Exam 2022

1. ***(2 points) Assume you are designing a NoSQL database to store students' profiles. Each student has a unique ID and some extra information, which are not fixed among students. What data model do you use in your database? Moreover, assume you want to use consistent hashing to store students' profiles across computers. Explain how you can use consistent hashing to spread data in your network.***

Based on the given use case we have students with unique IDs and some extra attributes where the attributes are not fixed. The best data model to use for this type of database is either key-value or column-oriented. Example of databases that use key value is dynamoDB and Voldemort. An example of a database that uses column-oriented is Cassandra.

I think for this specific use case key-value database is the best model. DynamoDB allows querying on a PK which in this case is the student ID to get all attributes for it. DynamoDB is built using consistent hashing to distribute data across nodes.

Consistent hashing works by distributing nodes and data over a hash ring. The nodes (computers) are hashed using a function that returns a location on the hash ring. In reality hash rings also use virtual nodes to more efficiently divide up the computer on the hash ring to more equally divide up the computation power.

Then the data that will be inserted like the student ID will be hashed using the same function which will return a location on the hash ring. Then in a clockwise order, the location will continue to walk until it finds the first node where it can be stored.

Consistent hashing is useful due to being able to in a good way distribute data and computation over nodes. The benefit of using a hash ring is that it can in a good way distribute data even if a specific node is added or removed. This is because typical direct hashing for dividing computation power is required to recalculate all data to evenly distribute it when a node is added or removed while consistent hashing allows for only a portion of the data/computation to be moved to another node.

2. ***(2 points) Imagine you are working in a big company, and your company is planning to launch the next big Blogging platform. Tomorrow morning you go to your office and see the following mail from your CEO regarding new work. How do you answer this email? Hint: use MapReduce to solve it, and explain what you do in the Map and Reduce phases. As you know, we are building a blogging platform, and I need some statistics. I need to find out, across all blogs ever written on our blogging system, how many times one-character words occur (e.g., "a", "I"), How many times two-character words occur (e.g., "be", "is"), and so on. I know it's a tremendous job. I am going on vacation for one week, and I must have this when I return. Good luck.***

Data process:
During the data process step, I would first read all of the blog posts.

Map:
During the map step, we will read all of the blog posts. Each blog post will be mapped in such a way that each word is assigned a key to it. The key will be determined by the number of characters that the word consists of. For example the word “hey” results in (3,1) where the 1 symbolizes the value (initial value of the count of 3 characters) and the word store results in (5,1) etc.

Shuffle & Sort: Handle by the system to group the characters.

Reduce:
After the shuffle and sort step each reducer will have a collection of a specific amount of characters for example the first reducer will have (1,1), (1,1)... of all the words that contain just 1 character, and reducer two will have (2,1), (2,1)... of all the words that contains just 2 characters. The reducer will then sum all of the elements to get the total amount of words that contain just 1 character, 2 characters, etc.

3. ***(2 points) Explain the difference between the Transformations and Actions in Spark. Moreover, explain how Spark handles failures.***

Transformations in spark are operations such as map, filter, flatMap, and groupByKey. These types of actions are lazy where they are not computed immediately but instead, remember the transformations and build a logical execution plan. By doing this and not immediately calculating it allows for optimizations.

Actions in spark are operations such as savetotxtfile, count, first, and collect. These actions are triggered immediately upon execution.

Spark handles failures by:

- Task failure, a task will be retried up to 4 times before giving up
- Node failure, if a node fails all running tasks will be executed on another node
- Uses data persistence and caching to retrieve data due to node failure which could allow for the spark to recompute them.
- Stagger handling

4. ***(1 points) Please list the differences between stateful and stateless operations in streaming processing programming models. Give two examples of each operation.***

Stateful operations in streaming processing programming models are actions that are performed over multiple different sessions. Such operations could be windowing where the goal is to process data between different sessions meaning it is stateful. Another operation is joining streams which, aggregations (count number events over a window, buffering, etc…

Stateless operations in streaming processing programming models can be operations such as map or filter which is stateless because it is operations that are only executed on the current data and don't involve different states.

