Big data: sample exam questions

16th January 2017

Contents

1 Databases for Big Data

1.1 NoSQL

Question 1.1.1. Explain the main reasons for why NoSQL data stores appeared.

Answer 1.1.1. Some reasons:

- RDBS are not good at being distributed (e.g., are not partition tolerant).
- Amounts of data a huge and it is not possible to cope with it without distribution.
- Data structure often changes or data is unstructured, but frequent schema changes are difficult to implement.

Question 1.1.2. List and describe the main characteristics of NoSQL data stores (any database or file is a series of bytes that, once stored, is called a data store).

Answer 1.1.2. Some characteristics:

- Schema-free.
- Unstructured or semi-structured.
- Usually distributed and network partitioned.

Question 1.1.3. Explain the difference between ACID and BASE properties.

Answer 1.1.3. RDBS focus on ACID properties: atomicity (if operation has started it will be finished or completely or not finished at all), consistency (all users see the same information), isolation (operations do not influence each other), durability (once a user has received confirmation that operation has completed, its result will be there, even the power is cut off the next nanosecond). NoSQL focus on BASE: basically available (system will give responses, but without guarantee that this is the latest response), soft state (system state might change even without external input due to eventual consistency model), eventually consistent (without external input the system will be consistent at some point in the future).

In terms of CAP theorem, ACID properties guarantee availability and consistency, while BASE properties guarantee availability and partition tolerance.

Useful links: explanation of BASE, wiki about CAP, wiki about ACID.

Question 1.1.4. Discuss the trade-off between consistency and availability in a distribute data store setting.

Answer 1.1.4. Consistency means that all users receive the same output given the same input (at the same point in time); availability means that we will always receive a (non-error) output. Assuming that we keep partition tolerance (system always operates despite partial failures) constant, we have to trade-off in the former two properties. Intuitively, this means that when something fails, we can either return error message, because we are not sure what is in the failed parts (consistency), or we can always return something (availability), but without a guarantee that it is the latest information (eventual consistency).

Question 1.1.5. Discuss different consistency models and why they are needed.

Answer 1.1.5. Different models:

- Strong consistency: at all points in time, the system is consistent. This is needed, if we must know that the returned result is consistent. E.g. in financial transactions: we cannot allow a customer buy something when his account is already zero, but the last query (because it did not have the latest information) returned the non-zero result.
- Weak consistency:
  - Window consistent: system might be inconsistent for some period of time, but then becomes consistent. Needed if need to know that the result is consistent, but latency is not an issue. E.g. probably in similar situations as eventually consistent.
  - Eventually consistent: without new external input, system will be consistent at some point in the future. Is needed when consistency is not an issue and other properties of data bases are preferred, for instance, availability. E.g. when a customer clicked to add something to a shopping cart, we want to save it, and we do not really care that it is added the milisecond after, but we want to eventually have the proper shopping cart (when the customer checks out).

Question 1.1.6. Explain how consistency between replicas is achieved in a distributed data store.

Answer 1.1.6. Via vector clocks.

Question 1.1.7. Explain the CAP theorem.

Answer 1.1.7. Cap stands for consistency, availability, and partition tolerance. The theorem states that a distributed system can guarantee only 2 of those.

Question 1.1.8. Explain the differences between vertical and horizontal scalability.

Answer 1.1.8. Vertical scaling is adding more powerful resources (larger hard drive, faster processor, etc). Thus a system is vertically scalable if it can perform reasonably better after these additions. Horizontal scaling is adding more instances of the same type (more nodes in a cluster, more processors in a node). Thus a system is horizontally scalable if it can perform reasonably better after these additions.

Question 1.1.9. Explain how consistent hashing works and what are the problems it addresses.

Answer 1.1.9. Have no idea. Please, someone explain it to me.

Question 1.1.10. Explain how vector clocks work and what are the problems they address.

Answer 1.1.10. Vector clocks attempts to recognize the order of events in a distributed system.

Each node has an ID. Vector consists of all IDs in the system and the respective counters. Whenever a node receives internal operation it increases its own counter. Whenever it receives a message from another node, it increases its counter and updates all other counters to max(its own, received). This vector is attached to every message.

If a node receives a message with a vector all elements of which are lower, then this is an old message (ignore it). If all elements are greater, it is a new message (act on it). If some are greater, some are lower—detect a conflict and attempt to resolve it (or just cry in the corner, if it is not possible to resolve it automatically).

