Examples of exam question types

Databases for Big Data

- NoSQL data stores and techniques
  - Explain the main reasons for why NoSQL data stores appeared.
    - Increasing numbers of concurrent users/clients
      - tens of thousands, perhaps millions
      - globally distributed
      - expectations: consistently high performance and 24/7 availability (no downtime)
    - Different types of data
      - huge amounts (generated by users and devices)
      - data from different sources together
      - frequent schema changes or no schema at all
      - semi-structured and unstructured data
    - Usage may change rapidly and unpredictably
  - List and describe the main characteristics of NoSQL data stores.
    - Ability to scale horizontally over many commodity servers with high performance, availability, and fault tolerance
      - achieved by giving up ACID guarantees
      - and by partitioning and replication of data
    - Non-relational data model, no requirements for schemas
      - data model limitations make partitioning effective
  - Explain the difference between ACID and BASE properties.
    - ACID:
      - Atomicity: Everything in a transaction need to excute successfully otherwise the whole transaction will not be excuted.
      - Consistency preservation: A transaction cannot leave the database in an inconsistent state.
      - Isolation: Transactions cannot interfere with each other and can be excuted concurrently.
      - Durability: Completed transactions persist, even when servers crash or restart the transactions remain committed.
    - BASE:
      - Basically Available: system available whenever accessed, even if parts of it unavailable
      - Soft state: the distributed data does not need to be in a consistent state at all times
      - Eventually consistent: state will become consistent after a certain period of time
    - BASE properties suitable for applications for which some inconsistency may be acceptable
  - Discuss the trade-off between consistency and availability in a distribute data store setting.
    - When choosing consistency over availability, the system will return an error or a time out if particular information cannot be guaranteed to be up to date due to network partitioning.

When choosing availability over consistency, the system will always process the query and try to return the most recent available version of the information, even if it cannot guarantee it is up to date due to network partitioning.

- Discuss different consistency models and why they are needed.
  - Strong consistency: after an update completes, every subsequent access will return the updated value, usually needed because
  - Weak consistency: no guarantee that all subsequent accesses will return the updated value
    - A type of Weak Consistency is eventual Consistency in which if no new updates are made, eventually all accesses will return the last updated value

- Explain the CAP theorem.
  Only 2 of the following 3 properties can be guaranteed at the same time in a distributed system with data replication
  - Consistency: the same copy of a replicated data item is visible from all nodes that have this item
  - Availability: all requests for a data item will be answered
  - Partition Tolerance: system continues to operate even if it gets partitioned into isolated sets of nodes

- Explain the differences between vertical and horizontal scalability.
  - Vertical scalability: Add resources to a server (e.g., more CPUs, more memory, more or bigger disks)
  - Horizontal scalability: Add nodes (more computers) to a distributed system

- List and describe the main characteristics and applications of NoSQL data stores according to their data models.
  - Key-value model:
    Characteristics:
    - Database is simply a set of key-value pairs
      keys are unique - values of arbitrary data types
    - Values are opaque to the system
    - Only CRUD (create, read, update, delete) operations in terms of keys
    - No support for value-related queries (no secondary index over values) because values are opaque to the system
    - Accessing multiple items requires separate requests, often not possible with one transaction
    - partition the data based on keys ("horizontal partitioning", also called "sharding") and distributed processing can be very efficient

    Applications:
    Whenever values need to be accessed only via keys:
    - Storing Web session information
    - User profiles and configuration
    - Shopping cart data
    - Caching layer that stores results of expensive operations (e.g., complex queries over an underlying database, user-tailored Web pages)
  - Document model:
  - Wide-column models:
  - Graph database models:

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)
    Aggregation of many computers/servers connected together as a single unit such that it can be controlled and scheduled to work on similar task
  - Parallel work (of a parallel algorithm)
    The total number of performed elementary operations
  - Parallel speed-up
    The factor by how much faster we can solve a problem with p processors than with 1 processor, usually in range (0,...,p)
  - Communication latency (for sending a message from node Pi to node Pj)
    The time interval for sending a message from node Pi to node Pj where high latency favors larger transfer block sizes (cache lines , memory pages, file blocks, messages) for amortization over many subsequent accesses
  - Temporal data locality
    The re-access of the same data element multiple times within a short time interval
  - Dynamic task scheduling
    Task scheduling method in which the priorities are calculated during the execution of the program

- PAR-Q2: Explain the following parallel algorithmic paradigm: Parallel Divide-and-Conquer.
  - If given problem instance P is trivial , solve it directly. Otherwise:
  - Divide: Decompose problem instance P into one or several smaller independent instances of the same problem, $P_1, ..., P_k$
  - For each i: solve P_i by recursion.
  - Combine the solutions of the P_i into an overall solution for P

  Where:

  - Recursive calls can be done in parallel.
  - Divide and combine phases can parallized when possible
  - Switch to sequential divide and conquer when enough parallel tasks have been created.

