CompSys Q&A

Chapter 15: Fault Tolerance & Paxos

- How precisely do we need to know the algorithms discussed in class by heart?
  For example in the Assignment on Paxos question 1.2 requires algorithm 7.13 from the script. Would we be given this as part of the exam question or do we need to have all algorithms memorized?

Our main policy on this is described on the DistSys course website: "Do not memorize the details of the remarks (e.g., what year UTC was established, or how the UTC format looks like; but you should know that UTC exists). Also, there is no need to learn all details of the algorithms and proofs. However, you should understand their concepts and ideas, so that you could explain them, or discuss variants."
Of course, if there is a question that requires you to analyze a minor detail in one of the algorithms, then we will provide the code of the corresponding algorithm.

Chapter 17: Byzantine Agreement

- In exercise 1.2b) the suggested solution is 27 but there is also a case with path length 29

We agree that there are better solutions possible, we will update this for the next year. The task mostly aims to give an understanding of what powers a byzantine adversary has in general.

- When assuming a worst-case scheduler, why do we assume that the worst-case scheduler can delay the messages sent from at most f nodes? For instance, why can't the worst-case scheduler delay all messages but those sent by byzantine nodes? Even in best effort broadcast, is it not guaranteed that eventually, every message will be delivered and accepted? If so, what is the advantage of waiting for n-f messages instead of all of them (in algorithm design) with respect to asymptotic runtime?

I guess you are talking about the asynchronous communication system. In this system, it is not possible to differentiate between messages that are delayed and messages from crashed (or byzantine) nodes, which will never arrive. A correct node cannot wait for all messages, because it does not know whether a message will arrive or not (and it cannot wait forever). Since we assume an upper bound f on the number of failures, a correct node knows that at least n-f messages will arrive for sure and so it can wait for them. The worst-case scheduler, on the other hand, knows that the nodes will wait for n-f messages. If it delays f messages long enough, the nodes will proceed with the next step of the algorithm and ignore the f delayed messages.

Chapter 19: Consistency & Logical Time

- Assignment 2.2. How to calculate the number of consistent snapshot in the system?

The number of consistent snapshots (CS) depends on the concurrency of the system and its calculation often boils down to a combinatorial problem - how many ways exist to combine the uninterrupted execution of one thread with the uninterrupted execution of another thread. To give an example, let's look at a system with 2 threads that execute the programs X (do some calculations), S (send a message), R (receive a message) in the following order:
Thread 1: XXSXXXR
Thread 2: XRXXSXX
Here, the first set of CS is all snapshots before thread 2 receives the first message. These can include from Thread 2 either the empty set {} or the set containing the first X - so 2 options. From Thread 1, as the send operation does not interrupt the execution, the CS before thread 2 receives the first message can include everything from the empty set up to {XXSXXX} - so 7 options in total. Therefore, the number of CS before thread 2 receives the first message is 2*7=14.
Next we focus on the CS before Thread 1 receives its first message, starting just after the first message has arrived at thread 2. From thread 2 this can include everything from the empty set to {RXXSXX} - 7 options. From thread 1 the empty set or anything between the send and receive operation (up to {XXX}) can be an option - so 4 in total. This yields 28 possible CS in this second set.
The last set of CS in this example includes all CS just after the second message has arrived at thread 1. That is, from thread 1 either the empty set or {R} and from thread 2 every possibility after the message has been send - empty set up to {XX}. So 2*3=6 CS in this part.
The total number of consistent snapshots is the sum of things that have to happen in sequence - counting every possibility in each part. As the first message has to be received before the second message, the total number of CS in this example is therefore 14+28+6=48.

Chapter 21: Quorum Systems

- This question is for chapter **21.4: Byzantine Quorum Systems**

  In the definition, we said that a quorum system S is f-masking if
  a. the intersection of two different quorums always contains 2f + 1 nodes, and
  b. for any set of f byzantine nodes, there is at least one quorum without byzantine nodes

  In the definition 21.28 (f-masking Grid) and 21.30 (M-Grid) only the property 1 (size of intersection of any two quorums) of the definition above is explained, while the validity of property 2 (a quorum without byzantine nodes exists) is only implied.

  Can you elaborate:
  - How the condition 2f+1n is derived in definition 21.28
  - How the property 2 is satisfied in definition 21.28
    This and the previous point are related. sqrt(n) is the number of quorums in the system, we need at least one quorum without byzantine nodes. Therefore, 1) there should be more quorums than byzantine nodes [sqrt(n) >= f+1] and 2) each quorum has to not overlap with another given quorum in at least f nodes so that the f byzantine nodes cannot affect all the quorums at the same time. If we increase the number of rows in the f-masking Grid quorum system condition 2) is not fulfilled and therefore the system is not f-masking. For

condition 2) to be met given 1), we need 2f+1 <= sqrt(n): Number of rows >= overlapping_nodes + non_overlapping_nodes => sqrt(n) >= f+1 + f = 2f+1

- How the condition fn-12 is derived in definition 21.30
- How the property 2 is satisfied in definition 21.30
  As above, this and the previous point are related. sqrt(n) is the number of quorums in the system, we need at least one quorum without byzantine nodes. To meet this condition we need f <= sqrt(n-1)/2 .