Question 1.1.11. List and describe dimensions that can be used to classify NoSQL data stores.

Answer 1.1.11. Dimensions:

- 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.

Question 1.1.12. List and describe the main characteristics and applications of NoSQL data stores according to their data models.

Answer 1.1.12. Models:

- Key-value stores:
  - Good: storing distributed data.
  - Bad: structures and relationships, accessing multiple items (since the access is by key and often no transactional capabilities)
  - Applications: storing web session information, user profiles and configurations, shopping cart data, caching layer.
- Column-family stores:
  - Good: storing distributed data, plus are better than key-value stores for querying and indexing.
  - Bad: structures and relations, highly dynamic queries.
  - Applications: document stores applications, analytic (web, personalized searches, inbox searches).
- Documents stores:
  - Good: storing something that needs to be representable as a document.
  - Bad: ???.
  - Applications: items with similar nature but different structure, blogging platforms, event logging, fast application development.
- Graph databases:
  - Good: for very interconnected data since they are efficient in traversing relationships.
  - Bad: for non-graph applications, bad at horizontal-scaling.
  - Applications: location-based services, recommendation engines, complex network-based applications, memory leak detections.

1.2 HDFS

Question 1.2.1. Explain what HDFS is and for what types of applications it is (not) good for.

Answer 1.2.1. HDFS is a distributes file system that runs on top of the native file system. Files are store in very large blocks—64/128 MB. It is also fault tolerant due to replication of each block on multiple nodes.

It is good for large files, streaming access, commodity hardware. But bad for small files, because it keeps location of all files in memory (which can run out); low latency data, multiple writers and arbitrary data modifications (can append files only).

Question 1.2.2. Explain the organization of HDFS.

Answer 1.2.2. HDFS consists of

- Namenode (master). It manages the file system name-space and metadata, store location of all blocks for a given file.
- Datanodes. They store and retrieve blocks, send heartbeat to namenode.

- Secondary namenode in case the master node fails. Note that it is node a backup node, but just a checkpoint.

Question 1.2.3. Explain the process of reading and writing to HDFS.

Answer 1.2.3. Writing:

1. Client asks namenode where to write files.
2. The namenode creates files and tells the locations to the client.
3. Client sends the file to one of the name nodes together with the information about 2 other nodes where to write it.
4. The first name nodes sends it to the second, etc.
5. When the desired number of replications have been written successfully, namenode informs the client that the write operation was successful.
6. Client informs namenode that the write operation was successful.

Reading:

1. Client asks namenode where the files are written.
2. Then accesses the namenodes and receives the file.
3. After the read operation is successful, client closes the file.

Question 1.2.4. Explain how high availability is achieved in HDFS.

Answer 1.2.4. All blocks are replicated (3 times by default) and information about their location is stored in the memory (for lower latency) of the namenode. If one of the replicas is corrupt, does not answer, we still have 2 more.

Thus, the only single point of failure is namenode, which has a secondary namenode in case it fails. The secondary namenode is not a backup node, and therefore, the information on it might be the most recent. Hadoop version 2+ allow for High Availability Namenode (which always runs and becomes a proper namenode as soon as the main one fails).

1.3 Dynamo

Question 1.3.1. Explain the data model and list main applications of Dynamo.

Answer 1.3.1.

Question 1.3.2. Explain the Dynamo design considerations and what are the advantages of Dynamo in comparison to RDBMSs.

Answer 1.3.2.

Question 1.3.3. Explain how basic NoSQL techniques are applied in Dynamo.

Answer 1.3.3.

Question 1.3.4. Explain versioning and semantic reconciliation in Dynamo.

Answer 1.3.4.

1.4 HBase

Question 1.4.1. Explain the difference between column-oriented and row-oriented storage.

Answer 1.4.1.

Question 1.4.2. Explain the data model of HBase.

Answer 1.4.2.

Question 1.4.3. Give example applications of HBase.

Answer 1.4.3.

1.5 Hive and Shark/SparkSQL

Question 1.5.1. Explain the problem that Hive and Shark/SparkSQL address.

Answer 1.5.1.

Question 1.5.2. Explain the data model of Hive and Shark/SparkSQL.

Answer 1.5.2.

Question 1.5.3. Discuss the trade-off between schema-on-read and schema-on-write approaches.

Answer 1.5.3.

Question 1.5.4. Explain the difference between OLAP and OLTP.

Answer 1.5.4.

Question 1.5.5. Explain the main differences between Hive and Shark and what are the advantages they lead to.

