Université Grenoble Alpes & Grenoble INP
M2 MoSIG
Academic year 2023-2024

Large-scale Data Management and Distributed Systems — Final Exam

Data Management (8 points)

Please use a distinct answer sheet for this part

All documents and electronic devices are forbidden except one handwritten A4 sheet of paper.

The number of points per exercise is only provided for indicative purposes.
The grade will take into account the quality of the presentation.

1 Some Statements (2.5 points)

For each of the following statements, answer **True** or **False** and explain in a few words. Answers without explanations will be ignored.

(a) Both scaling out and scaling up are useful for increasing the performance of big data processing systems.

(b) The Spark framework is more efficient than the Hadoop Map-Reduce framework especially for iterative jobs.

(c) The main concern in big data distributed processing is data consistency.

(d) NoSQL systems will replace SQL systems eventually.

(e) When dealing with a read-mostly workload, a row-oriented database will provide better performance than a column-oriented database because in a row-oriented storage all row values are stored next to each other.

2 Processing data (3.5 points)

Reminder: in Hadoop one job corresponds to a single map-reduce calculation.

2.1 You have a text file containing the data about the purchases of a group of people. Each line contains the information about one purchase. Namely it gives the name of the person, the category of the purchase, the amount and the date of purchase.

```
Pierre   Fun         25.90   10/10/2023
Lou      Clothes     55.00   04/05/2022
Ivan     Taxes       64.03   09/29/2023
Emilie   Food        25.90   05/13/2022
Emilie   Health      34.50   06/06/2023
...      ...         ...     ...
```

Give the pseudo-code for a Map-Reduce Hadoop program that calculates the average purchase amount per person, all categories included.

2.2 Let us change the setting. Now you have the information about purchases coming in (in almost) real time and you need to compute the average purchase amount per person once per week. What would be the design of your Hadoop Map-Reduce program? What jobs will you have and what intermediary data will you use? (The triggering of the Hadoop jobs is out of the scope of this question.)

3 Storing data (2 points)

3.1 Now that you are in the tech team of a big company, you need to decide of the database to store the data about people's purchases (similar data as the one you have the previous question). What kind of storage will you advise ? Cite at least two products providing this type of storage.

3.2 Your team leader announces the creation of a data warehouse to support the weekly computations of the average purchase amount and track changes in people's behavior. Her choice goes to the open-source PostgreSQL database. What do you think about this decision ?

3.3 How could you use Kafka to feed the data warehouse from the initial database ? Provide a schema of the proposed architecture and explain briefly.

Distributed Systems (12 points)

Please use a distinct answer sheet for this part

1 Ordering events (2 points)

1.1 Copy the execution scenario of Figure 1 on your answer sheet, and tag each *send* and *receive* event with its vector clock.

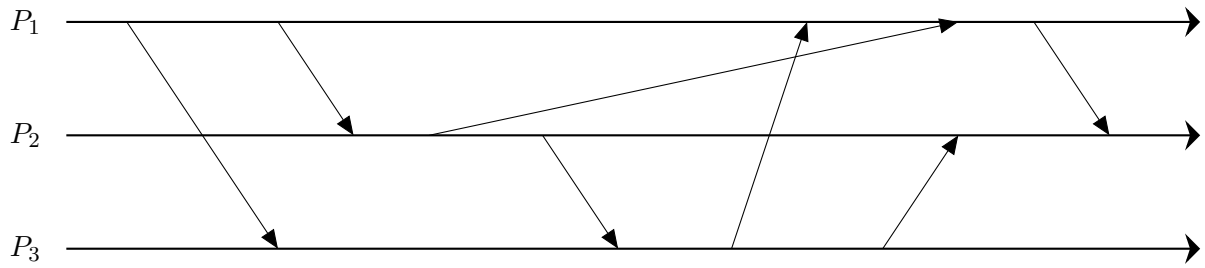

Figure 1

1.2 Let C be a consistent cut defined by the tuple $(c_1, c_2, \ldots, c_n)$. Let cut C' be $C \cup \{e_2^{c_2+1}\}$.

For each of the following cases, is C' **always** a consistent cut? Justify your answer.

(a) $e_2^{c_2+1}$ is a local event

(b) $e_2^{c_2+1}$ is a send event

(c) $e_2^{c_2+1}$ is a receive event

Note: The notation $e_2^{c_2+1}$ means "the next event on process p_2, after the consistent cut C ".

