Name: AMIT BORASE (A53095391)

**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?

**Answer:** Optimal window size will be 5 times the original window size which is 5120 bytes.

Let L be the original latency. Since the latency is directly proportional to the length of the cable. The new latency L' will be half the original latency. ( L' = L/2).

Now, Optimal window size = Latency x Bandwidth

Hence, 1024 bytes = L x 100Mbps ----- equation: 1

New optimal windows size = L' x 1Gbps = L/2 x 1000Mbps

= 5 x (L x 100Mbps)

= 5 x (1024 bytes) = 5120 bytes ----- using equation 1

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

**Answer:** Data link layer 'frames' data instead of streaming it to the destination. Because,

- Framing allows the data to be broken into small chunks called frames which can be individually packaged with well-defined boundaries and transmitted to the destination in an ordered fashion. In cases of transmission failures or loss of any such frame (arising due to collisions etc), we only have to re-transmit the concerned frame and not the entire data. If we were to use streaming instead of framing, we would not need entire data to be re-transmitted in case network loss of the portion of the data.
- With framing, error checking and error correction can be performed for individual frames by stuffing in the error detection and correction codes inside the frame, which are then consumed by the receiver to verify the validity of the received frame. The same is not possible easily if the data is to be streamed bits-by-bits.

(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?

**Answer:** The greatest improvement will be seen in bandwidth of the communication networks. This is because, the latency of the network is dictated by the medium propagation speeds, which are limited by the speed of light, and we already have optical fibre based networks that operates at fraction of the speed of light. Hence, no drastic improvement is possible w.r.t. Latency. Whereas, bandwidth of the network can be improved by laying more efficient and cheap submarine communication cables. Additionally technologies such as optical IP-switching will further improve it.

## 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?

**Answer:** Yes it is possible to implement a pass-by-reference mechanism. It can be done by assigning a key to each object that is globally unique across the RPC system and uniquely locates the remote object. This key can then be passed around to clients as a reference to remote object. But such approach requires us to implement additional mechanism that make this unique key consistent upon the changes to the remote object. This can be achieved by using leases or locking, so that in case the key needs to be modified as a result of modification to remote object, leases/locks can be recalled and remote object can be updated.

Another possible way of doing this is to implement a global memory systems, wherein the single global address-space exists across entire RPC system. This way we can use traditional pass-by-address mechanism to achieve the same purpose.

(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)

**Answer:**

i) Some architectures may take 16 bit to store an int while others may take 32 bits. Additionally systems architectures can be little-endian or big-endian. RPC system has to agree upon the standard width of the built-in data-type such as ints, floats and booleans, as well as the end of the 32 bit word to be transmitted first. All nodes in the RPC systems can then use these standards to perform marshalling and unmarshalling of data.

ii) Same set of problems as in i) above arise when it comes to handling complex data-types such as strings and structs. Some architectures may enforce padding while packing these data-types, while others may not. Additionally architectures may use different termination codes to mark boundaries of these data-types. Hence RPC systems has to agree upon the standards to be used while dealing with these complex data-types.

iii) Higher-order data-structures reside in heap and involve run-time memory references. Different languages enforce different restriction on such references such as mutable vs immutable. RPC system needs to further standardize these to enable coherent access to memory.

iv) Lets take an example of c++ that implements a virtual table to enable polymorphism. Implementations like these are very different from how java does it. Similarly for other programming paradigms like functors, decorators, each languages ex. Python/java, implements it in a whole different way and there is no way it can be ported from one to another.

(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?

**Answer:** This can be done with aid of the JAVA's reflection library. All we need is .class file, which can then be reflected by a proxy objects at runtime, to make remote invocations.

The advantage that one would get from this is it eliminates the need to re-compile client and server programs, in case the change is limited to RMI objects. All we need is to compile modified RMI object and feed it to the existing server and client programs.

The main disadvantage of this approach is that the Reflection library of the java is not optimized for performance. It incurs lot of overhead which is otherwise avoided by the standard RMI facility of JAVA. Java reflections are primarily to be used for the run-time debugging and analysis.

(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.

**Answer:** It would really be a uphill task to do this. To be able to manually serialize the objects we'd need a .class file for each such objects along with code to serialize each of them. Additionally we'll need to reflect on each of such objects at run-time perform serialization at run-time. Looking at the complexities of the modern distributed systems, this seems to be near impossible task and not-worthy effort.

## 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?

**Answer:** Under very high contention scenario, the Token ring based approach requires as low as single message for every entry to critical section, this is because under heavy contention almost every token passing message will yield entry to critical section by next node in the ring.

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

**Answer:** The majority voting approach requires the highest number of messages for each critical section entry. It involves sending request message to all of the nodes and then wait for permission from at least half of them before entering critical section, followed by sending a

release message to each of them that permitted us. Additionally under heavy contention situations it'll involve large number of vote-relinquishes, further increasing the number of messages required.

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

**Answer:** Token ring based approach seems to be the most robust of all. It can handle the node failure in both cases while in a CS and otherwise. If node was in CS and went down with token, then the other nodes in the systems can initiate the token generating message to overcome it. Otherwise, the predecessor of the node will simply skip the node during token passing and will pass it directly to next successor of the node and so on.

(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?

