Name: AndrewID:

15-440 Homework #1
Fall 2013 (Kesden)
Due: Tuesday, 9/26/2013

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 sliding window size is related to the product of the bit-rate and round-trip time. When the bit rate is increased by 10x, the optimal size is also increased by 10x. Since the RTT is a function of the propagation time, which is proportional to the length of the cable, when the length is cut in ½, the optimal window size is also cut in ½. So, we have (10x * 1/2x) = 5x.*

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

Framing is crucial. Unless we have some periodic synchronization point, such as gaps between frames and/or special bit patterns, the sender cannot figure out when a message starts or ends. It may start listening in the middle and never sync with the sender. Additionally, we need discrete units for other aspects of management, such as retransmissions, computing checksums, etc.

(c) In class, we discussed IP multicast. Does it guarantee that messages will be received by all hosts at the same time? What about in the same order? Why or why not?

Certainly not. IP multicast is based upon IP, which is a best-effort protocol. These guarantees would require some type of atomic or totally ordered protocol. And, those are very slow and expensive!

(d) In class, we discussed IP multicast. Is it typically an option for communications in global-scale distributed systems? Why or why not?

Nope. It is usually only enabled within an enterprise. Once we get out of an administrative domain, it is an insecure mess and also consumes a lot of resources at the routers, making it a potential tool for DoS attacks.

2. Middleware/RPC/RMI

(a) Consider the implementation 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?

This was a actually a soft ball in disguise. Consider the Remote Object References (RORs) that you are implementing for your project, and the "model" implementation in Java. RORs don't function exactly like local references, because of the latency of the network (including retransmission) and the possibility that the other end is down. So, we have to be able to handle time-outs and other remote and network related error conditions.

(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 (think back to 213 for how they can differ from system to system)

"Minor" issues such as byte ordering, data type sizes, and numerical properties such as behavior upon overflow or shift become "fun". They can limit use to the lowest common denominator and require some translation, e.g. byte ordering.

ii) Complex data types, including structs and strings

Now we get into all sorts of different conventions and translation. Representation, padding, end-of-string conventions, etc. Manageable – but more work and more complexity.

iii) Higher-order language and library data structures, such as linked lists, maps, etc.

This rapidly becomes a lost cause. What looks similar to a human may be very different in functionality and implementation. We can try to define common forms and routines to translate into and out of them, but we are going to lose in the translation.

iv) Programming paradigms (function pointers, jump table, functors, etc)

This is almost certainly a lost cause. Function pointers and jump tables are about as idiosyncratic as they come. And, functors are an abstraction that is very rich and model dependent. We might be able to agree on the serialization of a simple linked list. But, once we get into paradigm issues, nuance matters.

(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. The advantage is that it makes it really clean for the programmer. Everything is just magic. At runtime, everything gets inspected with reflection, etc, and built. The disadvantage is that it is very slow – it has to be done at runtime, every time.

(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.

Nope. There is no way of knowing whether the program wants a local copy or a remote copy, unless the programmer tells us. A solution where everything is remote isn't really practical. And, if everything is serialized, we just have a client-server RPC system, not an RMI. Beyond that, we can't really know if we should make shallow or deep copies of structures, etc, unless the programmer tells us – it is something that is inferred from the scope of use, not the definition.

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 requires the fewest messages of these options under heavy contention. If every participant requires the critical section, there is only one message per request as the token gets passed along.

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

Voting. Each participant is making a request that initiates a reply from every other host. Additionally, under heavy contention, nodes also end up requesting votes back, which requires another pair of messages per out-of-order request.

(c) Which of these systems is most robust to failure? Why?

Token ring can function with any number of losses – be it slowly. It is worth noting that a majority vote scheme only requires a majority and is reasonably robust.

(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?

A node's id serves as its priority. Should a node notice that it has voted for a lower priority requestor, it requests its vote back. In the case where that node has not yet entered the critical section, which is the only case that risks deadlock, it gives the vote back. Since there are no cycles in the precedence of node ids, no circular wait is possible.

(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.

Each of the two tasks picks one of many participants to be special. The essential difference is that everyone needs to know the coordinator, whereas with mutual exclusion the participants just need to know "in" or "not in".

(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?

The quick fix in most cases is to require a majority to keep playing.

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?

Lamport time provides a very limited guarantee about the ordering of time stamps among hosts. It does this at a very low cost, because it does it only when messages sent for other purposes are received.

(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?

Causality/Potential Causality. In other words, we can discovery the relationship between messages sent and events after they are received.

(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?

Nope. Correctness in light of concurrency requires synchronization at the instruction level. Unless the speed of light gets faster or communication leaves the emag spectrum, there is, and always will be, too much latency. It would take dozens of milliseconds for light to travel the circumference of the earth, even in a vacuum.

5. Databases and Friends

(a) What is the *CAP conjecture [Theorum]?* Please expand the acronym and explain it, and also explain the intuition behind it.

See the lecture notes. This is taken directly from the lecture.

(b) What does *BASE* stand for*? Please also explain each property.*

See the lecture notes. This is taken directly from the lecture.

(c) What does *ACID* stand for? Please also explain each property*?*

See the lecture notes. This is taken directly from the lecture.

(d) Please consider *BASE* and *ACID*. Please provide general guidance of the circumstances under which each is a uniquely appropriate model. Please also provide and explain one concrete example for each case.

ACID is appropriate any time the answer must be correct now and/or in the future, and future correctness may depend upon the present action.

BASE is appropriate when stale answers may be acceptable now and in the future.

(e) Assume a participant in a 2PC aborts after acknowledging a pre-commit, how does the system ensure correctness?

Upon recovery, the system consults the log, sees the pre-commit, contacts the coordinator (or other participants), finds out if the request ended in a commit or abort, finishes the operation accordingly, and resolves all such outstanding operations in this way before making each affected item available.

(f) Consider a distributed transaction implemented via *2PC*, at what point does a resource used by a transaction become unavailable? What about become available again? Why is this locking necessary?

It becomes unavailable with the pre-commit and becomes available again with the commit or abort. Otherwise, we can end up with interleaved transactions.

(g) How does *2PL* ensure that deadlock is not possible?

2PL is based on enumerating the resources and acquiring resources only in increasing order. Since one can never hold a higher-valued resource while waiting for a lower-valued resource, cycles are not possible. Given this, there can't be a circular wait. Deadlock is not possible.