Distributed Systems - Assignment 1

Markus Roth

September 28, 2016

1 Exercise 1

I use the definition of five aspects of heterogeneity from chapter 1.5.1 in Coulouris. I understand "to invoke a method" to mean that we are talking about remote method invocation, meaning that object-orientation rather than a pure procedure call is a necessity.

This is a very open-ended question with a wide variety of possible answers. I have limited my answer to three problems in each of the five categories.

1.1 Networks

- The client must be able to locate the server using some kind of service look-up process (e.g. DNS).
- Client and server might be on very different networks (WLAN vs. LAN behind NAT). The protocols these networks use must inter-operate in a transparent manner to enable information transfer between client and server.
- The routing of traffic between client and server must be able to change with changing network topologies while guaranteeing a consistency of order (i.e. correctly matching responses to method calls).

1.2 Computer Hardware

- Client and server might use a different number of bits to express numbers. For example, the client might use 32 bits to express a floating point number, while the server might use 64 bits.
- Client and server might use a different bit order for their stored data (little-endian vs. big-endian).
- Wildly different processor speeds, for example when a Internet of Things-device communicates with a traditional server, can lead to the situation where the result data to a message call cannot be processed fast enough by the client.

1.3 Operating Systems

- Client and server operating systems might use different special characters to denote newlines (newline vs. newline and carriage return) or ends of files (does a file end in a newline?).
- The file system exposed through the service might have features like symlinks or access control lists not supported on the file system of the client operating system.
- File locators (paths) on client and server operating system might work very differently ("C:\Users\markus" vs. "/home/markus").

1.4 Programming Languages

- Client and server programming languages might use different encoding formats for strings (ASCII vs. Unicode).
- Client and server programming languages might have different function call semantics. For example, the server might be able to return multiple values from a single function call, while the client might not have a way built into the programming language to handle this.
- Client and server may have different inheritance models in their object systems. For example, the server might return an object that inherits from two classes, while the client language might not have such a concept.

1.5 Implementations by different developers

- Client and server might have different ways of representing information. A common standard needs to be developed.
- Even if client and server conform to an agreed-upon communication protocol, subtle bugs may exist and edge-cases might be handled differently on client and server.
- Even if a protocol is followed perfectly, varying approaches to exception- and error-handling can lead to unpredictable outcomes.

2 Exercise 2

An open distributed system relies on the publication of protocols. Without a published protocol, it is impossible to add new services to the system to be used by all clients. This makes it necessary for the remote file system discussed in question 1 to adhere to the standard protocol of the open distributed system it wants to be a part of.

This severely limits the heterogeneity available in the system. For example, it is not possible to have one service fully expose a FAT16 file system and another service to fully expose an EXT3 file system using the same published protocol. This is due to the fact that these file systems have very different features sets, and a protocol used for both must be the lowest common denominator of the two feature sets.

In general, the needs of openness - adherence to published standards - are different from the needs of heterogeneity, which are that each member of the network might to whatever it pleases.

3 Exercise 3

3.1 Restricting Access

The core problems here are authentication and protection of the access control mechanism:

- The server must be sure that the client is who he says he is (client authentication).
- The client must be sure that the server is who he says he is (host authentication).
- It must not be possible to inject a message into an existing message exchange between client and server.
- It must not be possible to modify or delete a message between the client and the server.
- It must be possible to change which users have access to the operations, even while these users are in communication with the server.
- There must be no way to use the public operations to change the list of the named users or the classification of operations.
- It must not be possible to lock out named users from the service by repeated but invalid access attempts through non-named users.
- It must not be possible to overload or shut down the service, thereby preventing any access by users, by overloading the server with public operation access.

3.2 Sensitive Information

The core problems here is encryption of the responses:

- It must not be possible to extract meaningful information from an intercepted message between the client and the server.
- It must not be possible to perform a man-in-the-middle-attack by impersonating the server to the client.
- The client must protect the sensitive data against second-channel access attempts by non-named clients.

4 Exercise 4

4.1 Advantages of Distributed Systems

- Scalability: A distributed system can be scaled up almost limitlessly, while a centralized system can only be scaled up to the limits of the current computer hardware. This means that a distributed system can both deliver more performance and can better adapt to changing load than a centralized system. This also means better cost-efficiency in many cases.
- Reliability: A distributed system can adapt to a single component system failing, while a centralized system has a much higher probability of no longer providing a service upon failure of a component. Systems like the Internet have been working without significant downtime for decades.
- Locality: Some workloads are necessarily distributed and can profit from being worked on in a distributed system. For example I might want to search the logs of of all my point-of-sale systems. Rather than copy all the logs to a centralized system and run the search there, I can save storage space and bandwidth by having a search run on each of the component systems and aggregate the results.

4.2 Disadvantages of Distributed Systems

- Complexity: A distributed system is inherently more complex than a centralized system. It needs to deal with communication between its components, handle failing components, handle the lack of global time and the lack of instant error-free data propagation. It exists in a world of congested networks and transmission errors.
- Security: A distributed system has a higher security risk due to necessarily being exposed to some type of network. Any exposure leads to risk. A centralized system may be air-gapped and thus fully secured against network attacks from the outside.
- Overhead: A distributed system has some overhead to deal with synchronization and coordination. If the workload can be done on a single centralized system, it will be more efficient and thus cheaper to run it on a single system.

4.3 Loosely Coupled Systems vs. Tightly Coupled Systems

A tightly coupled system consists of multiple component systems that communicate with each other. They are typically located close to each other, have synchronized hardware and are operated by the same entity. They might communicate by a network, but also by a bus or other system. Often they share services such as main memory or persistent storage. All components mostly trust each other. A component joining of leaving the system is an exception, not the rule. A multiprocessor system is a tightly coupled system, as is a cluster for grid computing.

A loosely coupled system on the other hand can consist of wildly different components that are often geographically distributed. Components do not trust each other, and malevolent components must be assumed to exist. Different components must have a common interface, but might be developed by different groups with varying interest. Components must be allowed to join and leave the system at will, and failure of components is the rule, not the exception. BitTorrent is a loosely coupled system, as is Email.