Distributed Systems

18r. Preview for Exam 3: Selected questions

Paul Krzyzanowski
Rutgers University
Fall 2013
You have a huge amount of data on stock transactions. Each record contains the following information:

\[
\{ \text{timestamp, ticker\_symbol, sale\_price, shares} \}
\]

You want to find the average share price of each traded company for the year 2005 and will do this using MapReduce.

Explain the operations in your map and reduce functions. Assume that a default partition function is used.

For the map function, assume that the function is called once for each line of input and contains the parameters mentioned above. Explain any processing logic that takes place in the map worker to pre-process, discard, duplicate, or combine data. Also, be sure to state the sequence of data that each map worker emits, with the first item being the key. You may use the pseudocode `emit(a, b, c, …)` to represent the sequence of data that a map worker emits.

For the reduce function, assume that each worker is invoked with the parameters `<key, data\_list>`. You may use “for each X in data\_list” pseudocode to iterate over each of the items in data\_list. You may also use the pseudocode `emit(a, b, c, …)` to represent the sequence of data that a reduce worker emits. Explain any processing logic that takes place in the reduce worker.
You have a huge amount of data on stock transactions. Each record contains the following information:

{ timestamp, ticker_symbol, sale_price, shares }

You want to find the average share price of each traded company for the year 2005 and will do this using MapReduce.

Map(timestamp, ticker_symbol, sale_price, share_count):

    if  (year(timestamp) == 2005)
        for (i = 0; i < shares; i++)
            emit(ticker_symbol, sale_price);

Reduce(key, data_list):

    total =0; i = 0;
    for each price in data_list {
        total += price
        i++;
    }
    emit(key, price/i)
You have millions of files of user comments to various blog articles. Each file contains

\{ original-article-ID, this-article-ID (ID of this response article),
   author-ID (the author of the response), date, message \}

You want to create a list of authors with a per-author count of the number of unique articles that the author commented on.

Explain how you use Map-Reduce to compute this.

Specifically, explain what the map function does, what data the reduce gets, and what the reduce function does.

(1) Map: parse out \{author-ID, original-article-ID\}.
   Write these pairs to the intermediate files.

(2) Reduce gets as input:
    each call to Reduce gets author ID and a list of all original-article-IDs

(3) Reduce does: (a) sort original article IDs, (b) remove duplicates, (b) generate a count of unique original article IDs, (c) output author-ID & count.
Identify two reasons why you might want to use a higher replication factor for files in GFS.

1. High availability
2. Load balancing
3. Distribution for geographic proximity

The same chunk can be read from multiple machines
1. If one or more machines are dead, it can be read from surviving machines
2. Access to the chunk can be balanced across all systems that have replicas
3. The chunk can be read from the nearest machine that has a replica
What compromise must be made in a distributed system with replicated data if you must have high availability and partition tolerance?

Consistency

Brewer’s CAP theorem states that you can have **at most two** out of three of:

- consistency + availability + partition tolerance

Consistency + availability:
- means that we need to have 100% uptime network connectivity to make replicas

Consistency + partition tolerance:
- means that data may not be available if a replica cannot ensure it has an update

Availability + partition tolerance:
- means that some replication may not be able to propagate; leading to inconsistency
6. The partitioning function in MapReduce
   (a) Determines which shard will be assigned to a specific map worker.
   (b) Filters out unnecessary input data prior to being processed by the map worker.
   (c) Determines the division of available servers into map workers and reduce workers.
   (d) Determines which reduce worker will process data associated with a particular key.

A partitioning function runs on the map worker to determine which partition a (key, value) pair will go to. All (key, value) data is stored in an intermediate file.

When ALL map workers finish, the reduce workers start and contact all the map workers to fetch (key, value) data for their partition.

The default partitioning function is simply \( \text{hash(key)mod } N \), where \( N \) is the number of reduce workers.
7. In MapReduce, the first reduce worker starts processing data
(a) At the same time as map workers start in order to maximize concurrency.
(b) As soon as at least one map worker begins to emit output.
(c) When least one map worker completes.
(d) When the last map worker completes.

MapReduce has to wait for ALL map workers to finish.

