CSE 444 – Homework 6
Parallelism and Distribution

Name: Anupam Gupta

Question	Points	Score
1	20	
2	20	
Total:	40	

1 Parallel Data Processing

1. (20 points)

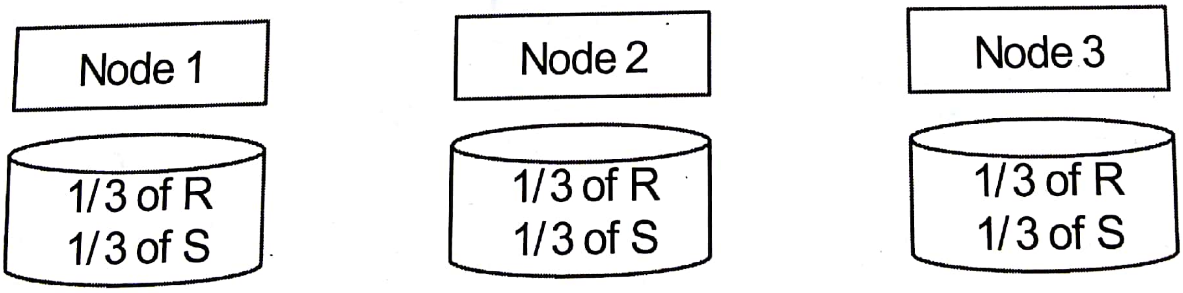
   (a) (10 points) Consider two relations R(a,b) and S(c,d) that are both horizontally partitioned across N = 3 nodes as shown in the diagram below. Each node locally stores approximately $\frac{1}{N}$ of the tuples in R and $\frac{1}{N}$ of the tuples in S. The tuples of R are randomly organized across machines (i.e., R is block partitioned across machines) while the tuples of S are hash-partitioned on S.c.

   Show a relational algebra plan for the following query and how it will be executed across the N = 3 machines. Pick an efficient plan that leverages the parallelism as much as possible. Include operators that need to re-shuffle data and add a note explaining how these operators will re-shuffle that data. For example, if you need to re-hash the data, add a "hash" operator into your query plan.

