Distributed Systems

20r. Exam 2 Review

Paul Krzyzanowski
Rutgers University
Fall 2015

Question 1a

(a) State one advantage of early binding over late binding.

- Ability to cache the name resolution and use it multiple times.
- Reduced latency when the binding is actually needed.

Bad answer:

- *More efficient*

Question 1b

(b) State one advantage of late binding over early binding.

- You get the most up-to-date results.
  For example, better support for resources that move around.

Bad answer:

- *More efficient*

Question 2

How does virtual synchrony handle a fail-recover situation? That is, what happens when a failed node come back?

A node that fails is taken out of the group.

When it comes back, it has to re-join the group.

Doing this requires a state transfer from another group member.

Question 3

Eric Brewer discusses consistency, availability, and partition tolerance in the CAP Theorem. Explain how it applies to traditional ACID semantics and the two-phase commit protocol?

Note: the question does NOT ask to explain why ACID semantics are not always the best to use and why you might want BASE (eventual consistency) instead. It asks you to explain the trade-off in providing ACID semantics.

Availability is lost to preserve consistency & partition tolerance.

If a network is partitioned & some processes cannot be reached, the coordinator will wait, trying again & again.

Sub-transactions cannot commit and will hold on to their locks, sacrificing availability.

ACID: Locking & 2PC forces consistency across partitions.

Question 4

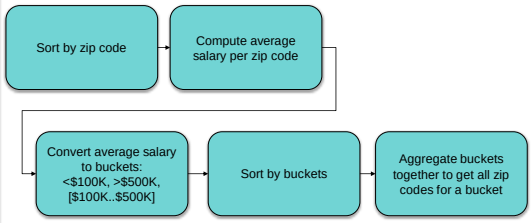
You have a collection of household information with each record containing information that includes:

{ name, age, address, zipcode, salary }

Use MapReduce to produce a list of zip codes where the average salaries are in the ranges: (1) under $100K, (2) $100K-500K, and (3) over $500K.

Explain the MapReduce operations needed to accomplish this. Use C/Java style pseudo code.
Feel free to use <= and >= comparisons for dates and assume that basic math operations and functions such as *average()* are available. Assume that the *map* function is called once for each parsed record of data and a function called *emit()* exists in the form emit(key, value). For reduce, assume that a print function exists that prints results; for example, *print(name, result)*. You will need to use MapReduce twice, using the output of one run as the input of the second.

Question 4 (continued)



Question 4 (continued)

{ name, age, address, zipcode, salary }

All we care about is zip & salary
1st map() will parse that out
We need to compute average salary per zip code ⇒ zip becomes key

{ 36310, $18K }	sort (so we can find average) →	{ 08540, $1514K }
{ 07931, $400K }		{ 08540, $82K }
{ 92651, $637K }		{ 08540, $949K }
{ 23451, $46K }		{ 08540, $700K }
{ 08540, $1514K }		{ 07620, $821K }
{ 08540, $82K }		{ 07931, $400K }
{ 07620, $821K }		{ 07931, $633K }
{ 07931, $633K }		{ 23451, $46K }
{ 36310, $48K }		{ 36310, $18K }
{ 36721, $22K }		{ 36310, $48K }
{ 92651, $135K }		{ 36721, $22K }
{ 92658, $300K }		{ 92651, $135K }
. . .		. . .

Question 4 (continued)

Map_1(name, age, address, zip, salary):
emit(zipcode, salary)
}
Reduce_1(key, list) {
print(key, average(list))
}

Each reduce worker gets
- Unique key (zip)
- List of values (salaries for the zip)

M-R framework does this

{ 36310, $18K }	sort (so we can find average) →	{ 08540, $1514K }	08540, { $1514K, $82K, $949K, $700K }
{ 07931, $400K }		{ 08540, $82K }	
{ 92651, $637K }		{ 08540, $949K }	
{ 23451, $46K }		{ 08540, $700K }	
{ 08540, $1514K }		{ 07620, $821K }	07620, { $821K }
{ 08540, $82K }		{ 07931, $400K }	
{ 07620, $821K }		{ 07931, $633K }	07931, { $400K, $633K }
{ 07931, $633K }		{ 23451, $46K }	
{ 36310, $48K }		{ 36310, $18K }	
{ 36721, $22K }		{ 36310, $48K }	
{ 92651, $135K }		{ 36721, $22K }	
{ 92658, $300K }		{ 92651, $135K }	
. . .		. . .	

