Big Data Analytics
Exam 2020-06-02
Grading Info

Olaf Hartig, Christoph Kessler, Jose M. Pena

Question 1 (1p)

Solution: The claim is wrong. Read scalability can also be achieved by scaling up. For instance, adding more RAM into a server may increase the cache of a DBMS, which then may allow the DBMS to handle an increased number of reads with the same performance with which it handled fewer reads when it had a smaller cache due to fewer RAM. Similarly, upgrading the CPU or replacing a slower hard disk with a faster and bigger one may help to achieve read scalability in a server.

Remarks:

- Some of your answers try to make an argument about fault tolerance. However, the question has nothing to do with fault tolerance.
- The question is also not about whether read scalability is "easier" or "better" to achieve by scaling horizontally instead of vertically. Of course, in contrast to scaling horizontally (scale out), there is a limit to scaling up a single server. Nonetheless, this does not mean that we cannot scale up a server at all in order to achieve read scalability (or data scalability or write scalability).
- If you didn't explicitly say that you think the claim is wrong (or correct), you cannot get the full point.

Question 2 (1p)

Example Solution 1: One possibility would be to have one document per report and have the data about the corresponding projects nested within these documents. Here is what this may look like for the example data:

```
291 →   { id: 121
          projects: [    { name: "Bee"
                             budget: 1,300,000
                             final report: 121 }
                     ]
          pages: 70
          location: "http://acme.com/beerep"
        }

297 →   { id: 391
          projects: [    { name: "UsMis"
                             budget: 1,000,000
                             final report: 391 },
                         { name: "AMee3"
                             budget: 3,700,000
                             final report: 391 }
                     ]
          pages: 350
          location: "http://acme.com/r391"
        }
```

```
299 →  { id: 699
         projects: [ ]
         pages: 100
         location: "http://acme.com/Other"
       }
```

Note that I have chosen some arbitrary numbers for the document IDs (291, 297, and 299). In practice, these IDs would typically be generated by the document store.

Example Solution 2: Another possibility would be to have two separate document collections: one with a (non-nested) document for each report and the other one with a (non-nested) document for each project. Here is what this may look like for the example data:

```
collection: "reports"

    291 →  { id: 121
             pages: 70
             location: "http://acme.com/beerep" }

    297 →  { id: 391
             pages: 350
             location: "http://acme.com/r391" }

    299 →  { id: 699
             pages: 100
             location: "http://acme.com/Other" }
```

```
collection "projects"

    291 →  { name: "Bee"
             budget: 1,300,000
             final report: 121 }

    297 →  { name: "UsMis"
             budget: 1,000,000
             final report: 391 }

    299 →  { name: "AMee3"
             budget: 3,700,000
             final report: 391  }
```

Remarks:

- Some of you have multiple key-value pairs with the same key within the same document. That’s not possible; the keys (field names) are unique within a document.
- Some of you have documents that primarily represent the projects (i.e., one document per project) and, then, nested within these documents there is data about the corresponding reports (i.e., in some sense the opposite of my first example solution). While this is not a problem per se, you should not forget that there is a report without a project, and the data about that report must be captured as well.
- The document IDs must be unique (at least, within a document collection).

Question 3 (1p)

Example Solution:

1. In the key-value database model, users can choose what the keys are, whereas the document identifiers in the document model are typically system-generated.
2. The key-value pairs in the key-value database model cannot be grouped whereas, in a document database, we can group key-value pairs into separate documents; moreover, some forms of document databases allow us to even group these documents further, namely into so called collections or domains.

Example Solution 2:

1. While the documents in a document database have an internal structure that is clearly defined (and, thus, can be operated on by the DBMS; for instance, to create indexes), the same is not the case for the values in a key-value database where any possible internal structure of such values is opaque from a DBMS perspective.
2. In the key-value model, access to multiple database entries (key-value pairs, in this case) requires separate requests. In the document model, on the other hand, multiple database entries (documents, in this case) may be retrieved in a single request.

Remarks:

- The question asks explicitly about differences. Hence, the question cannot be answered by writing two separate paragraphs with one of them describing what a key-value database is and, totally separate from that, the other one describing what a document database is. In contrast, I wanted to see two points where each of these points *compares* the two database models with one another in terms of some feature or aspect of these models (see the example solutions).
- Some of you wrote that key-value databases are not schema-free. That's wrong; we can choose the keys arbitrarily, and also the values do not need to all have the same, predefined internal structure.
- Some of you wrote that a difference is that key-value databases have key-value pairs whereas document databases have fields that consist of a (unique) name and a value. That's not actually a difference. It is just slightly different terminology but, from a conceptual perspective, the notion of a field in the context of document databases is essentially the same as the notion of a key-value pair.
- The fact that queries over document databases can be expressed via a programming API or in a query language is not a distinguishing feature. Key-value stores may also have a programming API to invoke the CRUD operations and, on the other hand, the CRUD operations may also be considered to be expressions of some form of a query language.
- It is not correct to say that the values in a key-value database are not using data types or are not structured. They may very well be! The thing is just that the internal structure or the data types used for the values is something that is not know to the DBMS (but to the application).
- Speculations about which type of database can make things "faster" or "more distributed", or can become "bigger", or whatever, are irrelevant for this question.
- Similarly, the question was not what these types of databases can be used for.

