**Grenoble Universities (Grenoble INP & Université Grenoble Alpes)**
**M2 MoSIG – MSIAM**
**Academic year –**

# Data Management in Large Scale Distributed Systems — Final Exam
# XXX

- Duration: **2 hours**
- Authorized documents: **No documents**
- The number of points per exercise is only provided for indicative purposes.

**This exam is made of two parts. Use distinct answer sheets for each part.**

# Part I

## 1 About Replication (4 points)

**1.1** Replication is one of the main strategy that is employed when dealing with data at large scale. In the context of a data center based on a shared-nothing infrastructure, replication can be used both to increase the reliability and the performance of the system.

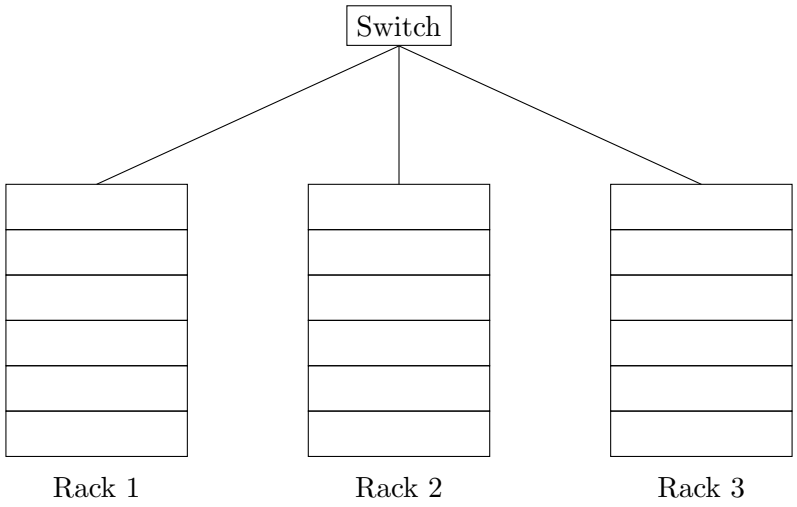


Figure 1: Data Center Architecture

(a) Explain how replication can be used for **fault tolerance**.

(b) Considering the architecture depicted in Figure 1, explain what is in your opinion the best strategy for placing replicas for **fault tolerance**.

(c) Explain how replication can be used for **performance**.

(d) Considering the architecture depicted in Figure 1, explain what is in your opinion the best strategy for placing replicas for **performance**.

**1.2** Name a system studied during the course that makes use of replication and briefly explain the replication strategy that it uses.

## 2 Resilient Distributed Datasets (4 points)

RDDs (Resilient Distributed Datasets) is the key concept in the Spark approach. A RDD is defined by a data source and a sequence of transformations.

**2.1** To optimize performance, Spark uses an approach called *Lazy evaluation* of the RDDs.

(a) Explain the concept of *Lazy evaluation*

(b) Explain why *Lazy evaluation* can help optimizing the performance of the system

**2.2** The driver of a Spark application analyzes the RDDs defined in the user program and generates tasks to be executed by the workers that will apply the transformations on the data. Transformations create dependencies between RDDs. We can distinguish between two kinds of dependencies: the *narrow* and the *wide* dependencies.

The Spark engine works more efficiently for programs that only use transformations inducing *narrow* dependencies than for programs having transformations inducing *wide* dependencies.

Gives 3 reasons why a program with only *narrow* dependencies can be run more efficiently than a program including *wide* dependencies and explain briefly.

## 3 Data processing at LinkedIn (5 points)

`LinkedIn` is an online social network dedicated to professional relationships. Among other things `LinkedIn` puts in relation employers posting jobs and job seekers posting their CVs.

Here is a description of 3 major applications used internally at `LinkedIn`:

**Inception.** The `Inception` application is in charge of extracting exception information from error logs generated by all `LinkedIn` services and to raise alarms.

**Standardization.** The `Standardization` application goes over the information provided by each user to generate a standardized description of her profile. This application is started every time the model used for standardizing profiles is updated.

**Data popularity.** The `Data popularity` application is calculating the top k most relevant items over a period of time for a data category (for instance, the *skills*).

