15-440/15-640: Homework 1

Due: September 27, 2016 10:30am

Name: _____

Andrew ID: _____

1. Alice wants to send files to Bob over the Internet at the fastest possible speed. To find the best network setting among those available to her, she transfers a 1000 KByte file over three different network settings. Calculate the total time taken to transfer this file in each of the following scenarios and help Alice decide which of those is the network setting she should use.
   For each of the following three scenarios calculate the total time taken to transfer the file assuming that the packet size is 1KByte, the RTT is 100ms, and that an initial 2*RTT of handshaking is performed before the actual data is sent.

   (a) Data packets are sent continuously on a 1.5Mbps bandwidth link **[5 points]**

   (b) After sending each data packet, there is a wait time of 1 RTT before sending the next packet. The link bandwidth is 1.5 Mbps. **[5 points]**

   (c) Up to 20 packets can be sent per RTT, and the bandwidth is infinite, i.e., the transmit time is zero **[5 points]**

Solution: Assume we are talking about transfer time from the perspective of transmitting at the source. We also accepted solutions that included propagation time to the destination.

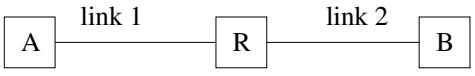
(a) $t = t_{handshake} + t_{propagation} + t_{transmission}$
(2 * RTT) + (0.5 * RTT) + (datasize / bandwidth)
$2 * (0.1s) + 0.05s + \frac{(1000*1024*8\ bits)}{1.5*10^6 bps}$
$= 5.71s$

(b) $t = t_{handshake} + t_{transmission} + t_{wait}$
(2 * RTT) + (datasize / bandwidth) + (1000 - 1) * RTT
$2 * (0.1s) + \frac{(1000*1024*8\ bits)}{1.5*10^6 bps} + 999 * (0.1s)$
$= 105.6s$

(c) Because $t_{transmission}$ is zero:
$t = t_{handshake} + t_{wait}$
(2 * RTT) + (49.5) * RTT
$2 * (0.1s) + 49.5 * (0.1s)$
$= 5.15s$

Common errors included not converting bytes to bits, forgetting the 2 * RTT in problems (b) and (c), and forgetting that 1 KByte = 2^{10} bytes.

2. Consider the following network topology, with hosts A and B connected through router R:



Router R operates in store-and-forward mode. Links 1 and 2 are both one megabit per second links with 10ms one-way latency.

(a) How long does it take to send a 1000 bit packet from A to B? **[5 points]**

(b) Recall that in Stop-and-Wait flow control mechanism, the sender sends a packet and then 'stops and waits' for an ACK before sending the next packet. Using stop-and-wait flow control, how long does it take for A to send a 100,000 bit file to B? Assume that the data packets are 1000 bits long. **[5 points]**

(c) What should be the minimum appropriate window size (the number of packets sent by the sender before waiting for an ACK) to utilize the full link bandwidth? **[5 points]**

(d) Generalize the appropriate window size W from part (c), for an arbitrary bandwidth B, a packet size N and an RTT R. **[5 points]**

Solution:

(a) prop_delay_packet = 1000 bits /(10^6 bits/s) = 0.001s = 1 ms
latency = 10ms
packet_time = 2*(prop_delay_packet + latency) = 2*(1ms + 10ms)
= 22ms

(b) packet_time=22ms
Assuming ACKs are negligible in size:
prop_delay_ack=0 bits/(10^6bits/s)= 0 ms
Ack_time = 2*(prop_delay_ack+latency) = 2(0 + 10) = 20ms
Number of packets = 10^5 / 10^3 = 100
Total time taken = 100*(packet_time+ack_time) = 100*(22ms+20ms) = 4200ms
= 4.2s

(c) Bandwidth-delay product = 10^6 bits * 10 ms * 2 = 2 * 10^4 bits
For 100% utilization, it will be = 1.0 * 2 * 10^4 = 2 * 10^4 bits
Window size = 2 * 10^4 / 10^3 = 20 packets

(d) Bandwidth-delay product = B * R
For 100% utilization, it will be = 1.0 * B * R = B * R
Window Size = B * R / N packets

3. Answer the following with respect to Remote Procedure Calls [**8 points**]

   (a) Describe two ways in which RPC calls differ from local procedure calls

   (b) Describe how you could help deal with each of the differences listed in part (a).

