15-440/15-640: Distributed Systems

Homework 1

REFERENCE SOLUTION

Last update: February 8, 2016

Question 1

Bob wants to send a file of F bytes to Alice. The file is split into blocks of size exactly M bytes each. You can assume that F is a multiple of M, and that Alice knows these values in advance. The network between Bob and Alice has a latency of L milliseconds and a bandwidth of B bytes per second. Bob is considering two different approaches.

In the *Hare* approach, Bob sends each block as a UDP packet and only requires Alice to send a single ACK packet after all blocks have been received. Individual UDP packets are not acknowledged. After sending all the blocks, Bob waits for a time T to receive Alices (single) ACK. If the ACK does not arrive within T, he repeats transmission of the entire file.

In the *Tortoise* approach, Bob again sends each block as a UDP packet. However, Alice must now ACK receipt of that packet. If no ACK arrives within time T, Bob retransmits that packet. He repeats this indefinitely, until the ACK arrives. He then proceeds to send the next packet, and so on until he receives the ACK for the last packet.

Bob is infinitely patient, and never gives up. For the different networking conditions described below, explore the merits of the Hare and Tortoise strategies for file sizes of 1 KB, 1 MB, and 1 GB. For all file sizes, the block size M is chosen so that exactly 10 packets are sent. As a simplifying assumption, you can assume that ACKs are ultra-reliable and therefore never lost. Clearly state and justify any other assumptions that you make, including how you choose a good value for T.

Solution: The key of this question is to consider the case when retransmission also fails. In addition it is crucial to know that actual wait time would be at least round trip time (RTT, 2 * single direction propagation time) plus transmission time.

To simplify analysis, let the round trip time (time between two transmissions) be x and let probability of transmission success for a single packet (or multiple packets that are treated as one retransmission unit) be p. x and p could be computed according to the context, while we do the analysis in a more general setting first. Let $\mathbb{P}$ denote the probability measure. Then

- $\mathbb{P}$[No retransmission] $= p$. Waiting time $T_0 = x$.
- $\mathbb{P}$[1 retransmission] $= p(1-p)$. Waiting time $T_0 = 2x$.
- $\mathbb{P}$[2 retransmission] $= p(1-p)^2$. Waiting time $T_0 = 3x$.

- $\mathbb{P}[k \text{ retransmission}] = p(1-p)^k$. Waiting time $T_0 = (k+1)x$.

Summing up this geometric series, we can write the expected waiting time for each transmission, T', as:

$$\begin{aligned} T' = \mathbb{E}[\text{Waiting time}] &= \sum_{k=0}^{\infty}(k+1)x \cdot (1-p)^k p = px \cdot \frac{d}{dp}\left[\sum_{k=0}^{\infty} -(1-p)^{k+1}\right] \\ &= px \cdot \frac{d}{dp}\left[-\frac{1-p}{p}\right] \\ &= \frac{x}{p} \end{aligned} \qquad \textbf{(EQ I)}$$

For retransmission, we regard each transmission process as independently repeated experiment. In the *Hare* approach case this is easy: each transmission should take, theoretically:

$$\underbrace{2 \cdot \text{latency}}_{\text{RTT}} + 10 \cdot \frac{\text{packet size}}{\text{bandwidth}}$$

(see Piazza posts on latency & transmission time if you are confused). For the *Tortoise* approach, the independence of each each transmission allows us to use the linearity of expectation. In other words, let X_i be a random variable denoting the time needed to successfully transmit the i^{th} packet, for $i = 1, 2, \ldots, 10$. Then:

$$\mathbb{E}[\text{Total waiting time}] = \mathbb{E}[X_1 + X_2 + \cdots + X_{10}] = \sum_{i=1}^{10} \mathbb{E}[X_i] = 10 \cdot \mathbb{E}[X_1]$$

where the last equality follows from the fact that all X_i's are identically distributed in probability (Bernoulli variables of success probability p). And of course, the calculation of $\mathbb{E}[X_1]$ simply follows from **EQ I** above since it is a single transmission.

We will use the conclusion above (more exactly, **EQ I**) for the following problems.

