

**Qtext :**

In the following types of application scenarios, point out what is most important - consistency or availability, when a system failure results in a network partition in the backend distributed DB. Explain briefly the reason behind your answer. **(04 Marks)**

- (a) A limited quantity offer on a product for 100 items at an online retail store is almost 95% claimed
- (b) An online signature campaign on a social media platform is gathering user input
- (c) A hotel booking website is trying to sell rooms at a destination that is seeing very little uptake despite offers
- (d) An online remote multi-player game needing a lot of sequential team coordination in executing a set of steps to reach a milestone.

**User Answer :**

Q1.

(a)

For a product with limited quantity, it is very important that the store does not sell the product to the customer when the product has actually been sold out. To ensure this, it is important that the DB is always updated to the latest numbers. Therefore, **consistency** is most important in this scenario since all the partitions must report the same updated numbers.

(b)

For an online campaign that is collecting user input, the final tally of the scores is not necessary to be updated in real-time. In this scenario, it is more important to have the availability of the platform to the end-user so that they are encouraged to participate in the campaign.

Therefore, **availability** is most important in this scenario.

(c)

Since the hotel booking website is seeing very little uptake, it can be safe to assume that there is sufficient vacancy in the hotel. With this in mind, the priority of the website goes towards user experience towards booking rooms online. Therefore, **availability** will be more important. However, once most of the rooms have been booked, the system must quickly switch to *consistency* to avoid overbooking.

(d)

Since the key factor here is sequential team coordination, and the order of steps is very important, it is very important that the system prioritizes **consistency** over availability. That way the team will be working with the latest information and not have to make decisions based on obsolete data.

Qtext :

You have a map-reduce program on a Hadoop cluster. If you run the program on a single node, it takes total 345 s and 5% of the overall time is spent in a sequential reduce operation. The rest of the map reduce application code and runtime can be parallelized. **(06 Marks)**

- (a) If you had a 10 node cluster with similar nodes for the same program and data set, how much time would you theoretically expect the program to take ?
- (b) Is there any reason to expect higher time if you actually measured the program execution on the 10 node cluster ?
- (c) Suppose you ran this program, with necessary modifications, but with a larger data set trying to accomplish more work with the 10 node cluster. What is a theoretical speed up you could target ?

User Answer :

Q2

$$T(1) = 345 \text{ seconds}$$

$$f = 0.05$$

(a)

$$N = 10$$

$$\text{Time taken by 10-node cluster} = T(N) = f*T(1) + (1-f)*T(1)/N$$

$$T(10) = 0.05*345 + 0.95*345/10 = 17.25 + 32.775 = \mathbf{50.025 \text{ seconds}}$$

$$(\text{Speedup} = 6.896551724)$$

**The program in the 10 node cluster would take 50.025 seconds.**

(b)

The time of 50.025 seconds is a theoretical time that assumes fixed workload. Additionally, all system or communication overhead are ignored in the calculation. And finally, the I/O time or exception handling time is also not included in the analysis. By factoring in these times, **it is definitely possible to get a higher time** with a 10 node cluster.

(c)

Using Gustafson's Law:

$$S(N) = f + (1-f)N$$

$$\text{Therefore, } S(10) = 0.05 + 0.95*10 = \mathbf{9.55}$$

**Therefore, theoretical speedup = 9.55**

**Qtext :**

It has been noted that there has been a random failure in one of the nodes of a cluster of 4 servers. The failure is observed always on the same node and is happening on the 25th of every month. An hour is taken to recover this node. Incidentally, the applications on these servers need an additional half hour to be started. Consider 30 days in a month.

**(06 Marks)**

- a) Calculate the availability of the cluster.
- b) If the cost of downtime is \$2k per hour, then what is the quarterly cost?

**User Answer :**

Q3.