5. ***(1 points) What type of process (ETL or ELT) is executed in the Dataware house and Data lake? Briefly explain their differences.***

ETL (Extract Transform Load):

- Data is extracted from a source system
- Data is transformed before loading into the data warehouse, this might involve cleaning, filtering aggregating, and converting data
- Load data into the warehouse
- **Reason:** ETL processes have been associated with data warehouses which in many cases require data processing before it's loaded due to the data being structured.

ELT (Extract Load Transform):

- Data is extracted from a source system
- The data is loaded into the data processing environment
- After loaded data is transformed in its destination environment.
- **Reason:** Commonly used in Data lakes and modern data warehouses. This is because data lakes support raw data to be ingested

Performance:
ELT leverages the power of modern cloud data warehouses which allows for better scalability and performance

Flexibility:
ELT allows to storage of raw data.

6. ***(2 points) In an undirected graph, each of two directly connected nodes is called neighbors. Assume you have an undirected graph in which each node stores a pair (id, #neighbors), where the first item is the node's ID, and the second one is the number of node's neighbors. Write three pseudo-codes in the Pregel, Graphlab, and PowerGraph to find the node ID with the smallest number of neighbors. If two nodes have the same number of neighbors, choose the one with the smallest node ID.***

Exam 2018

1. ***Distributed file systems such as GFS store files in chunks. The size of a chunk has implications on the system's performance. What are the advantages and disadvantages of storing files in large chunks?***

Storing large chunks using GFS has both some advantages and disadvantages. The advantages are that less metadata is required to be stored by the GFS master. Reducing the meat data stored in the master will also lead to fewer requests that need to be made to the master.

The disadvantages of using larger chunks in GFS often lead to more wasted space. Have a single point of failure. This could lead to a longer recovery time.

2. ***Explain briefly the similarities and differences between document-based and column-oriented databases. What is the difference between storing data in the row model and a column model? When is better to use the column model?***

Document-based databases store their data as documents which in many cases will be in either JSON or BSON format. MongoDB is an example of a document-based database. They often allow for advanced queries.

Column-oriented database is databases where data is associated with a column. Doing this allows for more optimization in storage costs. Cassandra uses column column-oriented database. Storing it as columns allows for queries where we might want to get a SUM or AVG of a column which is a cheap action to perform while it might be much more expensive in a document-based database.

3. ***Assume we want to implement a MapReduce code to multiply a one-dimensional matrix with a two-dimensional matrix. Briefly explain what map and reduce functions should do. (5 points) Reminder. The matrix product of matrices A and B is a third matrix C, where C = AB. If A is of shape m × n and B is of shape n × p, then C is of shape m × p, such that: cij = X k aikbkj 1***

Mapper:

Input:

Each record fed into the mapper would either be a row from matrix A or a column from matrix B. These records can be differentiated using a tag (e.g., 'A' or 'B').

For matrix A (rows):
`(Row number, [Row elements])`

For matrix B (columns):
`(Column number, [Column elements])`

Output:

The mapper should produce intermediate key-value pairs where the key is composite `(i, j)` indicating a cell in the resulting matrix C, and the value is `(Tag, Value)` where Tag is either 'A' or 'B' to indicate the matrix and Value is the corresponding value from matrix A or B.

For each element `a_ik` in matrix A:
`((i, k), ('A', a_ik))`

For each element `b_kj` in matrix B:
`((k, j), ('B', b_kj))`

Reducer:

Input:

All key-value pairs emitted by the mapper with the same composite key `(i, j)` will be grouped.

For a particular `(i, j)`:
`((i, j), [('A', a_i1), ('B', b_1j), ('A', a_i2), ('B', b_2j), ...])`

Output:

The reducer should compute the dot product for each group and produce the resulting matrix C.

