Solution to Homework 1

Problem 1. It is challenging to achieve scalability in distributed systems. Discuss two scalability challenges in providing a video conferencing application.
Answer:

- Geographical scalability: video conferencing appliccations are highly interactive. Long communication latency between two distant participants can result in inferior quality of experience.
- Size scalability: if many participants want to join the conference call, each participant will need to receive the live video stream from all other participants. However, the participant's network capability may not be able to support the high down stream bandwidth requirement. In addition, participant's device may also not capable of decoding all the streams in time.

Problem 2. TCP and UDP are two different transport layer protocols. Please briefly explain their differences.
Answer:

- TCP is a connection-oriented service: two end points must first set up a connection via three-way handshake before data can flow between them, while UDP is a connectionless service, no connection setup is required.
- TCP provides reliable service: data segments are retransmitted if lost, while UDP is an unreliable service.
- TCP provides flow control, preventing the sender from overwhelming the receiver by sending data too fast, while UDP does not.
- TCP provides congestion control, refraining senders from overwhelming the Internet, while UDP does not.

Problem 3. CDN services such as Akamai redirect clients to a best replica server using DNS. Explain how this DNS-based redirection works.
Answer: Consider a web service video.company.com that uses CDN service from e.g., cdn.com. When a client tries to resolve the IP address of video.company.com, the authoritative name server of company.com will return the client a CNAME record of a hostname managed by the CDN service, e.g., a1234.cdn.com. This redirects the client from company.com to cdn.com. The client would then move on to contact cdn.com's authoritative name server to find out the IP address of a1234.cdn.com.

You can also refer to Lecture 3 slide 25 for the correct redirection procedure.

Problem 4. (Chapter 4 Problem 18 in TBook)

Explain why transient synchronous communication has inherent scalability problems, and how these could be solved.
Answer: Synchronous communication requires the caller be blocked until it receives the desired response (e.g., ACK of request, acceptance of request, or the reply message with the RPC results). This stops the caller from doing anything meaningful while waiting for the response, and the wait can be very long if the receiver is physically far away from the caller. This can be solved by having the caller doing some other useful work while waiting for the response, essentially establishing a form of asynchronous communication.

Problem 5. Traditional RPC mechanisms cannot handle pointers. What is the problem and how can it be addressed?
Answer: Pointers refer to an address of a local memory location, and only make sense in its local address space. Therefore, if a pointer is passed to a remote machine, it would be meaningless. The problem can be addressed by sending a copy of the referenced data structure to the server, and on return replacing the client's copy with that modified by the server.

Problem 6. Consider a client-server application where the client makes remote procedure calls to the server. The link latency between the client and the server is 20 ms each way. Marshalling and un-marshalling of the parameters and return value take approximately 1 ms, respectively. The server takes 150 ms to process each request and can only process one request at a time.
(a) How long would it take to complete two back-to-back requests in a synchronous RPC system?
(b) How long would it take in an asynchronous RPC system? Assume the client is interested in the response and waits for the acceptance of the request before continuing.
Answer:
(a) In a synchronous RPC system, the client sends a call to the server, wait for the response to come back, and then issues the second call. For each call, the client need to first marshall the parameters (1 ms), send the request message (20 ms), the server receives the message, unmarshalls the parameters (1 ms), performs the call (150 ms), marshalls the result (1 ms), sends the reply message (20ms), when the reply arrives at the client, the client unmarshalls the result (1 ms). In total, each call takes $1 + 20 + 1 + 150 + 1 + 20 + 1 = 194$ms. Two back-to-back calls would take: $194 \times 2 = 388$ms.
(b) In an asynchronous RPC system, the client can send a second call after its first call is acknowledged by the server, without having to wait for the first call to complete. Therefore, the timeline would look like the following:
At $T = 0$ ms, the client marshalls the paramters of the first call;
At $T = 1$ms, the client finishes marshalling the parameters and sends out the request message;
At $T = 21$ms, the server receives the message, and responds the client with an acknowledgement;
At $T = 22$ms, the server finishes unmarshalling the parameters and starts to execute the call;
At $T = 21 + 20 = 41$ms, the client receives the acknowledgment for the first request, and continues to send out a second request;
At $T = 41 + 1 = 42$ms, the client finishes marshalling the parameters of the second call, and sends out the second call request;
At $T = 42 + 20 = 62$ms, the second call arrives at the server side;
At $T = 22 + 150 = 172$ms, the server finishes the execution of the first call;
At $T = 172 + 1 = 173$ms, the server finishes marshalling the response, and sends out the response;
At $T = 173 + 1 = 174$ms, the server finishes unmarshalling the paramters of the second call and starts the execution;
At $T = 173 + 20 = 193$ms, the marshalled response for the first call arrives at the client;
At $T = 193 + 1 = 194$ms, the client unmarshalls the response to the first call;
At $T = 174 + 150 = 324$ms, the server finishes the execution of the second call;
At $T = 324 + 1 = 325$ms, the server finishes marshalling the result of the second call;
At $T = 325 + 20 = 345$ms, the marshalled response for the second call arrives at the client;
At $T = 345 + 1 = 346$ms, the client finishes unmarshalling the result of the second call.

