15-440 Distributed Systems
Fall 2009 Final

Name:
Andrew: ID

November 30, 2010

- Please write your name and Andrew ID above before starting this exam.
- This exam has 11 pages, including this title page. Please confirm that all pages are present.
- This exam has a total of 131 points.

Question	Points	Score
1	4	
2	4	
3	4	
4	4	
5	8	
6	16	
7	4	
8	4	
9	3	
10	6	
11	8	
12	15	
13	16	
14	20	
15	15	
Total:	131	

A Short Questions

1. (4 points) Which techniques are used in DNS to improve scalability?
   A. Replication
   B. Recursive queries
   C. Caching / Soft-State
   D. Partitioning of Namespace

Solution: A, C, D

2. (4 points) Which of the following are true about the Chord distributed hash table?
   A. Has O(log(N)) lookup latency
   B. Directly supports fuzzy or wildcard lookups for content
   C. Uses content-based naming
   D. Provides anonymity

Solution: A, C

3. (4 points) Which of the following are commonly provided in RPC libraries?
   A. Application-level failure handling
   B. Data marshalling/unmarshalling
   C. Asynchronous function calls
   D. Exactly-once semantics
   E. At-most-once semantics

Solution: B, E

4. (4 points) Which of the following are advantages of using content-based naming for p2p file transfers?
   A. The receiver can download a chunk using parallel TCP streams.
   B. The receiver can route around failed links.
   C. The receiver can download a chunk of data from any source, not just the original seed.
   D. The receiver can verify that it received the correct data.

Solution: C, D

5. (8 points) For each of the following distributed mutual exclusion solutions, which properties does that solution have? Circle all that apply.

(a) Central mutex server:	fair	reliable	long synchronization delay
(b) Ring-based algorithm:	fair	reliable	long synchronization delay

(c) Shared priority queue: fair reliable long synchronization delay

(d) Majority rules: fair reliable long synchronization delay

(e) Maekawa voting: fair reliable long synchronization delay

Solution: fill me in, bro!

6. (16 points) Which of the following statements are true? Please circle True or False:

(a) True False In Google File System (GFS), when a client needs to access data, it contacts the master and gets the data from the master.

Solution: False

(b) True False An NFS file system appears at the same file path on all clients.

Solution: False

(c) True False An AFS file system appears at the same file path on all clients.

Solution: True

(d) True False AFS provides sequential consistency by ensuring that writes are observed by all clients immediately.

Solution: False

(e) True False In AFS, servers transfer the entire contents of directories and files to clients in chunks.

Solution: True

(f) True False Using Paxos, suppose a majority of participants replies "Accept-OK" of some proposed value V_1. True or false: A (possibly different) majority of participants might later reply "Accept-OK" of a different proposed value V_2.

Solution: False

(g) True False Write-ahead logging can require twice as many disk writes as no logging.

Solution: True

(h) True False Recall the global "happens-before" relationship ($\rightarrow$) that captured the logical notions of time using both the local happens-before relationship at each server and the fact that a message must be sent before it can be received. True or false: For a Lamport logical clock L, if $e_1 \rightarrow e_2$ then $L(e_1) < L(e_2)$.

Solution: True

7. (4 points) Describe a transactional workload for which we should expect write-ahead logging (WAL) to achieve a throughput similar to as if no logging were being done. Your workload should contain some writes. If your workload consists of multiple transactions, be clear about which writes are done in each transaction. Explain briefly why you expect WAL to have similar performance. (One to three sentences should suffice for your entire answer.)

Solution: A workload consisting of many transactions, each writing to the same single disk page. Each transaction writes a single disk page regardless of whether WAL is used, and with WAL the disk page itself can be cached in memory while only the log is written.

8. (4 points) In the context of databases and distributed systems, what is ACID? Tell us what each letter stands for and give a short phrase or sentence explaining what each letter means.

Solution: Atomic (all or nothing), Consistent (if consistent before transaction then consistent after), Isolated (each transaction runs as if alone), Durable (once a transaction commits, the system will not undo the transaction)

9. (3 points) Very briefly (1-2 sentences) explain what the CAP theorem states.