For each `(i, j)`:
`(i, j, ∑ a_ik * b_kj)`

Here's a more concrete example:

Assuming we are multiplying a one-dimensional matrix A with 2 rows and a two-dimensional matrix B with 2 columns, the mapper and reducer operations would be as follows:

Mapper:

Input:

A: `(1, [a_11, a_12])`
B: `(1, [b_11, b_21])`

Output:

`((1, 1), ('A', a_11))`
`((1, 2), ('A', a_12))`
`((1, 1), ('B', b_11))`
`((2, 1), ('B', b_21))`

Reducer:

Input:

For `(1, 1)`:
`((1, 1), [('A', a_11), ('B', b_11)])`

Output:

`(1, 1, a_11 * b_11)`

This process continues for all rows of A and columns of B.

4. ***Draw the lineage graph for the following code and mention which connections are narrow and which are wide.***

   val a = sc.parallelize(...)
   val c = sc.parallelize(...)
   val e = sc.parallelize(...)
   val b = a.groupby(...)
   val d = c.map(...)
   val f = d.union(e)
   val g = b.join(f)

Narrow transformations: These transformations do not require data shuffling between partitions. Examples are maps, filters, etc. Narrow transformations maintain a one-to-one relationship between partitions of the parent and child RDDs.

Wide transformations: These transformations require data shuffling between partitions. Examples are groupByKey, reduceByKey, join, etc.

Given the code, let's first draw the lineage graph (textually, since I can't draw actual graphs):

a, c, and e are base RDDs created using parallelize.
b = a.groupby(...) – b is derived from a using group, which is a wide transformation.
d = c.map(...) – d is derived from c using a map, which is a narrow transformation.
f = d.union(e) – f is created by combining d and e using union, which is a narrow transformation.
g = b.join(f) – g is derived from joining b and f, which is a wide transformation.

5. ***Explain how using DataFrame in Spark improves the programming performance compared to RDD. What is the relation between DataFrame and DataSet in Spark?***

Dataframe in terms of performance improved over RDD by utilizing Spark Catalyst optimizer. Dataframe and Dataset both provide a more high-level API which makes them much more user-friendly for developers.

DataSet is a subset of Dataframe that combines both RDDs with Dataframes. The dataset supports strong typing.

6. ***Explain briefly how spark join two tables, if***
   (a) both tables are so big that none of them can be loaded into the memory of one computer
   (b) one table is big and the other one is small, such that only the small one can be loaded into the memory of one computer.

a) Reduce join, when the tables can't be loaded into the memory they are joined after the shuffle and sort.

b) Map-side join, when a table can fit into memory using map-side join is much more efficient which results in that there is no need for a reduced stage.

7. ***How can a batch system be used to process streams, and how can a streaming system be used to process batches?***

A Batch processing system is a system that groups tasks (jobs) and executes them most of the time during low computation periods. This means that if we want to use a batch system to process streams of data we only need to batch them together and execute them at a later time and process them.

A streaming system and vice versa can be used to process batches in the same sense that instead of waiting and processing them later they will be processed right away. In terms of right away, their still might be windowing, etc. being used on the streaming system.

8. ***Explain how Kafka provides scalability and fault tolerance.***

Kafka is a messaging system. Kafka is built on the architecture of using brokers (handle requests). Brokers are divided into multiple machines. It also utilizes partitions by dividing up partitions of different machines to distribute computing power. Each partition can read and write.

Each partition is replicated on multiple brokers. One partition acts as a leader.

9. ***Compare the graph processing models in Pregel, GraphLab, and X-Stream.***

1. Pregel:
Model: Pregel was developed by Google and is inspired by the Bulk Synchronous Parallel (BSP) model. In Pregel, computation is vertex-centric.

Execution: During each superstep, each vertex can receive messages sent in the last superstep, perform computation, and send messages to other vertices for the next superstep.

Fault Tolerance: Pregel provides fault tolerance by periodically checkpointing the state of the computation. If a worker fails, computation is rolled back to the last checkpoint.

Communication: Vertices communicate by sending messages to other vertices.

Scalability: Pregel scales to large graphs and can handle billions of vertices.

2. GraphLab:
Model: GraphLab was designed to address some of Pregel's limitations, particularly around its use of the BSP model which can be inefficient for some algorithms. GraphLab introduces a more flexible consistency model.

