Quiz 2 Solution Set

Q1. Digital Twins. This question is about the concept of a *digital twin* for an IoT application that has sensors and control units (actuators), such as the rocket motors Larry Felser told us about.

[Q1.1] Define the digital twin concept.

A digital twin is just a database or some other data representation of an external real-world entity. For example, a twin could be a schema for a rocket engine and sensor readings for the status of its various instrumented components. IoT data is used to update the digital twin records, and then when applications act upon the objects that are "twins" of any control elements, the actions are pushed back to the outside world via actuators.

[*Background for Q1.2*] Suppose that connectivity with the IoT devices is pretty good, but with some brief outages that never last longer than 100ms, but can sometimes break the TCP connection from the device to the platform, after which a new TCP connection is automatically created.

[Q1.2.a] Would an application that does read-only access to the digital twin layer "observe" these outages? Focus on a case where the read is issued during an outage: what will the application see? Will it know that connectivity is disrupted?

In this situation the IoT data might lag if we briefly lose connectivity, so any software tracking our rocket needs to realize that some records could be as much as 100ms stale. We might not have any indication of this because presumably there can be brief "pauses" even if the TCP connection doesn't fail.

[Q1.2.b] Larry Felser told us about situations where the control application runs on the cloud and interacts with the digital twin, actively adjusting the rocket engine continuously to keep temperatures and pressures within the ideal range. Why would connectivity outages make this hard?

With very good timing data on the IoT devices, we could just wait 100ms to be sure we have the data from 100ms ago and we would have the full picture. But if we need to do continuous real-time control, 100ms lags might be too much and the rocket could explode or crash. And doing control with some missing data potentially would be less stable or accurate.

[Q1.2.c] The engine team believes that safe control of the rocket engine is feasible despite the connectivity outages. What is this statement telling us about timing deadlines for control actions?

This just builds on Q1.2.b: The team would need confidence that even if the rocket receives a command with a bit of delay, it would still work correctly. For example, it might be that they do small tweaks now and then but in fact the engine is still "safe" even without those, but just less efficient, so that if a control adjustment shows up 100ms late nothing blows up. Or it could be that rocket engine control is so imprecise that a 100ms delay isn't significant – maybe delays less than a few seconds are tolerable.

[*Background for Q1.3*] Now, assume that our digital twin storage layer is implemented purely by a DHT – updates go directly to the appropriate shard, and reads for a given device return the current value. Moreover, assume that if a DHT put or get happens to be issued when a TCP connection breaks and must be reestablished, the request returns an error code, "disrupted by loss of connectivity". You can

always reissue the request, but would not have any information at all about whether the disrupted put was successful.

[Q1.3.a] As a first step towards fixing this issue, define the concept of a **versioned** key-value object.

Some DHT systems track each update and consider that (key,value) pairs really evolve through a history of versions. In fact some even can keep past versions.

[Q1.3.b] Now tell us what a **conditional DHT put** does.

*The update comes with a desired version number, and the **put** only succeeds if this is the next version and no other component already did an update with the identical version number but some other value.*

[Q1.3.c] Finally, tell us how versioned objects and a versioned get and a versioned conditional put can be used to solve the problem described above. Your solution should guarantee that we update the digital twin data just once, not twice, and also that we never lose updates.

*After a connectivity loss, any **put** that was disrupted can be reissued with the same version number. We are guaranteed to see exactly-once behavior (each update is eventually recorded, and no update ever creates two versions of the underlying object).*

Q2. Consistent cuts used in a banking application. Suppose that you are creating a new purely electronic bank. Customers talking to the e-bank deposit money, withdraw money, or transfer money (a transfer occurs in two steps: the server hosting account A debits its balance by X, and then sends a message to the server hosting account B, which adds X to the destination account).

An *audit* is a tally of all the money in a system. Consider a purely online bank (an e-bank). To audit it, we would need to calculate the amount of money in each account and also all the transfers that are pending but not yet finalized. The audit should exactly match the total of deposits and withdrawals since the e-bank was created.

Now, suppose that the e-bank is large and accounts are sharded so that different accounts might be stored on different servers. Each shard is replicated using Paxos, and the e-bank system would be unavailable if a crash causes more replicas to be unavailable than the quorum needed for the request operation. For example, with 5 replicas per shard and QR=2, QW=4, customers could check their balance if 3 replicas crashed simultaneously, but couldn't do deposits – they would need to wait until at least 4 were operational again.

[Q2.1] Thinking about the guarantees provided by state machine replication and Paxos, what can you tell us about the guarantees customers can expect from the e-bank system, relative to their accounts? To answer, remind us about the properties of Paxos. For each Paxos property, "translate" it to the guarantee the bank is able to describe to its clients.