Only then are we sure that all (key, value) sets have been generated and are ready for input and sorting by the reduce workers.
8. A **tablet** in Bigtable is a subset of a table containing
(a) A range of consecutive rows and a range of consecutive column families.
(b) All rows but a subset of column families
(c) Frequently used rows and column families cached in memory for performance.
(d) A range of consecutive rows and all column families.

Each instance of Bigtable is just one table.

The table is sorted by the row key.

The table is broken up into **tablets**.

Each tablet is just a slice of a set of consecutive rows of the table.
Which of the following is not a responsibility of a map worker in the MapReduce framework:

(a) Generate (key, value) pairs.
(b) Target (key, value) data for one of R reduce workers.
(c) Partition original data into shards.
(d) Discard data of no interest.

The master (coordinator) assigns workload to the map workers.
Map workers partition intermediate (key, value) pairs.
Bigtable data is:

(a) Sorted by row keys.
(b) Sorted by column keys.
(c) Unsorted and assigned to a node by a hash function of the row key.
(d) Unsorted and assigned to a node by a hash function of the row and column keys.

Bigtable keeps all of its data sorted alphabetically by a row key.
Accessing consecutive rows is efficient and usually stays on the same machine.
Bigtable does not use Chubby to:
(a) Discover tablet servers.
(b) Store Bigtable schema information.
(b) Ensure there is only one master server running.
(c) Forward client requests to the proper tablet server.

Chubby is used to allow tablet servers to grab locks and to bootstrap a Bigtable master.
Client requests go to a Bigtable master if they require looking up a row.
Otherwise, they go directly to the tablet server holding that portion of the table.
Which of the following is not a property of Chubby?
(a) A distributed lock service that manages leases for resources.
(b) Uses active replication for fault tolerance.
(b) Uses Paxos to ensure consistency among servers.
(c) Uses load balancing across all replicas to respond to multiple client requests.

Chubby is a single server with backups on standby. It offers no load balancing.
10. Which distributed mutual exclusion algorithm does not require a participant to know anything about the composition of the group?

(a) Centralized
(b) Lamport
(c) Ricart and Agrawala
(d) Token Ring

Lamport and Ricart & Agrawala algorithms both require each participant to multicast to all other participants and get acknowledgements.

Token ring algorithm requires a participant to know the next participant in the ring. It does not need to know the entire group but needs to know at least one member.

The centralized algorithm does not require a participant to talk to any other participants. It only needs to know how to contact the mutex (lock) server.
11. Which distributed mutual exclusion algorithm does not result in a higher number of requests (and hence network traffic and system load) when many processes want a resource at the same time?

(a) Centralized
(b) Lamport
(c) Ricart and Agrawala
(d) Token Ring

a. **Centralized**: you’ll have a request per message
b. **Lamport**: you’ll have a multicast per message (+ ACKs)
c. **Ricart & Agrawala**: you’ll have a multicast per message (+ACKs)
d. **Token Ring**: a token is always circulating from process to process, regardless of how many processes need a lock – even if no processes need a lock.
12. Which mutual exclusion algorithm creates replicated request queues on each process?
   (a) Centralized
   (b) Lamport
   (c) Ricart & Agrawala
   (d) Token Ring

   a. **Centralized**: Only the central mutex server keeps track of requests

   b. **Lamport**: Yes. Each request message is multicast to the entire group (including the requesting process). Each process keeps a queue of requests sorted by timestamp. If your process ID is at the top, you can access the resource.

   c. **Ricart & Agrawala**: Requests are multicast but if a process doesn’t have the resource then it just returns an ACK and doesn’t queue the request.

   d. **Token Ring**: No queuing – you can access the resource whenever you get the token.
14. Chubby presents itself to clients as this service:
   (a) Centralized mutual exclusion
   (b) Hierarchical mutual exclusion
   (c) Token-based mutual exclusion
   (d) Contention-based mutual exclusion.

   Chubby is a centralized lock manager and file system.
   A lock service enables mutual exclusion (grab a lock & access the resource)
   Chubby has replicas of its state for fault tolerance but only one server handles requests.
15. Differing from a token-based algorithm, a contention-based mutual exclusion algorithm relies on:
(a) Reliable message delivery
(b) Unique Lamport timestamps in request messages
(c) A coordinator process
(d) Constructing a logical ring of processes.