Solution: You can have two of three: Consistency, Availability, and Partition-tolerance. But not all three.

10. (6 points) For each of the following scenarios, indicate which replication scheme you would use: either primary-backup or Paxos. Give a *short* explanation of why.

    (a) A lock server for a distributed filesystem that supports buying and selling stocks on the new york stock exchange.

    Solution: Paxos: The service must be extremely available and consistent, and it's very lightweight, so replication is reasonably cheap.

    (b) Storing data on data nodes in a distributed filesystem that stores your movie collection.

    Solution: Primary-backup: The data is large, so doing too many replicas is expensive. It's also not super-critical data nor does it need consistency.

    (c) A login server (accepts username/password, tells whether OK, and allows users to change passwords) used to support both of the above services, and many others.

    Solution: Paxos: The service is lightweight, so replication isn't too expensive, but is used by real-time and important services that can't go down.

11. (8 points) Dave wants to send Vijay a copy of the final exam over the network. Vijay and Dave have public/private keys and know each others' public keys. They also have a pre-shared secret key. These keys have not been compromised.

    Suppose students can intercept and modify packets on the network. For each of the following methods, circle whichever is appropriate to indicate whether the protocol is safe, whether the exam can be overheard, whether the exam can be modified, or whether it can be overheard and modified.

    (a) Dave sends the exam in plaintext.

    Safe Overheard Modified Overheard & Modified

    Solution: The exam can be stolen/replaced - no protection in place.

    (b) Dave signs the exam with his private key, sends it.

    Safe Overheard Modified Overheard & Modified

    Solution: The exam can be stolen - no encryption. Signing prevents replacement.

    (c) Dave encrypts the exam with Vijay's public key, sends it.

    Safe Overheard Modified Overheard & Modified

    Solution: Can be replaced - everyone has Vijay's public key. Encryption prevents stealing.

    (d) Dave encrypts the exam with Vijay's public key, signs it with his private key.

    Safe Overheard Modified Overheard & Modified

    Solution: Works...

    (e) Dave signs and encrypts the exam with a shared secret key.

    Safe Overheard Modified Overheard & Modified

    Solution: Works...

B Diffie Hell, Man

12. (15 points) Recall the Diffie-Hellman key-exchange protocol where two users, Alice and Bob, establish a secret, symmetric key:

    Alice generates a, g and p where p is a large prime and a is Alice's secret integer. She computes

    $$A = g^a \bmod p$$

    and sends g, p, and A to Bob. Bob, whose secret integer is b, generates

    $$B = g^b \bmod p$$

    and sends B back to Alice. The shared secret key, S, is then computed by Alice and Bob as $S = A^b \bmod p$ and $S = B^a \bmod p$ respectively. As discussed in class, everything sent in the clear cannot be used to discover the secret key S if p is a large prime.

    Extend the above Diffie-Hellman protocol to establish a secret shared key S between *three* parties, Alice, Bob, and Charlie. Explain what secret(s) each party has, what data is sent over the network at each step, and what computations each person performs to establish the shared secret.

    Solution: $A = g^a \bmod p$ $B = g^b \bmod p$ $C = g^c \bmod p$

    Alice sends A to Bob Bob sends B to Charlie Charlie sends C to Alice

    Alice computes SA as $C^a \bmod p = g^{ca} \bmod p$ Bob computes SB as $A^b \bmod p = g^{ab} \bmod p$ Charlie computes SC as $B^c \bmod p = g^{bc} \bmod p$

    Alice sends SA to Bob Bob sends SB to Charlie Charlie sends SC to Alice

    Alice computes S as $SC^a \bmod p = g^{bca} \bmod p$ Bob computes S as $SA^b \bmod p = g^{cab} \bmod p$ Charlie computes S as $SB^c \bmod p = g^{abc} \bmod p$

    Only $A, B, C, SA, SB, SC, p, g$ are sent in the clear. There is not enough information to discover $g^{abc} \bmod p$ using this information.

C Fail Whale

13. (16 points) Distributed systems often need to detect whether or not a server has failed. A *perfect failure detector* is a function `detectFailure(`C`)` that will eventually (correctly) return `true` if server C has failed, and `false` otherwise. `detectFailure` is allowed to require an arbitrarily long time: it must eventually return the correct answer, but it does not need to return within any fixed time bound.

