CS5412 Prelim.    Your name: ________________________________    Your Netid: ________

1. **Context: Azure IoT Hub maintains a database of sensor information, containing many billions of records, with a total read and update rates in the millions per second.**

a) [10 pts] In just a few sentences summarize Jim Gray's concerns about scalability for large databases that use locking to avoid concurrency-control update conflicts.

*Jim argues that because of lock conflicts and deadlocks that trigger rollback/retry or transaction failures that cause transactions to be resubmitted, a database system will slow down drastically as a function both of the number of servers that share equally in managing the load, and as a function of the number of transactions (operations) it is asked to perform. His real contribution centers on an analytical model used to show that performance could degrade as $N^3$ in the number of servers (N), or $T^5$ in the rate of transactions (T). If your answer talked about reads and writes or queries and updates instead of transactions or other database terminology, you will still get most of the credit as long as you captured this basic scalability concern, and that it grows as a polynomial function both of the number of servers and the workload the servers see. For full credit we looked for polynomial growth in the number of servers and number of transactions, but we awarded partial credit for aspects you explicitly talked about even if you didn't use the magic words that would earn full credit. We gave no credit if your answer didn't mention any of these points.*

b) [5 pts] Jim recommended *sharding*. Define this term. If we shard the Azure IoT hub database, what would you recommend that we use as the "key"?

*Sharding is a term for splitting some sort of database or file into smaller chunks that can be accessed independently, so that a single operation might go to a single chunk and be completely performed by it. Normally we use a (key,value) model. For the Azure IoT Hub database, the sensor-id is the most obvious choice of key, but you could also consider sharding by location ("Farmer's Brown's cow barn") with some form of secondary indexing by id. The advantage of a sharding key that somehow uses location is that sensors close to one another map to the same shard, so that if a program needed to compare data, or combine it, it could still do all the work at one server. But the key still needs to uniquely identify the sensor, to avoid conflicts.*

**c)** [10 pts] Suppose that you store a really big file on a sharded file system, where the single file will end up spread over many shards (like in GFS, HDFS, FFFS, or other similar systems). Give an example of a very simple action on the file (one involving many operations, but doing something simple) that would have been fast to do if your program was running on a single file server with the entire file on that same server, but would become very slow if you did it the identical way on the sharded representation. Then suggest a way to modify the slow program so that it would be smart about sharding and might even run faster on the sharded version than on the unsharded single server.

*In a sharded representation, our single file will be broken into chunks, and each chunk will reside on a different shard – hence on different servers. With a really big file, this form of sharding might be unavoidable because the file itself might easily be too big to fit on one server (the word "striping" is sometimes used too, it relates to sharding a single file over many servers). Consider a program that wants to do some form of sequential scan of the original file, like a word search in an editor. If the program ran on a single machine with the file on the local file server, it would run pretty fast due to automated prefetch. But if we ran that identical program on the sharded version, it would have to fetch data chunk by chunk over the network. This is sure to be much slower due to all the network overheads.*

*You can speed up this kind of slowed-down program in a few ways. One step would be to pick a sharding policy that will co-locate related data – but keep in mind that in a big-data setting it may be impossible to fit all the data on one machine, no matter how much you would love to do that. So a more important step is to consider modifying it into a parallel program that will have a subtask running at each shard. Since each subtask is local to the shard, it can scan that* *single chunk, and then we can combine the outputs. MapReduce uses this approach. In our word-search example, each chunk can be searched rapidly, and then we just combine the subresults.*

**2. Today's cloud relies heavily on the CAP principle to gain better scalability.**

a) [10 pts] Define the "C", "A" and "P" in CAP, just a sentence or so each. *Not a paragraph!*

*<u>C</u>onsistency: This can be understood to reference database consistency (transactional serializability) property. But in CS5412 it could also just refer to staleness: consistent data isn't stale. Either is fine. So in effect, data is consistent if (1) all replicas see the same <u>sequence</u> of updates, (2) failures cannot cause <u>loss</u> of data, (3) updates are seen in a timely manner and after that, <u>stale data won't be served up</u>. There are also weaker consistency models, but CAP is about the strong one.*

*<u>A</u>vailability: The system immediately responds to requests. We can understand this as meaning timely/fast reads..*

*<u>P</u>artition Tolerant: The system remains available even if the network connection to the cloud is broken or some service it normally uses is temporally down.*

b) [10 pts] Give a concrete example in which today's cloud has services or $\mu$-services that relax "C" to gain better "A" and "P". Be very specific about the sense in which "C" is being relaxed, so that we can understand exactly what "C" means for your example, and so that it will be clear to us that "C" it is not guaranteed (but "A" and "P" are).

