y]}{E[Uptime_y]+E[Downtime_y]} = \frac{80}{100} = 0.8$

1

b) $P = P(X_{up}) * P(Y_{up}) + P(X_{up}) * P(Y_{down}) + P(X_{down}) * P(Y_{up})$
$= 1 - P(X_{down}) * P(Y_{down})$
$= 1 - (1 - 0.6) * (1 - 0.8) = 0.92$

1

c) $P(XisUP|YisUP) = \frac{P(XisUP \cap YisUP)}{P(YisUP)} = \frac{60}{80} = \frac{3}{4} = 0.75$

2

Tim Schubert (3229613)
Denis Moslavac (3227767)
Haoqing Wu (3493407)
Johannes Haberstock (3065727)

$P(XisUP|YisDOWN) = \frac{P(XisUP \cap YisDOWN)}{YisDOWN} = \frac{0}{20} = 0$
X and Y don't seem to be fault independent because every time Y is down, X is down too.

d) In the case every Node depends on each other, the worst case of the Availability of the System ist the multiplication of each Availability
: $A_S = A_0 \cdot A_1 \cdot ... \cdot A_{i-1} \mid i \in [0, ..., |Nodes| - 1]$.
But if you can derive a correlation between Nodes, the Availability is actually higher than the worst case. Looking on our example in Figure 1, we can se the correlation between Y an X, that X is always down when Y is down but not the other way round. So let this be a correlation, the Availability of the system is:
$A_S = A_Y - P(XisDOWN|YisUP)$.
So
$A_S = A_Y - P(XisDOWN|YisUP) = 0.8 - \frac{2}{8} = 0.8 - 0.25 = 0.55$
As already said, Y seems to be independent of X and so we dont have to consider this in our calculations. Now observing the worst case in our example, we notice that the Availibility is much lower, if we dont include the correlation between X and Y.

$A_S = A_Y \cdot A_X = 0.8 \cdot 0.6 = 0.48.$

P(X = up \cup Y = up) =
P(X = up, Y = up) + P(X = up, Y = down) + P(X = down, Y = up)
= 0.6 + 0 + 0.2 = 0.8

If we compare now the Availability with task 4b), we can definitly see a difference. Depending on if the nodes are independent of each other or not, the Availability in- or decreases. 1