2 Broadcast algorithms (5 points)

2.1 Here are the four properties that define Atomic Broadcast:

- *Integrity:* Each process delivers message m at most once, and only if was broadcasted by some process.
- *Validity:* If a correct process broadcasts a message m, then every correct process eventually delivers m.
- *Uniform Agreement:* If a message m is delivered by some process (whether correct or not), then m is eventually delivered by every correct process.

- *Uniform total order:* If some process (correct of faulty) adelivers m before m', then every process adelivers m' only after it has adelivered m.

For each property, say whether it is a liveness or a safety property. Justify your answers briefly.

2.2 Figure 2 presents the Uniform Reliable Broadcast algorithm studied in the course. This algorithm implements the following properties:

- *Integrity:* Each process delivers message m at most once, and only if was broadcasted by some process.
- *Validity:* If a correct process broadcasts a message m, then every correct process eventually delivers m.
- *Uniform Agreement:* If a message m is delivered by some process (whether correct or not), then m is eventually delivered by every correct process.

Answer the following questions and justify your answers:

(a) To reduce the cost of the algorithm, we propose the following modification: we propose to call `tryDeliver()` in line 24 only if the condition "$[s, m] \notin pending$" is true (that is, we move the call to `tryDeliver()` inside the *if* block). Does the algorithm remain correct with this modification?

(b) The uniform reliable broadcast algorithm presented in Figure 2 relies on a perfect failure detector $\mathcal{P}$. Assume that instead of *strong completeness*, we have a failure detector that ensures *weak completeness*. What would be the impact for the considered algorithm?

```
1   Implements:
2   UniformReliableBroadcast, instance urb.

4   Uses:
5   BestEffortBroadcast, instance beb
6   PerfectFailureDetector, instance 𝒫

8   Variables:
9   correct = Π # The set of processes considered correct
10  delivered = ∅
11  pending = ∅
12  for all m:
13      ack[m] = ∅

15  Upon urb.broadcast(m):
16      pending = pending ∪ [self,m]
17      beb.broadcast([self, m])

19  Upon beb.deliver(p, [s, m]) # s is the id of the source
20      ack[m] = ack[m] ∪ p
21      if [s, m] ∉ pending:
22          pending = pending ∪ [, m]
23          beb.broadcast([s, m])
24       tryDeliver([s,m])

26  Upon event crash(q) raised by 𝒫:
27      correct = correct \ {q}
28      for all [s, m] ∈ pending:
29          tryDeliver([s,m])

31  Function canDeliver(m):
32      return (correct ⊆ ack[m])

34  Function tryDeliver([s, m]):
35      if canDeliver(m) and m ∉ delivered:
36          delivered = delivered ∪ m
37          Trigger urb.deliver([s, m])
```

Figure 2: Implementation of Uniform Reliable Broadcast.

3 About Consensus (5 points)

3.1 We defined the FloodSet Consensus in the Synchronous Round Model.

(a) To which system model does this round model corresponds to?

(b) Give an important drawback of the Synchronous round model if we want to implement it in practice and explain briefly.

3.2 Consider the OTR consensus algorithm (Presented in Figure 3), an execution of the algorithm that is compatible with the assumption of OTR, and GSR=10.

```
1    Variables:
2    x_p = {v_p}

4    Round r:
5      S_p^r : # denotes the sending step of p in round r
6          send (x_p) to all processes

8      T_p^r : # denotes the state transition step of p in round r
9          if number of messages received ≥ n − f:
10             x_p = most frequent value received (if more than one, take the smallest)
11         if at least n − f values received are equal to x̄:
12             DECIDE(x̄)
```

Figure 3: The *OTR* algorithm

Answer the following questions. For each question, justify your answer by describing briefly the execution scenario corresponding to your answer.

(a) With adequate initial values, what is the minimum number of rounds needed for all correct processes to decide?

(b) With any initial values, what is the minimum number of rounds needed for all correct processes to decide?

(c) What is the maximum number of rounds needed for all correct processes to decide?

3.3 We still consider the OTR consensus algorithm. In Figure 3, assume line 9 is suppressed. Is the algorithm still correct? If yes, explain why. If no, what property (properties) of consensus is (are) violated? Construct an execution that illustrates the violation(s).