Question 4 (continued)

Next – create buckets
Map_2(zip, avg_salary):
if (avg_salary < 100K)
emit("1", zip)
else if (avg_salary >= 500K)
emit("3", zip)
else
emit("2", zip)
}

Reduce_2(key, list) {
if (key.equals("1"))
print("Under 100K: ")
else if (key.equals("2"))
printf("100-500K: ")
else
printf("Over 500K: ")
for zip in list
print(zip, " ")
}

Input
08540, $811K
36721, $22K
07931, $516K
23451, $46K
07620, $821K
36310, $33K
92651, $135K
. . .

Output from map
"3", 08540
"1", 36310
"3", 07620
"1", 23451
"1", 36721
"3", 07931
"2", 92651
. . .

Input to reduce
"3", { 07620, 07931, 08540, … }
"1", { 23451, 36310, 36721, …}
"2", { 92651, … }
. . .

Question 5

Which of the following is an example of a pure name?:

a) An email address (e.g., pxk@cs.rutgers.edu).
b) An Ethernet address (e.g., 00:1d:72:ae:cd:c6).
c) A file pathname (e.g., /usr/src/linux-headers/arch/arm/include/asm/syscall.h).
d) A URL (e.g., http://www.cs.rutgers.edu/~pxk/417/exam/index.html).

- A pure name has no context in it.

Question 6

The DNS (Domain Name System) server for the EDU domain knows:

a) The list of names and owners of domains under .edu but not their IP addresses or name servers.
b) The IP addresses for all domains under .edu, including subdomains (e.g., cs.princeton.edu).
c) The IP addresses for all domains directly under .edu and name servers for subdomains.
d) The domain name servers for each domain directly under .edu.

- When you register a domain name, you supply the domain name registrar with your ID info and IP addresses for name servers for your domain
- The domain name registry operator keeps track of name servers for the domains registered under it; it sends referrals to those servers.

Question 7

The two-army problem demonstrates this about communication over faulty lines:

a) Reliable communication can never be achieved.
b) Reliable communication can be achieved but requires having the receiver send an acknowledgement.
c) Reliable communication can be achieved but may require several back-and-forth acknowledgements.
d) Reliable communication can be achieved only in one direction.

- To be 100% certain, you'll need an infinite # of back and forth ACKs

Question 8

In virtual synchrony, a *view change* is defined as the event that takes place when:

a) The system hosting the group membership service changes.
b) A different process starts sending messages.
c) A message has been received by all group members.
d) The group membership changes.

Question 9

In virtual synchrony, a message is considered *stable*:

a) When the sender promises not send any updates for that message.
b) Once it has been delivered to the application.
c) Once an incoming message has been fully received and validated to be error-free.
d) If it has been received by all group members..

- A message that has been confirmed to have been received by all group members is stable.

Question 10

In a Paxos system with *P* proposers and *A* acceptors, how many *proposers* need to be alive for the algorithm to work?

a) 1.
b) [(P+1)/2] (a majority of proposers).
c) [(A+1)/2] (one proposer for a majority of acceptors)
d) A (one proposer for each acceptor)

- Paxos can use any number of proposers – but only one needs to be functioning
  - Usually, one is elected as a *leader* and the others are not used
  - One proposer avoids message rejection due to proposing earlier sequence numbers

Question 11

In a Paxos system with P proposers and A acceptors, how many acceptors need to be alive for the algorithm to work?

a) 1.
b) [(P+1)/2] (one acceptor for a majority of proposers).
c) [(A+1)/2] (a majority of acceptors)
d) P (one acceptor for each acceptor)

- Paxos requires a majority of acceptors to be functioning
  - Ensures that at least one acceptor in the current group of acceptors has knowledge from previous proposals

Question 12

Which is NOT a property of a transaction?

a) All-or-nothing: a transaction cannot complete partially.
b) Available: a transaction cannot keep other transactions from running.
c) Serializable: the result of concurrent transactions must be the same as if they ran in some serial order.
d) Permanent: once completed, the results of a transaction are made permanent.