The two contention-based algorithms we looked at were Ricart & Agrawala and Lamport’s. The “contention” takes place when two processes want the same resource at approximately the same time.

With either algorithm, all processes agree that the process with a message containing an earlier timestamp gets to go first.

For the algorithm to work, it’s essential that all timestamps are unique.
16. The Chang & Roberts algorithm optimizes the ring algorithm by:
   (a) Using UDP instead of TCP for message delivery.
   (b) Testing higher-numbered processes first
   (c) Diving the ring into sub-rings and using a divide-and-conquer approach
   (d) Stopping multiple election messages from circulating.

Chang & Roberts is a refinement of the ring election algorithm and adds two optimizations:

1. Instead of affixing the process ID of every live process to the circulating message, the voting takes place with each received message so that the election message contains only one process ID (e.g., the highest or lowest process ID)

2. If a process gets an election message but has already processed one earlier (but that message did not circulate around yet), it may ignore the message to avoid multiple election messages circulating.
   If we’re voting for the highest # process ID and an election message comes in with a lower process ID than the one we already processed, ignore it.
17. A group of 10 processes (P0..P9) uses the bully algorithm to pick a leader with the highest numbered process ID. Process 6 detects the death of process 9 and holds an election. How many *election* messages are sent in the system as a whole (include failed messages to process 9)?

(a) 3  
(b) 6  
(c) 10  
(d) 45

1. Every process participating in the election sends an *election* message to all higher-numbered processes (some may not respond because they’re dead).

2. Each process that receives an *election* message sends an acknowledgement to the sender and holds its own election (step 1).

3. If no higher numbered process responds, than the process declares itself the winner.
17. A group of 10 processes (P0..P9) uses the bully algorithm to pick a leader with the highest numbered process ID. Process 6 detects the death of process 9 and holds an election. How many election messages are sent in the system as a whole (include failed messages to process 9)?

(a) 3
(b) 6
(c) 10
(d) 45

3 messages from P_6
2 messages from P_7
17. A group of 10 processes (P0..P9) uses the bully algorithm to pick a leader with the highest numbered process ID. Process 6 detects the death of process 9 and holds an election. How many election messages are sent in the system as a whole (include failed messages to process 9)?

(a) 3
(b) 6
(c) 10
(d) 45

3 messages from P6
2 messages from P7
1 message from P8
6 total messages
18. The two-army problem demonstrates that reliable communication with unreliable communication links:
(a) Can be achieved with $n^2$ message exchanges for a system of $n$ processes.
(b) Can be achieved with a simple message acknowledgement protocol.
(c) Requires a two-way acknowledgement.
(d) Cannot be achieved with 100% certainty.

Two army problem summary:

Two armies, A & B need to decide to attack the enemy. If both attack, they win. If only one attacks, it will die.

A sends a messenger to B: “let’s attack in the morning”. But A needs to know B received the message so it asks for a return message. If the return message does not arrive, A does not know if the return messenger didn’t make it and B got the message or if B never got the message.

If A receives a response, B isn’t sure if the messenger made it to A or not. B can ask for an acknowledgement to the acknowledgement but then A will not be sure if the second ACK made it to B. We can do this indefinitely…

The two-army problem demonstrates that you can never achieve certainty with unreliable asynchronous networks.

Since message delivery times are uncertain, you can never be sure that a system is not responding or the message is just taking a really long time.
19. Paxos reaches agreement when:
(a) All proposers agree on a value to send to the acceptors.
(b) All acceptors agree to a proposed value.
(c) The majority proposers agree on a value to send to the acceptors.
(d) The majority of acceptors agree to a proposed value.

This is tricky as stated:

- Paxos requires ALL acceptors to agree to a proposed value.
- However, only a MAJORITY of acceptors need to be running (quorum)

Paxos requires this quorum (majority of acceptors) to agree to a proposed value.

If any acceptor does NOT agree, that means that it has agreed to a higher value from some proposer (and the acceptors that do agree must have been down when that request came in).
A hierarchical lease:

(a) Allows clients to get both exclusive and shared leases.
(b) Allows multiple clients to request leases for parts of an object.
(c) Allows a client that has a lease for an object to get a lock for that object.
(d) Uses an elected coordinator to manage a set of leases.

