Examples of exam question types

Databases for Big Data

NoSQL data stores and techniques

?Explain the main reasons for why NoSQL data stores appeared.
List and describe the main characteristics of NoSQL data stores.
1.A broad category of disparate solutions
2.simple and flexible non-relational data models
3.High availability & relax data consistency requirement (CAP theorem)
4.Easy to distribute – horizontal scalability
5.Data are replicated to multiple nodes
6.Cheap & easy (or not) to implement (open source)
Explain the difference between ACID and BASE properties.
ACID: properties needed to guarantee consistency and availability
• Atomicity ☾ A transaction is an atomic unit: It is either executed completely or not at all
• Consistency ☾ A database that is in a consistent state before the execution of a transaction, is also in a consistent state after the execution of the transaction
• Isolation ☾ A transaction should act as if it is executed in isolation from the other transactions
• Durability ☾ Changes in the database made by a committed transaction are permanent

BASE: properties come into play if availability and partition tolerance is favored (schema-free, easy replication support, simple API, eventually consistent/BASE)
• Basically Available ☾ an application works basically all the time (despite partial failures)
• Soft State ☾ is in flux and non-deterministic (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.

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
• Weak consistency
– Eventual consistency – if no new updates are made to the object, eventually all accesses will return the last updated value
• Popular example: DNS

Explain how **consistency** between **replicas** is achieved in a distributed data store.
N – number of nodes that store replicas
R – number of nodes for a successful read
W – number of nodes for a successful write

R + W > N strong consistency: Consistency (& reduced availability)

W=N – R + W ⩽ N eventual consistency: Inconsistency window – the period until all replicas have been updated in a lazy manner

Explain the **CAP theorem**.
CAP Theorem: Consistency, Availability, Partition Tolerance

Explain the differences between **vertical and horizontal scalability**.
Scalability: system can handle growing amounts of data without losing performance.
• 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.
distributed data storage, replication (how to distribute the data) Consistent hashing
• 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!

Explain how **vector clocks** work and what are the problems they address.
– VC1: Initially, Vi [j] = 0, for i , j = 1, 2, ... N
– VC2: Just before pi timestamps an event, it sets Vi [i] := Vi [i] + 1
– VC3: pi includes the value t = Vi in every message it sends
– VC4: When pi receives a timestamp t in a message, it sets Vi [j] := max(Vi [j]; t [j]), for j = 1, 2, ... N

List and describe dimensions that can be used to classify NoSQL data stores.(HBase)
• Data model – how the data is stored
• Storage model – in-memory vs persistent
• Consistency model – strict, eventual consistent, etc.
  – Affects reads and writes requests
• 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.
(Impacts application, querying, scalability)

• Key-Value Stores
Characteristics:
Schema-free
– Keys are unique
– Values of arbitrary types
Efficient in storing distributed data
Limited query facilities and indexing
– get(key),put(key,value)
– Value ☾ opaque to the data store ☾ no data level querying and indexing
Applications:
– Storingwebsessioninformation
– User profiles and configuration
– Shoppingcartdata
– Using them as a caching layer to store results of expensive operations (create a user-tailored web page)

• 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
• MongoDB, CouchDB, Couchbase
• Applications:
– Items with similar nature but
different structure
– Bloggingplatforms
– Contentmanagementsystems
– Eventlogging
– Fast application development

• 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
• Types
– Googles BigTable, Hadoop HBase
– No column families –
Amazon SimpleDB, DynamoDB
– Supercolumns-Cassandra

• Graph Databases
• Graph model
– Nodes/verticesandlinks/edges

– Properties consisting of key-value pairs
• Suitable for very interconnected data since they are efficient in traversing relationships
• Not as efficient
– as other NoSQL solutions for non-graph applications – horizontal scaling
• Applications:
– location-basedservices
– recommendationengines
– complexnetwork-basedapplications
• social, information, technological, and biological network
– memory leak detection

HDFS
Explain what HDFS is and for what types of applications it is (not) good for.

Explain the organization of HDFS.
Explain the process of reading and writing to HDFS.
Explain how high availability is achieved in HDFS.

Dynamo
Explain the data model and list main applications of Dynamo.
Explain the Dynamo design considerations and what are the advantages of Dynamo in comparison to RDBMSs.
Explain how basic NoSQL techniques are applied in Dynamo.
Explain versioning and semantic reconciliation in Dynamo.

