ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Autumn Term 2022 Prof. R. Wattenhofer

Distributed Systems
Exam

Date: 02-02-2023

Do not open or turn before the exam starts!
Read the following instructions!

This examination lasts for 90 minutes and comprises 90 points.

You may answer in German, English, or combine German and English.

Unless explicitly stated, you do **not** have to justify your answers. Writing down your thoughts (math, text, or annotated sketches), however, might help with the understanding of your approach. This may then result in points being awarded even if your answer is not correct. Please write legibly. Unreadable answers will not be graded.

Please write your name and student number on every additional sheet. Please write your name and student number in the following fields on this cover sheet.

Family Name	**First Name**	**Student Number**

Task	**Achieved Points**	**Maximum Points**
1 - Paxos		12
2 - Byzantine Agreement		22
3 - Consistency		16
4 - Blockchain		18
5 - Quorum Systems		22
Total		**90**

1 Paxos (12 points)

Remember the Paxos protocol you have seen in the lecture:

Algorithm 1 Paxos

	Client (Proposer)	**Server (Acceptor)**
	Initialization	
	c ◁ *command to execute* $t = 0$ ◁ *ticket number to try*	$T_{\max} = 0$ ◁ *largest issued ticket* $C = \perp$ ◁ *stored command* $T_{\text{store}} = 0$ ◁ *ticket used to store C*
	Phase 1	
1:	$t = t + 1$	
2:	Ask all servers for ticket t	
3:		**if** $t > T_{\max}$ **then**
4:		$T_{\max} = t$
5:		Answer with `ok`(T_{store}, C)
6:		**end if**
	Phase 2	
7:	**if** a majority answers `ok` **then**	
8:	Pick (T_{store}, C) with largest T_{store}	
9:	**if** $T_{\text{store}} > 0$ **then**	
10:	$c = C$	
11:	**end if**	
12:	Send `propose`(t, c) to same majority	
13:	**end if**	
14:		**if** $t = T_{\max}$ **then**
15:		$C = c$
16:		$T_{\text{store}} = t$
17:		Answer `success`
18:		**end if**
	Phase 3	
19:	**if** a majority answers `success` **then**	
20:	Send `execute`(c) to every server	
21:	**end if**	

We will look at a system consisting of five servers S_1, S_2, S_3, S_4, S_5. The clients have five commands $C \in \{a, b, c, d, e\}$ they want executed. Answer the questions about the system states provided below. Assume that no messages are currently in transit.

a) [3] Consider the following state. Which command could be proposed next?

S_1	S_2	S_3	S_4	S_5
$T_{\text{store}} = 1$	$T_{\text{store}} = 1$	$T_{\text{store}} = 2$	$T_{\text{store}} = 2$	$T_{\text{store}} = 2$
$C = a$	$C = a$	$C = b$	$C = b$	$C = b$

○ a.

○ b.

○ a or b.

○ any of a, b, c, d, e.

b) [3] Consider the following state. Which command could be proposed next?

S_1	S_2	S_3	S_4	S_5
$T_{\text{store}} = 1$	$T_{\text{store}} = 0$	$T_{\text{store}} = 3$	$T_{\text{store}} = 2$	$T_{\text{store}} = 3$
$C = a$	$C = \emptyset$	$C = b$	$C = a$	$C = b$

○ a.

○ b.

○ a or b.

○ any of a, b, c, d, e.

c) [2] Can the state below be a valid state of Paxos?

S_1	S_2	S_3	S_4	S_5
$T_{\text{store}} = 2$	$T_{\text{store}} = 3$	$T_{\text{store}} = 2$	$T_{\text{store}} = 3$	$T_{\text{store}} = 2$
$C = a$	$C = d$	$C = d$	$C = d$	$C = a$

○ Yes.

○ No.

d) [2] Can the state below be a valid state of Paxos?