Execution: In GraphLab, the computation is also vertex-centric, but instead of super steps, it has an asynchronous execution model that allows vertices to compute simultaneously without waiting for global synchronization.

Data Scope: GraphLab provides three consistency models: full consistency, edge consistency, and vertex consistency. This allows algorithms to choose the appropriate consistency model based on their needs.

Communication: Similar to Pregel, vertices in GraphLab communicate by exchanging messages. However, GraphLab also supports shared-memory execution, where neighboring vertices on the same machine can read and write to shared variables without explicit messaging.

Dynamic Graphs: GraphLab supports dynamic graph algorithms, meaning the graph structure can be changed during the computation.

3. X-Stream:
Model: X-Stream is an edge-centric model. Instead of focusing on vertices, the computations are centered around edges, making it different from both Pregel and GraphLab.

Execution: X-Stream uses an iterative streaming model where each iteration processes all edges exactly once. During this processing, X-Stream reads and updates edge and vertex data.

Storage: Instead of storing the graph in main memory, X-Stream streams the graph from disk. This makes it possible to handle extremely large graphs that do not fit into main memory.

Communication: Since X-Stream is edge-centric, messages are not exchanged between vertices. Instead, updates are performed on vertices when their edges are processed.

Scalability: X-Stream can handle very large graphs because of its streaming approach. However, it may require multiple passes over the data, which can be IO intensive.

Summary:
Pregel is vertex-centric and uses the BSP model for computation. It's designed for scalability and fault tolerance.

GraphLab is also vertex-centric but offers more flexibility in its execution model and consistency. It's designed for performance and supports dynamic graph algorithms.

X-Stream is edge-centric and uses a streaming model to handle very large graphs. It's designed for scalability, especially in scenarios where the graph does not fit into the main memory.

10. ***Assume we have two types of resources in the system, i.e., CPU and Memory. In total, we have 10 CPUs and 20GB RAM. There are two users in the system. User 1 needs h1CP U, 4GBi per task, and user 2 needs h2CP U, 1GBi per task.***

How do you share the resources fairly among these two users, considering asset fairness and DRF? (5 points)

…

Exam 2018

1. ***True/False questions. Please briefly explain the reasoning behind your choice (8 points)***
   (a) With MapReduce, each of the R reducers is responsible for producing 1/R th of the amount of output data (true/false, why?)

False, This is due to the data divided to the different reducers not being balanced where certain reducers have more data to process.

(b) GFS (HDFS) is used to store input, intermediate, and output files for MapReduce jobs (true/false, why?)

True, this is because of how MapReduce works during the different stages and processes data is locally written and not constantly saved in memory.

(c) Assume we have a Dynamo storage with a hash function H that produces IDs between 0 and 32 to store and locate objects on the servers. If we have five servers with IDs 0, 3, 8, 18, 23, and H(X) = 26, then object X will be stored on the server with ID 23 (true/false, why?)

False, based on how hash rings work the H(X) = 26 will clockwise find the next available node (server) which in this case will be 0.

(d) BigTable provides fault tolerance by replicating data on multiple tablet servers (true/false, why?)

True, BigTable utilizes GFS for fault tolerance where each tablet is stored in GFS which has replica servers for fault tolerance.

2. ***MapReduce question (8 points) Suppose that you are given two documents with the following content:***
   • Document1: Hello world Hello Hadoop
   • Document2: Hello Spark

   We want to generate a list of locations (i.e., word number in the document and identifier for the document) for each word occurrence. The output generated by your program should look like this:
   Hello → Document1 : 1, 3 | Document2 : 1
   world → Document1 : 2
   Hadoop → Document1 : 4

Spark → Document2 : 2
Write out in pseudo-code the steps taken in Hadoop's map and reduce phases to generate the above output. Please also specify the input and output of the map() and reduce() functions. Assume the identifier for each document is provided as the key to the map() function.

