Study Log of Distributed System

1. Introduction
   1. What is distributed system?

A distributed system contains two of more independent computers/machines/servers. They may share some data, keep communication with each other, and act as uniformly to the client. To the outside world, they are treated as a single system. The client even doesn’t know there may be hundreds and thousands of machines behind the user interface.

* 1. Features for distributed system
     1. Complexity

Compared with single node system, a distributed system is more complicated. For example, the data may not be able to be fetched from local, you have to communicate with other machine. And other challenges like how to synchronize time, how to share data, how to replicate data. Finally, how to make it act as a single machine to the client, along with keep being correct, stable and robust.

* + 1. Multi-task support

A distributed system shall have the ability to work on multiple tasks simultaneously. This is the main advantage why we use it instead of a standalone system.

* + 1. Robustness

Another advantage of distributed system is robustness, compared with the single node system. If a machine of a distributed system is shutdown or dead, it will be abandoned. System will find a substitute machine to continue handle the task.

* + 1. Uniform interface

Though, the structure of a distributed system may be very complicated. However, the inside components must be packaged as a whole unit. All of the internal components are put into a black box. When we are using it, it is not necessary to know how its components cooperate with each other. We just need a uniform interface and the identical inputs and outputs.

* + 1. Transparency

Whatever the architecture of the distributed system is and however the complexity of the system, it is always transparent to the end user. Transparency includes following aspects:

* Access – How a resource is accessed is unknown to end user. The user just care about he/she the result that the resource is fetched.
* Location – Where the resource is located is unknown to end user. There may be lots of duplicated data in several machine or location. To the end user, he/she just gain one of the copy, and it doesn’t matter, where it comes from.
* Migration – The resources may be moved to other places, but the end user doesn’t realize and he/she actually doesn’t need to know that, since the system still works as nothing has been changed.
* Relocation – Sometimes, the resource the user is asking for may not be available from the original location. So the system may relocate the user to another machine or sub system to get it. The end user has no idea about the details about this procedure.
* Replication – Generally, a distributed system create several duplicated data in different machines. This is so-called replication. The data the user is operating on may be replicated and the end user doesn’t know.
* Concurrency – Since a distributed system can handle multiple tasks simultaneously, it is often happened that one resource may be accessed by several users at the same time. The distributed system must solve the conflict problem, keep the atom transaction for each user and prevent from creating dead lock. For each end user, he/she doesn’t realize the detailed situation, it seems only one person is operating on the resource.
* Failure – Error/exceptions are hard to completely avoid. When a failure occurs, the task would be executed again on another machine or at another time. To end user, he/she totally has no idea what has happened, since the success result will still be returned to him/her.
  1. sdfsd

1. sdfsdf

* Interface Definition Language(IDL), is used to define uniform interfaces for the distributed system to let others access. Generally, it at least has two arguments: one is the length of the data; another is actual data contains a string with that length. Relevant terms: Open system, IDL, Client Stub, Server Skeleton(Stub).
* Distributed Algorithm, fully decentralized algorithm:
* No complete information about the whole state
* Independent machine makes decision based on limited local knowledge
* One machine’s failure doesn’t affect to others or ruin the whole system.
* No global clock.
* False assumption when developing distributed system
* Network is reliable
* Network is secure
* Network is homogeneous
* Topology doesn’t change
* Latency is zero
* Bandwidth is infinite
* Transport cost is zero
* There is an administrator
* Transaction(Atomic, consistent, isolated, durable), Distributed Transaction
* Process, IPC(InterProcess Communication) used to communicate between processes. User shared memory and message to store data.
* User ‘Test and set’ to preserve atomic operation in local machine.
* Hard to use ‘Test and set’ for remote machine, which has different memory spaces.
* Hard to implement shared memory to make distributed system coordinate well.
* Clients and servers: a server can be client to other servers. Communicate with messages.
* Web: TCP/IP, HTTP, CGI, URL, MIME types, HTML

1. Architecture

* Architecture styles: layered, object-based, data-centered, event-based
* For event-based, publish to and subscribe from event bus, decoupled between processes.
* Application layer: user-interface, processing, data level.
* Heavy client/ heavy server: depends on requirement.
* DHT(Distributed Hash Table), add a new node to DHT, create a random id.

1. Processes

* User space/Kernel space
* Virtual machine
* UDP, one way, from sender to receiver, cheap, no setup, no guarantee, data may lost
* TCP, two way, each sends or receives data, reliable, in order, buffer required, delay
* Synchronous call: block, wait, sequential, inefficient, simple
* Asynchronous call: no hang, no wait, multi-thread, efficient, complex
* Socket/Port
* Marshalling/Unmarshalling
* XML namespace,
* Serialization/Deserialization
* Request-reply communication:
* RPC exchange protocols: R, RR, RRA

1. Communication

Protocol Layer: Application, Presentation, Session, Transport, Network, Data Link, Phisical

Steps of RPC: Client->Local OS->Remote OS->Server->Local OS->Client OS->Client

Asynchronous RPC:

Persistent communication

Berkeley Sockets

Reduce jitter: use buffer

Multicast