- PAR-Q3: Discuss the performance effects of using large vs. small packet sizes in streaming.
  - When the packet size is small the throughput (operations per second) will be small and might not utilize the available memory where as large packet size might overflow the memory and also cause a delayed 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?
  - It helps with load balancing

- 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).

```
# instantiating a skeleton template in a user-provided function without parallelism concerns
float max(int a, int b)
{
  return (a > b) ? a : b;
}

# Using the user-provided function in a Reduce skeleton
auto array_max = Reduce(max);

# Excuting the code
array_max(array);
```

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

  Advantages:

  - Abstraction, hiding complexity (parallelism and low level programming)
  - Parallelization for free
  - Easier to analyze and transform

  Advantage/Drawback:

  - Enforces structuring, restricted set of constructs

  Drawbacks:

  - Requires complete understanding and rewriting of a computation
  - Available skeleton set does not always fit
  - May lose some efficiency compared to manual parallelization

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

For parallel algorithm A

Where:
sequential part A^s works only on one processor
parallel part A^p can be sped up by p processors

$$\text{Total Work } w_A(n) = w_{A^s}(n) + w_{A^p}(n)$$

Total Work = number of elementary operations performed by the sequential part + number of elementary operations performed by the parallel part

$$\text{Time } T = T_{A^s} + \frac{T_{A^p}}{p}$$

If the sequential part of A is a fixed fraction of the total work irrespective of the problem size n, that is, there is a constant β with:

$$\beta = \frac{w_{A^s}(n)}{w_A(n)} \leq 1$$

The relative speed up of A with p processors is limited by:

$$\frac{p}{\beta p + (1-\beta)} < \frac{1}{\beta}$$

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

$$\text{Absolute Speedup } S_{abs} = \frac{T_s}{T_{(p)}}$$

$$\text{Absolute Speedup } S_{rel} = \frac{T_{(1)}}{T_{(p)}}$$

Where for algorithm A:
T_s: Time to excute the best serial algorithm for a problem on one processor of the parallel machine
$T_{(1)}$: Time to excute parallel algorithm A on 1 processor
$T_{(p)}$: Time to excute parallel algorithm A on p processors

$$S_{abs} \leq S_{rel}$$

- 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.)
- Logistic Regression

```python
# Reading from file => getting the required data by map() =>
# persisting in memory for faster access
points = sc.textFile(...).map(...).persist()

# Intial random weights
w = np.random.ranf(size = D)

# Compute logistic regression gradient for a matrix of data points
def gradient(matrix, w):
  Y = matrix[:, 0]  # Labels
  X = matrix[:, 1:] # Coordinates
  # For each point (x,y), compute gradient function, then sum these up
  return ((1.0/(1.0 + np.exp(-Y * X.dot(w))) - 1.0) * Y * X.T).sum(1)

for i in range(iterations):
  w -= points.map(lambda m: gradient(m, w).reduce(lambda a,b: a+b))
```

- KNN

```python
import numpy as np
from math import sqrt
from pyspark import SparkContext

sc = SparkContext(appName = "KNN")

# getting the data in a (y, (x1,...xn)) format where y is the class label
# and x1,...,xn are the predictive attribute values
# mydata = sc.textFile().split().map(lambda x: (x[0], (x[1:])))
# example of mydata
mydata = ((0,(2,1)), (1,(4,5)), (0,(1,3)), (0,(-2,1)), (1,(5,3)), (0,(1,1)), (1, (2,2.5)))
# we use parallelize to partition the tuple so spark can work on it in parallel
mydata = sc.parallelize(mydata)

def getDistance(x1, x2):
  """
  calculates the distance between two points
  """
  n = len(x1)
  distance = 0.0
  for i in range(n):
    distance += (x1[i] - x2[i])**2
  return sqrt(distance)


# (k, (x1,...,xn)) where k is the number of nearest neighbors
# and (x1,...,xn) is the point of interest, in this case it have 2 attributes
k = 3
parameters = (3, (2,3))
# Broadcast the parameters to all nodes
bc = sc.broadcast(parameters)

# map the data to (class, (distance, 1)) => sort it from smallest to largest
# => and take the ones with the k shortest "smallest" distances
kNeighbors = mydata.map(lambda x: (x[0], ((getDistance(x[1], bc.value[1]), 1)))) \
                         .sortBy(lambda k: k[1][0], ascending=True) \
                         .take(bc.value[0])
kNeighbors = sc.parallelize(kNeighbors)
print(kNeighbors.collect())

# map to (class, 1) => sum the counts => sort from largest to smallest this time
# => and take the largset one
pred = kNeighbors.map(lambda x: (x[0], x[1][1])) \
                 .reduceByKey(lambda a,b: a+b) \
                 .sortBy(lambda x:x[1], ascending=False) \
                 .take(1)

print(f"predicted class is: {pred[0][0]}")
```

- Cross-Validation

```
# number of folds
K = 3
fold_size = int(len(mydata) / K)

CVdata = tuple((mydata[k:k + fold_size],k) for k in range(0, len(mydata), fold_size))
CVdata = sc.parallelize(CVdata)

error = sc.emptyRDD()
p = (2,3)
for i in range(0, len(mydata), fold_size):
    train = CVdata.filter(lambda x: x[1] != i).flatMap(lambda x: x[0])
    test  = CVdata.filter(lambda x: x[1] == i).flatMap(lambda x: x[0])
    testError = test.foreach(lambda x: (x[1][0] - LR(p,i))**2, 1)
    testError = sc.parallelize(testError).reduce(lambda a,b: (a[0]+b[0], a[1]+b[1]))
    error[i] = x[0]/x[1]

MSE = mean(error)

print(f"Average Test Error: {MSE}")
```