Name: Kaiwen Sun

CSE 291 (Distributed Systems)
Spring 2016 (Kesden)
Homework #1

1. Networks

(a) Given a 100Mbps network with an optimum sliding-window size of 1024 bytes, what would be the optimal window size if the bitrate was increased to 1Gbps and the maximum allowable run of the cable was cut in half? Why?

The round-trip time is T = 1024bytes/100Mbps seconds to send all data in the window. Immediately after finishing sending, the ACK of the first sending comes back. If the maximum allowable run of the cable was cut in half, the rout-trip time was cut in half, i.e. T/2. During T/2 seconds, a 1Gbps bitrate can send **1024bytes/100Mbps/2*1Gbps=5120bytes** data.

So the optimal window size is 5120bytes.

(b) When transmitting at the link layer, we normally "frame" data. Why? For example, why do we not just stream bits?

By framing data, we can figure out the start and end of a unit of data (frame), can manipulate the unit of data easily, especially when we want to detect link layer's bit error, and fix the error by using check sum or retransmit. At the link layer, it is very hard to manipulate a stream of data if we are unsure about the start, end and length of stream.

(c) Consider global scale communication over the next 30 years. In which dimension, latency or bandwidth, will we mostly likely see the greatest improvement? Why?

I think the **bandwidth** will be greatly improved, because it is relatively easier to be improved by smart encoding, higher frequency, or simply more wires. But the latency is hard to be improved because so far we don't know anything that runs faster than light, and unlikely to know in next 30 years.

2. Middleware/RPC/RMI

(a) Consider the implemesntation of an RPC system in a homogenous environment (same hardware, same OS, same language, same, same, same). Is it possible to implement a pass-by-reference (not necessarily pass-by-address) mechanism? If not, why not? If so, in what ways might it be best to relax the semantics of a typical local pass-by-reference situation? Why?

Yes, it is possible to implement a pass-by-reference mechanism. I can think of two approaches:

One approach is to receive the changed argument from server, and write the updated argument to where the reference points to.

The other approach is to implement a pass-by-address mechanism by implementing distributed shared memory, or a global page table, such that the address is globally valid. But this approach is less efficient.

(b) Consider the implementation of an RPC system in a heterogeneous environment (different processor architecture, different OS, different programming language, different, different, different). How might the heterogeneity complicate the model? Please consider each of the following:

i) Simple primitives (consider endian-ness, width, etc)
ii) Complex data types, including structs and strings,
iii) Higher-order language and library data structures, such as linked lists, maps, etc.
iv) Programming paradigms (function pointers, jump table, functors, etc)

i) Simple primitives
The compatibility of primitive types is **relative easy**, because the number of primitive types is quite small . For each primitive type, we can manually design a “global format” understood by every computers. Each computer translates its local format to and from the globally agreed format. For example, the endianness problem can be resolved by agreeing on the “network byte order”. All data sent to and received from network should be in “network byte order”, and thus understood by every computer. Similarly, width problem can be resolved by agreeing on a format that can interpret all primitive types and understood by every machine.

ii) Complex data type, including struct and strings
The compatibility of complex data type is **a little bit harder**, because the number of types if complex data is infinite. But it is still easy to find a way to describe a globally agreed format. A struct or array is a collection of sub-struct, sub-array and/or primitive, plus some meta data such as the number of fields of a strut, the length of an array, etc.. It is easy to use a formalized (means globally understood) way to describe or list what sub-struct/sub-array/primitive are contained in this struct/array, and recursively describe the sub-struct/sub-array until reaching primitive or forming a loop.

A string can be interpreted as an array of letter/char. The compatibility of array can be achieved, so can that of string.

iii) Higher-order language and library data structures, such as linked lists, maps, etc.
It is hard to achieve the compatibility of higher-order language and library data structures. For example, the implementation of maps is map in C++, HashMap in Java, dict in Python. Their inner implementations are very different. Also, the operations on these data structures are different too. It is not always possible to translate from one library to another. But since we can achieve compatibility of struct, array and primitive, it is applicable to use them to describe higher-order language's functionality and more complex data structures, regardless of the library implementation.

iv) Programming Paradigms
The programming paradigms of programming languages can be very different, and thus it is very hard to achieve the compatibility. For example, it is easy to do type conversion in a weekly typed language, but it is hard in a strongly typed language. How can you translate the type conversion from a weekly typed language to a strongly typed one?
But theoretically, every program is describing a Turing machine, which means they are equivalent. One approach is to compile, a Java code for example to machine code, and decompile the machine code into a C code. But obviously this is very hard. Much much harder than agree on global format of primitives.

