CS5412 / REVIEW FOR FIRST FEW WEEKS

Ken Birman

Fall, 2022

THESE SLIDES ARE FOR PRELIM REVIEW

Our exams definitely can question anything that we covered in class even if they are not on these slides

The goal of the slides is to help you review basic ("fundamental") ideas that more subtle questions might build on

Often, prelim questions in cloud computing are thought and understanding questions. Memorizing every slide won't necessarily help.

WHAT IS A MEMORY VERSUS A THOUGHT QUESTION?

Some students come from academic traditions in which books and slides have every single thing you must know.

Exams would show something from some page or some slide and you would be able to get an A+ by filling in the blank from memory

A thought question shows you a scenario you could encounter as part of your job, working in the cloud field with tools like the Azure ones. And you are asked what advice you would give your boss. This is not fill-in-the-blank.

IDEAS FROM THE FIRST WEEKS

The first weeks were really broad and introductory

For this reason there is less “specific” material in this review slide set than in the ones for state machine replication, or the ones on time and time-based concepts

Still, these broad ideas are important

IDEAS OFTEN TESTED FROM THE FIRST WEEKS OF CLOUD COMPUTING

Concept of a key-value store (DHT)

Idea that a DHT could offer a file system API and yet still be a DHT underneath

Idea that we can encode almost any kind of data structure into a DHT. A prelim question might offer an example and ask you to do the encoding.

A few details to be sure you understand:

- Sharding: How it works, and why we use it!
- How to pick a good key for each key-value pair you need to create
- Costs that arise when doing **put/get**
- Concept of **watching** a key
- Concept of customizing an elastic μ-service

IDEAS OFTEN TESTED FROM THE FIRST WEEKS OF CLOUD COMPUTING

Why do we split monolithic services into collections of smaller μ-services?

How is this actually done? What role is gRPC playing? What about message bus or queuing solutions like Kafka?

Where did the 100ms goal come from, and why do we talk about it so often?

A few details to be sure you understand:

- Jim Gray's analysis for a monolithic replicated database that tries to gain more capacity by scaling to more nodes
- The actual formula: N^3T^5
- Concept of sharding for scalability
- Need to modify SQL applications to never try and combine data from more than one shard in a single atomic action
- Why we call this NoSQL

IDEAS OFTEN TESTED FROM THE FIRST WEEKS OF CLOUD COMPUTING

CAP theorem. Meaning of BASE

We learned about C+A systems that don't need P. Why does the soft state (versus hard state) idea led to designs in which μ-services need P?

Why does Eric focus on relaxing consistency, not one of the other properties? What kind of inconsistency is he focused on? Is it arbitrary, or a more specific kind of inconsistency?

Can you list some things that definitely do need consistency? How would you decide, if you were looking at some new scenario?

Can you give examples of relaxing C?

Why does relaxing C not hurt if the application is mostly focused on caching immutable content?

- What does immutable content mean?
- If Facebook resizes photos, why is this not "mutating" the photos?
- Why cache the photos at all, if we can resize them as needed?
- What was the intellectual value of the type of study we looked at (for Facebook image cache hit rates)?

IDEAS OFTEN TESTED FROM THE FIRST WEEKS OF CLOUD COMPUTING

What does it mean to customize some sort of preexisting elastic framework?

How does this concept related to the IaaS versus PaaS distinction?

We discussed how C+A μ-services do make sense, if the μ-service or set of μ-services somehow holds "all the needed data". What would this mean, in practice?

Can you give examples of μ-services that can hold data in memory and won't need persistent data (logging)?

What might that kind of in-memory service be more easily elastic?

What does "stateless" mean, when we talk about a stateless first tier? Do we need to do all our coding in a functional language?