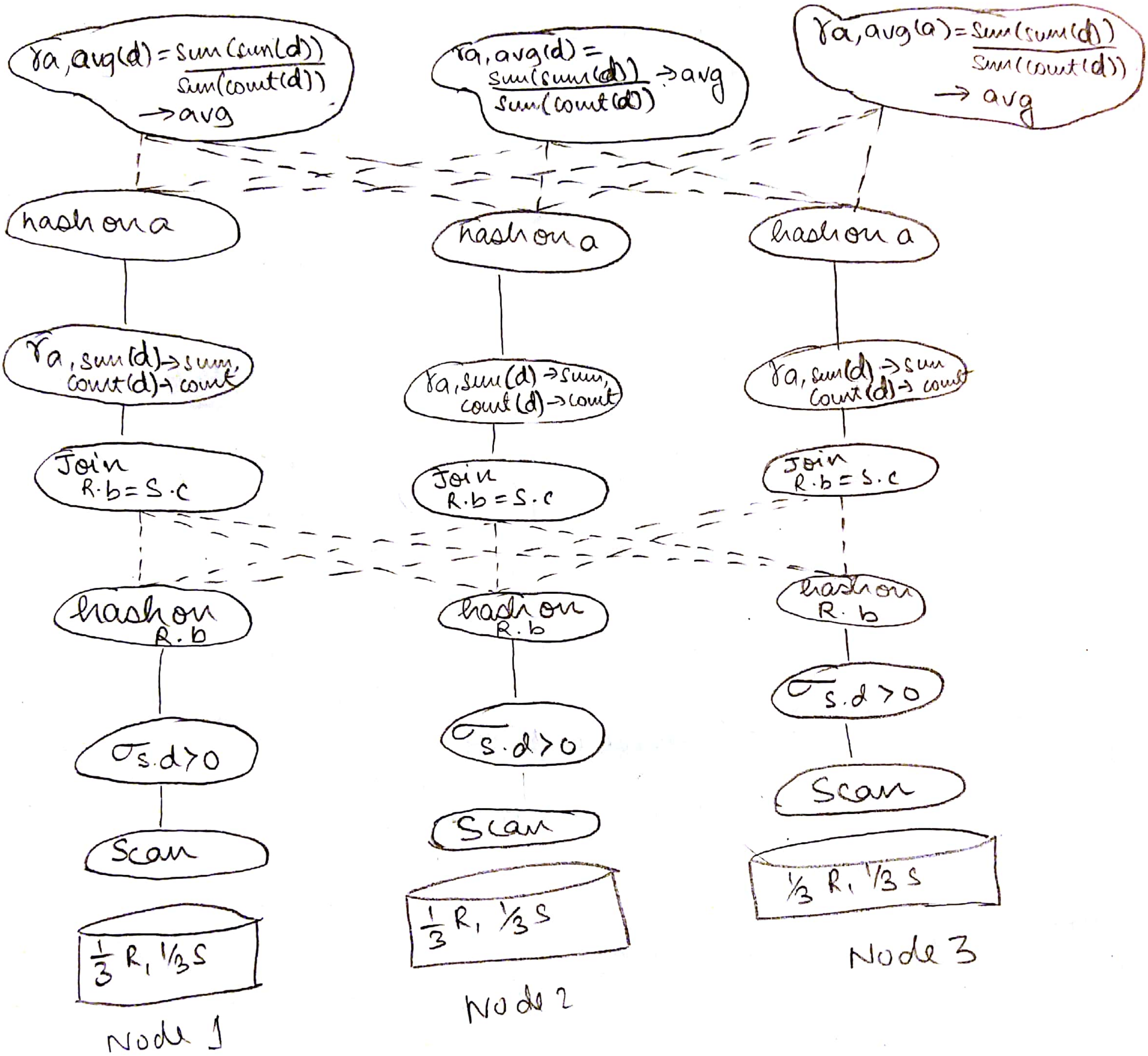
   Draw the parallel query plan. Indicate the edges that re-shuffle data across machines by drawing them as dashed lines:

   Note: Your plan will be more efficient if you push aggregations down. Can you compute partial aggregates before shuffling data? Can you compute partial aggregates before the join?

```
SELECT a, avg(d) as avg
FROM R, S
WHERE R.b = S.c
AND S.d > 0
GROUP BY a
```



Answer:


γa, avg(d) = sum(sum(d)) / sum(count(d)) → avg
γa, avg(d) = sum(sum(d)) / sum(count(d)) → avg
γa, avg(a) = sum(sum(d)) / sum(count(d)) → avg
hash on a
hash on a
hash on a
γa, sum(d) → sum, count(d) → count
γa, sum(d) → sum, count(d) → count
γa, sum(d) → sum, count(d) → count
Join R.b = S.c
Join R.b = S.c
Join R.b = S.c
hash on R.b
hash on R.b
hash on R.b
σ s.d>0
σ s.d>0
σ s.d>0
Scan
Scan
Scan
1/3 R, 1/3 S
1/3 R, 1/3 S
1/3 R, 1/3 S
Node 1
Node 2
Node 3

First we scan the tables at each of the nodes, then apply a selection to only use those tuples with s.d > 0. Then we hash & repartition on R.b and then compute the join. Following that we group by a and compute our partial aggregates sum(d) and count(d). Then we rehash and partition on a and compute our final average as $\frac{sum(sum(d))}{sum(count(d))}$ at each node for each a.

(b) (10 points) Explain how the query would be executed in MapReduce. Make sure to specify the computation performed in the map and the reduce functions. You do not need to show pseudocode. An English-language description will suffice. Also, you can use a query plan that is different from the one you chose in the previous section.