(c) Consider Java's RMI facility, which generates stubs at compile time. Could it, instead, generate the stubs at runtime? For example, could it disassemble a class file, or inspect an object's properties at runtime, rather than at compile time? If not, why not. If so, what would be the advantages and disadvantages of this model?

Yes, it could. We can use java reflection and dynamic proxy to achieve it at runtime. The advantage of doing it at runtime is that the stub is more flexible and generous purpose. The RMI factory generate any types of stub. However, the disadvantage is that using dynamic proxy reduces the execution efficiency and slows down the program.

(d) Consider Java's RMI facility, which only plays nicely with classes that implement the *Serializable* or *Remote* interfaces. Would it be possible to implement an RMI facility in Java that worked for all classes? For example, by using a combination of the class file, as well as reflection and other Java mechanisms to decompose, serialize, and reconstitute instances by brute force? If so, please explain any necessary limitations. If not, please example why not.

No, it is hard to implement RMI for unserializable classes. Writing to/reading from network is an operation on a one dimensional stream of data. It is impossible, for example, to directly write a graph/heap/tree/table to a one dimensional network stream.

3. Distributed Concurrency Control

In class we discussed enforcing mutual exclusion, among other ways, via a central server, majority voting, and token ring.

(a) Which of these systems requires the fewest messages under heavy contention? How many messages are required per request?

Token ring. Under heavy contention, every one wants to enter the critical section. Once a node receives the token, the node makes use of the token. So, with token ring, one message is required per request.

(b) Which of these systems requires the most messages under heavy contention? Why?

Majority voting requires most messages.
Comparing the three ways:
- Token ring: 1 message per request. (passing token)
- Central server: 2 messages per request. (request and reply)
- Majority voting (without retrieving): at least 2 × #voters per request. (1 soliciting and 1 voting per voter)

(c) Which of these systems is most robust to failure? Why?
Token ring is the most robust. Even if one or more nodes loose connection to the partition of nodes holding the token, they can generate their own new token and passing it. The original token can be passed ignoring those nodes which loose connection.

(d) The *voting protocol*, and the *voting district* protocol, are based upon participants reaching an agreement as to who can enter the critical section. What happens in the event of a tie that could otherwise risk deadlock?
If there is tie, we can use a **total ordered time stamp** to determine the priority of requests. If the a voter has voted before it receives a higher priority request, it can **relinquish its previous votes**. Finally the vote will goes to the host with higher priority, and breaks the tie. It doesn't matter when a relinquishing is refused due to the less prior host's already being in the critical section, because if that occurs then the tie has already been broken.

(e) Many coordinator election protocols have analogous mutual exclusion protocols and vice-versa. What is the most important similarity of the two problems? What is the most important difference? Focus your answer on the nature of the problem, itself, not the solution.

Similarity: Both coordinator election and mutual exclusion want to select a globally special and unique computer.
Difference: Coordinator election requires all nodes to know which one is the special node. Mutual exclusion only requires each node to know whether itself is special or not.

(f) In the event of a partitioning, techniques such as token ring can result in two or more distinct groups, each with its own coordinator. How can this be prevented, while enabling progress?

If we can guarantee the token will never be lost, then we can disable generating new tokens, so that only the group holding the token can have a coordinator. However, this mechanism may not provide a good performance, since the group holding the token may be very small.

So another approach is to allow only the majority group to elect a coordinator. If there is a tie among multiple majority groups, allow only the majority group having a particular computer to elect a coordinator.

4. Logical time

(a) One form of logical time is per-host sequence numbers, e.g. “5.1” is time 5 on host 1. Another form is Lamport’s logical time. What advantage does Lamport time offer? At what cost?