HBase
Explain the difference between column-oriented and row-oriented storage.
Explain the data model of HBase.
Give example applications of HBase.
Hive and Shark/SparkSQL
Explain the problem that Hive and Shark/SparkSQL address.
Explain the data model of Hive and Shark/SparkSQL.
Discuss the trade-off between schema-on-read and schema-on-write approaches.
Explain the difference between OLAP and OLTP.
Explain the main differences between Hive and Shark and what are the advantages they lead to.
Explain how fault tolerance is achieved in Shark/SparkSQL.

Parallel Computing
PAR-Q1: Define the following technical terms:
(Be thorough and general. An example is not a definition.)
Cluster (in high-performance resp. big-data computing)
Parallel work (of a parallel algorithm)
Parallel speed-up
Communication latency (for sending a message from node Pi to node Pj)
Temporal data locality

Dynamic task scheduling

PAR-Q2: Explain the following parallel algorithmic paradigm: Parallel Divide-and-Conquer.

PAR-Q3: Discuss the performance effects of using large vs. small packet sizes in streaming.

PAR-Q4: Why should servers (cluster nodes) in datacenters that are running I/O-intensive tasks (such as file/database accesses) get (many) more tasks to run than they have cores?

PAR-Q5: In skeleton programming, which skeleton will you need to use for computing the maximum element in a large array? Sketch the resulting pseudocode (explain your code).

PAR-Q6: Describe the advantages/strengths and the drawbacks/limitations of high-level parallel programming using algorithmic skeletons.

PAR-Q7: Derive Amdahl's Law and give its interpretation.

PAR-Q8: What is the difference between relative and absolute parallel speed-up? Which of these is expected to be higher?

PAR-Q9: The PRAM (Parallel Random Access Machine) computation model has the simplest-possible parallel cost model. Which aspects of a real-world parallel computer does it represent, and which aspects does it abstract from?

PAR-Q10: Which property of streaming computations makes it possible to overlap computation with data transfer?

MapReduce
MR-Q1: A MapReduce computation should process 12.8 TB of data in a distributed file with block (shard) size 64MB. How many mapper tasks will be created, by default? (Hint: 1 TB (Terabyte) = 10^12 byte)

MR-Q2: Discuss the design decision to offer just one MapReduce construct that covers both mapping, shuffle+sort and reducing. Wouldn't it be easier to provide one separate construct for each phase instead? What would be the performance implications of such a design operating on distributed files?

MR-Q3: Reformulate the wordcount example program to use no Combiner.

MR-Q4: Consider the local reduction performed by a Combiner: Why should the user-defined Reduce function be associative and commutative? Give examples for reduce functions that are associative and commutative, and such that are not.

MR-Q5: Extend the wordcount program to discard words shorter than 4 characters.

MR-Q6: Write a wordcount program to only count all words of odd and of even length. (There are several possibilities.)

MR-Q7: Show how to calculate a database join with MapReduce.

MR-Q8: Sometimes, workers might be temporarily slowed down (e.g. repeated disk read errors) without being broken. Such workers could delay the completion of an entire MapReduce computation considerably. How could the master speed up the overall MapReduce processing if it observes that some worker is late?

Spark-Q1: Why can MapReduce emulate any distributed computation?

Spark-Q2: For a Spark program consisting of 2 subsequent Map computations, show how Spark execution differs from Hadoop/Mapreduce execution.

Spark-Q3: Given is a text file containing integer numbers. Write a Spark program that adds them up.

Spark-Q4: Write a wordcount program for Spark. (Solution proposal: see last slide in lecture 8.)

Spark-Q5: Modify the wordcount program by only considering words with at least 4 characters.

Cluster Resource Management

YARN-Q1: Why is it reasonable that Application Masters can request and return resources dynamically from/to the Resource Manager (within the maximum lease initially granted to their job by the RM), instead of requesting their maximum lease on all nodes immediately and keeping it throughout the job's lifetime? Contrast this mechanism to the resource allocation performed by batch queuing systems for clusters.

YARN-Q2: Explain why the Node Manager's tasks are better performed in a daemon process controlled by the RM and not under the control of the framework-specific application.

Machine Learning for Big Data

Implement in MapReduce or Spark a machine learning algorithm,
e.g.
logistic regression,
k-means,
EM algorithm,
support vector machines,
neural nets, etc.
(The pseudo-code of the algorithm will be provided in the exam.)