**Answer:** The event of tie is possible when a node gets REQUEST with lower request_timestamp than the one for which it has already sent a YES message and hasn't received a RELEASE yet from it. Under these circumstances, both the nodes requesting might form a deadlock. To resolve this, the approach allows for a vote to be recalled. To break this tie, the node sends an INQUIRE message to the node it has already granted permit. In case this node has already won the election then no deadlock actually happened, hence it simply ignores the received INQUIRE message. Otherwise if it has not won election yet, it returns back the vote to the granting node using RELINQUISH message. Upon receipt of RELINQUISH'ed vote, the node grants the vote to the node with a lower request_timestamp thereby allowing it to enter critical section w/o any deadlocks.

(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.

**Answer:** Most notable similarity between both the problems is that both of them try to select a single machine, which is either a coordinator or the machine that accesses some resource. Most important difference of the two problems is that in case of coordinator election problem, every other node needs to know which node has been elected as a coordinator. But in case of synchronization problem, other nodes need not which node has been selected for CS access, only the selected node needs to know that it has been selected for access to critical section.

(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?

**Answer:** Existence of multiple coordinator originating from network partitioning can be prevented by adding constraints such as requirement of majority nodes to form a group and elect a coordinator. This approach is used in IBM's GPFS to maintain lock manager synchronization during network partitions.

## 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?

**Answer:** Lamport logical time provides a way of ordering of events originating from the transmissions of messages between two communicating machines, that satisfies the happens-before primitive. The logical timestamp is incremented based on the local known previous timestamp and the obtained timestamp from sender. As a result, the increase in timestamp may not be continuous, thereby they may become substantial big, and may occupy considerable space inside the messages.

(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?

**Answer:** Lamport's logical time does not provide a way to detect a causality violations but the vectors based timestamps does.

(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?

**Answer:** With ever increasing performance improvements in communication networks, there has been equal amount of performance increase in computational processing capabilities. As a result time between two interleaving events has been reducing as well. Thereby requiring smaller and smaller bound on maximum allowable drift between clocks, making it more and more difficult to synchronize clocks to maintain such small drifts. Additionally, the communications networks can only get faster upto some point (medium propagation speeds are limited by the speed of light), which may not be enough to achieve rapid clock syncing.

## 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?

**Answer:** Write-one and read-all strategy can result in a write-write conflict. It can be avoided by using logical timestamps based version numbers. And then performing a read quorum before every write, to get the base version number to be incremented after write, to get the globally latest and correct 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.

**Answer:** Consider the initial CVV of [1, 1, 1, 1] for all nodes in 4 node clusters. Now if the network partition breaks the clusters into two groups (of host 1, 3 and 2, 4). Then in case both the groups

separately work on the same file, they might perform writes that result into CVV's of [2, 1, 2, 1] for group1 of host 1 and 3, and CVV of [1,2,1,2] for group2 of hosts 2 and 4. Both these CVV's are concurrent to each other.
Concurrent CVV's indicate that the group1 of servers has seen some changes not seen by group2 and group2 has seen some changes not seen by group1. This indicates that the network may have been partitioned resulting in distinct and concurrent changes to the data.

(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.

**Answer:** Below are the set of steps that need to be performed during write to ensure that the version numbers remain correct.

i) Acquire a write-quorum of 2 nodes.

ii) Select another two nodes to form a lock quorum of 4 nodes (Max (2, 4)) (including both nodes from write quorum selected in previous step).

iii) Take a write lock on all of the nodes in the write quorum, and read lock on the remaining 2 nodes of lock quorum. This will ensure that no one can update the latest version numbers while we read it.

iv) Now perform the read quorum on these 4 nodes and read the latest version number.

v) Perform a write operation on the 2 nodes of write quorum.

vi) Increment the version number based on the version number read in step iv above.

vii) Release write lock from the 4 nodes and mark the write complete.

(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.

**Answer:** First of all we'll need a client request handling thread to be initiated at each of the server, that can handle request from clients request..

Furthermore, the this thread uses the standard 2-PL mechanism to atomically acquire list of locks upon receiving the above mentioned ACQUIRE message. 2-PL helps in avoiding the deadlock scenarios. It can use inter-cluster RPC/RMI facilities to acquire locks on behalf of client.

Armed with above system, the mobile client only needs to perform READ / WRITE operations to a single node in the cluster. The locking thread running on that node will enforce the write-all/read-one constraints using the 2-PL mechanism described above. In case of READ, it'll simply acquire a READ lock on given object on any of the servers (possibly least-loaded one) and READ the contents and return it to client, followed by release of the lock.

In case of write, it'll simply acquire write lock using 2-PL mechanism, followed by atomically write data using 2PC/3PC mechanisms on all servers. And then reply to client, followed by release of the locks one-by-one.

The above protocol provides a simple READ/WRITE interface to mobile client without it having to worry about the read-one/write-all constraints or the atomicity constraints of the distributed system. Furthermore a load-balancer can be placed between client and the system, to re-direct client request to

least loaded server in the system. We can also enforce various time-outs to avoid resource constraints arising from lock holds.

(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.

**Answer:** Consider the initial CVV of [1, 1, 1, 1] for all nodes in 4 node clusters. Now if the network partition breaks the clusters into two groups (of host 1, 3 and 2, 4). Then in case both the groups separately work on the same file, they might perform writes that result into CVV's of [2, 1, 2, 1] for group1 of host 1 and 3, and CVV of [1,2,1,2] for group2 of hosts 2 and 4. Both these CVV's are concurrent to each other.

Concurrent CVV's indicate that the group1 of servers has seen some changes not seen by group2 and group2 has seen some changes not seen by group1. This indicates that the network may have been partitioned resulting in distinct and concurrent changes to the data.