S_1	S_2	S_3	S_4	S_5
$T_{\text{store}} = 2$	$T_{\text{store}} = 3$	$T_{\text{store}} = 2$	$T_{\text{store}} = 3$	$T_{\text{store}} = 2$
$C = \emptyset$	$C = d$	$C = \emptyset$	$C = d$	$C = \emptyset$

○ Yes.

○ No.

e) [2] Can the state below be a valid state of Paxos?

S_1	S_2	S_3	S_4	S_5
$T_{\text{store}} = 2$	$T_{\text{store}} = 3$	$T_{\text{store}} = 4$	$T_{\text{store}} = 5$	$T_{\text{store}} = 6$
$C = a$	$C = b$	$C = c$	$C = d$	$C = e$

○ Yes.

○ No.

a) b.

b) a or b.

c) No. Two different commands that are stored cannot have the same T_{store}, as to store a command the clients would have to receive a ticket from the majority of the servers. There cannot be two majorities for the same ticket.

d) No. If $T_{\text{store}} > 0$, C cannot be $\emptyset$.

e) No. There can be at most 3 unique commands stored in a system with 5 servers because to store a command you need to receive a ticket from the majority of the servers.

2 Byzantine Agreement (22 points)

Alice claims that Algorithm 2 achieves *Byzantine Agreement* in a **synchronous network** where, out of the n nodes, $f < n/3$ are byzantine, and each node's input is a bit (either 0 or 1).

Algorithm 2 Alice's algorithm

```
1: x_0 := input value
2: for i = 1 ... f + 1 do
3:     Send x_{i-1} to every node (including yourself)
4:     R := multiset of values received
5:     D := multiset obtained after discarding the lowest f and the highest f values from R
6:     x_i := (min D + max D)/2
7: end for
8: If x_{f+1} < 0.5, set y := 0. Otherwise, set y := 1.
9: Output y and terminate.
```

In this task, we will investigate whether Alice is right. Let us first recall the definition below.

a) [2] Fill in the gaps. An algorithm where each node holds an input $x \in \{0, 1\}$ and outputs a value y upon termination achieves *f-resilient Byzantine Agreement* if, even when at most f of the nodes are byzantine, the following properties hold:

- *Correct-Input Validity*:
  If an honest node outputs y, then _____.
- *Agreement*:
  If two honest nodes output y and y', then _____.
- *Termination*: Every honest node outputs a value y.

We can now analyze the algorithm. Let us first see what happens in the scenario below.

b) [3] Assume all nodes have input 0 in Algorithm 2. Which of the following are true for iteration $i = 1$?

○ True ○ False An honest node may obtain a multiset R containing both 0 and 1.

○ True ○ False An honest node may obtain a multiset D containing both 0 and 1.

○ True ○ False An honest node may obtain $x_1 > 0$.

We first show that honest nodes obtain non-empty multisets D in iteration $i = 1$, hence their values x_1 are well defined. In fact, this is the case in each iteration, which ensures *Termination*.

c) [2] Fill in the gaps.

In iteration $i = 1$, _____,
hence $|R| \geq n - f$ for every honest node.

This implies that, for every honest node $|D| \geq$_____> 0, and therefore $D \neq \emptyset$.

We can now focus on the more interesting properties.

d) [4] Let $\mathcal{X}_{i-1}$ denote the multiset of values x_{i-1} sent by honest nodes in iteration i. Show that, in iteration i, honest nodes obtain values x_i such that $\min \mathcal{X}_{i-1} \leq x_i \leq \max \mathcal{X}_{i-1}$.

e) [3] Show that Algorithm 2 achieves *Correct-Input Validity*.

It remains to check whether the *Agreement* property holds.

f) [8] Alice claims that Algorithm 2 achieves *Agreement*. Do you agree with Alice?

If you agree, prove that *Agreement* holds (you can assume *Termination* and *Validity*).