Answer 1.5.5.

Question 1.5.6. Explain how fault tolerance is achieved in Shark/SparkSQL.

Answer 1.5.6.

2 Parallel Computing

Question 2.0.1. 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 P_i to node P_j).
- Temporal data locality.
- Dynamic task scheduling.

Answer 2.0.1. Definitions:

- Cluster: cluster is a set of connected computers that work together and perform the same task (source).
- Parallel work: number of instructions performed by all processors, where in each step as many processors are available as needed to execute the step in constant time.
- Parallel speed-up: the ratio of execution time of an algorithm by a single to the execution time by p cores.
- Communication latency: time that is needed to send a minimal message of 1 bit message from one node to another. Highly depends on of *start-up* and *per-hop* time (source)..
- Temporal data locality: re-accessing the same data multiple times within a short period of time and execution of the same task.
- Dynamic task scheduling: allocating resources (processors) at run time based on their availability.

Question 2.0.2. Explain the following parallel algorithmic paradigm: Parallel Divide-and-Conquer.

Answer 2.0.2. The (sequential) divide-and-conquer paradigm consists of the following steps:

- **Divide:** the problem is trivial, solve it. If not, then recursively split it into independent problems of the same type and check whether they became trivial.
- **Combine** solutions into a common solutions (recursively, thus, combine solutions of solutions until we get *the* solution).

In parallel version we parallelise the recursion and solving of trivial independent tasks. Also try to parallelise the combining phase.

Question 2.0.3. Discuss the performance effects of using large vs. small packet sizes in streaming.

Answer 2.0.3. Streaming attempts overlap the processing of each packet of data in time with access of subsequent units of data and/or processing of preceding packets of data. If the packets can be processed independently, then large packets should be better, because the sum of latencies is reduced. If the packet processing depends on the result from the previous packet, then small packets should perform better, because accessing the data element can be performed while the computation of the previous element is still in progress, and with smaller packets there will be more such overlaps.

Question 2.0.4. 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?

Answer 2.0.4. In I/O intensive tasks actual I/O output is the bottleneck. Therefore, CPU is underutilised. To decreases its underutilisation, we should assign more tasks for each CPU.

Question 2.0.5. 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).

Answer 2.0.5. We will need to use the **reduce** skeleton.

```
y = reduce(data = x, function = max)
```

Thus, we have created a reducer which would compare all elements in the $\boldsymbol{x}$ array and return the max of them.

Question 2.0.6. Describe the advantages/strengths and the drawbacks/limitations of high-level parallel programming using algorithmic skeletons.

Answer 2.0.6. Advantages:

- Abstract and easy to use.
- Does not require special knowledge about parallelisation.
- Easier to analyse and transform.

Drawbacks:

- Can be less efficient than manual parallelisation.
- Available skeletons does not always work.

Question 2.0.7. Derive Amdahl's Law and give its interpretation.

Answer 2.0.7.

$$\begin{aligned} S(p) = \frac{T(1)}{T(p)} &= \frac{w_A}{w_{A^s} + \frac{w_{A^p}}{p}} = \frac{w_A}{\beta w_A + (1-\beta)\frac{w_A}{p}} = \frac{1}{\beta(1-\beta)\frac{1}{p}} = \frac{p}{\beta p + (1-\beta)} \\ &= \frac{p}{\beta p + \underbrace{1-\beta}_{\geq 0}} \leq \frac{\cancel{p}}{\beta\cancel{p}} = \frac{1}{\beta}. \end{aligned} \tag{1}$$

Question 2.0.8. What is the difference between relative and absolute parallel speed-up? Which of these is expected to be higher?

Answer 2.0.8. Relative speed-up is speed-up compared to the same algorithm but using 1 core; absolute speed-up is the speed-up compared to the best sequential algorithm. Since best sequential algorithm will always perform on one core not worse (or even better) than a parallel algorithm on one core, relative speed-up is larger or equal to the absolute speed-up.

Question 2.0.9. 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?

Answer 2.0.9. It represents multiple cores. It abstracts from communication costs, limited number of processors, scheduling overhead, and also assumes that local computations take 1 unit of time and memory access also takes 1 unit of time.

Question 2.0.10. Which property of streaming computations makes it possible to overlap computation with data transfer?

Answer 2.0.10. Streaming computations and data transfer are independent tasks.