Question 4 (1p)

Solution: The claim is wrong. While it is true that, when considering each database object separately, there is only one compute node that is the master for that database object, it is not necessarily true that one node is the master for *all* the database objects. If that was be the case, then this node would have to have a complete copy of the whole database, and every write operation would have to be handled by that one node, which would defy the purpose of using a distributed system.

Remarks:

- This question was not about distributed computation (MapReduce, etc) but about distributed database management where we have multiple replicas of each database object..

Question 5 (1p)

Solution: All four nodes have to confirm the writes in this case. To achieve strong consistency, we need to have W+R > N. In this case, N=4 and R=1. Therefore, we need W=4.

Question 6 (1p)

Correct answers: (i) high overhead for communication of a fixed amount of data in many messages due to many small messages (linear model for time of communicating a block of N elements, from the lecture), (ii) high relative tasking overhead due to many light-weight tasks, (iii) large number of entries in the HDFS name node, making querying the name node for the block locations a performance bottleneck.

Grading: 1p if at least two of the three reasons above were given with proper explanations. 0.5p for one only. Partial points may be given only, depending on the quality of explanations. A correct keyword without explanation gives no points.

Remark: Many of you wrote that the name node would run out of memory – for 1K blocks this would typically not be a problem yet, virtual memory could be large, and long before that limit would be reached the name node will be swamped with lookup requests from the workers so it becomes the performance bottleneck in a large system. Admittedly, individual key-value pairs as blocks would mean that the name node directory is about of the same size as the distributed data operated on; for that reason I still gave the 0.5p for this reason, provided that a reasonable explanation was given. Remark: It is not correct that smaller blocks lead to a larger number of nodes required or used in a cluster. Instead, it leads to a larger number of mapper tasks and thus higher tasking overhead.

Question 7 (1p)

Correct answer:

(a) record reader (mapper task reads a block from HDFS),
output formatter (reducer task may write to HDFS).

(b) partitioner (mapper task writes file(s) with key-value pairs to local disk),
shuffle-and-sort (in the s&s phase, a reducer task first downloads relevant files from the disks where the mappers' partitioner had written them).

Grading: Quite a number of students did not read this question carefully and answered this question as if it was one from an old exam that looked similar. Not or wrongly referring to HDFS resp. to (a) and (b) gives no points. Moreover, only partial or no points may be given depending on the quality of the purpose explanations.

Question 8 (1.5p)

Correct answer: The right answer can be found on the slides. The difference in dependence structure should be mentioned (elementwise vs. global), the difference in producing an RDD or not, as well as the difference in evaluation (lazily vs. immediately).

Grading: 0.5p per correct description of a transformation resp. action characteristics as given above, 0.25p per correct example. Deduction of points depending on the quality of explanations.

Remark: This was an easy question, known from previous exams, and almost all had reasonable answers to it. Many however forgot to explain the difference in dependence patterns (although my lecture and lecture notes emphasized it), which led to a 0.5p loss.

Question 9 (1.5p)

Correct answer: Computations consisting of sequences of maps / transformations (esp. iterative computations, such as in ML training by gradient search), so the execution order within the sequence can be adapted (a run-time tiling optimization that is enabled by lazy execution and the element-wise dependence pattern of transformations) such that RDD data can be forwarded across map chains in memory instead of in a global (HDFS) file, even for very large data sizes.

Grading: Deduction of points may be applied depending on the quality of the explanation. For a right buzzword (“iterative”, “sequences”) but insufficient explanation, only 0.5p are given.

Remark: Sequences of full mapreduce where each reduce phase actually performs a reduction (global dependence pattern) do not qualify here – a reduction operation terminates a RDD lineage.

Remark: Skipping replication of data blocks to write for fault tolerance in the normal (fault-free) case can also contribute to the better performance of Spark; partial points are given here if the corresponding structure of computations is mentioned and the reason is explained properly.

Remark: Caching of the same RDD value with .cache() for multiple subsequent reads is only a rather minor benefit in Spark over MapReduce. The important one is the normal use case of Spark programs i.e. chains of transformations with single write / single read access patterns, which Spark speeds up by in-memory forwarding as described above. For the effort, I have given up to 0.5p for well-explaining answers mostly focusing on .cache().