The Mapper part of the MapReduce function needs to keep track of the key which is the word and the value which is the document name and the position of the word. For example the document "Document1: Hello world Hello Hadoop" will result in a mapping of:

("Hello", ("Document1", 1), ("world", ("Document1", 2), ("Hello", ("Document1", 3), ("Hadoop", ("Document1", 4), etc.

The program then will handle the shuffling and sorting part of MapReduce.

The Reducer will after receiving the different groupings based on the key SUM the first the ones that are in the same document and the ones that are in different documents. Doing this for example "Hello" will generate "Hello → Document1: 1, 3 | Document2: 1".

```
Documents = {name: content}

Function map(docID, docContent):
  words = split(docContent by space)
  for i in 1 to length(words):
    word = words[i]
    emit(word, (docID, i))

Function reduce(word, list<docID, position>):
  result = ""
  docDict = {} // Dictionary to group positions by document ID

  for each (docID, position) in list:
    if docID not in docDict:
      docDict[docID] = []
    docDict[docID].append(position)

  for docID, positions in docDict:
    sortedPositions = sort(positions) // Sort the positions
    result += docID + " : " + join(sortedPositions, ", ") + " | "

  emit(word, substring(result, 0, length(result) - 3))
```

3. ***Spark questions***
(a) Draw the lineage graph for the following code and mention which connections are narrow and which are wide.
val a = sc.parallelize(...)
val c = sc.parallelize(...)
val e = sc.parallelize(...)

val b = a.groupby(...)
val d = c.map(...)
val f = d.union(e)
val g = b.join(f)

Join, groupBy is wide and map, the union is narrow

(b) Explain briefly how spark joins two tables if
i. Both tables are so big that none of them can be loaded into the memory of one computer
ii. One table is big and the other one is small, such that only the small one can be loaded into the memory of one computer.

(i) Both tables are so big that none of them can be loaded into the memory of one computer:
Reduce-side join:

In this case, when both tables are large and can't fit into memory, a reduce-side join is used. Here's how it works:

Shuffle & Partition: Both tables are shuffled such that rows with the same key from both tables end up on the same machine. This step ensures that each machine has all the data required to perform a local join.

Sort: Data on each machine is sorted by key. This makes the join process more efficient.

Local Join: The join operation is performed locally on each machine. Each machine will output joined rows where keys match.

Aggregate: If needed, results from all machines are combined together.

Reduce-side join requires shuffling data across the network, which can be costly in terms of performance, but it's the only choice when both tables are too large to fit into memory.

(ii) One table is big and the other one is small, such that only the small one can be loaded into the memory of one computer:
Map-side joins (Also known as Broadcast Join in Spark):

In situations where one table is significantly smaller than the other, a map-side join is often preferred.

Broadcast: The smaller table is broadcasted to every node in the Spark cluster. This means each machine will have a complete copy of the smaller table in its memory.

Local Join: Each partition of the larger table is then joined locally with the broadcasted smaller table. Since the smaller table is in memory, this join operation is very efficient.

Map-side join skips the shuffling step entirely, making it faster for cases where one table is small. However, it's essential to ensure the smaller table can indeed fit into the memory of each worker node.

4. ***Streaming questions (8 points)***

   (a) Assume you want to use Storm to implement the word count application. Explain what spouts and bolts you need, and what types of grouping you use between them.

   (b) Briefly compare the fault tolerance model of Storm, Spark Streaming, and Flink.

5. ***Graph questions (8 points)***

   (a) What is the difference between message passing and shared memory models in graph processing platforms?

Messages passing:

- A Distributed model where each node exists on different machines
- Each node has its own memory and processing power
- Communicate by sending and receiving messages

Shared Memory:

- All nodes in a unified memory space
- Threads can access any part of the graph without sending and receiving messages

   (b) Assume we have a graph, in which all vertices have a local numeric value. Write a vertex-centric Gather-Apply-Scatter pseudo-code to update the local value of the vertices with the minimum value in the graph. For example, if a graph has three vertices A, B, and C, with values 4, 2, and 5, respectively, we would like to end up with value 2 at all vertices.