Quiz 2 discussion

Multiple choice: Answer the following questions

1. Which of the following assumptions should not be made about a distributed system? Mark all that apply.

A. The network latency is always low and predictable

B. The set of computers working together (called group of nodes) can change over time

C. New nodes cannot join a distributed system

D. The nodes of a distributed system may fail at any time

Solution: A, C

2. Which of the following are reasons to use NoSQL over relational databases? Mark all that apply.

A. Most of the data in present day applications is unstructured and cannot be fit in schemas

B. There is a need to maintain all the data in a single server

C. The volume of data is huge

D. Database queries need to be returned quickly

Solution: A, C, D

3. Which of the following statements about MongoDB are false? Mark all that apply.

A. All MongoDB documents should have the same set of fields

B. MongoDB is a wide column database

C. MongoDB supports JSON documents

D. Adding a field to a document will add that field to all documents

Solution: A, B, D

4. Which of the following about in-memory databases are true?

A. If the machine running the in-memory database restarts, all the data is lost

B. Fetching data from memory (RAM) is orders of magnitude faster than fetching data from the disk

C. In-memory databases need to have a predetermined schema

D. In-memory databases can never store data on the disk

Solution: A, B

5. Failure detectors have a property called completeness. When a process fails, if at least one other processor identifies that the process has failed, the failure detector is said to be complete. 100% complete failure detectors are complete under all conditions. Which of the following failure detectors is not 100% complete?

A. All-to-all heartbeating

B. Centralized heartbeating

C. Gossip protocol

D. Ping-based protocol

Solution: B

Question 2:

Are relational databases obsolete? Is there a reason to prefer them over NoSQL databases? Answer in 3-5 sentences.

Answer: Relational databases are still around, and are not obsolete. There are still applications that use structured data and benefit from the guarantees of consistency provided by SQL databases. Enhanced versions of relational databases, like MariaDB and BlinkDB, try to make SQL more suitable for present day workloads.

Question 3: Cassandra provides different consistency levels that can be ensured at query time. ANY is a level of consistency for write queries, where the query to a key succeeds if it is written to any one of the nodes, including ones that may not be managing the key. QUORUM is a level of consistency where the write succeeds only after a majority of the nodes (i.e., more than half) managing the key have been written to. Answer the following questions in 3-5 sentences each.

1. What can you say about the availability of these two consistency levels?

2. If there are n nodes in Cassandra managing a key, how many failures can the two consistency levels handle?

Answer: 1. ANY ensures high availability at the cost of eventual consistency. QUORUM has strong consistency, at the cost of low availability.

2. ANY: At least n-1. QUORUM: n/2 - 1

Question 4: Answer the following question about Chord:

A P2P system using Chord with a ring size of 64 has nodes at positions 1, 5, 7, 17, 32, 34, 40, 50, and 62. Answer the following questions and show how you arrived at the result (will be especially useful for partial credit):

1. How many entries does each finger table have?

2. A key k is hashed to 49. Which node is this key-value pair stored in?

3. Write down the finger table of node 5.

4. Write down the finger table of node 40

5. If a request for a file with key k hashed to value 49 comes to node 5, how will the request be redirected to the appropriate node? Write down the finger table entry each intermediate node looks at, and the node it forwards the request to.

6. What failure detection mechanism do you suggest for this system, and why?

Solution:

1. 6

2. 50

3. {6:7, 7:7, 9:14, 13:14, 21:32, 37:40}

4. {41:50, 42:40, 44:50, 48:50, 56:62, 8:14}

5. 50 is above 5+2^5 = 37 and below 5+2^6 - 1= 4, so node 5 looks up finger table entry 6 and redirects the request to 40. Now, 50 is above 40+2^3 = 48 and 40+2^4 = 56. So, it looks up the finger table entry for 48. Key 48 is stored in node 50, so it redirects the request to 50, the node that contains the request for 50. Node 50 then serves the request.

6. At least one node has to know about a node failing and concurrent failures should be supported. A ping-based method, or gossip work well, where "neighbors" are picked using finger tables. All-to-all heartbeating also works, but involves a large number of broadcasts.

Question 5: For her research, Alice is building an application that pulls data from twitter. She wants to study mentions of other users in tweets. Given a user, she wants to study who the user has mentioned and how many times. E.g., she wants to know who Bob mentioned in his tweets. She also wants to maintain how many times each user was mentioned in Bob’s tweets.

1. She is not sure what kind of a database to use for this application. What database would you suggest to her, and why?

2. Will Alice benefit from using an in-memory cache? Why?

Solution:

1. Graph database.

2. No for two reasons. Although it is a read-heavy workload, same data is not read repeatedly. As a cache for the graph db, in-memory writes will make the data stale.