Solution:

(a) Any two of: Call by value vs Call by reference Performance Differences OS/Language Differences Failure Semantics

(b) This can have many answers depending on the response in part A. Any answers with good justification should get credit. Possible Answers: Call by value vs Call by reference can be helped by minimizing the amount of large data structures being passed in rpc calls, or using copying/restoring for data structures instead of passing references.

Failure Semantics can be helped by minimizing state on the server, and by using RPC semantics such as at-least-once or at-most-once.

4. In the following situations, describe which RPC semantic is the most suitable (at-most-once or at-least-once), and why.[**8 points**]

   (a) Saving a file in a distributed file system

   (b) Buying a laptop online

   (c) Checking the price of a stock

   (d) Uploading a blog post

Solution:

(a) At-least-once. After saving the file once, the action becomes idempotent. Sending multiple calls to have the file saved does no harm to the client, and the client would want to ensure that the file is actually saved.

(b) At-most-once. The client can always try to repurchase the laptop if the previous attempt is not successful. The client would not, however, want to accidentally buy multiple laptops.

(c) Either: At-least-once. The act of checking for a stock price is idempotent, and no global state is changed. At-most-once. The client can always recheck later, and indefinite blocking is avoided.

(d) Either is acceptable: At-least-once. It wouldn't be too harmful for identical blog posts to be posted in a row. The client could simply delete any copies. At-most-once. The client can always repost the blog if the previous one is not successful; in addition, indefinite blocking is avoided.

5. There is a classical problem in concurrency known as the "Dining Philosophers." The problem is as follows: There are 5 philosophers sitting around a circular dining table. Each philosopher has a bowl of rice in front of him, and there is a single chopstick in between each two philosophers. Each philosopher alternates between thinking and eating. A philosopher needs a pair (i.e. fork to their left AND right) of chopsticks in order to eat. When the philosopher is done eating and goes back to thinking, he will put down his chopsticks.

(a) Given that this is an example of a concurrent system with a need for synchronization, a solution should aim for i) correctness, ii) efficiency, and iii) fairness. Describe what each goal means in the context of the Dining Philosophers problem. **[3 points]**

`Step 1:` think until the left chopstick is available; when it is, pick up;
`Step 2:` think until the right chopstick is available; when it is, pick up;
`Step 3:` when both chopsticks are held, eat for a fixed amount of time;
`Step 4:` then, put the right chopstick down;
`Step 5:` then, put the left chopstick down;
`Step 6:` repeat from the beginning.

(b) Does this solution address the desirable properties of concurrent systems listed above? If not, which one is violated and give an example of when it is violated? **[6 points]**

(c) Assume that we change the solution outlined in part (b) above such that there is an additional waiter who allows only 4 philosophers at the table at any given time. The waiter will only allow another philosopher to join once there are <4 philosophers at the table. Is there a desirable property of concurrent systems that is still violated? If so, give an example of when it is violated. **[6 points]**

Solution:

(a) Correctness: No two philosophers should be using the same chopsticks at the same time. Efficiency: Philosophers do not wait too long to pickup chopsticks when they want to eat. Fairness: No philosopher should be unable to pick up chopsticks forever and starve

(b) This can violate efficiency. If every single philosopher reaches for the left fork at the same time, then every philosopher is stuck waiting for the right fork, which results in a deadlock.

(c) This can violate fairness. If the 4 philosophers at the table at any given time never leave, then the philosopher not at the table will never have a chance to eat.

6. After forgetting your AFS password, you decide to create your own distributed file system. personal distributed file system to synchronize your files between your different devices.