There will be two map-reduce jobs for this problem

```
Map function 1:
    Input → [key] value (tuple from either relation)
     if (value. relation = R): emit Intermediate (value.b, (1, value))
     else if (value.d > 0): emit Intermediate (value c, (1, value))

Reduce function 1.
    Input → key, Iterator values
     R = [ ], S = [ ]
     for each value in values:
         if (value. type = 1): R.add(value)
         else S. add (value);
     for R1 in R , for S1 in S
         emit( R1 , S1);

Map function 2:
    Input → tuple from Reduce function2 ( R1 , S1 )
                                       R1.a R1.b S1.c S1.d
    emit Intermediate (R1. a , S1.d);

Reduce function 2: 
    Input → [key], Iterator values (from Mapfn.2)
     Sum = 0, count = 0.
     for (value in values){
         sum = sum + value
         count = count + 1
     }
     emit (key, sum/count);
```

P. T. O →

We will basically have a map function that both filters S(removes tuples with $S.d < 0$) and sends out tuples for a join. The Reduce function picks up these tuples and emits the joined tuples(as a result of the join). This output goes back into another Map function that will be used for groupby. It basically emits the a attribute along with the 'd' attribute(whose average needs to be taken.) The second reduce function gets all the d values for a particular 'a' and computes the avg. and emits them correctly which are the final results of the query.

2 Distribution and Replication

2. (20 points)

(a) (10 points) In the two-phase commit protocol, describe what happens if a subordinate receives a PREPARE message, replies with a YES vote, crashes, and restarts.

In the 2 phase commit protocol, when a Subordinate receives a PREPARE message and replies with a YES vote (supposing all other subordinates reply with a YES as well), the co-ordinator will log and Force write a commit record and send the commit message to all the subordinatesand wait for ACK messages. However, since the subordinate has crashed by this time, it will not recieve the commit message. When the sub-ordinate restarts, it will analyze its logs in the analyze phase, see there's a PREPARE message but no commit/abort message and will ask the co-ordinator what to do. Now since the co-ordinator has not received all the ACK messages, it hasn't removed that transaction yet (assuming its not a presumed abort state machine) and will find the transaction in its transactions table, see it has a logged commit and will send the commit message to the subordinate following which the subordinate will receive the message, force write its commit and end and send back an ACK message following which the co-ordinator will also end and forget the transaction.

P. T. O →

If any of the other transactions have a conflict and reply with a NO message for prepare, the co-ordinator logs an Abort and sends an ABORT message to all the sub-ordinates which send back ACK messages & forget the transaction. When the subordinate that crashed, restarts it checks in with the co-ordinator, aborts and sends an ACK as well at which point the co-ordinator logs an End record and forgets the transaction.

(b) (10 points) Explain the benefits and challenges of asynchronous replication (also called lazy replication) in contrast to synchronous replication. Discuss both the configuration that uses a single master and one that uses multiple masters.

Asynchronous replication provides us with more availability and better performance (OLTP) than synchronous replication. Since, there's just one machine we have to deal with that writes to other machines asynchronously, the transactions are completed faster. With a single master system, the benefits are that it is much faster than synchronous replication (since it doesn't have to wait to write to other machines) and it increases/improves the availability. However, the drawback here is that since it's all dependent on one machine, when the primary crashes, there ~~is~~ are potentially some recent transactions that are lost (since they hadn't been replicated) and the system is then not consistent anymore. In a similar fashion with a group master approach, the availability and performance is much better since they're all working simultaneously, however, it is possible that in these cases, the system is not consistent anymore. There is a higher chance of conflicts happening with this approach than with a synchronous approach. In the case of conflicts, there's an additional overhead involved with abortion of transactions. In a similar fashion if the computer crashes, there's some overhead with waiting for the computer to restart while simultaneously trying to do tentative transactions that could be decided to abort later! In terms of productivity & availability, asynchronous replication is better than synchronous replication, however it is much worse with consistency.