If you disagree, describe an execution (inputs, byzantine strategy, etc.) where Algorithm 2 does not achieve *Agreement*.

a) The dots can be completed as follows:

- *Correct-Input Validity*:
  If an honest node outputs y, then y is the input of some honest node.
- *Agreement*:
  If two honest nodes output y and y', then $y = y'$.

b)
- (True) Honest nodes only send 0, but corrupted nodes can send 1.
- (False) At most the f corrupted nodes may send values different than 0, hence these values are discarded. D only contains zeroes.
- (False) Since D only contains zeroes.

c) In iteration $i = 1$, at least the $n - f$ honest nodes send their values x_{i-1}, hence $|R| \geq n - f$ for every honest node.

This implies that, for every honest node $|D| \geq n - 3f > 0$, and therefore $D \neq \emptyset$.

d) Each honest node receives at most f (corrupted) values, hence outside $[\min \mathcal{X}_{i-1}, \max \mathcal{X}_{i-1}]$. Values outside this interval are discarded, hence $D \subseteq [\min \mathcal{X}_{i-1}, \max \mathcal{X}_{i-1}]$ and therefore $x_i = (\min D + \max D)/2 \in [\min \mathcal{X}_{i-1}, \max \mathcal{X}_{i-1}]$.

e) The output condition ensures all honest nodes output a bit, and if all honest nodes have input b, Task **d)** ensures that every honest node obtains $x_{f+1} = b$ and hence outputs $y = b$.

f) Alice is wrong.

We split the honest nodes into three sets: S_0 and S_1 of f nodes each, and $S_{0.5}$ of size $n - 3f$. Nodes in $S_0 \cup S_{0.5}$ have input 0, while nodes in S_1 have input 1. In each iteration, all corrupted nodes send 0 to nodes in S_0 and 1 to nodes in $S_{0.5} \cup S_1$.

We show that, in each iteration i, nodes in S_0 obtain $x_i < 0.5$, while nodes in $S_{0.5} \cup S_1$ obtain $x_i = 0.5$.

When $i = 1$, nodes in S_0 receive $n - f$ zeroes and f (discarded) ones, hence $x_i = 0 < 0.5$. Nodes in $S_{0.5} \cup S_1$ receive $n - 2f > f$ zeroes and $2f > f$ ones, hence $x_i = 0.5$.

Assuming the statement holds for $i - 1$, in iteration i, nodes in S_0 receive f zeroes, f values < 0.5, and $n - 2f > f$ values 0.5, and then obtain $x_i < (0.5 + 0.5)/2 = 0.5$. Nodes in $S_{0.5} \cup S_1$ receive f (discarded) values < 0.5, $n - 2f$ values 0.5, and f (discarded) ones, hence obtain $x_i = 0.5$.

Therefore, nodes in S_0 output $y = 0$, while nodes in $S_{0.5} \cup S_1$ output $y = 1$.

3 Consistency (16 points)

a) [8] The following shows the operations for each of three nodes in the order of execution.

```
Node 1:  R(x)=2, W(x=1), R(x)=a

Node 2:  W(x=2), R(x)=1, W(y=2)

Node 3:  R(y)=b, W(x=3), R(x)=c
```

Here we used the notation `W(x=val)` for a write operation writing value `val` to variable `x`, and `R(x)=val` for a read operation on variable `x` returning the value `val`.

Provide all possible combinations for `a`, `b`, and `c`, such that the execution above is sequentially consistent. Assume that at the beginning the variables `x` and `y` are initialized to zero.

b) [8] Consider a system with n nodes. The nodes communicate through messages. The system is asynchronous. A node is either active or asleep. An active node can go to sleep at any time, but once asleep the node only becomes active if it receives a message. The system has terminated, when all nodes are asleep *and* there are no more messages in transit. Each node i maintains two variables: r_i := total number of messages received by node i, and s_i := total number of messages sent by node i.

Bob claims that if he sequentially checks if each process is asleep and if the total number of messages sent and received are equal, then the system must have terminated (Algorithm 3).

