CSE 601: Distributed Systems

Toukir Ahammed

Design Goals of a Distributed Systems

Design Goals of a Distributed Systems

- Four important goals that should be met to make building a distributed system worth the effort.
  - a distributed system should make resources easily accessible;
  - it should hide the fact that resources are distributed across a network;
  - it should be open;
  - it should be scalable

Design Goals of a Distributed Systems

Resource Sharing

Making Resource Accessible

- An important goal of a distributed system is to make it easy for users (and applications) to access and share remote resources.
- *Example*: printers, computers, storage facilities, data, files, services, networks etc.
- Economic reason: it makes economic sense to share costly resources such as supercomputers, high-performance storage systems, e.g., it is cheaper to
  - have a single high-end reliable storage facility be shared than having to buy and maintain storage for each user separately.
  - it is cheaper to let a printer be shared by several users in an office than having to buy and maintain a separate printer for each user

Making Resource Accessible

- More advantages
  - Connecting users and resources also makes it easier to collaborate and exchange information
  - collaboration tools such as software for collaborative editing, teleconferencing, and so on that became (more easily) available due to the COVID-19 pandemic

Design Goals of a Distributed Systems

Distribution transparency

Distribution transparency

- An important goal of a distributed system is to hide the fact that its processes and resources are physically distributed across multiple computers, possibly separated by large distances.
- In other words, it tries to make the distribution of processes and resources **transparent**, that is, invisible, to end users and applications.
- A system that is able to present itself to users and applications as if it were only a single computer system is said to be transparent.

Distribution transparency

- A system that is able to present itself to users and applications as if it were only a single computer system is said to be transparent.
- To make a distributed system transparent (i.e., conceal its distributed nature), you have to hide the underlying distribution.
- You create abstractions that hide the system resources so that the location and implementation of these resources can be changed without having to change the distributed application.
- Resources should be abstracted and addressed logically rather than physically.
- Middleware is used to map resources referenced by a program onto the actual physical resources.

Types of distribution transparency

Transparency	Description
Access	Hide differences in data representation and how an object is accessed
Location	Hide where an object is located
Relocation	Hide that an object may be moved to another location while in use
Migration	Hide that an object may move to another location
Replication	Hide that an object is replicated
Concurrency	Hide that an object may be shared by several independent users
Failure	Hide the failure and recovery of an object

Access Transparency

Access transparency deals with hiding differences in data representation and the way that objects can be accessed.

Example:

- A distributed system may have computer systems that run different operating systems, each having their own file-naming conventions.
- Access issues that should preferably be hidden:
  - differences in naming conventions
  - differences in file operations, or
  - differences in low-level communication with other processes

Location Transparency

- **Location transparency** refers to the fact that users cannot tell where an object is physically located in the system.
- **Naming** plays an important role in achieving location transparency. It can often be achieved by assigning only logical names to resources.
- **Example**: **uniform resource locator** (**URL**) such as https://iit.du.ac.bd, gives no clue about the actual location of the Web server.

Migration Transparency

- Distributed systems in which resources can be moved without affecting how those resources can be accessed are said to provide **migration transparency**.
- In the previous example, the URL also gives no clue whether files at that site have always been at their current location or were recently moved there.

Relocation Transparency

- **Relocation transparency** supports the mobility of processes and resources initiated by users, without affecting ongoing communication and operations. In such cases, resources can be relocated while they are being accessed without the user or application noticing anything
- **Example**:
  - communication between mobile phones: regardless whether two people are actually moving, mobile phones will allow them to continue their conversation.
  - online tracking and tracing of goods as they are being transported from one place to another

Replication Transparency

- In distributed systems, resources may be replicated to increase availability or to improve performance by placing a copy close to the place where it is accessed.
- **Replication transparency** deals with hiding the fact that several copies of a resource exist, or that several processes are operating so that one can take over when another fails.
- *Note*: a system that supports replication transparency should generally support location transparency as well, because it would otherwise be impossible to refer to replicas at different locations.