**3.1** In the course, we have seen that there are two main categories of processing engines: Batch processing engines and Stream processing engines.

For each of the applications described above, explain whether you think that a Batch processing engine or a Stream processing engine would be more adapted to implement this application.

Answer the question for each application independently, and:

- Justify your answer
- If you make additional assumptions about an application, state these assumptions explicitly.

**3.2** Independently of the processing model that is used to implement these applications, name the application for which it is going to be the harder to get performance improvement by scaling out the underlying hardware infrastructure. Justify your answer.

## **Part II** (remember to use a separate answer sheet for this part)

# 4 NoSQL databases (5 points)

In this part of the exam, you have to **answer if each statement is true or false. Each incorrect answer cancels out a correct one**. The minimum score is zero (if you have more incorrect than correct answers, you are not receiving a negative score), the maximum is 5 points if you answer correctly to every question.

**4.1** Regarding the motivations behind NoSQL databases and their differences to traditional relational databases, is it correct to say that:

**4.1.1** The use of a NoSQL database is more adapted for an agile development cycle.

(a) True

(b) False

**4.1.2** Traditional relational databases typically require data organization in tables following a pre-defined schema.

(a) True

(b) False

**4.1.3** All NoSQL databases provide the same ACID (Atomicity, Consistency, Isolation, Durability) properties as relational databases, but with better performance.

(a) True

(b) False

**4.1.4** With NoSQL databases it is easier to scale-out systems, i.e., to improve the number of clients they can serve by using more distributed nodes.

(a) True

(b) False

**4.2** About NoSQL databases and distributed storage systems in general, is it correct to say that:

**4.2.1** The CAP theorem (also called Brewer's theorem) states a distributed storage system can provide the ACID properties.

(a) True

(b) False

**4.2.2** The CAP theorem shows a distributed storage system is never able to provide both Consistency and Availability.

(a) True

(b) False

**4.2.3** In the context of the CAP theorem, consistency means a client asking for a piece of data always gets the most recent version of it.

(a) True

(b) False

**4.2.4** In the context of the CAP theorem, availability means nodes where data is stored never become unavailable.

(a) True

(b) False

**4.2.5** NoSQL databases do not need to worry about dealing with network partition as they prefer to have consistency and availability.

(a) True

(b) False

**4.2.6** When merging conflicting versions of the same piece of data, the use of vector clocks allows us to always find the most recent version.

(a) True

(b) False

# 5 Neo4j (2 points)

Imagine a social network where users may follow other users' updates, but that is not necessarily a symmetrical relationship. For instance, user Alice can follow user Bob to see his posts, but that does not necessarily mean that Bob follows Alice's updates as well. Real-life examples of this are Twitter and Instagram.

Information about this social network is stored as a graph in Neo4j. Users are represented by nodes of type *User* containing an attribute called *username*. An edge of type *Follows* from node A to node B means A follows B, and there are other types of edges in this graph to represent other relationships.

The application that displays information in each user's profile page has to query the Neo4j database to obtain the corresponding data. for each of the queries below, **answer if they provide**

**the information they should provide. If they don't, explain in a few words what is wrong** (without a justification your answer will not be considered). All errors are related to the logic of the query, and **not** with Neo4j syntax.

Remember that in Neo4j:

- *()* represents a node;
- *(:NodeType)* represents a node of type *NodeType*;
- *({attr: "Value"})* represents a node that has the *attr* attribute with value *"Value"*;
- *-[]-* represents a relationship (edge), its direction can be specified with *-[]->* or *<-[]-*;
- similarly to nodes, *-[:RelType]-* represents a relationship of type *RelType*.

**5.1** For a given username ("Alice"), return the number of users followed by Alice.

```
MATCH(:User{username: "Alice"})-[]->(o:User)
RETURN count(o)
```

**Is the provided code correct? If not, explain.**

**5.2** For a given username("Alice"), return the name of the users followed by Alice but that do not follow Alice back.

```
MATCH(u:User{username: "Alice"})-[:Follows]->(o:User)
WHERE NOT (o)-[:Follows]->(u)
RETURN o.username
```

**Is the provided code correct? If not, explain.**