University of Waterloo
CS 486, Winter 2024
Assignment 1

Question 1

List and define three techniques for scaling distributed systems. For each one of the listed techniques, state if it complicates the system design and development, and discuss how? (Hint: check Chapter 1).

1. Hiding communication latencies:

   This complicates the system design and development because it means to construct the application to use **asynchronous communication**. Which means that the software needs to handle callback, states across interrupted processes, and ensuring the entire system is still responsive when parts of it is blocked when waiting for results. Which is very difficult to design and debug, as the states such system might end up with is indeterministic.

2. distribution of work:

   This complicates the system design and development because it involves splitting components into smaller pieces and spreading these pieces across the system. And in order to achieve this, the system itself must have ability to dynamically assigning tasks to different processing nodes, and the error handling associated with it. All of which will be hard to design debug similar to the first point.

3. replication:

   This complicates the system design and development the most because it involves duplicating components across a distributive system. One major issue associated with doing this is ensuring data integrity / coherence and avoid data divergence / conflicts across the entire system, especially for distributive databases. Which is also very hard to design because trade-offs must being made from using different strategies like 2 piece commit, and there is no single best solutions for all scenarios. Also, the process of determining which approach is ideal is not straightforward as well. Sometimes multiple prototypes must being made and a real-world tests on the deployment environment is needed to quantify the performance difference between different approaches.

Question 2

In Xerox RPC we studied in class:

a) How the server detects duplicate RPC calls?

The server first requests client's call to contain a unique process ID `(client IP, proc id)`, and a strictly increasing sequence number `seq#`, and in the sever end after received the request, server first check **export table** (contains function id, function name, call back pointer) to verify the validity of the request, then check **Prev_Calls table** (find the last sequence number the server successfully executed for the given `proc id` and `client ip`) to check if the call is **duplicated**.

(For server crashing, server re-binds `fid` so no previous registered client functions' remote request can pass. Then it's up to the client to establish a new relation with the restarted server process.)

b) Assume the Xerox RPC server has three responses to an RPC call: success, duplicate, error. If a client is making the following sequence of RPC calls, what will be the response from the server for each one of the following RPC calls? Explain your answer if you think the server will respond with a "duplicate" or "error".

The server has the following functions in its export table:

(Index, Fid, Function pointer)

(1, 10, `First_fun()`)

(2, 11, `Second_fun()`)

RPC call format is `rpc_call(client ip, process id, seq #, fid, args)`

1. `rpc_call(10.0.0.4, 100, 50, 10, 5)`

   response: success

2. `rpc_call(10.0.0.4, 100, 51, 10, 5)`

   response: success

3. `rpc_call(10.0.0.4, 100, 52, 12, 5)`

   response: error

   Because there is no function with `fid=12` in the export table.

4. `rpc_call(10.0.0.4, 100, 54, 10, 5)`

   response: success

5. `rpc_call(10.0.0.4, 100, 53, 10, 5)`

   response: duplicate

   Because the sequence number `53` is less than the last executed sequence number, so this call is earlier than 54, the previous succeed call. Server will not accept this request, but what error respond server will reply with is up to the exact implementation of Xerox RPC (Not covered in class).

6. `rpc_call(10.0.0.4, 100, 01, 11, 1)`

   response: duplicate

   Because the sequence number `01` is less than the last executed sequence number. (sequence number is per `proc id, client IP`)

7. `rpc_call(10.0.0.4, 100, 55, 10, 5)`

   response: success

8. `rpc_call(10.0.0.4, 101, 54, 10, 5)`

   response: success

Question 3

Google opted to build a distributed storage system that can leverage the storage resources of compute nodes in a data center, unfortunately, this approach may increase network overhead due to replication. In this question you will estimate the performance of a distributed file system. To write a file in our distributed file system:

- A client sends the data to one node (primary node). The file is stored in a buffer in memory
- Once the entire file is received. The primary node writes the data from memory to disk and concurrently replicates the file to two secondary nodes.
- The secondary replicas write the file to disk then acknowledge the operation to the primary.

Only when the primary gets the acknowledgment from the two secondary nodes, the primary will acknowledge the operation to the client. For reliability reasons, Google selects one secondary to be in the same rack as the primary node, and one secondary in a different rack. For reads, the client can read from any of the three nodes. The client is located on a rack that is different from the primary and secondary nodes.

Assume the following hardware characteristics. (Simplification: assume GB and MB are base 2 numbers.)

Hardware	Latency	Bandwidth
RAM	100ns	20 GBps
SSD	60 μs	200 MBps
Network within Rack	70 μs	1,280 MBps
Network in DC	500 μs	320 MBps

Table 1: Hardware characteristics

For the following questions, assume there is only one client in the system. Consider each sub-question separately, i.e., an optimization in one part does not affect the other parts.

