1.1


γa, avg(d) = Sum(count(d)) / sum(count(d)) → avg
γa, avg(d) = Sum(count(d)) / sum(count(d)) → avg
γa, avg(d) = sum(count(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
σS.d > 0
σS.d > 0
σS.d > 0
Scan
Scan
Scan
node 1
node 2
Node 3
1/3 of R 1/3 of S
1/3R, 1/3S
1/3R, 1/3S

1.2:

Map function 1:

- If the map function process a block of the S Relation, it applies the selection predicate to each S tuple in that block (S.d > 0), and if the tuple passes the selection, it outputs a record with the key = S.c and value = ('S', S.d).
- If the map function processes a block of the R relation, it outputs b record with key = R.b and value = ('R', R.a)

Reduce function 1:

- Input to the reducer: the same R.b and S.c as the key b., and a list of R and S tuples ('R', R.a) or ('S', S.d). In other words, we have (b, (value from R, value from S, value from S, etc...))
- Computation: the reducer performs the local join of R and S, and finds the avg(S.d) value.

Map function 2: Take the block from the results of Reduce 1: key = a, and value are d.

Reduce function 2: for the same a from Map 2, caculate total sum of a, and count of a. Finally use sum to divided by count to get average.

we need to have 2 map-reduce functions. The first map function filters S (remove tuples whose S.d < 0) and sends out tuples for a join. The reduce function picks up those tuples and emits the joined tuples as the result of the join. This output goes back into another Map function that will be used for groupby. It basically emits the a attributes with the d attribute. The second reduce function gets all the d's value for the same a and computes the avg of d. which are the final results of the query.

2.1:

In the 2 phase commit protocal, when a subordinate receives a PREPARE message and replies with a YES vote (supposing all others 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 subordinates and wait for ACK messages.

However, since the subordinate has crashed by this time, it will not receive the commit message. When the subordinate restarts, it will analysis its logs in the analyze phase, and receives 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 reserved that transaction yet (assuming its not a preserved abort state) and will find the transaction in its transactions table, and 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.

If any other transaction has a conflict and replies with a No for PREPARE, The co-ordinator logs an Abort and sends an ABORT to all subordinates which send back ACK messages to forget the transaction. when the subordinate that crashed, restarts, check-in with the co-ordinator, aborts and sends an ACK as well as which point the co-ordinator logs the END record and forgets the transaction.

2.2

Asynchronous replication provides us with more availability and better Performance (OLTP) than synchonized replication. Because there's just one machine that has to deal with that writes to other machines asychronously, the transactions are completed faster. With a single master system, the benifits are that it is much faster than synchronous replication (since it doesn't have to wait to write to other machines) and it increase/improves the availability.

However, the drawback is that since it is all dependent on one machine, when the primary crashes, there will be some transactions be lost. (because they hemdn't been replicated), and then the system will not be consistent anymore. In asychronose with a group approach, the availability and performance is much better since they are all working simultaneously However, it is possible that in these cases, the system is not consistent anymore.

In contrast, Synchronous replication provides way much better consistency, and it makes sure that every thing is correct and up to date at all time. It's drawbacks are: ① slow for throughput transactions. ② It is very expensive to get votes between nodes and mak really high runtime overhead.

For synchronouse replication with group method coun provide really fast reads when it implements 'Read-locks-one, write-locks-all' method.

The configuration for synchronous replication has two different

methods. 1) Synchronous Mast Replication

2) Synchronous Group Replication.

1) In the Master method, ther is only one node chose as primary, and use 2PL at the end to propogate the transactions on other nodes. When the primary fails, the rest node run election and chose a new primary.

2) In the group method, Any node can initiate a transaction, but need to gather a number of nodes to agree on that transaction.

The Configuration for asynchronous is similar, and I have talked a little bit about it earlier.

1) Master method for asynchronous:

There will be one replica update by the transactions, and the other nodes will be updated asychronous later. And when the master fails, there is still elections, but it needs to find which node is most up-to-date.

2) Asychronous Group Replication:

There are multiple master, each nodes can initiate a transaction, but cannot guarentee one-copy serializability