(a)

$$\text{MTTR} = 1 + 0.5 \text{ hours} = 1.5 \text{ hours}$$

Assuming, there are 30 days, the total hours =  $30 \times 24 = 720$  hours

Therefore, since the node fails consistently on 25th of every month, we can say that the

MTTF = 720 hours

Therefore,

$$\text{Availability} = \text{MTTF} / (\text{MTTF} + \text{MTTR}) = 720 / 721.5 = \mathbf{0.99792 \text{ or } 99.792\%}$$

(b)

One quarter = 3 months = 90 days = 2160 hours

Downtime cost = \$2000 per hour

$$\text{Quarterly cost} = 2000 \times 1.5 \times 2160 / 720 = \mathbf{\$9000}$$

Therefore, quarterly cost of downtime = **\$9000**

**Close**

**Qtext :**

"spotify, an on-demand music providing platform, uses Big Data Analytics, collects data from all its users around the globe, and then uses the analyzed data to give informed music recommendations and suggestions to every individual user".illustrate different phases in the life cycle of this big data project. **(04 Marks)**

**User Answer :**

Given are the stages:

#### **1. Business case evaluation**

Spotify has to define a clear business case towards this project. In this case, it is to use the user information like music choices, follow-up songs, minutes played, etc to make a recommendation system to every individual user.

Justification - a good recommendation system will attract more users and retain existing users since good recommendations lead to more usage  
Motivation - to build a large user base and be the most profitable music platform in the industry

Goals - To create a personalized recommendation system for each user based on the information collected from all users in the system

#### **2. Data Identification**

Almost all of the data sources will be internal since spotify will be able to get its users usage pattern to extract information needed for the recommendation system.

Here, identifying the relevant inputs is the most important. The inputs could be as follows:

- Minutes listened per song
- Which song was followed up after the current song
- Most listened genre
- Which decade of the songs most listened to?
- Songs liked/disliked

#### **3. Data Acquisition and Filtering**

In this phase, the data decided from the previous step is gathered and filtered. A basic data validity check is performed to see whether the data is useable or not.

#### **4. Data Extraction**

Most of the data here may not be in any strict structural format. This stage deals with converting all the unstructured data into a structured format which can then be passed to the model training.

#### **5. Data Validation and Cleansing**

In this stage, the extracted data undergoes a thorough integrity check. The data is checked for outliers, skewness, missing fields, etc.

#### **6. Data Aggregation and Representation**

In this stage, all the different structured data from the previous stage is collated into a single unified view. Any key related discrepancies are corrected

## 2. Data Identification

Almost all of the data sources will be internal since Spotify will be able to get its users' usage pattern to extract information needed for the recommendation system.

Here, identifying the relevant inputs is the most important. The inputs could be as follows:

- Minutes listened per song
- Which song was followed up after the current song
- Most listened genre
- Which decade of the songs most listened to?
- Songs liked/disliked

## 3. Data Acquisition and Filtering

In this phase, the data decided from the previous step is gathered and filtered. A basic data validity check is performed to see whether the data is useable or not.

## 4. Data Extraction

Most of the data here may not be in any strict structural format. This stage deals with converting all the unstructured data into a structured format which can then be passed to the model training.

## 5. Data Validation and Cleansing

In this stage, the extracted data undergoes a thorough integrity check. The data is checked for outliers, skewness, missing fields, etc.

## 6. Data Aggregation and Representation

In this stage, all the different structured data from the previous stage is collated into a single unified view. Any key related discrepancies are corrected and the data structure is made uniform.

## 7. Data Analysis

This stage contains the actual analysis that leads to feasible results. Here, a mix of descriptive, diagnostic and predictive analytics takes place in order to determine the users' behaviour and how that can be translated to getting a good score on their recommendation logic.

Once done, the test results are tested against any or all alternate hypotheses that may have been suggested during the course of the project. All the features are explored in order to see which features form the final feature list that define the recommendation system.

## 8. Data Visualization

In this stage, the results of the previous stage are visualized. These results could be in the form of an accuracy index, a correlation plot, or something as simple as a probability matrix of the recommended music tracks based on a test input.

This stage is important to explain the performance efficiency of the recommendation system to the business, since business does not necessarily have analytical skills to understand the numerical results.

