732A54 Big Data Analytics

Examples of Exam Questions

Ashraf Sarhan

December 25, 2016

Databases for Big Data:

1. NoSQL data stores and techniques:

- Explain the main reasons for why NoSQL data stores appeared.
  - Frequent schema changes, management of unstructured and semi-structured data
  - Huge datasets
  - RDBMSs are not designed to be distributed, continuously available
  - High read and write scalability
  - Different applications have different requirements
- List and describe the main characteristics of NoSQL data stores.
  - A broad category of disparate solutions
  - Simple and flexible non-relational data models
  - High availability & relax data consistency requirement (CAP theorem)
  - Easy to distribute – horizontal scalability
  - Data are replicated to multiple nodes (Down nodes easily replaced, No single point of failure)
  - Cheap & easy (or not) to implement (open source)
- Explain the difference between ACID and BASE properties.
  - According to the CAP Theorem (Consistency, Availability, Partition Tolerance)
    - ACID: Atomicity, Consistency, Isolation, Durability
      - Atomicity : All operations in a transaction will complete, or none will
      - Consistency: before and after the transaction, the database will be in a consistent state
      - Isolation: operations cannot access data that is currently modified
      - Durability: data will not be lost upon completion of a transaction
    - BASE: Basically Available, Soft State, Eventual Consistency
      - Basically Available: an application works basically all the time (despite partial failures)
      - Soft State: is in flux and non-deterministics (changes all the time)
      - Eventual Consistency: will be in some consistent state (at some time in future)
- Discuss the trade-off between consistency and availability in a distribute data store setting.
  - According to the CAP Theorem (Consistency, Availability, Partition Tolerance)
    - Availability:
      - all requests will be answered, regardless of crashes or downtimes.
      - Example: a single instance has an availability of 100% or 0%, two servers may be available 100%, 50%, or 0%.
    - Consistency:
      - after an update, all readers in a distributed system see the same data.
      - all nodes are supposed to contain the same data at all times.
      - Example: single database instance will always be consistent, if multiple instances exist, all writes must be duplicated before write operation is completed.

- Discuss different consistency models and why they are needed.
  - Strong consistency – after the update completes, any subsequent access will return the updated value.
  - Weak consistency – the system does not guarantee that subsequent accesses will return the updated value. (inconsistency window)
- Explain how consistency between replicas is achieved in a distributed data store.
  - Consistent hashing: distributed data storage, replication (how to distribute the data)
  - Logical Time: recognize order of distributed events and potential conflicts, attach timestamp (ts) of system clock to each event.
  - Vector Clock: A vector clock for a system of N nodes is an array of N integers, Each process keeps its own vector clock, V i , which it uses to timestamp local events, Processes piggyback vector timestamps on the messages they send to one another, and there are simple rules for updating the clocks. * Explain the CAP theorem.
  - CAP Theorem: Consistency, Availability, Partition Tolerance
  - Theorem (Gilbert, Lynch SIGACT'2002): only 2 of the 3 guarantees can be given in a shared-data system.
    - Availability:
      - all requests will be answered, regardless of crashes or downtimes.
      - Example: a single instance has an availability of 100% or 0%, two servers may be available 100%, 50%, or 0%.
    - Consistency:
      - after an update, all readers in a distributed system see the same data.
      - all nodes are supposed to contain the same data at all times.
      - Example: single database instance will always be consistent, if multiple instances exist, all writes must be duplicated before write operation is completed.
    - Partition Tolerance:
      - system continues to operate, even if two sets of servers get isolated.
      - Example: system gets partitioned if connection between server clusters fails, failed connection will not cause troubles if system is tolerant.
- Explain the differences between vertical and horizontal scalability.
  - Vertical Scalability (scale up):
    - add resources (more CPUs, more memory) to a single node
    - using more threads to handle a local problem
  - Horizontal Scalability (scale out):
    - add nodes (more computers, servers) to a distributed system
    - gets more and more popular due to low costs for commodity hardware
    - often surpasses scalability of vertical approach
- Explain how consistent hashing works and what are the problems it addresses.
  - NoSQL: Techniques – Consistent Hashing
    - Task:
      - find machine that stores data for a specified key k
      - trivial hash function to distribute data on n nodes: h(k; n) = k mod n
      - if number of nodes changes, all data will have to be redistributed!
    - Challenge:
      - minimize number of nodes to be copied after a configuration change
      - incorporate hardware characteristics into hashing model
- Explain how vector clocks work and what are the problems they address.

  - NoSQL: Techniques – Consistent Hashing
    - Task:
      - A vector clock for a system of N nodes is an array of N integers.
      - Each process keeps its own vector clock, V i , which it uses to timestamp local events.
      - Processes piggyback vector timestamps on the messages they send to one another
    - Challenge
      - recognize order of distributed events and potential conflicts
      - most obvious approach: attach timestamp (ts) of system clock to each event e → ts(e) + error-prone, as clocks will never be fully synchronized + insufficient, as we cannot catch causalities (needed to detect conflicts)