3 MapReduce

Question 3.0.1. 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)

Answer 3.0.1.

$$M = \frac{12.8 \times 10^1 2}{64 \times 10^6} = \frac{128 \times 10^1 1}{64 \times 10^6} = 2 \times 10^5 = 200\,000 \tag{2}$$

Question 3.0.2. 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?

Answer 3.0.2. Using the whole MapReduce construct, shuffling stage can start earlier than all maps have finished execution, this allows to save time. If reducer is commutative and associative, then reducer also can start execution earlier. Moreover, the whole point of the MapReduce framework is to use as abstract function as possible, while specifying each of the constructs would be a lower (still quite high-level though) programming.

Question 3.0.3. Reformulate the wordcount example program to use no Combiner.

Answer 3.0.3. Removing combiner function everything should still work. Mapper produces a (key, 1) pair, which can still be interpreted by the reducer correctly. It would just be slower.

Question 3.0.4. 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.

Answer 3.0.4. Values with same keys might be on different nodes, therefore the result will not change only if the function is associative and commutative. Example of associative and commutative: `add(a, b)`, because: $a + (b + c) = c + (a + b)$. Example of a function that is not associative and commutative: `subtract(a, b)`, because $a - (b - c) \neq c - (a - b)$.

Question 3.0.5. Extend the wordcount program to discard words shorter than 4 characters.

Answer 3.0.5. In combiner:

```
for w in wordcounts.keys():
  if wordcounts[w] <= 4:
    print '%s\t%s' % (w, wordcounts[w])
```

In reducer:

```
if current_word:
  if current_count <= 4:
    print '%s\t%s' % (current_word, current_count)
```

Question 3.0.6. Write a wordcount program to only count all words of odd and of even length. (There are several possibilities.)

Answer 3.0.6. In mapper:

```
for word in words:
  if len(word) % 2 == 1: # or 0, if count even length words only
    print '%s\t%s' % (word, 1)
```

Question 3.0.7. Show how to calculate a database join with MapReduce.

Answer 3.0.7. Probably not the most effective way, but we could have a mapper than returns key-value pairs for each row in the database and key is the column(s) on which to join. while value consists of all other columns of interest **and** a variable that indicates whether this is a left of right table. Then in the reducer stage, we would return all possible combinations of values of left and right column for each key.

Question 3.0.8. 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?

Answer 3.0.8. Assign the task to another worker, and when one of them finishes computations, stop the other one.

4 Spark

Question 4.0.1. Why can MapReduce emulate any distributed computation?

Answer 4.0.1. Any distributed computations consists of local tasks and exchange of information. Local tasks can be done by mapper, exchange can be done via reducer (source).

Question 4.0.2. For a Spark program consisting of 2 subsequent Map computations, show how Spark execution differs from Hadoop/Mapreduce execution.

Answer 4.0.2. MapReduce would write the result to a file, then distribute it again in the second mapper, while Spark would keep this intermediate result in memory.

Question 4.0.3. Given is a text file containing integer numbers. Write a Spark program that adds them up.

Answer 4.0.3. Assume that all numbers are in one line, then

```
sum = (sc
  .textFile("some_input_file")
    .split()
    .flatMap(lambda a: a)
    .reduce(lambda a, b: (a, a[1] + b[1])))
print(sum)
```

Question 4.0.4. Write a wordcount program for Spark. (Solution proposal: see last slide in lecture 8.)

Answer 4.0.4.

```
# Read file
lines = sc.textFile("some_input_file").flatMap(lambda line: line.split("␣"))
words = lines.split().map(lambda w: (w, 1))
wordcount = words.reduceByKey(lambda a, b: a + b)
```

Question 4.0.5. Modify the wordcount program by only considering words with at least 4 characters.

Answer 4.0.5.

```
# Read file
lines = sc.textFile("some_input_file").flatMap(lambda line: line.split("␣"))
words = lines.split().map(lambda w: (w, (1, len(w)))
longwords = words.filter(lambda a: a[1][1] >= 4)
wordcount = longwords.reduceByKey(lambda a, b: a + b)
```

5 Cluster resource management

Question 5.0.1. 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.

Answer 5.0.1. For example, mapper tasks usually utilise a lot of resources, while reducer tasks, usually underutilise resources of a cluster. However, both of them are the same job. Therefore, by dynamically allocating resources over the job's lifetime, we can lease free resources when we are in underutilisation states, e.g., reducer tasks.

Question 5.0.2. 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.

Answer 5.0.2. Jobs can come from different frameworks and have different priorities. Therefore, framework-specific is not able to decide which of them has a higher priority.