## 9. Utilization of Analysis Results

In this final stage, the recommendation model is deployed to a small section of the user base to test out the feedback and the performance of their model. Once the results are in, stages 7-9 may be repeated to enhance the performance.

Finally the model is deployed for the entire user base.

**Qtext :**

Write the pseudocode of mapper and reducer classes to find Call data records which has call type as ‘sms’ and call duration more than 10mins. Consider the datasets with the following fields: subscriber\_phone\_number, cell\_id, timestamp, call\_duration, phone\_id, status and type\_of\_call. **(06 Marks)**

**User Answer :**

**Mapper 1 - Extracts call records with type "sms"**

**mapper1():**

```
import sys

for line in sys.stdin:
    // Assume data is comma separated ","
    data = line.strip().split(",")
    if len(data)!=7:
        continue

    if (data[6] == '\'sms\''):
        for item in data:
            print(item, end='\t')
        print()
```

**Reducer 1- Returns records with type "sms" and duration more than 10 mins**

**reducer1():**

```
import sys

for line in sys.stdin:
    data = line.strip().split("\t")
    if len(data)!=7:
        continue

    if(int(data[3]>=10):
        for item in data:
            print(item, end='\t')
        print()
```

---

**Qtext :**

You are given a data set with the following fields in each record: name, age, address, zipcode, salary. You need to output zip codes where avg salary is in the range \$100k- \$500K using MapReduce. Show the logic of Mapper and Reducer functions to solve the given problem. **(04 Marks)**

**User Answer :**

Mapper 1 - Extracts a tuple of (zip, salary)

**mapper1():**

```
import sys

// Assume that the fields are comma separated ","
for line in sys.stdin:
    data = line.strip().split(",")
    if len(data) == 5:
        // Outputs (zip, salary)
        print(data[3], "t", data[4])
```

Reducer 1 - Calculates average salary per zip as a tuple (zip, average(salary))

**reducer1():**

```
import sys

oldZip = None
salTotal = 0
salCount = 0

// Assume that the fields are tab separated
for line in sys.stdin:
    data = line.strip().split("t")
    if len(data)!=2:
        continue
    // assume something has gone wrong and skip

    zip, sal = data

    if oldZip and oldZip != zip:
        avgSal = salTotal/salCount
```

## Reducer 1 - Calculates average salary per zip as a tuple (zip, average(salary))

**reducer1():**

```
import sys

oldZip = None
salTotal = 0
salCount = 0

// Assume that the fields are tab separated
for line in sys.stdin:
    data = line.strip().split("\t")
    if len(data)!=2:
        continue
    // assume something has gone wrong and skip

    zip, sal = data

    if oldZip and oldZip != zip:
        avgSal = salTotal/salCount
        print(oldZip, "\t", avgSal)
        oldZip = zip
        salTotal = 0
        salCount = 0

    oldZip = zip
    salTotal += float(sal)
    salCount += 1

if oldZip != None:
    avgSal = salTotal/salCount
    print(oldZip, "\t", avgSal)
```

## Mapper 2 - Collects those zips whose salary is between 100K-500K

**mapper2():**

```
import sys:

for line in sys.stdin:
    data = line.strip().split("\t")
    if len(data)!=2:
        continue
```

```
salCount += 1
```

```
if oldZip != None:  
    avgSal = salTotal/salCount  
    print(oldZip, "\t", avgSal)
```

#### Mapper 2 - Collects those zips whose salary is between 100K-500K

##### **mapper2():**

```
import sys:  
  
for line in sys.stdin:  
    data = line.strip().split("\t")  
    if len(data)!=2:  
        continue  
  
    zip, sal = data  
  
    if ((sal>=100000) & (sal <= 500000)):  
        print("100K-500K", "\t", zip)  
    else:  
        print("Others", "\t", zip)
```

#### Reducer 2 - Print out all the zips with key as "100K-500K"

##### **reducer2():**

```
import sys  
  
for line in sys.stdin:  
    data = line.strip().split("\t")  
    if len(data)!=2:  
        continue  
  
    flag, zip = data  
  
    if flag == "100K-500K":  
        print(zip) // prints the final required zips
```