(a) You want to design your distributed file system for personal use so you can synchronize your files between your different devices. Of the three main tradeoffs of distributed file systems: consistency, performance, and scalability, which would you optimize? Describe one way you would optimize this goal. **[5 points]**

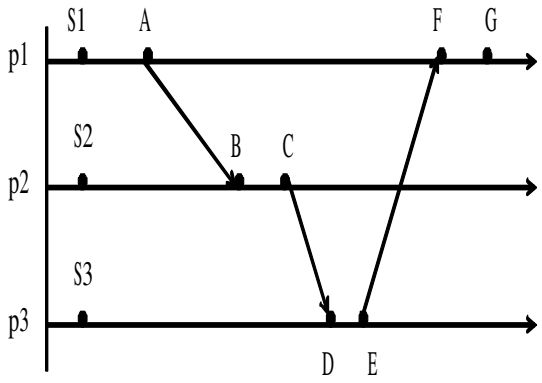
(b) In order to work on group projects, you decide to open up your distributed file system to your group members. To ensure your work is not lost, you wish to replicate data in other servers. However, you have trouble deciding how data replication should be handled. Name one benefit of using AFS's data replication method and one benefit of using Coda's data replication method. **[6 points]**

Solution:

(a) Performance. Since it is only for personal use, consistency would not a big of an issue (no write-sharing if there's only one user), and scalability would not matter unless the user plans to store a high amount of personal files. The user would want to optimize speed and availability of their files. Other answers are okay if justification is provided.

(b) A benefit of AFS's data replication is that consistency is better ensured. This would be especially useful if multiple people are editing the same files for the group projects. A benefit of Coda's data replication is that availability is better ensured. This would be useful if any group member had to work on a file without having connection, or if there were a network partition.

7. Processes 1, 2, 3 are using a logical clock to keep a consistent time. The horizontal arrow represents physical time and the crossing arrow represents a message being passed. Each point on the horizontal lines is an event.



(a) Andrew finds out that the processes above are using Lamport's logical clock, but not all the clock values are not known to him. The initial logical time S1, S2 and S3 were 11, 1 and 0 respectively, and Andrew observed that the clock was 16 at E. What is the value of the clock at A, B, C, D, and F? **[5 points]**

(b) This time, Andrew finds out that the processes above are using vector clocks, but not all the clock values are not known. The initial logical time S1, S2 and S3 were (0,0,0) , (0,1,0) and (0,0,11) respectively. What's the clock values for A, B, C, D, E and F? **[6 points]**

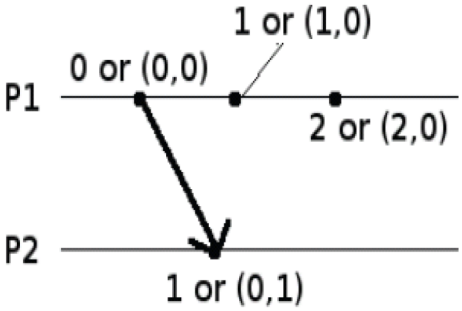
(c) Give an example that highlights the shortcomings of Lamport's clock in comparison to vector clocks, i.e., the Lamport's algorithm cannot clearly conclude that an event e happens before e', even though $L(e) < L(e')$, where $L(e)$ is the Lamport's timestamp of the event e and that the vector clock algorithm concludes clearly whether an event e happened before e' or not. **[4 points]**

Solution:

(a) A=12 B=13 C=14 D=15 F=17

(b) A=(1,0,0) B=(1,2,0) C=(1,3,0), D=(1,3,12), E=(1,3,13), F=(2,3,13)

(c) Consider the following example in the figure of 2 processes. The Lamport timestamp 2 in P1 is clearly larger than Lamport timestamp 1 in P2, but the corresponding events are not in happen-before relation. On the other hand with vector timestamps (2,0) and (0,1), corresponding to those two events, we can clearly conclude that these two events are not in happen-before relation (since VT1[P1] >VT2[P1] and VT1[P2] <VT2[P2]).



8. An NTP server B receives server A's message at 11:25:12.350 bearing a timestamp 11:25:2.300 and replies to it. A receives the message at 11:25:4.620, bearing B's timestamp 11:25:14.600. Estimate the offset between B and A and the accuracy of the estimate. [**8 points**]

Solution: Let a = Ti-2 – Ti-3 = 12.35 - 2.30 = 10.05, b = Ti-1 – Ti = 14.60 - 4.62 = 9.98. Then the estimated offset o_i = (a+b)/2 = 10.015s, with estimated accuracy = ± d_i / 2 = ± (a-b)/2 = 0.035s (answers expressed to the nearest millisecond).