A. Suppose Alice and Bob live in different CMU dorms. The network between them is the CMU LAN. Latency L is one millisecond, and bandwidth B is one gigabit per second. The network is extremely reliable: the probability of a UDP packet being lost is only 10^{-10}. For different file sizes using the *Hare* and *Tortoise* strategies (i.e., total of 6 different combinations), how long is Bobs expected waiting time for successful file transmission?

Solution: Define PS = packet size. FS=file size
The waiting time T for each transmission should be at least $2L + \frac{PS(\text{or}FS)}{B}$ (FS

for Hare and PS for Tortoise). In this case, therefore, we set $x = (2L + \frac{PS(or FS)}{B})$ in **EQ I**. Since the loss probability is negligible, the expected total waiting time can be approximated in situation where no packets get lost, namely, $p = 1$ in **EQ I**.

With Hare approach, the waiting time for each of Bob's transmission is $2L + \frac{FS}{B}$. The expected waiting time T' for different packets are:

1) $FS = 1KB = 8 \times 10^3$ bits, $T' = 2.008 \times 10^{(-3)}s$

2) $FS = 1MB = 8 \times 10^6$ bits, $T' = 0.01s$

3) $FS = 1GB = 8 \times 10^9$ bits, $T' = 8.002s$

With Tortoise approach, the waiting time is $2L + \frac{PS}{B}$ each. If the file size is $1KB = 8000bits$, then each packet will contain $PS = 800$ bits. Applying **EQ I**, we have that the expected time of transmission of this single packet is

$$\mathbb{E}[X_1] = \frac{2L + \frac{PS}{B}}{p} = 2L + \frac{PS}{B} = 2 \cdot 0.001s + \frac{8 \cdot 10^2 \text{ bits}}{10^9 \text{ bits/s}} \approx 2 \cdot 10^{-3}s$$

Then by the linearity of expectation, the total waiting time expected for a complete transmission of 1KB file is $10 \cdot \mathbb{E}[X_1] = 2 \cdot 10^{-2}s$. This cases for other file sizes follow similarly. Thus the expected total waiting time T' for files of different sizes are: $2 \times 10^{(-2)}s$, $2.8 \times 10^{(-2)}s$ and $8.02s$.

Note: if you used $1K = 2^{10}$, $1M = 2^{20}$ and $1G = 2^{30}$, you will get:

1. Hare approach: $2.0076 \cdot 10^{-3}s$, $9.8 \cdot 10^{-3}s$ and $8.002s$.
2. Tortoise approach: $0.02s$, $0.02781s$ and $8.02s$.

B. Repeat (A) if the probability of UDP packet loss rises to 10^{-1} due to a flood in the Cyert Hall networking center.

Solution: Now, with Hare approach the probability that a retransmission is required is $1 - 0.9^{10} = 0.65$, thus set $p = 0.35$ in **EQ I**.
Expected times are:

1) $FS = 1KB = 8 \times 10^3$ bits, $T' = 5.74 \times 10^{(-3)}s$

2) $FS = 1MB = 8 \times 10^6$ bits, $T' = 0.029s$

3) $FS = 1GB = 8 \times 10^9$ bits, $T' = 22.86s$

With Tortoise, the probability of retransmission is 0.1, thus set $p = 0.9$.
Expected times are then:

1) $PS = 1KB/10 = 8 \times 10^2$ bits, $T' = 2.22 \times 10^{(-2)}s$

2) $PS = 1MB/10 = 8 \times 10^5$ bits, $T' = 3.11 \times 10^{(-2)}s$

3) $PS = 1GB/10 = 8 \times 10^8$ bits, $T' = 8.91s$

Note: if you used $1K = 2^{10}$, $1M = 2^{20}$ and $1G = 2^{30}$, you will get:

1. Hare approach: $5.74 \cdot 10^{-3}s$, $0.028s$ and $22.86s$.
2. Tortoise approach: $2.22 \cdot 10^{-2}s$, $3.09 \cdot 10^{-3}s$ and $8.91s$.