Algorithm 3 Bob's termination algorithm

```
1: r = 0, s = 0, i = 1
2: while i ≤ n do
3:    if node i awake then
4:       r = 0, s = 0, i = 1
5:    else
6:       r = r + r_i, s = s + s_i , i = i + 1
7:    end if
8:    if i == n + 1 and r ≠ s then
9:       r = 0, s = 0, i = 1
10:   end if
11: end while
12: Output: System has terminated
```

Does Algorithm 3 correctly determine termination? Prove your answer.

a) There are 6 valid combinations for the values a, b, c:

- $a = 3,\ b = 0,\ c = 3$
- $a = 1,\ b = 0,\ c = 1$
- $a = 1,\ b = 0,\ c = 2$
- $a = 1,\ b = 0,\ c = 3$
- $a = 1,\ b = 2,\ c = 3$
- $a = 3,\ b = 2,\ c = 3$

b) The algorithm is not correct. Consider the displayed Figure 1.

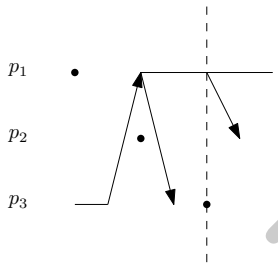

Figure 1: Example of incorrect behavior for Algorithm 3.

The black dots indicate the moments where Bob checks. Algorithm 3 will output "System has terminated" at the dashed line, even though p_1 is still active.

4 Blockchain (18 points)

a) [4] Alice owns one unspent transaction output worth 0.2023 Bitcoins. She wants to send these Bitcoins to Bob within 3 hours. While Alice is willing to pay a maximum amount δ for fees, she wants to save on fees. Suggest a strategy by filling in the gaps below. The strategy should consist of creating 10 transactions, from t_0 to t_9, at time T, where T is the current blockchain height.

$$t_0 = \{\text{input: } \{pk_A, 0.2023\},\ \text{output: } \{pk_B, 0.2023 - \_\_\_\_\_\},\ \textit{timelock: } T + \_\_\_\_\_\}$$

$$\vdots$$

$$t_i = \{\text{input: } \{pk_A, 0.2023\},\ \text{output: } \{pk_B, 0.2023 - \_\_\_\_\_\},\ \textit{timelock: } T + \_\_\_\_\_\}$$

$$\vdots$$

$$t_9 = \{\text{input: } \{pk_A, 0.2023\},\ \text{output: } \{pk_B, 0.2023 - \_\_\_\_\_\},\ \textit{timelock: } T + \_\_\_\_\_\}$$

b) [6] Alice wants to explore other ways of saving on fees. She recalls that miners will always try to earn as much as possible by including the most worthwhile transactions in their block. The size of a transaction (including inputs, outputs and header) is measured in *vbytes* and each block is limited to 1 million *vbytes*. For each statement, mark whether it is true or false.

○ True ○ False To save money, Alice can observe the transaction pool before picking how much she wants to spend on fees.

○ True ○ False Instead of sending Bitcoins immediately to Bob, Alice can wait until she also wants to make a payment to Charlie. By batching both transactions together she can save on fees.

○ True ○ False The most profitable strategy for miners is to always include transactions with the highest fee to size ratio, where the size is measured in *vbytes*.

c) [4] Alice also considers using Ethereum. Name **two** differences between Bitcoin and Ethereum in how fees are defined. (One sentence for each.)

d) [4] Imagine a new rule stating that whenever a miner observes a new Bitcoin block at the current maximum blockchain height, the miner tosses a fair coin to decide whether to replace the current head of the blockchain with the new block. What statements are true?

○ True ○ False The profitability of a selfish miner can increase.