Chapter 22: Eventual Consistency & Bitcoin

- Question 2.1 b) especially the used word - maturing - is not clear in this content. It was not introduced during the lecture.

The term “maturing” is not defined in the lecture. But the regular English meaning for the word “maturing” can be used here. As a verb, it means that you have to wait for a period before the object of the verb is ready to be used. In this case, the object of the word is the reward outputs of a block. Maturing them means that the miner has to wait for 100 blocks before they can use them as inputs in other valid transactions. The solution sheet gives the actual rationale for this maturation.

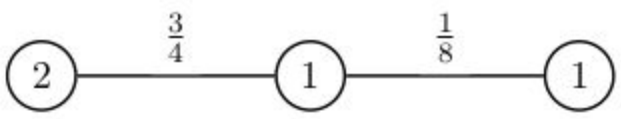
Chapter 23: Game Theory

- When doing proofs about truthfulness of an auction (e.g Theorem 23.20) are we allowed to assume that all bidders know all the bids of all other bidders? For example in proof of Theorem 23.20 it says: "By not stating the truth and decreasing his bid to b1 - eps > b2, player one could pay less and thus gain more".
  However according to Algo 23.18, b1 doesn't know the bids of the other bidders as they only tell their bid to the auctioneer (according to Def. 23.17).

According to definition 23.19, an auction is truthful if for each player, bidding his true value is a dominant strategy. Even if player 1 does not know the bid of player two in practice, he would change his strategy to b2 + eps if he knew the other bid. Thus playing z1 is not a dominant strategy.

- Is there a better way to finding the Social Optimum in Selfish Caching than just trying all possible combinations of nodes caching? In the solution of assignment 11 the Social Optimum is always a Nash Equilibrium and therefore is found with the best response strategy. However it is not generally the case that a Social Optimum is a Nash Equilibrium, or is this the case with Selfish Caching?

The Social Optimum is not always a Nash Equilibrium for the Selfish Caching game, for example :



where the demands are the numbers inside each node.
(1, 1, 0) or (1, 0, 1) are the Social Optima but the only NE is (1, 0, 0).
There are better ways to find the Social Optimum using some case analysis and taking advantage of the topology of the network, like the analysis in the solution to find the Nash Equilibria.

Chapter 24: Distributed Storage

- Can you talk about exercise 2.4 of the assignment. What is the degree of a node in the multiple skiplist?

The degree of a node in the multiple skiplist is the total number of edges connected with that node. Notice that in this question, we assume each list would wrap around at the ends. For solution to (b), you can also argue that the minimum degree is 2l+1 as it is not very useful to have two nodes connected with two edges.

Chapter 25: Authenticated Agreement

- In Chapter 17 we Assume that node can forge an incorrect sender address, then in Chapter 25 every node can sign its messages such that a node can reliably identify which node sent a message. What is now the difference from the signature to the inability of forging an incorrect sender address?

In Chapter 25 we are interested in practical byzantine fault tolerance. That is, we are fine with practical, yet theoretically imperfect assumptions. As such, we have introduced cryptographic signatures (Def. 25.1) and are confident that they will work in practice to certify the origin of a message. However, asymmetric cryptography is (nowadays?) based on the assumption that an attacker does not have enough time/computing power to try out all possible private keys (brute-force attack). In other words, the system is only secure with the assumption that an attacker is polynomially bounded in its computation time.

- Why in the prepare stage (phase 2) the node needs to wait for 2f messages (Def. 25.16) and in the commit stage for 2f+1 (phase 3) (Def. 25.17)?

In Phase 2 of the PBFT protocol, we have already received a 'pre-prepare' message from the primary. This message can be seen as a primary's 'prepare'-message and is thus incorporated into the count. 2f 'prepare'-messages + 1 'pre-prepare' message then add up to the required 2f + 1 threshold.
In the Execute phase, the primary is included just like any backup node and we thus do not differentiate between these nodes.

Chapter 27: Blockchain Research

- What is expected from us to know for the exam?

The main ideas of the December 16 lecture are relevant for the exam. We will not ask any detailed questions, and the papers were uploaded as additional material that, as such, only serve as a background.

- Some papers refer to term - close the channel. What does it mean and why it is expensive?

The closing of the channel is the process where one of the parties publishes on the blockchain a “commitment” transaction, hence spending the output of the “funding” transaction (opening of the channel). From that point on (closing of the channel), no valid off-chain transactions can be executed between the parties in this channel. It is costly because this process includes at least one on-chain transactions which costs the blockchain fee to the miners.