C. Alice graduates from CMU and moves to Mars (the planet, not Mars, PA). Latency L is now 8 minutes (one way) and bandwidth B is one megabit per second. Fortunately, the network is still very reliable and the probability of UDP packet loss is still only 10^{-10}. For different file sizes using the *Hare* and *Tortoise* strategies, how long is Bobs expected waiting time for successful file transmission?

Solution: In this case, the expected time for ACK would be $x = 960 + \frac{PS(or FS)}{B}$ and the retransmission probability can be negligible.
With Hare, the expected times are:

1) $FS = 1KB = 8 \times 10^3$ bits, $T' = 960.01s$

2) $FS = 1MB = 8 \times 10^6$ bits, $T' = 968s$

3) $FS = 1GB = 8 \times 10^9$ bits, $T' = 8960s$

With Tortoise, the expected times T' are $9600s, 9608s, 17600s$, respectively.

Note: if you used $1K = 2^{10}$, $1M = 2^{20}$ and $1G = 2^{30}$, you will get:

1. Hare approach: $960.01s$, $968s$ and $9152s$.

2. Tortoise approach: $9600s$, $9608s$ and $17792s$.

D. Suppose a solar storm temporarily increases the probability of UDP packet loss to 10^{-1}. Repeat (C) under these conditions.

Solution: With Hare, the expected times are ($p = (\frac{9}{10})^{10} = 0.35$):

1) $FS = 1KB = 8 \times 10^3$ bits, $T' = 2742.88s$

2) $FS = 1MB = 8 \times 10^6$ bits, $T' = 2765.71s$

3) $FS = 1GB = 8 \times 10^9$ bits, $T' = 25600s$

With Tortoise,

1) $PS = 1KB/10 = 8 \times 10^2$ bits, $T' = 10666.68s$

2) $PS = 1MB/10 = 8 \times 10^5$ bits, $T' = 10675.56s$

3) $PS = 1GB/10 = 8 \times 10^8$ bits, $T' = 19555.56s$

Note: if you used $1K = 2^{10}$, $1M = 2^{20}$ and $1G = 2^{30}$, you will get:

1. Hare approach: $2742.88s$, $2765.71s$ and $26148.57s$.

2. Tortoise approach: $10666.68s$, $10675.56s$ and $19768.89s$.

E. Reflecting on your answers to A through D, crisply state the set of conditions under which the Hare strategy dominates the Tortoise strategy and vice versa.

Solution: Hare is more sensitive to loss probability and thus it's conceivable that if the loss probability keeps rising, Tortoise would dominates over Hare, i.e. Tortoise dominates over Hare when loss probability is high.

But Tortoise is sensitive to delay (propagation delay) thus if the delay is prolonged, the Hare would dominate.

Question 2

For each of the following use cases, identify the weakest RPC semantics that can be used (exactly-once, at-most-once, or at-least-once). In each case explain your answer.

A. Transacting a stock trade online

> **Solution:** Exactly-once. No matter which semantics we choose, at the upper level of the application we have to make sure the transaction happens— and happens once only.

B. Checking on the status of a flight

> **Solution:** At-least-once. The operation is idempotent, meaning the global state is preserved even if multiple calls may have been issued. We can keep sending requests and this semantic guarantees that the `checkStatus()` call executes at least once.

C. Submitting a blog post that has already been composed in a local file and includes a timestamp and the users name at the beginning of the blog text.

> **Solution:** Either one of the following is correct, as long as the explanation is reasonable.
>
> - At-least-once. It is not harmful, in this case, to keep sending call requests to the server machine until we get an ACK. In other words, it wouldnt hurt to post two identical blogs in a row.
> - At-most-once. User can simply re-post if the previous attempt is not successful. This semantic helps avoid indefinite blocking, or any other reasonable concerns.

D. Creating a subdirectory in a distributed file system.

> **Solution:** Either one of the answers can be accepted, with appropriate justification:

- At-least-once: The same directory can only be created once, after which the operation becomes idempotent. Also, in this case, issuing the same calls multiple times (when no ACK is received from the DFS server) is harmless to the user.
- At-most-once: Server could crash (or unreachable), and if it happens, we dont want to wait forever. Instead, we can use at-most-once semantic to declare a timeout.

