Problem Set 1 Solutions

Q1. Suppose you wish to create an RPC mechanism for a Java client on an Android smartphone (ARM hardware) to communicate with server code written in C that is running on a Linux server (Intel Xeon x86 hardware). Relative to the RPC mechanism you just implemented for Project 1, what additional complexity would your new RPC mechanism need to handle? How would you address each of the complexities you identify? Consider both primitive data types (such as ints, floats, chars) and complex data types (such as structs and strings).

Additional complexities:

- Ints, floats, chars and other primitive data types may be of different sizes or representations (for example, C uses 1 byte chars but Java uses 2 bytes). Additionally, systems may use different endianness, so byte ordering issues may arise. Other issues involve numerical properties such as behavior upon overflow or shift.
    - These problems are not so hard to solve. We could define a common representation for all primitive data types, and each entity participating in the RPC would need to make sure to convert the data type to the common representation before sending it to the other one. Additionally, each entity would need to implement "deserializers" that convert from the common representation to the local representation.

- If we wanted to support complex data types, such as structs and strings, we would need to define a lot more of conventions and implement a lot of translation routines. For example, define end-of-string conventions, padding conventions for structs, representation standards (ascii vs utf-8 for strings for example).
    - Again, a solution to this problem is to define a common representation for all of these complex data types. We need to handle much more complexity in this case (padding issues, end-of-string conventions and representation standards for strings, etc), but we would still be able to implement this.

Q2. Jane loves to listen to music. In fact, her mornings cannot begin until she listens to "Comfortably Numb" by Pink Floyd. She has a Spotify music subscription service which streams her favorite melody, which is S seconds long, at T bytes per second. Jane, as you may not be aware, is also one of the first human inhabitants on Mars. Unfortunately, all the Spotify servers are located on Earth.
The network between Earth and Mars has a one way latency of L seconds (i.e. RTT is 2L seconds), and a bandwidth of B bytes per second. Fortunately, end to end transmission reliability between Earth and Mars is so good that no acknowledgement packets are sent. Considering each condition separately, answer the following:

A. Once Jane requests for the song to be played, how long will it take before she can hear the first note of the song?
B. Suppose sunspot activity increases dramatically. It is predicted that Earth-Mars network communication will be disrupted because many packets will be dropped. Jane hates scratchy sounding music, and decides to wait until the entire song is buffered, before playing it. What is the minimum time that it will take for the song to buffer completely after Jane presses the play button? What is the maximum time?
C. Consider the same conditions as mentioned in (B) but assume that B >> T. What is the shortest time for the song to be completely buffered?
D. Consider the same conditions as mentioned in (2) but assume that T >> B. What is the shortest time for the song to be completely buffered?

Ans.
Length of the song: *S*T*
Bandwidth: *B*

A. Jane hears the first note of music after *2L* seconds i.e. the RTT from Mars to Earth.

B. Maximum Time: *infinity.* Minimum Time: *2L + (S*T)/ min(T, B)*.

C. *2L + S*. Note that the limiting factor here is the rate at which the server sends its data.

D. *2L + S*T/B*. The limiting factor here is the bandwidth. So even though the server can transmit at T bytes/second, the underlying transmission channel proves to be the limiting factor.

Q3. For a use case involving files of modest size, you are considering whether to add whole file caching at clients to a distributed file system. Your experiments show that it takes 200ms for a client to retrieve a typical file from the remote system, but only 5ms for it to retrieve it from the local disk. You also determine that in a trace of 1000 file accesses, 724 were to files that had been previously accessed. Based on these measurements, answer the following questions :-
A. What is the cache advantage of your system?
B. What is the hit ratio, miss ratio, and expected cost of a reference?
C. Based on these figures, do you think caching should be implemented for this system? Explain your answer.
D. Identify a situation where you would not use a cache, despite there being a high cache advantage.
E. Identify a situation where you would use a cache, despite there being a low cache advantage.
F. Identify two shortcomings of full replication (e.g. DropBox) relative to on demand caching.