*Many cloud systems respond to web queries using data in caches, without checking for more recent updates. If the cache becomes stale, the web page will be constructed using old content. For example, the count of "Likes" on a Facebook post is generally a bit stale. The same risk arises when a cloud function uses cached data to react to an IoT event. Photo caches often can relax consistency too, in the sense that it would be rare for a photo in a cache to be damaged in some way, and for Facebook, very unusual for a photo to be updated (it happens mostly if someone defaced a photo or deletes a photo). So we can relax consistency by not checking very actively to see if the cached copy is still valid.*

c) [5 pts] Describe an IoT cloud application scenario in which we might need "C" at all times. Does CAP tell us that such applications are *impossible*? Relate your answer to the idea behind having Azure IoT Edge run close to the sensors, rather than sending all data to the main cloud datacenter.

*A system guiding a set of drones to survey a farm might adapt their flight plans to take advantage of wind patterns. Such a system needs consistency because if some drones switch to the new plan and some use the old plan, they could collide, survey some areas twice and others not at all, etc. Azure IoT edge can offer consistency because it runs right at the farm, so in some sense it never experience a partitioning failure. We can take P off the table and focus on C+A. Banks need consistency when they allow a withdrawal – otherwise someone might overdraw their account. A hospital would need consistency to be sure that a patient is given her meds, exactly once, using the correct current dosage.*

**3. John has been accused of insider stock trading. It turns out that every communication event (phone, messaging, Facetime, Zoom, WhatsApp, etc) between John, Lilly, and John's stock broker were logged by time, but not by content (so we know exactly when they communicated, but not what they talked about). The SEC obtained this data and has linked it into a single database.**

a) [10 pts] The prosecutor says that she can show that there was an event A, where Lilly spoke to John right after learning in a confidential IBM briefing about a breakthrough in quantum computing, and an event B, where John told his stock broker to buy 5,000 shares of IBM before the announcement. The prosecutor claims that A $\rightarrow$ B. Define Lamport's "$\rightarrow$" symbol and then tell us what the prosecutor is saying, as a sentence in plain English.

*This is a reference to Lamport's happens before relation. A happens before B if information could have flowed from A to B either locally through some computer where A happened and then later B happened, or through messages. In the real world, perhaps A happened to someone and then B happened to that same person, or A spoke to B. The prosecutor is thus claiming that Lilly gained information in an event that happened before she called John, that she passed the* *information to John in an event that happened before John phoned his broker, and thus that John's IBM stock transaction was causally related to Lilly's advance insider knowledge. (This was a very long-winded answer. A shorter one will be absolutely fine).*

b) [5 pts] John's defense attorney points out that event A has an actual clock time on it, 10:02am, and that the clock time on B is 9:59am. Does this prove that John bought the stock before Lilly called?

*No, it is not a way to prove this property. The clocks could have been slightly skewed.*

c) [5 pts] Define the term "concurrent" using Lamport's $\rightarrow$ relation. Do this in words, and then again as an equation.

*In English: Two events are concurrent if they occurred simultaneously, so that neither caused the other.*

*Lamport: A and B are concurrent if neither $A \rightarrow B$, nor $B \rightarrow A$.*

d) [5 pts] In the Freeze Frame File System, when we read data using a time index, we are guaranteed temporal accuracy, but also guaranteed that we will see a consistent cut. Define a consistent cut briefly in terms of Lamport's $\rightarrow$ relation. Now give a concrete example of a task where this guarantee is important.

*If your answer used the term "file system snapshot" instead of "at a time index" this is absolutely fine. FFFS offers two APIs, one using time in a fancier form of "seek" operation, and one based on a POSIX snapshot. Either way of expressing your answer would get full credit from us.*

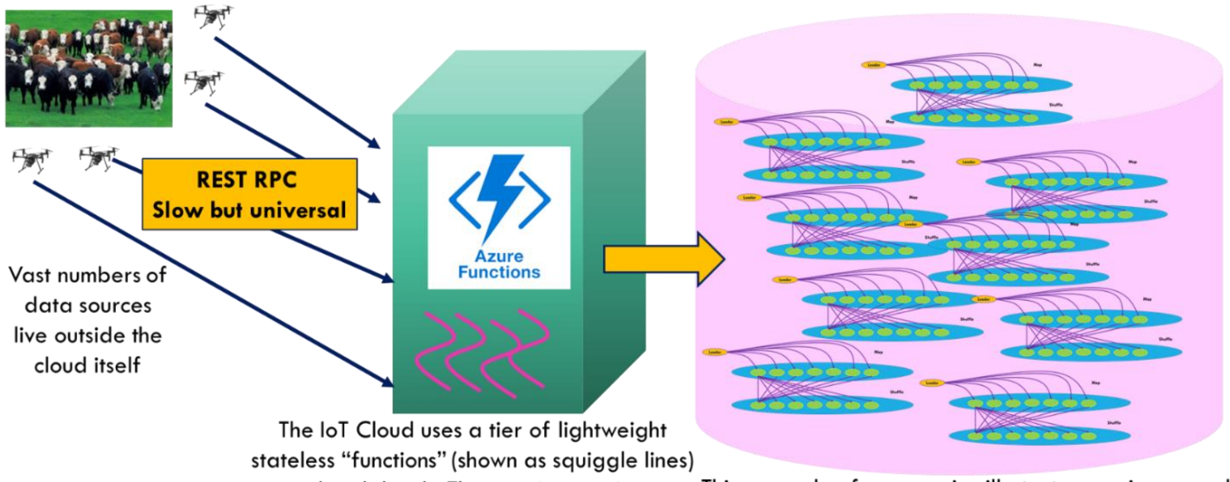
*A consistent cut is like a snapshot or photo of a distributed system, along which we have no gaps in the causal past. For example, if the cut includes event B and B was caused by A ("A happens before B"), A should be included too. The most common issue would be an <u>in</u>consistent cut in which a message was shown as received but never sent.*

