Data Intensive Computing – Review Questions 6

Adriano Mundo, Riccardo Colella (Group 29)

1. **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?**

   A possible solution may be to use a document-based data model. A document-based database is a system where every field is stored in a well-formatted document containing specific characteristics. Whatever field may be needed can be added to the document. Every student, in this particular example, will have their ID as key and all the other information needed, that can differ from one student to the other, as values. The document may equally be an XML, or a YAML or a JSON file.

2. **When do we need to use `cache` in Spark?**

   The `cache` function is very useful in contributing to better performance for frequently accessed DataFrames or tables. So, `cache` in Spark should be used whenever there's the need to reuse a DataFrame or a table. It is not important to have enough memory to `cache` all the DataFrames, the process will be managed by Spark and it will store as many of the partitions read in memory across executors as memory allows. It is important to `cache` after reading data from source to reuse that data. Finally, in the case of not enough space available, data will be cached to local disk, that's faster than reading again from source where partitions need to be recomputed slowing down the job. By caching, then, it can be also created a "checkpoint" in case some failures happen.
   In addition, cache is an action, not a transformation. This implies that when using the *.cache* command, you are forcing the computation of that command, as it is not a lazy operation.

3. **Explain how Kafka provides scalability and fault tolerance?**

   Kafka as a commit log service is high-reliable, scalable and provides fault tolerance. The main components, producers and consumers are fully decoupled and agnostic to each other, and this is a key element to achieve the high scalability. The events registered are stored in topics and these topics are partitioned hence a topic is spread over a number of different buckets. This distributed placement of data is important for scalability because it allows clients to both read and write data from/to many brokers at the same time.

   Also, fault tolerance capabilities are integrated natively within Kafka. Since partitions are highly available and replicated stream data is persisted is available even if the application fails and needs to reprocess it. Tasks in Kafka leverage the fault-tolerance capability offered by consumers to handle failures. If a task runs on a machine that fails, the task is automatically restarted in one of the remaining running instances of the application. In addition, Kafka Streams makes sure that the local state stores are robust to failures. Thus, for each state store, it maintains a replicated changelog in which it tracks any state updates. These changelog topics are partitioned too.

4. **Assume we have two types of resources in the system, i.e., CPU and Memory. In total we have 28 CPU and 56GB RAM (e.g., 1 CPU = 2 GB). There are two users in the systems. User 1 needs ⟨1CPU, 2GB⟩ per task, and user 2 needs ⟨1CPU, 4GB⟩ per task. How do you share the resources fairly among these two users, considering (i) the asset fairness, and (ii) DRF.**

- *Asset Fairness case*: we have 2 users which want to share resources and we can calculate how to share the sources by giving weights to resources and equalizing total value given to each user. x and y are the tasks of User 1 and User 2

```
max(x,y)
subject to
x + y <= 28 (CPU constraint)
2x + 4y <= 56 (Memory constraint)
2x = 3y
```

User 1: x = 12 (12 CPU, 24 GB) (43% CPU, 43% GB)
User 2: y = 8 (8 CPU, 32 GB) (28% CPU, 57% GB)

- *DRF (Dominant Resource Fairness) case*

User 1 allocation (1 CPU, 2GB): 1/28 = 3.57% CPU and 2/56 = 3.57% RAM.
Dominant resource of User 1: equal share for CPU and RAM
User 2 allocation (1 CPU, 4 GB): 1/28 = 3.57% CPU and 4/56 = 7.14% RAM
Dominant resource of User 2: RAM

x and y are the tasks of User 1 and User 2

```
max(x,y)
subject to
x + y <= 28
2x + 4y <= 56
1/28x = 4/56y
```

User 1: x = 14 (14 CPU, 28 GB) (50% CPU, 50% RAM)
User 2: y = 7 (7 CPU, 28 GB) (25% CPU, 50% GB)