Question 3

On a lightly-loaded 10 Gbps LAN in a Facebook data center, end-to-end RPC communication between a client and a server for some operations is taking almost 100 milliseconds. None of the RPC operations involve large data transfers. Which of the following is the most plausible reason for this poor performance? Explain your answer.

Solution:

1. A is not correct, since the 10Gbps LAN is lightly-loaded, as the question points out. There shouldnt be a serious problem of congestion.
2. B is not correct. Usually hardware assumptions and standard marshalling libraries (for instance, json marshalling) guarantee that the overhead of marshalling and unmarshalling shouldnt be a major concern.
3. C is correct. It is very hard for heavily loaded servers/clients to promptly handle the requests they receive from each other and ACK their responses. Instead, the requests are usually queued at the hosts to be processed later. This could increase the RPC communication time required enormously.

Question 4

You have been hired by Uber as their lead architect for their driverless automobile system. Give one example each (with explanation) of a safety property and a liveness property involved in this system that you are designing.

Solution:

1. Safety: This is an open-ended question. Some sample answers:
   (1) Collisions should be avoided whenever possible.
   (2) Trivial partial failure should not affect the overall functionality of the vehicle.
   (3) Pull over in the case of an emergency (sensor failure, navigation problem, etc.).
   (4) Route to the destination!
2. Liveness: This is an open-ended question. Correct as long as the answer concerns timely execution progress (dealing with problems similar to starvation, etc.). Some sample answers:
   (1) Lane merging.
   (2) No customer waits forever.

Question 5

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

Solution:

1. Error handling (failure): RPCs can fail because of network communication problems, which is something that local calls dont suffer from. Therefore, RPCs need to implement semantics (at-least-once, etc.) to handle the potential failures such as timeout.
2. Local calls are more "one-to-one: one caller invokes a function call, and one function is executed accordingly. But for RPC, they may be multiple callers sending the same request to a server.
3. The data involved in RPC needs to be serializable (i.e. marshalled). For local procedure call, this is not a major concern.
4. ...

Question 6

To get around intellectual property laws that protect copyrighted material, the founder of EBooks_R_US has a novel business model. There is only one copy of each e-book legally acquired and paid for by the company. All of these e-books are located on a single server. If you ssh into the server, a user-friendly e-reader application can be used on any currently unused e-book. EBooks_R_US charges customers by how long they are logged in and using their e-reader. It is effectively "renting out its e-books without making illegal copies of any e-book. The company does very well initially, with a rapidly growing customer base. Unfortunately, as the company expands, there is growing concern about the security risk of allowing each customer to `ssh` into the server. The founders legal counsel assures them that a small variant of their current business model would also be lawful. In the new model, each e-reader would be wrapped in software that allows its operations to be invoked remotely. Instead of logging into the server, the user merely runs client software that remotely operates the e-reader. This is just a different way of "renting out an e-book without copying it. You may assume that all client-server TCP connections are authenticated and encrypted.

You have been hired to rapidly implement the software needed to support the new model. Your business partner says you should use REST to implement the communication between server and client. Your Technical Advisor. in contrast, suggests that you use RPC to export e-reader functionality. Time is of the essence. The e-reader is completely implemented in C++. Would you recommend use of REST or RPC? Explain your answer.

Solution: In this case, RPC is a recommended choice. Here are some potential reasons behind this:

1. REST does not really look like local procedure calls. Its based around the web technologies (e.g. JSON) which means a lot of changes may be needed to the old implementation.

2. REST requires a much more complicated implementations. In particular, in a REST system, you also have to provide code that transforms the data contained in REST requests and responses to the data representation that you have in your system— which could be a potential problem.

3. REST is based on HTTP. It would be hard if you want to provide your own protocol of communication (e.g. using your design of sliding window, designation of sequence number, etc.).

4. In this particular project of e-book, clients and server are tightly coupled (meaning they are designed to work together as an entire system).

5. The original app was built using C++, not the typical web-friendly frameworks. Therefore, a lot of work is expected to connect it to a REST interface without significant rewrite.