Advangate: The advantage of Lamport time is the relationship “happen-before”. With the help of host-id, it defines a total order among global events.
Cost: t requires sending one timestamp along with each message. It may cause “missing time” (one node at local time 3 receives a message sent at the sender’s time 6). Also it can’t detect potential causality violation.

(b) In class, we discussed a simple form of vector time. What relationship(s) among timestamps can be discerned from this form of vector logical time, but not Lamport logical time?

Vector logical time can detect **(potential) causality violation**.

(c) Consider the amazing progress we've made in data communication networks over the years. Will it –ever-- be possible to sync physical clocks rapidly enough to use as the basis for correct synchronization of a global distributed system? Why or why not?

No. It is very unlikely. Because the communication speed is very unlikely to be faster than the speed of light. Regarding the worldwide distance, **the latency can't be ignored**.

5. Replication and Quorums

(a) Consider replication to multiple servers based upon a write-one, read-all static quorum with version numbering. What steps must be taken upon a write to ensure the correctness of the version number?

The writer has to check the version number in a read quorum, in order to correctly decide the latest version number.

(b) Consider Coda Version Vectors (CVVs). Give an example of two concurrent CVVs. What is indicated by concurrent CVVs? Please describe one situation in which this might occur.
A conflict exists if two CVVs are concurrent, because concurrent vectors indicate that each server involved has seen some changes to the file, but not all changes.

For a system with 4 servers, #1&#2 in one partition, #3&#4 in another partition.

Original CVV:
[1,1,1,1], [1,1,1,1], [1,1,1,1], [1,1,1,1]

Concurrent write to #1&#2, and another write to #3&#4:
[2,2,1,1], [2,2,1,1], [1,1,2,2], [1,1,2,2]

After partition repaired:
[2,2,1,1] > [1,1,2,2] in the sense of the first entry, but [2,2,1,1]<[1,1,2,2] in the sense of the fourth entry. Conflict.

(c) Consider a situation with 5 replica servers employing a write-2/read-4 policy and version numbering. Please design and describe a locking scheme that ensures that version numbers remain correct, even in light of concurrent writes. You do not need to consider failure. But, you do need to describe all necessary communication and state, such as communication between or among servers and/or participants.

The lock should be of size l = max(w,r) = max(2,4) = 4. Pick any 4 nodes, lock them to prevent their version number from being updated by other writers, then read version numbers from them. Since l = r and r+w>n, the latest version number is guaranteed to be read in. Then pick 2 of the 4 nodes, write the increased new version number to the 2 nodes. Since all 4 nodes including the 2 are locked, no write-write conflict will occur. Finally, release the lock on the 4 nodes to finish the entire operation.

In addition, since r+w>n and l = max(w,r), no other read or write operation will be completed during the l nodes are locked. So no read-write or write-write conflict will occur.

(d) Please consider the special case of a mobile client using a read-one/write-all quorum. Please design an efficient locking protocol to ensure concurrency control. But, please ensure that your protocol permits the lock to be acquired at one replica and released by another.

A mobile client such as a smart phone may be relatively slow. So in order to make the locking protocol efficient, the following ideas can be applied:

i) Send a request to a fast server, and let the fast server to complete the write-to-5 operation on behalf of the mobile client.

ii) The write quorum is equal to “all”, whenever there is at least one node is being locked, no other writer can writes; whenever a writer acquires the lock, no other writers or readers can block the write. So a writer can release a node node right after it finishes writing the node. It doesn’t have to release all nodes at once after finishing all write. When the write quorum is partially released, a reader, which acquires only one node to be locked, can read the latest version from one of the released nodes.

(e) Coda Version Vectors (CVVs) are a form of logical time stamp, specifically a vector time stamp, which are used to aid in the management of replication. What could cause two CVVs be *concurrent* (and non-identical)? Why? Illustrate your answer with an example involving 2 servers and 2 clients.

If client1 writes to server1 and client2 writes to server2 concurrently, then the there would be two different new versions. When a client reader wants to read the latest version, it will find two different version with identical version number.