- List and describe dimensions that can be used to classify NoSQL data stores.
  - Data model – how the data is stored
  - Storage model – in-memory vs persistent
  - Consistency model – strict, eventual consistent, etc.
  - Physical model – distributed vs single machine
  - Read/Write performance – what is the proportion between reads and writes
  - Secondary indexes - sort and access tables based on different fields and sorting orders
  - Failure handling – how to address machine failures
  - Compression – result in substantial savings in raw storage
  - Load balancing – how to address high read or write rate
  - Atomic read-modify-write – difficult to achieve in a distributed system
  - Locking, waits and deadlocks – locking models and version control
- List and describe the main characteristics and applications of NoSQL data stores according to their data models.
  - Key-Value Stores
    - Schema-free (Keys are unique, Values of arbitrary types)
    - Efficient in storing distributed data
    - (very) Limited query facilities and indexing
  - Column-Family Stores
    - Schema-free (Rows have unique keys, Values are varying column families and act as keys for the columns they hold, Columns consist of key-value pairs)
    - Better than key-value stores for querying and indexing
  - Document Stores
    - Schema-free (– Keys are unique, Values are documents – complex (nested) data structures in JSON, XML, binary (BSON), etc.)
    - Indexing and querying based on primary key and content
    - The content needs to be representable as a document
  - Graph Databases
    - Graph model (Nodes/vertices and links/edges, Properties consisting of key-value pairs)
    - Suitable for very interconnected data since they are efficient in traversing relationships

2. HDFS:

- Explain what HDFS is and for what types of applications it is (not) good for.
  - HDFS (Hadoop Distributed File System)
    - Runs on top of the native file system
      - Files are very large divided into 128 MB chunks/blocks
    - To minimize the cost of seeks

            - Caching blocks is possible
            - Single writer, multiple readers
            - Exposes the locations of file blocks via API
            - Fault tolerance and availability to address disk/node failures
    - Usually replicated three times on different compute nodes
    - Based on GFS (Google File System - proprietary)
    - HDFS is Good for ...
        * Store very large files – GBs and TBs
        * Streaming access
            - Write-once, read many times
            - Time to read the entire dataset is more important than the latency in reading the first record.
        * Commodity hardware
            - Clusters are built from commonly available hardware
            - Designed to continue working without a noticeable interruption in case of failure
    - HDFS is currently Not Good for ...
        * Low-latency data access
            - HDFS is optimized for delivering high throughput of data
        * Lots of small files
            - the amount of files is limited by the memory of the namenode; blocks location is stored in memory
        * Multiple writers and arbitrary file modifications
            - HDFS files are append only – write always at the end of the file

- Explain the organization of HDFS.
    - Namenode (master)
        * Manages the filesystem namespace and metadata
        * Stores in memory the location of all blocks for a given file
    - Datanodes (workers)
        * Store and retrieve blocks
        * Send heartbeat to the namenode
    - Secondary namenode
        * Periodically merges the namespace image with the edit log
        * Not a backup for a namenode, only a checkpoint

- Explain the process of reading and writing to HDFS.
    - Read a File in HDFS:
        * Client opens the file it wishes to read by calling open() on the FileSystem object, which for HDFS is an instance of DistributedFileSystem.
        * DistributedFileSystem. Calls the namenode using RPC to determine the locations of the blocks for the first few blocks in the file
        * For each block the namenode returns the addresses of the datanodes that have a copy of that block and datanodes are sorted according to their proximity to the client.
        * DistributedFileSystem return an FSDataInputStream to the client for it to read data from. FSDataInputStream in turns wraps the DFSInputStream which manages the datanode and namenode I/O
        * Client calls read() on the stream. DFSInputStream which has stored the datanode addresses then connects to the closest datanode for the first block in the file.
        * Data is streamed from the datanode back to the client, which calls read() repeatedly on the stream. When the end of the block is reached DFSInputStream will close the connection to the datanode and then finds the best datanode for the next block.

        * If the DFSInputStream encounters an error while communicating with a datanode, it will try the next closest one for that block. It will also remember datanodes that have failed so that it doesn't needlessly retry them for later blocks. The DFSInputStream also verifies checksums for the data transferred to it from the datanode. If a corrupted block is found, it is reported to the namenode before the DFSInputStream attempts to read a replica of the block from another datanode
    - Write a File in HDFS:
        * DistributedFileSystem object do a RPC call to namenode to create a new file in filesystem namespace with no blocks associated to it
        * Namenode process performs various checks like a) client has required permissions to create a file or not 2) file should not exists earlier. In case of above exceptions it will throw an IOexception to client
        * Once the file is registered with the namenode then client will get an object i.e. FSDataOutputStream which in turns embed DFSOutputStream object for the client to start writing data to. DFSoutputStream handles communication with the datanodes and namenode.
        * As client writes data DFSOutputStream split it into packets and write it to its internal queue i.e. data queue and also maintains an acknowledgement queue.
        * Data queue is then consumed by a Data Streamer process which is responsible for asking namenode to allocate new blocks by picking a list of suitable datanodes to store the replicas.
        * The list of datanodes forms a pipeline and assuming a replication factor of three, so there will be three nodes in the pipeline.
        * The data streamer streams the packets to the first datanode in the pipeline, which then stores the packet and forward it to second datanode in the pipeline. Similarly the second node stores the packet and forward it to next datanode or last datanode in the pipeline
        * Once each datanode in the pipeline acknowledge the packet the packet is removed from the acknowledgement queue.

- Explain how high availability is achieved in HDFS.
    - The namenode is single point of failure:
        * If a namenode crashes the cluster is down
    - Secondary node
        * periodically merges the namespace image with the edit log to prevent the edit log from becoming too large.
        * lags the state of the primary prevents data loss but does not provide high availability
        * time for cold start 30 minutes
    - In practice, the case for planned downtime is more important
    - Pair of namenodes in an active stand-by configuration:
        * Highly available shared storage for the shared edit log
        * Datanodes send block reports to all namenodes
        * Clients must provide transparent to the user mechanism to handle failover
        * The standby node takes checkpoints of the active namenode namespace instead of the secondary node