*We might need a consistent cut if we wanted to collect a set of photos from different cameras and needed to avoid mashups, like HDFS can give, with a mix of photos that might even jump backwards in time. Another famous example is that if you had a bank in the cloud, and different servers holding different sets of accounts (like shards with each branch on a different shard), to audit the bank you would need a consistent cut in order to be sure that no money gets double counted or "lost".*

**4. In Azure IoT, we can write code by creating new functions that will run in the function server, or new $\mu$-services that the functions can talk to.**

a) [10 pts] Define the terms <u>function</u> and <u>$\mu$-service,</u> using a small diagram to illustrate your points (we left extra room to let you draw a small picture). Explain why a cloud using this approach can scale up and down easily (offers inexpensive elasticity). In your answer, first discuss read-only actions, and then actions that cause an update.

*Here is an image from one of our slides. It only shows one $\mu$-service, but many applications would talk to several.*



The IoT Cloud uses a tier of lightweight stateless "functions" (shown as squiggle lines) to absorb load. They run in containers.

This example of a μ-service illustrates running a machine-learning task that the function layer might talk to.

*An Azure function (or AWS lambda) is just a simple program coded in a language like C# or F# (or any other language you like). It is triggered by some event, and is launched by a service like the Azure function server, shown in the image. A function would ideally handle the event entirely: for example, if a drone requests "the next search area", the function might be able to load that data and send it back from a cached copy. Functions run in containers and nothing can be stored on the local (virtual) disk: this is erased each time the function runs. But a function can save data into a μ-service. It uses "bindings" to declare which of those μ-services it needs to talk to.*

*A μ-service is just some pool of nodes on which you run a group of processes that cooperate to solve some problem on behalf of the function layer, such as intelligently diagnosing possible cow hoof problems, or computing a search plan for a big field that optimizes for wind patterns, or telling a smart car what it has permission to do on a smart highway. These services can hold state, and in fact Azure provides standard ones like key-value storage, file storage, databases, image compression and storage, message queuing or message bus, etc. But you can also add new ones.*

*We use functions for very cheap elasticity: each event triggers a new function instance, hence we can scale very easily and inexpensively. We use μ-services for stateful purposes, where the event causes an update. A function that gets an update would simply pass the request to the relevant μ-service.*

b) [5 pts] Give one example of something a μ-service can do, but that a function can only do with help from a μ-service (or from some other standard cloud service provided by the cloud operator).

*In a smart farm, a function could tell a drone what to do next, but if the drone needed analysis of a photo, the function would probably need to forward it to a photo-analysis μ-service. Updates of all kinds also need to be sent to a μ-service.*

c) [5 pts] Denote by $F_A$ a function triggered by event A. Now, suppose that event A triggers function $F_A$ and a while later event B triggers function $F_B$. Where would we save data from event A, if we will need it to run $F_B$ when event B later occurs?

*We would save the data in a storage layer, like a key-value store or a database, running as a μ-service. We could also use the global file system, or really any kind of smart service that remembers data, like a smart "photo ID service for cows", or a smart "insect pests currently present in this area" service.*

a) [5 pts] Suppose that your team has an ML expert who creates a machine-learning classifier for cow foot and hoof health. It consists of a small piece of code that takes an image as input and requires a precomputed 6GB parameter file. As long as the parameter data has been loaded in advance, it can classify a new photo in a few seconds Would you recommend deploying this as an Azure IoT function, or as a new μ-service? Why?

*A 6GB parameter file would take a very long time to load into memory, so this task definitely cannot run as a function in the function-server layer. The problem is that if we store the parameters on disk, then every time the function launches it would start by needing to read the parameter file as an input, and this could take a long time. It would still be an issue even if we store it on HDFS or some other sharded file server, because then in addition to loading it from disk we would have to transfer it over the network. And bundling the parameters into the container that holds the function won't work either, because that container itself would still have to be loaded by the function server when it wants to launch it.*

*So we are clearly forced to put this into a new μ-service. The code in the μ-service would just run the classifier. And the parameters could then be loaded just once, when the μ-service is initially launched, after which it would be held in memory. Thus the only object that has to be moved around will be the new photo that we want to classify, and hopefully that wouldn't be a huge object (good quality photos are usually about 1MB in size).*