Remark: Quite a number of answers claimed that MapReduce could handle much larger inputs than Spark. This is not true in principle (albeit individual implementations may have size limitations), they basically can provide the same functionality, only do it in different ways/order. Note that, when forwarding RDD operand values in memory between transformations, Spark does not need to hold all values of an RDD in memory at the same time. Because of the elementwise dependences of the transformations, Spark is free to compute them one-by-one in smaller blocks (this is called strip-mining or tiling) that fit the sizes of the upper levels of the memory hierarchy.

Question 10 (4p)

Example Solution:

```
tr = sc.textFile("trainingdata.csv").cache() # Important because I re-use the RDD.
te = sc.textFile("testdata.csv").cache()

# training
n = tr.map(lambda x: (x[0])).reduce(sum) # I assume that x[0] is the target value.
d = tr.count()
r = n/d

# testing
n = te.map(lambda x: abs(x[0] - r)).reduce(sum)
d = te.count()
e = n/d
```

Question 11 (1p)

Example Solution 1: An example of an application for which it would be important to be able to handle increasing amounts of data without loosing performance (i.e., data scalability) is a user analysis and product recommendation application in an online shopping portal. The reason is that the number of users of the portal may increase, and so may the number of products or, for instance, the number of product reviews written by users. As a consequence, the amount of data that would have to be analyzed by the application may also increase. Of course, it would be undesirable if the increased amount of data has a negative impact on the performance of the application (or even of the shopping portal as a whole).

Remarks:

- If your description is primarily about handling user requests, that's not correct as an answer to the question because this is more related to read scalability rather than data scalability.
- Similarly, if your description is primarily about writes or "uploads", then this is more related to write scalability.
- Generic descriptions of companies or start-ups that may have to deal with increasing amounts of data is not sufficiently concrete.
- Also, as stated explicitly in the question, it is not enough if your application / use case simply has to do with a huge amount of data. In contrast, it must have to do with *increasing* amounts of data because that's what data scalability is about.
- Talking about scalability in general is not what was asked for in the question. Similarly, any discussion of how data scalability may be achieved is irrelevant, and so is a discussion of horizontal scalability or vertical scalability.

Question 12(a) (1p)

Solution: The only way to query a key-value database is by `get`(*key*) operations. Since they keys in the given database are the user IDs and we cannot assume to know the IDs of the users named Alice, we have to go through *all* the keys of the database. That is, for every userID, we have to request the corresponding value by doing `get`(userID), then we have to look into the value to check whether the name related part of the value is the string "Alice" and, if that's the case, we can add the birth year related part of the value to our result. Notice that processing the values (i.e., checking their name related part and extracting their birth year related part) is something that would have to be implemented in an application program (rather than expressed as queries for the database system).

Remarks:

- Saying that you would simply do get(alice_in_se) and get(selaya) is incorrect because we cannot assume to know which user IDs belong to users whose name is Alice.
- You cannot do something like `get(username="Alice")` or do a *"get with a user name as key"* (as some of you wrote). Similarly, you cannot do something like `get key(birth year)`

Question 12(b) (1p)

Example Solution 1: A possible extension of the given database may consist of two parts: First, we extend the keys of the existing key-value pairs by prefixing them with the string `"UserID:"`. Second, we may add an additional key-value pair for each unique user name such that the key of such a key-value pair is the corresponding user name, prefixed with the string `"UserName:"`, and the value is an array of the user IDs of all users that have this name. For instance, for the given example database, this extension may look as follows.

```
"UserID:alice_in_se" → "Alice, 1987, [bob95 charlie]"

"UserID:bob95" → "Bob, 1995, [charlie]"

"UserID:charlie" → "Charlie, 1996, []"

"UserID:selaya" → "Alice, 1974, [charlie], [alice_in_se selaya]"

"UserName:Alice" → "[alice_in_se selaya]"

"UserName:Bob" → "[bob95]"

"UserName:Charlie" → "[charlie]"
```

Given this extension, we may now do `get("UserName:Alice")`, which would give us the value that contains the array with the user IDs of all users named Alice (i.e., `alice_in_se` and `selaya`, in our current example database). Next, we only need to do one more `get` operation for each of these user IDs, and for the values that we would retrieve by these `get` operations, we can immediately extract the birth year related part and add that to our result.

Remarks:

- Just writing that *"we can add redundancy to the key-value pairs"* is not enough.
- Simply creating additional key-value pairs in which the users' names are the keys is problematic because there may be collisions between user IDs and user names (e.g., what if there is a user with user ID "Alice"?). The example solution addresses this issue by extending all keys with a prefix, which makes it possible to distinguish the resulting two types of keys.
- Extending the keys by adding the names (or the birth dates) into them would not help because we would still have to know the user IDs in this case.
- It was not the intention of this question to have you propose the use of a different database model (e.g., a wide-column database or a document database). In contrast, the question was explicitly about changing or extending the given database (rather than creating a new one).

