Name : Muhammad Ziam Saddique (fdai6519)

Subject : Distributed Applications (Exercise 2)

Submitted to : Rainer Todtenhoefer

Dated : 10 November 2020

Summary:

Distributed systems is the modern way of designing systems. A lot of self-contained applications which run on a PC are the examples of distributes, e.g. image editors. They seem to be executed on single computer system but actually they rely on remote cloud systems for updates, storage, and other services. According to Tanenbaum and Van Steen, “A distributed system is a collection of independent computers that appears to the user as a single coherent system”. Some issues should be kept in mind, while designing a distributed systems, such as latency and maintainability. Because these factors directly affect the performance of system. There are, as well, many benefits of distributed systems, such as resource sharing, one can share resources with each other without investing a lot on resources, such as disks, printers, files, and compilers. Distributed systems are also open to software designed by different vendors. Distributed systems also support concurrency, they operate several processes at the same time on separate computers on the network. These processes may communicate with each other during their normal operation. The main benefit of distributed systems is scalability, user can use scale-up or scale-down the capacity as per it’s needs. If there is more traffic, user can scale-up the capacity and scale-down the capacity if there is less traffic. Distributed systems are also fault tolerant, if one system is facing any issue, the other system is there to replace and maintains the smoothness of operations. Though distributed systems are more complex as compared to centralized systems, but they are more effective.

While designing the distributed systems, some issues should be considered, such as transparency, openness, scalability, security, quality of service, and failure management.
If we talk about scalability, it reflects system’s ability to deliver high-quality service as demands on the system increase. There are three dimensions of scalability, size, distribution, and manageability. When a system is distributed, attackers may target any of the individual system components or the network itself. If a part of the system is successfully attacked, then the attacker may be able to use this as a “back door” into other parts of the system. Distributed systems are able to defend themselves from several attacks, such as interception, when an attacker intercepts communications between parts of the system so that there is a loss of confidentiality. Interruption, where system services are attacked and cannot be delivered as expected. Denial-of-service attacks involve bombarding a node with illegitimate service requests so that it cannot deal with valid requests. Modification, where an attacker gains access to the system and changes data or system services. Fabrication, where an attacker generates information that should not exist and then uses this information to gain some privileges. For example, an attacker may generate a false password entry and use this to gain access to a system.

The quality of service (QoS) offered by a distributed system reflects the system’s ability to deliver its services dependably and with a response time and throughput that are acceptable to its users. Ideally, the QoS requirements should be specified in advance and the system designed and configured to deliver that QoS. Unfortunately, this is not always practicable for two reasons, It may not be cost-effective to design and configure the system to deliver a high quality of service under peak load. The peak demands may mean that you need many extra servers than normal to ensure that response times are maintained. This problem has been lessened by the advent of cloud computing where cloud servers may be rented from a cloud provider for as long as they are required. As demand increases, extra servers can be

automatically added. The quality-of-service parameters may be mutually contradictory. For example, increased reliability may mean reduced throughput, as checking procedures are introduced to ensure that all system inputs are valid.

In a distributed system, it is inevitable that failures will occur, so the system has to be designed to be resilient to these failures. Failure is so ubiquitous that one flippant definition of a distributed system suggested by Leslie Lamport, a prominent distributed systems researcher, is: You know that you have a distributed system when the crash of a system that you've never heard of stops you getting any work done.

There are two main models of interaction in distributed systems, such as procedural interaction and message-based interaction. Middleware is something which manage the diverse parts of system written in different programming languages and ensure that they can communicate and exchange data.

There are client-server systems in which, the user interacts with a program running on their local computer, such as a web browser or application on a mobile device. This interacts with another program running on a remote computer, such as a web server. The remote computer provides services, such as access to web pages, which are available to external clients.

There are several architectural patterns for distributed systems, which are as under:

1. Master-slave
2. Two-tier client-server architecture
3. Multi-tier client-server architecture
4. Distributed component architecture
5. Peer-to-peer architecture

Master-slave architecture is usually used where interaction response times are needed. When centralizing the system for security reasons is required, two-tier-client-server architecture is leveraged. If there is a high volume of transactions, Multi-tier client-server architecture is recommended. Distributed component architecture, which is used when resources from different systems and databases need to be combined, or as an implementation model for multi-tier client–server systems. When clients exchange locally stored information and the role of the server is to introduce clients to each other, Peer-to-peer architecture is used.

Reference: "I. Summerville, Software Engineering Chapter 17"