Concurrency Transparency

- An important goal of distributed systems is to allow sharing of resources. In many cases, sharing resources is done *cooperatively* (e.g. communication channels). However, there are also many examples of *competitive* sharing of resources.
- **Example**: two independent users may each have stored their files on the same file server or may be accessing the same tables in a shared database. In such cases, it is important that each user does not notice that the other is making use of the same resource. This phenomenon is called **concurrency transparency**.
- An important issue of concurrent access is the consistency of the resource. It can be achieved through locking mechanisms, by which users are given exclusive access to the desired resource.

Failure Transparency

- **Failure transparency** means that a user or application does not notice that some piece of the system fails to work properly, and that the system subsequently (and automatically) recovers from that failure.
- Masking failures is one of the hardest issues in distributed systems and is even impossible in certain cases. The main difficulty in masking and transparently recovering from failures lies in the inability to distinguish between a dead process and a painfully slowly responding one.
- **Example**: when contacting a busy Web server, a browser will eventually time out and report that the Web page is unavailable. At that point, the user cannot tell whether the server is actually down or that the network is badly congested.

Design Goals of a Distributed Systems

Openness

Openness

- Another important goal of distributed systems is openness.
- An **open distributed system** is essentially a system that offers components that can easily be used by, or integrated into other systems.
- At the same time, an open distributed system itself will often consist of components that originate from elsewhere.
- Openness implies that system components can be independently developed in any programming language and, if these conform to standards, they will work with other components.

Openness: Interoperability

- **Interoperability** characterizes the extent by which two implementations of systems or components from different manufacturers can co-exist and work together by merely relying on each other's services as specified by a common standard.
- **Example**: a healthcare system where hospitals and clinics using different software platforms, exchange patient data seamlessly through open standards

Openness: Portability

- **Portability** characterizes to what extent an application developed for a distributed system A can be executed, without modification, on a different distributed system B that implements the same interfaces as A.
- **Example**: cloud-based application that can be easily moved from one cloud provider, like AWS, to another, such as Google Cloud, without significant modification (containerization technologies like **Docker)**

Openness: Extensibility

- It should be easy to add new components or replace existing ones without affecting those components that stay in place. In other words, an open distributed system should also be **extensible**.
- In an extensible system, it should be relatively easy to add parts that run on a different operating system, or even to replace an entire file system.
- **Example**: plug-ins for Web browsers, but also those for Websites, such as the ones used for WordPress.

Design Goals of a Distributed Systems

Scalability

Scalability

- The scalability of a system reflects its ability to deliver a high-quality service as demands on the system increase.
- Scalability of a system can be measured along at least three different dimensions
  - Size
  - Distribution
  - Manageability

Size Scalability

- It should be possible to add more resources to a system to cope with increasing numbers of users.
- A system can be scalable regarding its size, meaning that we can easily add more users and resources to the system without any noticeable loss of performance.

Geographical Scalability

- It should be possible to geographically disperse the components of a system without degrading its performance.
- A geographically scalable system is one in which the users and resources may lie far apart, but the fact that communication delays may be significant is hardly noticed.

Administrative Scalability

- It should be possible to manage a system as it increases in size, even if parts of the system are located in independent organizations.
- An administratively scalable system is one that can still be easily managed even if it spans many independent administrative organizations.

Scaling Up vs Scaling Out

- Changing the size of a system may involve either scaling up or scaling out.
- Scaling up means replacing resources in the system with more powerful resources.
  - For example, you may increase the memory in a server from 16 Gb to 64 Gb.
- Scaling out means adding more resources to the system
  - (e.g., an extra web server to work alongside an existing server).
- Scaling out is often more cost-effective than scaling up,
  - cloud computing makes it easy to add or remove servers from a system
  - however, this only provides performance improvements when concurrent processing is possible.