Question 13 (1p)

Solution: In consistent hashing, the key of every key-value pair is mapped by a hash function to a hash value, and each compute node is assigned a distinct range of possible hash values. Then, such a compute node is responsible for handling the key-value pairs that have a key whose hash value is in the range assigned to the compute node. Now, if a node is removed, the range of hash values that was assigned to this node is merged with the next range. As a consequence, the node that was assigned that next range will become responsible now for also handling all the key-value pairs that the removed node was responsible for.

Remarks:

- It is not enough to simply say that the keys need to be "redistributed" or "remapped."
- Similarly, it is not enough to simply say that the keys have to be "supported"/"copied"/"taken over" by "another node."
- If you only talk about "adjacent nodes" or "neighboring nodes" (i.e., without being explicit that it is the immediate *next* node), then you get only 0.5p.

Question 14 (0.5p)

Solution proposal/grading: I accepted all "yes" answers with reasonable explanations based on execution order and parallelization. In fact, nobody answered "no" here (even though one could argue that, as long as the mapper is only single-threaded, then within one mapper process the commutativity is not really required for a separate combiner because data within the same block is then, for the same key value, always combined in the original order; but in a global view across nodes – i.e., at block borders - it is required; moreover, a commutative combiner gives more flexibility for execution within a mapper task, e.g. allows at least in principle to internally multithread the mapper including the combiner across multiple cores).

Question 15 (0.5 + 1 = 1.5p)

Accepted answers for 15(a): MapReduce is a kind of "super-skeleton" that provides three different functionalities (map, sort, reduce) in a single programming construct (not all of which will be used in each MapReduce instance). Hence the analogy. By iterating multiple MapReduce steps, basically any distributed computation can be emulated.

Accepted answers for 15(b): Spark splits the super-skeleton functionality into separate simpler operations (transformations, actions) which can be composed directly (by forming RDD lineages), and this is an enabling feature for forwarding partial results in memory rather than via distributed files only.

Remarks:

- 15(a) could only be answered if one had paid attention to what I had said in the lecture, and that were apparently not so many; instead many wild guesses were made here. In some cases, traces of the right answer for 15(a) came across in the answer of 15(b). I also gave the 0.5p for answers along the lines of "by iterating MapReduce one can emulate all distributed parallel computations".
- 15(b) was not answered correctly by many students who wrote about the execution model rather than the programming interface asked for. I still gave partial points for reasonable explanations.

Question 16 (1.5p)

Solution: This is a standard question and the answer can be found on the slides. Important that the motivation (see below), the FT by replication of blocks in MapReduce and the FT by recomputing of lost RDDs in Spark based on RDD lineage information were mentioned and explained.

Remark: 0.5p for the motivation, 1p for the comparison. The motivation of fault tolerance was only answered in rather general terms by most answers; I gave the 0.5p nevertheless where the answer made sense, as long as some connection to large-scale distributed computations was made. In a number of cases this subquestion was apparently simply overlooked. For the comparison, up to 1p could be obtained depending on quality and quantity, and most students could say at least something about it.

Question 17 (1.5p)

Solution guideline: The sequential computation needs time 2 C * N because Mapper and Reducer run in sequence and each needs time C * N. Hence, even if the mapper's part of the work (50%) is speeded up to infinity by parallel processing and the reducer is kept sequential, not more than speedup factor 2 could be achieved (calculation required).

Grading: Correct and well commented application of Amdahl's Law gives full points. 0.25p for mentioning Amdahl's Law but stating or using it wrongly or not at all. About 0.5p for incomplete efforts in the right direction, depending on degree of partial correctness.

Question 18 (6p)

Solution:

```
def classifier(x, w): # It classifies x given weights w

   foo = x[0]*w[0]+x[1]*w[1]
   t = 1
   if foo < 0:
      t = -1
   return t

tr = sc.textFile("trainingdata.csv").cache() # Important because I re-use the RDD.

w[0] = 0
w[1] = 0
e[0] = 1
e[1] = 1

while abs(e[0])+abs(e[1]) > 0
   cl = tr.map(lambda x: (x[0], x[1], x[2], classifier((x[1],x[2]),w))) # I assume
that x[0] is the target value.
   e = cl.filter(lambda x: x[0] != x[3]).map(lambda x: (x[1]*x[0],
x[2]*x[0])).reduce(lambda x,y: (x[0]+y[0], x[1]+y[1]))

   w[0] = w[0] + alpha * e[0]
   w[1] = w[1] + alpha * e[1]
```