Availability is not part of transactional (ACID) semantics

(a) "all or nothing" = atomic

(c) "serializable" = isolated

(d) "permanent" = durable

Question 13

The first thing that happens in the second phase of the two-phase commit protocol is (assume it will commit):

a) The participant responds to the commit query message received in phase 1.
b) The coordinator sends a commit message to all participants.
c) The coordinator tells all participants to free resources held by the transaction.
d) The coordinator asks the participants if they have completed the transaction, which started in phase 1..

- Phase 1
  - Send a commit query to all participants
  - Get a response so we compute the outcome of the vote
- Phase 2
  - Send a commit/abort request to all participants
  - Get an acknowledgement that the commit/abort completed

Question 14

The purpose of an additional phase in the three-phase commit protocol is to:

a) Get unanimous agreement from all participants on whether to commit or abort.
b) Tell all participants the outcome of the commit vote before telling them to commit or abort.
c) Have the coordinator receive acknowledgement of whether a commit or abort was successful.
d) Log the status of the commit or abort decision to a write-ahead log.

- 3PC enables the use of a recovery coordinator even if some participants are down as well
  - Propagate knowledege of the commit vote BEFORE any participant takes action on it

Question 15

Differing from two-phase locking, in strict two-phase locking:

a) A transaction must promise to release every lock that it is granted.
b) A transaction must grab a lock for every resource that it needs to access.
c) A transaction must release all of its locks only at the end.
d) A transaction cannot acquire locks after it has released a lock.

- 2PL
  - Grab locks as you need them
  - Release locks once you don't need the resource anymore
  - BUT: promise NOT to acquire any lock if you already released a lock
- Strict 2PL
  - Don't release any locks until you're done with the transaction
  - Avoids cascading aborts

Question 16

The reason that strict two-phase locking was introduced as an enhancement of two-phase locking was to:

a) Increase the potential amount of concurrency possible.
b) Ensure that transaction executions are serialized.
c) Avoid cascading aborts when transactions read uncommitted data.
d) Achieve consistency by forcing transactions to get locks for any resources they need.

- With 2PL, a concurrent transaction may read data modified by another transaction if that transaction released its lock.
- That transaction may not have yet committed.

Question 17

The edge chasing used in the Chandy-Misra-Haas algorithm is used to:

a) Prevent deadlock by ensuring that a transaction can only wait on resources held by an older transaction.
b) Construct a global wait-for graph to determine whether deadlock exists.
c) Detect whether there will be a circular dependency on waiting for resources.
d) Ensure deadlock cannot occur by aborting the transaction that is currently using the needed resource.

- Send a probe message to the holder of the resource you want.
- That holder does the same …
- If the message comes back to you, then you know there will be a circular dependency if you get the lock (hence, deadlock)

Question 18

Deadlock will never occur if:

a) A resource can be held by at most one process.
b) Processes that hold resources are not allowed to wait for another resource.
c) A resource, once granted, cannot be taken away.
d) Two or more processes are waiting for resources held by one of the other processes.

Four conditions for deadlock
- Mutual exclusion
- Hold & wait
- Non-preemption
- Circular wait

(a) mutual exclusion – we might get deadlock

(c) non-preemption – we might get deadlock

(d) circular wait – we might get deadlock

(b) hold & wait is NOT permitted – the four conditions will not be met

Question 19

A callback in AFS:

a) Enables a server to control bandwidth by telling a client when it can issue a request.
b) Informs a client when its requested operation has completed.
c) Informs a client that a requested file lock is available.
d) Informs a client that a cached file is now invalid.