Paxos property	*Customer guarantee*
All replicas see every update.	*The customer software can safely access any replica.*
Updates are in the same order.	*The customer cannot tell which replica it accessed: results will be identical*
Committed updates are never lost.	*The bank never forgets a deposit, withdrawal or transfer.*

[Q2.2] Define Lamport's *happens before* relation, denoted $\rightarrow$.

We write A $\rightarrow$ B *to express that information could have flowed from event A to event B. In fact* $\rightarrow$ *is a transitive closure of two more basic relationships: A occurs before B in the same thread, or A is a send of a message and B is the receipt of that message. But in CS5412 we didn't talk about transitive closures and you would get full credit just for saying that information about A could have reached B.*

[Q2.3] A *consistent snapshot* has the property that if B is included in the snapshot, and A $\rightarrow$ B, then A is also included in the snapshot. A snapshot includes the state of each machine, and also the state of each network connection at the time the snapshot was created. Suppose we are auditing the e-bank. Explain why auditing along an *inconsistent* cut can cause buggy behavior but auditing along a *consistent* cut would eliminate that bug. What audit information would be found in the network portion of the state?

Assume that Sally and Henry have accounts and they were mapped to different shards.

An inconsistent cut could show that some message was received, but omit the sending of that message. For example, this could mean that Sally transfers $25 to Henry, and we see the deposit in Henry's account, yet the same money is still shown as being in Sally's account.

With a consistent cut, every dollar can be found either in an account, or in a message that is carrying out the deposit step of a transfer (a transfer reads and debits one account, then on the target account reads and rewrites the new increased balance, and the message was sent from the debit computer to the deposit computer).

[Q2.4] In class we learned about logical clocks and vector clocks. Which of these implements $\rightarrow$? For the one that *does not* implement $\rightarrow$, explain what it *does* implement, and how that relates to $\rightarrow$. Why don't we always use the one that fully implements $\rightarrow$?

Lamport's logical clocks, LC, require just a single integer per computer in the system and add space for that integer to each message, but only handle one direction of $\rightarrow$*: if A* $\rightarrow$ *B, then LC(A) < LC(B). Sometimes we might want to be able to conclude that if LC(A) < LC(B) we could conclude that A* $\rightarrow$ *B. For this we need to use a vector clock. The problem is that a vector clock is a vector with space for one integer per machine. In a datacenter with 100,000 machines the vector would be 100,000 elements long – making the overhead very high. So if a logical clock suffices we don't use vector clocks.*

Q3. **Blockchains used in cloud computing.** A blockchain is just a special form of file that provides some basic forms of protection against damage or deliberate tampering.

[Q3.1] What operations are supported?

A blockchain supports an append operation: the client has a new record and asks that the record be added to the end of the blockchain. Internally, "miners" aggregate records into blocks and then compute a proof of work, after which they propose a "block-append" operation. So the two forms of append are really the only update operations supported. A blockchain will also support reading the blockchain to see what each record contains. In practice any client normally has a copy of the whole blockchain and implements this as a local read against its replica.

[Q3.2] What are the protection guarantees and how does a blockchain implement them?

A blockchain is protected against any form of tampering. Any change will invalidate the chain from the point where that change is made forward.

[Q3.3] What is the difference between a permissionless wide-area blockchain, a permissioned wide-area blockchain, and a permissioned blockchain implemented within a single datacenter?

With a permissionless wide-area blockchain, anyone can join and propose records, or mine for new blocks. We assume a Byzantine fault model.

With a permissioned wide-area blockchain, only nodes with explicit permission can join and propose new records or participate in mining. This is useful when enterprises use blockchains. Generally, we would not see Byzantine faults in this case, but this is not a rigid aspect of being permissioned – one can imagine a permissioned blockchain that does assume Byzantine faults. (We wouldn't deduct if you insisted that it was one policy or the other.)

With a permissioned single data-center blockchain, we really have an append-only Paxos log, but sealed using cryptographic signatures to prevent tampering. We no longer assume Byzantine faults.

[Q3.4] What can cause a blockchain to roll-back? We learned that if you wait for about 6 records in the blockchain used by BitCoin, records of that age or older should be safe against rollback. Why?

In Blockchain systems, participants compete to be the next to extend the chain with a new block. You can have ties, and when this occurs the tie-breaker is that one of the participants manages to extend the chain sooner than the other, causing the shorter chain to roll back and accept the blocks from the longer one. But it is very rare to see rollbacks of more than 2 or 3 blocks, so by waiting for 6 blocks to be appended, BitCoin is delaying long enough to be sure that a one-hour old (6 blocks old) record is safely on the chain and won't rollback.