Ans.

A. cache advantage = (200ms/5ms) = 40
B. hit ratio = 724/1000 = .724, miss ratio = 1 - hit ratio = .276 ,expected cost of reference = (.276 * 200) + (.724 * 5) = 58.82ms
C. Yes, the expected cost of a reference is much cheaper than the cost of a miss, so caching should significantly improve performance.
D. One example would be a situation in which the hit rate would be low because data is typically only accessed once, such as for streaming video.
E. One possibility could be if there is some other cost involved besides time. For example, maybe you have to pay for the bandwidth for accessing the remote copy, but accessing a cached copy is free.
F. It caches everything so it may take up a lot of space, you always see updates which creates unnecessary traffic, and the consistency protocol is unclear.

Q4. What are some motivations for using the RPC paradigm?

Ans. Network is complex - it is convenient to abstract this away into another layer, making it transparent to clients so that application programmers have a way to make remote calls in a way that "looks and feels" just like local calls. That is, remote calls now emulate local procedure calls, so all the complexity of the "remote" part is taken care of (mostly). This in turn allows for more flexibility and freedom in application programming, when a lot of this complexity is hidden and related errors are already automatically handled - more focus can be given to develop other parts of the system. For example, RPC can be used to develop distributed parallel computing systems. Server(s) can be master scheduler(s) and distribute work to client/worker machines across a network.

Q5. Excluding issues that arise from lack of a shared address space across machines, explain two different ways in which programming using RPC is more complicated than programming using local procedure calls.

Ans. New failure modes - now we have to deal with things like network failure, machine failure (a server dies vs. a client dies), server fulfillment process dying (vs. a worker process dying) or software failures, etc. Network partitioning is another issue that must often be addressed. Network packets could be lost or mangled... In order to achieve a similar level of robustness and failure resistance, remote calls require more complex failure policies.

As with any communication over the network, security is an important consideration. Security modes and permissions need to be implemented according to some policy across different machines, there must be some designated source(s) of authority dictating security policies, cross-domain calls can raise unclear security questions, etc. And also, if you are using some RPC framework, you are implicitly trusting that implementation (not always a good idea)... there are questions including are RPC communications encrypted, etc.

Q6. Why is it typically harder to reduce end-to-end latency than to improve throughput in a typical distributed system?

Ans. Reducing end to end latency usually requires deep structural or protocol changes. Furthermore, there is a fundamental lower bound on latency due to the speed of light. No amount of additional resources can let you violate physics. Bandwidth can be improved by adding resources, e.g., more / wider links. It is not fundamentally limited, though cost may be an issue.
More complex systems often add latency (more layers or stops along a network == more middle men and communication == more latency), and every middle node along the way has its own processing to do, such as security checks, validity checks, ACKS, etc. These all add to latency, but since all such middle nodes operate in parallel, throughput or bandwidth may not be affected.

Q7. NeverLoseIt, a new startup with patent pending technology, claims that its storage enhanced network routers can greatly improve file transfer reliability in a network that is composed solely of such routers. Each router has sufficient storage to buffer an entire file. Now, a client application can just hand its file to the closest router, which then takes responsibility for getting it to the next router, and so on, until the last router in the chain hands it to the destination server. Each link of this chain is reliable, because each router retransmits a file to the next router until it is acknowledged. NeverLoseIt claims that client application code can be much simpler because it does not have to check for success acknowledgement from the destination server. Can this possibly be true? Explain your answer.

Ans. No this cannot be true.
- This is a violation of end-to-end argument. It makes the network much more complex, while not making client's life any easier. However, is does improve performance when network condition is bad.
- The client application still has to deal with the reliable transfer of file from the client to the first router. The NeverLoseIt technique does not automatically guarantees reliability for this first hop. Dealing with this hop makes application development no easier.
- The server who acknowledges the last router may not be the application itself (e.g. it can be transport layer). When the server delivers the file to the application, errors may occur. Extra effort is needed to ensure the reliability of this part.