○ True ○ False The profitability of a selfish miner can decrease.

a)

$$t_0 = \{\text{input: } \{pk_A, 2023\},\ \text{output: } \{pk_B, 2023 - 0\},\ \textit{timelock: } T + 1\}$$

$$\vdots$$

$$t_i = \{\text{input: } \{pk_A, 2023\},\ \text{output: } \{pk_B, 2023 - i * \frac{\delta}{9}\}, T + 1 + i\}$$

$$\vdots$$

$$t_9 = \{\text{input: } \{pk_A, 2023\},\ \text{output: } \{pk_B, 2023 - \delta\}, T + 10\}$$

The strategy consists of progressively paying more and more fees, until the transaction is eventually included on chain (if the max fee delta is sufficient to do so). All transactions can be created at time T without knowledge about current fee levels. Additionally, miners cannot exploit them to obtain higher fees, since transactions with higher fees can only be included later.

b)

- (True): The amount of fees required to be accepted in a block can vary widely, hence looking at new transactions is very helpful to gage the fee required to be included.
- (True): Performing two (or more) transactions in one reduces the amount of overhead, thus the amount of *vbytes* used, and thus even with a lower fee, miners will be more willing to include the transaction.
- (False): As block space is limited, it is possible for the miner to have just a few *vbytes* of capacity left, in which case it would be profitable for them to include a small transaction, even with very low fees. (In general this is simply a variation of the Knapsack-Problem)

c) Possible answers include:

(i) Unit of measurement: Ethereum uses gas cost per computation instead of cost per byte in Bitcoin.

(ii) Choosing fee: In Ethereum sender use the GasPrice field to pick the fee amount, in Bitcoin the difference between inputs and outputs defines the fee.

(iii) Redistribution: Fees in Bitcoin fully go to the miner, where as in Ethereum fees are to a large degree burned (introduced in EIP-1559).

(iv) Variable vs. fixed fees: In Ethereum fees can vary and are not always predictable at the time of emitting a transactions, unlike Bitcoin where the exact fee amount can be set a priori.

d)

- (True) If the selfish miner has no control of the scheduler, i.e., its block is rarely or never accepted by miners, the profitability will be increased after the introduction of the new rule.
- (True) If a selfish miner has had full control of the scheduler, this strategy will diminish its profitability. (In fact it is currently implemented in the *go-ethereum* client, as it improves the worst-case scenario of selfish mining.)

5 Quorum Systems (22 points)

Quorum systems can be a tool for achieving mutual exclusion. If a client holds a quorum (has access), no other client can hold a quorum. Alice wants to implement access to a resource using a quorum system involving $n \geq 10$ servers. However, instead of mutual exclusion, she wants that (i) at most **two** clients can have access (each client holds a quorum) simultaneously. Moreover, if currently one client holds a quorum, then (ii) a later-arriving client should be able to lock a different quorum, as otherwise the system degenerates to mutual exclusion. We call the two properties (i) and (ii) together *2QS*.

Alice first thinks to use the Majority quorum system: any client who successfully acquires $t = \lceil n/2 \rceil$ locks gains access; i.e. quorums consist of sets of t servers.

a) [2] Which statements regarding Alice's Majority approach are correct?

○ True ○ False No three clients can have access simultaneously.

○ True ○ False The system achieves 2QS.

b) [4] For the following values of t, is it true that the Majority system above achieves 2QS?

○ True ○ False $t = \lceil n/4 \rceil$

○ True ○ False $t = \lceil n/3 \rceil$

○ True ○ False $t = \lceil 2n/3 \rceil$

○ True ○ False $t = \lceil n/3 \rceil - 1$

c) [10] In the lecture we have learned that there are quorum systems beyond Majority. A quorum system just needs to fulfil certain properties. For each of the following properties, pick whether the property **Always (A)** / **Sometimes (S)** / **Never (N)** holds for a set family achieving 2QS.

○ A ○ S ○ N Any two distinct quorums intersect (have a common server).

○ A ○ S ○ N Any three distinct quorums intersect (at least one server is in all three quorums).

○ A ○ S ○ N In any three distinct quorums, there are two that intersect.