    For the parts below, you may use a `send` primitive to send the messages you need (i.e., you don't need to use sockets or anything silly like that). Also, a remote system does not fail or restart while `detectFailure` runs; it is either up or down for the duration of your measurements.

    (a) Assume the network is reliable and synchronous. Is it possible to write a perfect failure detector? If yes, write one using any reasonable pseudocode. If no, prove it. Clearly state your assumptions.

    Solution:

    (b) Assume the network is reliable but asynchronous. Is it possible to write a perfect failure detector? If yes, write one using any reasonable pseudocode. If no, prove it. Clearly state your assumptions.

    Solution:

D Tweet, Tweeter, Tweetest

14. (20 points) Recall the statuses twitter dataset we provided for Project 3, which contained data of the form:

```
postid <tab> status-message
```

In Twitter, some status-messages are repeats of earlier status messages. These repeated messages are preceded by "RT" followed by the original status message. For example:

```
142412141    I need puppies but youtube is blocked.
584923144    Puppies are awesome!
929483834    RT Puppies are awesome!
872935923    RT Puppies are awesome!
423135125    Meow mix meow mix please retwitter
723917388    RT I need puppies but youtube is blocked.
629359237    RT Puppies are awesome!
562983948    RT Meow mix meow mix please retwitter
729385724    RT Meow mix meow mix please retwitter
912383444    Taking 15-440 final exam, lolol.
```

In this problem, your goal is to identify the most "retweeted" status messages using the MapReduce style of programming. You must write one or more Map and Reduce functions whose final output contains a sorted list of post ids that have been "retweeted"/repeated, **sorted by the number of times they have been retweeted**.

If run on the above example, the output would be:

```
1       I need puppies but youtube is blocked.
2       Meow mix meow mix please retwitter
3       Puppies are awesome!
```

Only retweeted statuses should appear in the output. The output should be sorted by the number of retweets from fewest to most.

You can use any reasonable pseudocode you choose (Java-like or C++-like); we will grade based on the algorithm, not having semicolons correct. You can use two types: Integers and Strings. You can assume the existence of STL-like string functions for your convenience. You can iterate over lists using the *for* construct as shown below:

```
for item in list {
   (do something with item)
}
```

You should use the "emit(key,value)" pseudocode for the output of functions.

```
// Assume line has been parsed into post id and status already
Map(int postid, String status) {









}


// Fill in the appropriate types in the brackets
Reduce(<          > key, List<          > values) {









}
```

Use the page below should you need more Map and Reduce functions. Declare them as above.

Solution:

```
Map(int postid, String status) {
  if (status.substring(0,2) == "RT") {
    emit(v.substring(3), 1);
  }
}

Reduce(String key, List<int> values) {
  int sum = 0;
  for value in values {
     sum += value;
  }

  if (sum > 0) {
    emit(key, sum);
  }
}

Map2(String status, int count) {
  emit(count, status);
}

Reduce2(int count, String status) {
  emit(count, status);
}
```

E Noah Way to Avoid Locking

15. (15 points) Due to global warming, your friend Noah needs to gather animals from all over the globe. Noah doesn't have much time, and must collect **exactly two** of each type of animal. From his computer science classes, he knows that he can make use of distributed systems techniques to accomplish his task. Noah realizes that he can split the earth into continents and have other people assist him in collecting animals.

    Noah has a created a server that contains a hashtable of animals that have already been acquired. He needs a way to regulate access to this hashtable, so that he gets only one pair of every animal. Write the following functions using mutexes and condition variables. Make them efficient (e.g., no spinning, no unnecessary waiting). Mutexes alone are not enough to ensure efficiency.

```
hashtable<string> animals;

/* this function should insert 1 animal into the hashtable
 * of the type specified by the variable "animal" */

void insertAnimal(string animal){









}

/* this function is in the case of animals accidentally dying.
 * it should delete 1 animal from the hashtable
 * of the type specified by the variable "animal" */

void deleteAnimal(string animal){









}
```