A hierarchical lease is typically used to alleviate load from a centralized lease (lock) manager.

This top-level coordinator would give out coarse-grained leases: ones that last for a longer time and cover a big chunk of data, such as a table.

The second-level coordinator is responsible for handing out fine-grained leases for the objects it is responsible, such as rows within the table.
21. The purpose of the first phase in a two-phase commit protocol is to:
(a) Tell all processes participating in the transaction to start working on the transaction.
(b) Wait for all processes to commit their transactions.
(c) Find out whether processes are still working on the transaction.
(d) Get consensus from all processes participating in the transaction on whether to commit.

Phase 1: Query and get answers from ALL participants on whether they want to commit or abort

Phase 2: If 100% of participants vote to commit, then send the *commit* request. Else send *abort*. 
22. A three-phase commit protocol:
(a) Improves the consistency of the two-phase protocol.
(b) Tells the coordinator of the final commit vs. abort outcome.
(c) Sets time limits for the protocol.
(d) Gives cohort processes the ability to authorize the commit.

3PC overview on the next slide…
Three Phase Commit review

• The three phases are:
  1. Ask all participants if they can commit and wait for ALL responses (like Φ1 in 2PC)
  2. Tell all participants the outcome (commit or abort) and get ACKs … but don’t act yet!
  3. Tell all participants to commit
Three Phase Commit review

- The 3PC protocol accomplishes two things:
  1. Enables use of a recovery coordinator
     - If a coordinator died, a recovery coordinator can query a participant.
     - If the participant was in phase 2, that means that EVERY participant voted on the outcome.
       - The completion of phase 1 is guaranteed. It's possible that commits may have started. The recovery coordinator can start at phase 2.
     - If the participant was in phase 1, that means NO participant has started commits or aborts. The protocol can start at the beginning
     - If the participant was in phase 3, the coordinator can continue in phase 3 – and make sure everyone gets the commit/abort request
  2. Every phase can time out – no indefinite wait like in 2PC
     - Phase 1:
       - Participant aborts if it doesn’t hear from a coordinator in time
       - Coordinator sends aborts to all if it doesn’t hear from any participant
     - Phase 2:
       - If coordinator times out waiting for a participant – assume it crashed, tell everyone to abort
       - If participant times out waiting for a coordinator, elect a new coordinator
     - Phase 3
       - If a participant fails to hear from a coordinator, it can contact any other participant for results
22. A three-phase commit protocol:
   (a) Improves the consistency of the two-phase protocol.
   (b) Tells the coordinator of the final commit vs. abort outcome.
   (c) Sets time limits for the protocol.
   (d) Gives cohort processes the ability to authorize the commit.
23. Paxos avoids the “split brain” problem that can arise when a network is partitioned by:
(a) Placing proposers and acceptors on the same machine.
(b) Placing acceptors and learners on the same machine.
(c) Requiring over 50% of acceptors to be accessible.
(d) Using a two-phase commit protocol for each incoming request.

Because over 50% of acceptors have to be running, we avoid the problem of losing information.

Let’s consider the case of ≤ 50% acceptors running

• 4 acceptors: A, B, C, D
• Initially, only A & B run. They agree to accept request #100.
• They die and C & D come up. They don’t have A & B’s state
• C & D agree to accept a new incoming request #90 (but 90 < 100)

If a majority ran, then A or B would have to remain running and would reject a request of 90.
24. Which condition is not necessary for deadlock?
   (a) Mutual exclusion (a resource can be held by only one process).
   (b) Hold and wait (processes holding resources can wait for another resource).
   (c) Preemption (a resource can be taken away from a process).
   (d) Circular wait (a cycle of resource holding and waiting exists).

Four conditions must be met for deadlock to occur

1. Mutual exclusion – a resource can be held by just one process
2. Hold & wait – a process that wants a resource will wait for it indefinitely
3. Non-preemption – a process cannot temporarily release the resource
4. Circular wait – there is a circular dependency of Holds and Waits

The answer (c) is the opposite of condition #3.
25. False deadlock is caused by:
(a) Releasing one resource before waiting on another.
(b) Waiting on a resource before releasing one that is already held.
(c) Improper message ordering at the coordinator.
(d) Two processes competing to grab the same resource.