○ A ○ S ○ N No quorum is a strict subset of another quorum.

○ A ○ S ○ N Every quorum intersects at least one other quorum.

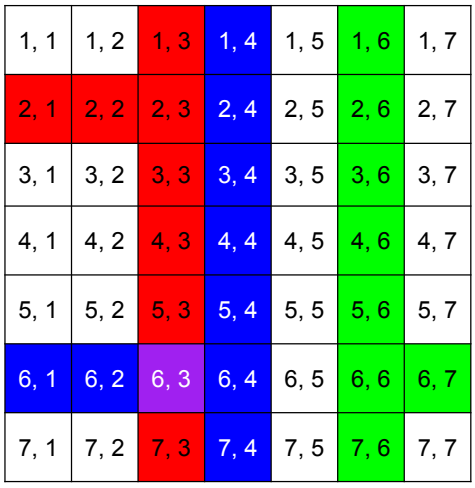

Figure 2: Alice's grid quorum system for $n = 7^2$. Two left quorums are depicted for cells $(2, 3)$ and $(6, 4)$, in red and blue respectively. One right quorum is depicted for cell $(6, 6)$ in green.

Alice now wants to scale her system with more servers, but notices that her Majority-based system comes with unreasonable load and work. Inspired by the Grid quorum system, she has an idea: assume that the servers are arranged in an $\sqrt{n} \times \sqrt{n}$ grid, with cells indexed by pairs (i, j) for $1 \leq i, j \leq \sqrt{n}$. Quorums are of two kinds: left quorums and right quorums. A left quorum is determined by a cell (i, j) and consists of all servers on column j and all servers on row i to the left of column j. A right quorum is similar, except for having all servers on row i to the right of column j. The system consists of all possible left and right quorums. Figure 2 shows an example of the construction for $n = 49$.

d) [2] What are the load and work of the system invented by Alice as a function of n? Use the asymptotic notation (ignoring constant factors, as in big $\mathcal{O}$) when answering.

- Load:
- Work:

e) [4] Does this left-right-grid system achieve 2QS? Explain your answer in 1-3 sentences.

a)

- **True**, because $3\lceil n/2 \rceil > n$.
- **False**, because for odd n we have $2\lceil n/2 \rceil > n$, so two clients can never gain access simultaneously in this case, failing to satisfy (ii).

b) **False**, **False**, **False**, **False**.

To see why, consider $n = 12$ and compute t in each case. Similarly to the previous part, in all cases we get that either too many or too few clients can get access simultaneously.

Note. For the second item, both answers were awarded full credits because the question was meant to have $n+1$ instead of n in all places defining t, but a typographical error was not spotted (this is the only item where this would change the answer).

c) **Never**, **Sometimes**, **Always**, **Sometimes**, **Sometimes**

For the first item, the property is incompatible with 2QS, as it implies mutual exclusion.

For the second item, one might be tempted to think that the answer is also **Never**, for similar reasons. This is almost true, with one pathological exception: consider a system consisting solely of two disjoint quorums, then the quantification "any three distinct quorums" is vacuously true.

For the third item, this is true because we require no three clients to have access simultaneously.

For the fourth item, one can imagine two systems satisfying 2QS: one being majority with $t = \lceil (n+1)/3 \rceil$, where no strict inclusions hold, and one consisting of three quorums A, B, C such that $A \subset B$ and $B \cap C = \varnothing$.

For the fifth item, one can imagine two systems satisfying 2QS: one being majority with $t = \lceil (n+1)/3 \rceil$, where every quorum intersects at least one other quorum, and one consisting of three quorums A, B, C such that $A \subset B$ and $B \cap C = \varnothing$, where quorum C intersects no other quorums.

d) Load is $O(1/\sqrt{n})$ and work is $O(\sqrt{n})$, similarly to the grid system seen in the lecture.

e) **No**, because if somebody locks the right quorum of $(1, 1)$, then nobody else can gain access.