[Q3.5] Suppose that we start mining BitCoin for use in transactions both between people living fully on Earth or fully on Mars, but also for transactions between the Earth and Mars City. But then there is a solar storm that lasts for one month. During this one month period both Earth and Mars continue to use BitCoins locally. What will happen when the two systems finally are able to reconnect? How would the Vegvisir system we learned about in class address the issue?

The worry would be that because Earth is larger and probably has more datacenters for mining, when we reconnect it will roll back all the Mars records. Vegvisir solves this using a merge protocol where both branches can be retained and instead of rollback, are merged into a single shared chain, but with no double-spending allowed.

Q4. **CryptDB used in a medical setting.** This is about the system created at MIT that we heard about in class. All aspects of this question including Q4.5 can be done purely from what we learned in class. You should not need to do any additional research about CryptDB (but you may need to think about what can and cannot be done with the technique CryptDB employs, particularly for Q4.5).

[*Background for Q4.1*] CryptDB runs on top of a standard SQL database system, but if you look at a database layout as the human user imagines it, and then compare it to the database relations (tables) actually stored in the server, you would notice some differences.

[Q4.1.1] A database schema is a list of columns in the table, and their meaning. How does CryptDB change the database scheme the end-user requested?

CryptDB will often need extra columns. Depending on the SQL used in the application, a single column in the real database may be encrypted multiple times to create multiple different columns, each supporting a distinct subset of SQL operations. So basically, our database gets larger.

[Q4.1.2] How does CryptDB change the actual values – the data held within the database relation?

Data is always encrypted, and often reencrypted 2 or 3 times.

[Q4.1.3] CryptDB requires that all the clients sharing a database agree on certain encryption policies (for example, to support exact value match, all the clients have to agree on the encryption rule that takes values and encrypts them prior to upload). Does this imply that the server would also need to know those policies? Explain your answer.

The server would in general be assumed to "know" the algorithm the client is using. But the clients could have secret keys used as part of the encryption and the server would not have those. So for example even if the server knew that kpb3 (Ken's netid) is stored in the database, and knows the "method" used to encrypt kpb3, the server should still not be able to know the exact encrypted value – without the key that should not be feasible. Thus ken's records would not be identifiable.

[Q4.2] In client-server software, we use the word *stub* to refer to the module that lets the client system interface to the server software over the network. In CryptDB the stub does some extra work, compared to a database stub for a product not using the CryptDB approach. What is this extra work?

In CryptDB the stub translates the client SQL queries and data into the encrypted and expanded SQL operations the server would actually see, and then translates the results back into SQL results the client would understand.

[Q4.3] At Cornell, if you want to check the results of your weekly Covid test, you can log into a cloud server a day or two later and look up your test, and will see whether it was negative or positive. Would CryptDB be able to support this functionality? Whether you say yes or no, explain your reasoning. ***Keep in mind that in New York State, HiPPA rules say that the individual patient can see their own data, or their authorized health care providers, but this would not include letting Azure or AWS administrators see your personal data – even if they happened to know your netid. So a "yes it can be done" answer needs to include your reason that an evil administrator would not be able to "see" your records. And a "no, impossible" answer needs to explain why it is impossible.***

This can easily be done with CryptDB. My netid and the test date and the outcome would be uploaded, but the server would just see columns of encrypted data. To do the lookup I could query by exact match on netid, and see all my test results, or I could also do an exact match on date and see just one specific result. The server learns nothing because even knowing my netid doesn't give it the key needed to actually figure out what my encrypted netid would be.

[Q4.4] Suppose that the New York Times is allowed to access the Cornell Covid tracing database. Would it be possible to give the Times a way to compute how *many* tests were done, and also *the percentage of positive results* without the Times being able to learn who was tested, or who was Covid-positive?

CryptDB does have a comparison capability, so we can use that feature for date ranges, and then as long as we tell the NY Times the needed keys to do exact-match on test results, they can perform that query without learning anything else, and without the server learning anything else either.

[Q4.5] Some database queries are very expensive to compute. Suppose we have IoT devices and mobile devices with limited power and compute resources, and they talk to a cloud that has unlimited compute power. If a client wanted to perform one of these extremely costly queries, would the computing occur on CryptDB or on the client system?

The computing should occur entirely on the cloud. Of course CryptDB cannot support every possible query, so there are SQL queries that can't be computed, at all, using this approach. But when a query can be performed, it is executed entirely on the cloud database platform.