False deadlock is caused when a centralized coordinator receives two messages that may have been sent concurrently by two different processes (transactions): a *hold* message and a *release* message

It receives the “hold” message before it receives a “release” message for some resource and detects a circular wait condition.

Had the messages been received in the opposite order, there never would be deadlock.
26. The wait-die algorithm is a technique of deadlock prevention that:
   (a) Ensures that circular wait will not exist.
   (b) Relaxes the use of locking to avoid waiting on resources.
   (c) Introduces time-outs if a process cannot get a resource within a time limit.
   (d) Schedules transactions in a serial order so that only one runs at a time.

The wait-die and wound-wait algorithms both ensure that a circular wait cannot exist.

You don’t need to remember the distinction:

Wait-die:
   Allows an old processes to wait on a resource that a young process is using.
   A young process that wants a resource in use by an old process will kill itself (and restart later)

Wound-wait
   Allows a young process to wait on a resource that an old process is using.
   An old process that wants a resource in use by a young process will kill the young process
27. Compared with two-phase locking, strict two-phase locking:
(a) Guarantees that there is only one growing and one shrinking phase per transaction.
(b) Ensures that a transaction cannot access data written by an uncommitted transaction.
(c) Uses a two-phase commit protocol to get a lock.
(d) Makes the use of resource locks mandatory.

Two phase locking allows a transaction to release locks ONLY when it knows that it will not acquire any more locks.

This ensures that transaction output will have the illusion of serial execution even if they execute concurrently

BUT – if a transaction aborts then any transaction that read data from resources it released will have to abort

Three phase locking holds resources until the transaction is committed or aborted. There is no opportunity to read non-permanent data.
28. Optimistic concurrency control schemes usually allow multiple transactions to run concurrently and:
(a) Grab locks for resources they need.
(b) Avoid the use of locks.
(c) Use a distributed consensus algorithm to agree on a commit order.
(d) Replicate data for fault tolerance.

Optimistic concurrency control avoids the overhead of grabbing locks.

Instead, at commit time, it checks to see if another transaction accessed resources that it modified in an invalid manner.

Optimistic concurrency control is useful in cases where there is a low probability of conflicts.
30. DFS tokens are most comparable to:
(a) *Shared locks and write locks* in concurrency control.
(b) The token in a token-ring mutual exclusion algorithm.
(c) Getting consensus in a Paxos leader election algorithm.
(d) A callback promise in AFS.

Shared (read) locks allow multiple readers and no writers of an object.
Write locks give exclusive access to one process.

These are not the same as DFS tokens (or SMB oplocks) but the concept is similar:

DFS tokens give a client permission on whether it can cache data or not based on whether other clients are accessing that same data.
31. Commands sent to a Chubby cell:
   (a) Are load balanced among the machines in the cell.
   (b) Must be sent to and are processed by the current master.
   (c) Are executed by whichever machine gets the request.
   (d) Go to the master and are then forwarded to whichever Chubby replica holds the needed data.

A Chubby cell consists of multiple machines: one master and the rest are replicas.

All requests have to go to, and are processed by the master.

The cells periodically run an election to elect a master. If the master is dead, then a new master is elected.
32. Which of these operations is most efficiently implemented on a large-scale GFS system?
(a) Read one 1 TB file.
(b) Read 1 million 1 MB files.
(c) Write one 1 TB file.
(d) Write 1 million 1 MB files.

a. To read a 1TB file, a client will contact the GFS master and get a list of chunks that make up the file. Each chunk is associated with a list of servers that contain replicas of the chunk. Then the client contacts the chunk servers containing the needed chunks.

b. To read 1 million 1 MB files, the client will have to contact the GFS master 1 million times. There is only one master, so this becomes a point of congestion.

c. Writing is far less efficient than reading because each new block has to be allocated to a chunkserver and replicated on replica chunkservers. With the typical 3-way replication, we will be writing 3 TB of data

d. Same as (b) – we’re hitting the master 1 million times but this time writing 3 TB of data
The End