1. What is the response time when a client writes a 1GB file? Show your calculation.

   Firstly, we can formulate the response time for a client as the chain of events thats going to take the longest time: (assume computation time is 0 since information is not given):

   Network latency between client to primary
   + send entire file from client to primary
   + network latency between primary and secondary in DC (absorb SSD latency)
   + send file from primary to secondary in DC (absorbing time take for the write / send to other node)
   + secondary's Latency to SSD (start to write to SSD)
   + time to write 1GB file to SSD (on the secondary node in DC)

+ Network delay from secondary in DC to primary (RTT, not counted) (assume acknowledgement is arbitrarily small)
+ Network delay from primary to client (RTT, not counted) (assume the acknowledgement reply is arbitrarily small)

By simplification given, we can compute:

The **time to write 1GB to SSD** is given by $\frac{1024}{200} = 5.12$ second.

The **time to send 1GB file over network in DC** is given by $\frac{1024}{320} = 3.2$ second.

So, the response time is:

$500\mu s + 3.2s + 500\mu s + 3.2s + 60\mu s + 5.12s$

Which evaluates to approximately 11.52106 seconds.

(RAM latency ignored by PIAZZA posts)

2. What will be the throughput and response time of writing a 1GB file if the client only waits until the data is stored at the primary and one of the secondaries (not two of the secondaries as in part a)? Show your calculation.

   Since clients only waits until one of the secondaries node, we can assume that we can base our computation with the secondary being the one within the same rack as the primary node.

   Secondly we can formulate the response time in a similar manner:

   Network latency between client to primary
   + send entire file from client to primary
   + network latency between primary and secondary within rack (absorb SSD latency)
   + send file from primary to secondary within rack (shorter than write to SSD, but the following chain is longer)
   + secondary's Latency to SSD (start to write to SSD)
   + time to write 1GB file to SSD (on the secondary node in DC)
   + Network delay from secondary within rack to primary (RTT, not counted) (assume acknowledgement is arbitrarily small)
   + Network delay from primary to client (RTT, not counted) (assume the acknowledgement reply is arbitrarily small)

   By simplification given, we can compute:

   The **time to send 1GB file over network within rack** is given by $\frac{1024}{1280} = 0.8$ second.

   So, the response time is:

   $500\mu s + 3.2s + 70\mu s + 0.8s + 60\mu s + 5.12s$

   Which evaluates to approximately 9.12063 seconds.

   (RAM latency ignored by PIAZZA posts)

   The throughput of this implementation is $\frac{1GB}{9.12063seconds} = 0.10964154888423278$.

   Thus the throughput is approximately 0.1096 GB per second.

Question 4

a) What is a stateless design?

A stateless design refers to the approach that make each requests from a client to a server containing all information needed to complete the requests, allowing the server to complete client requests without the need of any contextual but only the information given in each of the request itself. Hence the server does not store any information about the client's session.

b) What is its main advantage?

The main advantage of the stateless design is it improves the scalability of the server.

Because the server processes don't need to maintain a table of states from each clients, which means the server processes can easily handle requests from increasing amount of clients without a significant increase in memory or processing requirements for maintaining session states (note data coherence across server is not ensured in this model); Also the stateless nature means multiple servers can handle the request without the need to access session states, which might be on other servers.

c) What is its main disadvantage?

The main disadvantage of the stateless design is it increases network overhead on both clients and servers (also some process overhead for server as no optimization for requests from same client).

Because server process doesn't hold any information about the clients, more data is needed to be sent with each requests to provide the necessary context for processing, plus clients needs to add algorithms to handle server crashes.

Question 5

SEDA design divides the server into modules that communicate through queues instead of subroutine calls. What are the advantages and disadvantages of this design choice?

SEDA divides the server into multiple stages, where each stage follows a thread pool design, which communicate with each other using only queues. The advantages of such design includes but not limit to:

- Modular design allows easier design, program, testing / debug, because each stage can be treated individually as a stand -alone project, which also allow decoupling and minimized interference between different modules, where the different stages prevented overcommitment of resources and used dynamic resource controllers to ensured that.
- Thread-pool, even batching and adaptive load shedding design allowed better performance tuning (self-tuning), and the thread pool designs gave it capacity of satisfying massive concurrency demands, also allows fine granularity of control per stage.
- Simplified the construction of well-conditioned services, which shields application programmers from many of the details of scheduling and resource management.
- Enabled introspection, which allows the examination of system behaviour / performance during operation.
- Good fairness from multi-stage and thread-pool design.