Problem 7. Describe one application for which at-least-once semantics would be the best choice, and why?
Answer:

There is no unique answer to this question. One possible answer is: a read-only service can use at-least-once semantics since: i) the read operation is idempotent, so it is okay to execute the procedure more than once; ii) the at-least-once semantics is simpler and easier to support than at-most-once semantics - it only requires the client to retransmit the request, no server-side duplicate filtering is needed.

Problem 8. (Chapter 2 Problem 5 in TBook)

In a structured overlay network, messages are routed according to the topology of the overlay. What is an important disadvantage of this approach? (Hint: latency.)

Answer: In a structured overlay network, messages may be routed through multiple intermediate nodes before arriving at the destination node. These intermediate nodes may be physically far from each other, causing messages to travel through long physical paths that can incur very high latency (e.g., transoceanic paths that incur hundreds of milliseconds latency).

Problem 9. (Chapter 2 Problem 10 in TBook)

Not every node in a peer-to-peer network should become superpeer. What are reasonable requirements that a superpeer should meet?

Answer: A superpeer is expected to have consistent Internet connection. While regular peers may be on and off, a superpeer should be online consistently. Moreover, a superpeer is expected to have more bandwidth as it will receive queries from both within the group and outside of the group. Finally, a superpeer should have enough computing and storage capabilities for keeping track of all resources availble within the group annd looking them up.

Problem 10. Consider a Chord ring based on 9-bit node identifiers and keys, nodes {0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 456, 501} are present on the ring. Provide the finger tables for all nodes.

Answer:

node 0: [1, 2, 5, 8, 21, 34, 89, 144, 377]
node 1: [2, 3, 5, 13, 21, 34, 89, 144, 377]
node 2: [3, 5, 8, 13, 21, 34, 89, 144, 377]
node 3: [5, 5, 8, 13, 21, 55, 89, 144, 377]
node 5: [8, 8, 13, 13, 21, 55, 89, 144, 377]
node 8: [13, 13, 13, 21, 34, 55, 89, 144, 377]
node 13: [21, 21, 21, 21, 34, 55, 89, 144, 377]
node 21: [34, 34, 34, 34, 55, 55, 89, 233, 377]
node 34: [55, 55, 55, 55, 55, 89, 144, 233, 377]
node 55: [89, 89, 89, 89, 89, 89, 144, 233, 377]
node 89: [144, 144, 144, 144, 144, 144, 233, 233, 377]
node 144: [233, 233, 233, 233, 233, 233, 233, 377, 456]
node 233: [377, 377, 377, 377, 377, 377, 377, 377, 501]
node 377: [456, 456, 456, 456, 456, 456, 456, 0, 144]
node 456: [501, 501, 501, 501, 501, 501, 8, 89, 233]
node 501: [0, 0, 0, 0, 5, 21, 55, 144, 377]

Problem 11. List all hops for the following key lookups for the Chord system described in Problem 10:

source	key
0	500
5	145
21	88
89	150
233	1
501	1

Answer:

500@0: 0→377→456→501
145@5: 5→144→233
88@21: 21→55→89

150@89: 89→144→233
1@233: 233→501→0→1
1@501: 501→0→1

Problem 12. (Chapter 2 Problem 11 in TBook)

Consider a BitTorrent system in which each node has an outgoing link with a bandwidth capacity B_{out} and an incoming link with bandwidth capacity B_{in}. Some of these nodes (called seeds) voluntarily offer files to be downloaded by others. What is the maximum download capacity of a BitTorrent client if we assume that it can contact at most one seed at a time?

Answer: In a BitTorrent system, seeds only contribute their uplink bandwidth and do not download, while other peers may download and upload at the same time. Usually the number of seeds is much smaller than the number of peers in the systems. Suppose there are S seeds and N peers in the BitTorrent system, and $S \ll N$. The total aggregated uplink bandwidth capacity in the system is therefore $(S+N) \times B_{out}$. Since seeds don't need to download, assuming that the incoming bandwidth capacity of peers is unlimited, each peer can get $\frac{(S+N) \times B_{out}}{N}$ downloading capacity. If we take the incoming bandwidth capacity B_{in} into consideration, a peer can get $\min\{\frac{(S+N) \times B_{out}}{N}, B_{in}\}$ downloading capacity.