- When a client downloads a file, the server makes a *callback promise*
- When another client uploads that file, the server notifies each client that downloaded the file that its cached copy is invalid

Question 20

Which file system is not implemented at the operating system level?

a) Network File System (NFS).
b) Andrew File System (AFS).
c) Microsoft SMB.
d) Google File System (GFS).

- NFS, AFS, and SMB are available to normal apps and look like a native file system.
- GFS was designed to be accessible via user-level APIs

Question 21

Coda's client modification log enables:

a) Reintegration.
b) Read-write replication.
c) Consistent caching.
d) Disconnected operation.

- Coda supports disconnected operation (when the AVSG = ∅)
- At that time any file updates are logged in the client modification log (CML)
- When the system is re-connected to the network
  - The CML is played back to propagate changes to the server

(d) is correct in that the CML is used during disconnected operation.

- However, the CML is not needed to access files while disconnected
- (a) is the better answer since reintegration is the specific part of disconnected operation that the CML is designed to handle

Question 22

Credit-based flow control in SMB:

a) Allows the server to control the load from each client.
b) Provides a billing system for clients that sue a server.
c) Improves the level of concurrency in file system operations.
d) Enables the client to cache more data.

- The server gives each client a number of "credits"
- Each client request consumes a credit
- When the client runs out, it waits to get more credits from the server

Question 23

Which file system was designed to support concurrent appends to a file?.

a) NFS.
b) AFS.
c) SMB.
d) Coda.

- AFS & Coda: session semantics; no concurrent file modifications of any kind
- NFS: stateless design doesn't provide capability for appends

Question 24

An SMB oplock gives clients:

a) The possibility of caching file attributes.
b) The ability to lock a remote file.
c) The ability to check out a file for disconnected use.
d) The ability to synchronize local copies of remote files.

- Oplocks (similar to DFS tokens) give clients permission to cache file data
  Examples
  - Level 1: exclusive access – cache attributes, read-ahead, write-behind, cache lock data
  - Level 2: multiple readers, one writer – writer can cache file attributes & read-ahead

Question 25

Which file systems place file metadata on separate servers from file data?

a) GFS and Chubby.
b) Dropbox and GFS.
c) Chubby, Dropbox, and GFS.
d) AFS and GFS.

- Each AFS, NFS, SMB, DFS server stores complete file systems: data & metadata
- Chubby is a single-server file system that happens to be replicated
  - Each server contains all data
- GFS
  - Separate master server stores names and chunk IDs associated with each name
  - Actual data (which may be massive) distributed among chunkservers

Question 26

Why did Dropbox add notification servers to their architecture?.

a) To provide a mechanism to alert administrators when problems arise.
b) Dropbox servers needed to be notified when the file has been uploaded to Amazon servers.
c) To ensure that files are consistent among multiple clients.
d) To reduce load from clients polling the servers..

- Notification servers send notifications of changes to clients
- Before that, each client would poll the server

Question 27

Chubby does not support:

a) Partial file reads.
b) Event notifications.
c) Whole file uploads.
d) File locking.

- Chubby was designed to store small files (e.g., configuration data), provide locking, and notifications
- Since files are small, file I/O is only entire file reads & writes

Question 28

AS GFS master:

a) Identifies the addresses of all the name servers that keep track of file names and data.
b) Accepts every client request and routes it to the appropriate server.
c) Stores all the names in the file system along with the location of their data
d) Receives file write operations from clients that are then propagated to replicas.

- The GFS master stores metadata
- File content is distributed – and replicated – among chunkservers

Question 29

How does Bigtable manage the growth of a table?

a) Individual table cells are distributed among a large set of servers.
b) An entire column of a table can be migrated to a different server.
c) Each new row is allocated to one server in a large set of servers based on a hash of its key.
d) A table is split along rows into sub-tables.

- As a table gets bigger, it is split along a row into sub-tables (tablets)
- Rows in bigtable are always kept sorted by the row's key

(a) No. All columns of a row are kept together & adjacent rows are kept together

(b) No. Columns aren't split out and don't move.

(c) No. Rows are not distributed among servers; they stay together

The End