And it's disadvantages includes but not limit to:

- The event-driven nature of it makes it harder to implement, as outlined in the essays of SEDA "many developers believe that event-driven programming is inherently more difficult."
- It's hard to ensure cache consistency due to the presence of multiple thread pools and modules.
- Detecting when a service is overloaded or not is very difficult, and so is determining an appropriate control strategy to counter such overload. "Many variables can affect the delivered performance of a service, and determining that the service is in fact overloaded, as well as the cause, is an interesting problem."

Question 6

What is the difference between client streaming and server streaming in gRPC?

The main difference lies between client / server streaming is the dataflow direction and the designed scenario for them.

Client steaming: A client-streaming RPC is similar to a unary RPC, except that the client sends a stream of messages to the server instead of a single message. The server responds with a single message (along with its status details and optional trailing metadata), typically but not necessarily after it has received all the client's messages.

In this model, the communication starts with the client pushing data to the server over a period of time. After receiving all the messages from the client, the server processes these messages and sends back a single response, which includes the server's status details and any optional trailing metadata. This approach is useful in scenarios where the client needs to send a large amount of data or multiple pieces of information that are best sent in a stream rather than a single request.

Server streaming: A server-streaming RPC is similar to a unary RPC, except that the server returns a stream of messages in response to a client's request. After sending all its messages, the server's status details (status code and optional status message) and optional trailing metadata are sent to the client. This completes processing on the server side. The client completes once it has all the server's messages.

In this model, the client sends a single request to the server, and in response, the server sends back a stream of messages. This is typically used when the server needs to send back a large amount of data or multiple pieces of information that are naturally sequential or too large to be sent in a single response. The stream concludes with the server's status details and optional trailing metadata, marking the end of the server's processing. The client completes its process once it has received all messages from the server.

Question 7

Since year 2000, a new set of servers - the gTLD servers (global Top-Level-Domain servers) have been introduced to host the ".com", ".org" and other high level domains. Until 2000 these domains where hosted by the root servers themselves, thus, introducing the gTLD servers was similar to introducing a new level of indirection for DNS requests. Was the user-perceived performance / latency of DNS queries significantly affected? Why?

No, the introduction of gLTD servers didn't significantly affect the user-perceived performance / latency of DNS queries mainly for the following three reasons:

- DNS heavily relies on caching at various levels. Once a DNS query is resolved, the result is cached for a while (TTL). This means even if the latency itself drastically increases, only the first query is impacted, and subsequent queries for the same domain can be served from the cache without the need to go through the entire resolution process again, which means their latency is un-changed. Note that our typical internet-related activities' latency perceived by the user are from those subsequent queries.
- Introductions of gTLD servers allows a better distribution of DNS query load by offloading popular TLDs like ".com" and ".org", meaning there are less bottle-necks in the process chain of resolving a DNS query, potentially even increase response time in certain times.
- Global DNS server are usually connected to the internet via very fast commercial-grade exchange points. Which means the additional delay from this added layer should be very small.

Question 8

Part A Delegation chain for `WWW.CS.WISC.EDU`:

Queried Server	NS Result/Delegation
a.root-servers.net	b.edu-servers.net f.edu-servers.net i.edu-servers.net a.edu-servers.net g.edu-servers.net j.edu-servers.net k.edu-servers.net m.edu-servers.net l.edu-servers.net h.edu-servers.net c.edu-servers.net e.edu-servers.net d.edu-servers.net
a.edu-servers.net	adns3.doit.wisc.edu adns1.doit.wisc.edu adns2.doit.wisc.edu adns4.doit.wisc.edu
adns3.doit.wisc.edu	dns2.itd.umich.edu dns2.cs.wisc.edu dns.cs.wisc.edu dns3.cs.wisc.edu
dns.cs.wisc.edu	dns.cs.wisc.edu (`SOA`)

Part B Query chain for IP address:

First command2: `dig @a.root-servers.net PTR 96.167.97.129.in-addr.arpa`

Command	NS Result/Delegation	Result Type
`dig @a.root-servers.net PTR 96.167.97.129.in-addr.arpa`	a.in-addr-servers.arpa b.in-addr-servers.arpa c.in-addr-servers.arpa d.in-addr-servers.arpa e.in-addr-servers.arpa f.in-addr-servers.arpa	NS
`dig @a.in-addr-servers.arpa PTR 96.167.97.129.in-addr.arpa`	x.arin.net y.arin.net r.arin.net z.arin.net arin.authdns.ripe.net u.arin.net	NS
`dig @x.arin.net PTR 96.167.97.129.in-addr.arpa`	cn-dns-mc-new.uwaterloo.ca cn-dns-ec2-new.uwaterloo.ca ext-dns-azure-a.uwaterloo.ca	NS
`dig @cn-dns-mc-new.uwaterloo.ca PTR 96.167.97.129.in-addr.arpa`	haproxy-2004.cs.uwaterloo.ca	PTR