Distributed Systems

03r. Assignment review

Paul Krzyzanowski

TAs : Shaleen Garg & Baber Khalid

Rutgers University

Spring 2020

Question 1

How does an **omission failure** differ from a **general network failure**?

- An omission failure is a failure to send or receive certain messages
  - Usually due to buffer overflow
- A network failure is a complete failure
  - Usually due to a failed or broken network link

Question 2

Halting failure and **fail-stop** are terms that are often used interchangeably. What is the distinction between them?

- **Halting failure**
  - The component stops functioning without informing other components prior to stopping
- **Fail-stop**
  - Is a halting failure that notifies other components prior to stopping

Review: types of system failures

- **Fail-stop** (**fail-silent**)
  - Failed component stops functioning
    - Ideally, it may notify other components first
  - **Halting** = stop without notice
  - Detect failed components via **timeouts**
    - But you can't count on timeouts in asynchronous networks
      - And what if the network isn't reliable?
    - Sometimes we guess

- **Fail-restart**
  - Component stops but then restarts
  - Danger: **stale state** – the restarted system doesn't know what happened recently

Review: types of network failures

- **Omission**
  - Failure to send or receive messages
    - Queue overflow in router, corrupted data, receive buffer overflow
- **Timing**
  - Messages take longer than expected
    - We may assume a system is dead when it isn't
  - Unsynchronized clocks can alter process coordination
    - Mutual exclusion, timestamped log entries
- **Partition**
  - Network fragments into two or more sub-networks that cannot communicate with each other

Review: types of network & system failures

- **Fail-stop** (**fail-silent**) *[covered earlier, but we're looking at outputs here]*
  - A failed component (process or hardware) does not produce any output
- **Byzantine failures**
  - Instead of stopping, a component produces faulty data
  - Due to bad hardware, software, network problems, or malicious interference
- Design goal – fault tolerance
  - Failures are inevitable – try to design systems that can handle them
  - Avoid single points of failure
    - A single failed component causes the entire system to not work

Question 3

When is it preferable to use UDP over TCP?

- Applications where low latency is crucial, such as gaming, videoconferencing, and telephony

UDP vs. TCP discussion

- With UDP (Universal Datagram Protocol)
  - Operating system does not handle acknowledgements, retransmission, or proper sequencing of messages
  - No connection setup
    - TCP requires a three messages to establish who's talking to whom, allocate kernel buffers, and establish & send starting sequence numbers and buffer sizes
  - No connection setup means
    - Much lower latency if you just need to send a quick message to a server
    - Not that significant with a lot of traffic
      (the overhead of a connection setup doesn't matter so much then)
- UDP will not detect lost packets or re-request damaged ones
  - But for real-time audio or video, that's fine since we may hear/see a glitch anyway

UDP vs. TCP discussion

Some common services that use UDP

- Voice over IP services (VoIP)
  - Skype, SIP-based IP telephony, Apple FaceTime
- IPTV (Internet-based TV)
- TFTP (trivial file transfer protocol)
  - Commonly used for network booting – really lightweight protocol
- DNS (Domain Name Service)
  - Look up IP addresses for domain names
  - UDP is low latency and doesn't require DNS servers to store connection state
- DHCP (Dynamic Host Configuration Protocol)
  - Used to configure new hosts on a local area network
  - A client cannot send a message with TCP because it doesn't have a unique address yet

Question 4

How does caching differ from replication?

- Caching stores frequently-accessed data for rapid access
- Unlike replication, it may not store the entire set of data
- Cached data may also become stale
  - The original data may be updated while the cache contains old versions

Question 5

Why did Google create gRPC?

- To create a lightweight alternative to HTTP for accessing microservices
- To provide a language-independent way for clients and services to communicate
  - Java, Go, Python, and Ruby are supported
  - Protocol buffers are used for serializing the data

Discussion: microservices

- Microservices – just a form of the Service-Oriented Architecture (SOA)
- Collection of services
  - Each service is autonomous
    - Runs on its own with no dependence on other services
    - Idependently deployable
  - Single function
    - Generally small and focused on doing just one specific task
  - General purpose
    - Built as a general-purpose service rather than with a specific client application in mind
  - Applications can contact various microservices to do their job
  - Fault tolerant (ideally)
    - Does not store client state (client will pass any needed data)
    - Can be load balanced or re-initialized on other servers

Discussion: Apache Thrift

- Google created protocol buffers and, later, gRPC for their applications to access various underlying services
- Facebook took a similar approach
  - Created **Thrift** – now part of the Apache project
  - RPC framework to work with a variety of languages
    - C, C++, C+, Go, Java, Node.js, Perl, Ruby, Rust, etc.
- Interfaces are defined in Thrift IDL (interface description language)
- Thrift compiler generates client and server stubs that implement the interface
- Various marshaling protocols may be selected
  - A simple binary protocol (faster than text)
  - A compact binary protocol
  - JSON – a text-based protocol – slower to process

Question 6

RESTful Web Services use HTTP to request operations and provide parameters, rather than embedding the entire nature of the request in the message body. For normal web interactions, our browsers use the following HTTP methods:

`GET`: request a web page

`POST`: submit a form, send data to a server

What six HTTP methods are used in RESTful services and what does each one do?

1, 2. `POST`, `PUT`: insert a new resource or update a resource

3. `GET`: retrieve information about a resource

4. `DELETE`: delete a resource

5. `OPTIONS`: list allowed operations on a resource

6. `READ`: return response headers and no body

Question 6

RESTful Web Services use HTTP to request operations and provide parameters, rather than embedding the entire nature of the request in the message body. For normal web interactions, our browsers use the following HTTP methods:

`GET`: request a web page

`POST`: submit a form, send data to a server

What six HTTP methods are used in RESTful services and what does each one do?

1, 2	`POST`, `PUT`	insert a new resource or update a resource
3	`GET`	retrieve information about a resource
4	`DELETE`	delete a resource
5	`OPTIONS`	list allowed operations on a resource
6	`READ`	return response headers and no body

Question 7

What is the main distinction between the use of a PUT and a POST operation in REST interfaces?

- A `PUT` operation is **idempotent** and a `POST` is not
  - You can send the same `PUT` request multiple times with no side-effects.
  - Repeated `POST` requests may result in multiple resources getting created on the server
- A `PUT` request requires the client to specify the complete URI of the resource
  - This means that the client should be able to construct this URI even if it does not yet exist on the server: the client will have to choose a name

The end