- Author: Petre Iordanescu
- Publisher: RENware Systems
- Edition: 2021 August 15
- Tags: distributed systems
- Language: EN

About brief series. These materials are derived from author training sessions as course notes for students. It is intended as presentation of basic topics regarding a concept. These materials are intended to present for a concept whst is good for, its main topics and what to deep research to find out and learn more about it.

Distributed systems. Overview

A distributed system is mainly a form of distribution of load across more computers instead of growing one machine to accommodate a higher load.

Classification

The basic and most important classification of distributed systems is:

- centralised
- decentralised or peer ti peer

This classification is the most important and must be considered first in choosing the architecture of a distributed system.

Architecture basic components

Obviously the core architectural components are those systems that execute the processes and for which the system is created.

But for a consistent work, a distributed system needs some auxiliary components. As an example is a computers network where beside the 'workable computers' there are additional equipments and computers (like routers, bridges, switches) needed for a right and consistent operating of network.

These components are:

- (FD) failure detection
- (SD) service directory
- (SA) service availability
- (OR) orchestrator and balancer
- (MQ) messaging queue
- (LC) logical sequencing and ordering events
- (TP) time synchronisation
- (EL) election services

These components, named *services* (as representing the right logical role) are not always all present. Here I presented a list with *what is usual present*, a *theoretical model*.

In practice, sometimes some services are doing similar things, so they are combined in one service. Or new services are present such database systems, conversion services (used for compliance with standards), and so on. Or some services are just used as is, being already standardised and present as public services (for example time synchronisation).

Centralised architecture (CEN)

In a centralized architecture there is one system, hard coded (meaning already known / dedicatef and1 established at configuration) with master role. That means:

- that system cannot (be) changed dinamicaly in best cases by reconfiguring the whole system (and often restarting it).
- the whole system is a *Single Point of Failure* type.
- all components of the system need the master to properly work.

That last point is important because most of distributed systems need a **master** to properly run, but this can be dinamicaly established at run time, even the an initial choose at system start (see *election* service).

The most known systems are: Zookeeper, Amoeba and others less known as being such systems, for example Facebook, YouTube, Netflix, and so on, all of them being hybrids between centralised and decentralised architectures.

Peer to peer architecture (P2P)

In a P2P architecture, the components of the distributed system can work independently one of each other, discovering itself who is "an available partner(s)" to work with.

The main advantage is of course that there no single point of failure, the whole system working properly (coherent) even with 1 node up and still running.

Sure that strong feature generates some performance penalties, but remains the preferred one if is possible, often these kind of problems going to compromises.

Probably the most known such a system is BitTorent, but the basics was established by *gnutela*.

Description of services

Here is a short description of each service, what is its role (what is doing) and what is good for. Also, where applicable, the most known standards, algorithms and protocols, was referred.

(FD) failure detection

The FD service is responsible for detecting the availability state of each node from the system. There are many aspects of the alghoritms used, such as:

- establishing too early (or too easy) that a node is no more alive. The node can work perfectly as machine, but not respond because a network damage which is solved meanwhile

- informing all other nodes about any node state, when declare it no more available or when it wakes up, otherwise nodes will have different information at the same time.
- and more other aspects which are not in scope of this brief article.

The most known algorithm used for these kind of services is **gossip**. For more details about failure detection, you can start with references 3 and 4.

(SD) service directory

A SD service is useful to have a map of all services that nodes can do and what services a node can do. In other words this is like a "Golden Pages" book where you can find the following information: *where I need to call to solve the issue.

A service directory is a core component of a distributed system and allows for 'keep away' from hard coding the nodes addresses, creating the premise for a dynamic system and adapt to almost any network infrastructure.

There is no 'dedicated alghoritm' because this is not typically an algorithm, but a data structure. Therefore there are some protocols destinated for a service directory and anf of the most known is **WSDL** (see reference 5).

(SA) service availability

The SA service is tightly coupled to FD service, but in addition creates the premises for:

- load balancing and distributing load over more nodes
- finding the best node to solve an issue (best meaning as round trip time, ie **RTT**)
- present a 'live map' of nodes with their status
- determining early (a preemptive behavior) the nodes load and health status (**early watch alert**) systems

Usual systems used to implement a SA service are HA-PROXY, NGINX, and others.

(OR) orchestrator and balancer

This service is just a SA but more explicitly named and with SA's premises activated.

As orchestrator is a little bit more than a balancer because there is a component for scheduling which can preempt a better usage of nodes. The scheduling is not more different that found in operating systems theory and with exactly the same topics, like locking of shared resources for example.

(MQ) messaging queue

As in any "mono" system, the communication between services and processes is a vital necessity. And, here I'm not talking about just communication between nodes, which is obviously assured by the network. Here, I'm talking about communication at process level which needs a specialised mechanism which must assure:

- the consistency of transported message
- the guarantee of delivery
- to work on any network topology

- to be compliant with standards, in order to can integrate a new node in system, even is developed by anybody else

These are to be assured by a **message queue** which can be implemented by using a standard aystem. As examples are: KV Store, Redis, Rabbit MQ, AMPQ, IBM MQ, MS SQL Broker, Apache Spark, and many more others.

(LC) logical sequencing and ordering events

This is referring to sequencing of information itsels, in the whole distributed system. The objective is *THAT ALL NODES TO SEE THE SAME SEQUENCE OF INFORMATION*, as it was produced by any other node and being agnostic of nodes clock or physical time. More exactly, the objective is that information to be processed in the same order by any / all nodes interseted to do that.

The fundamental algoritm was issued by *Leslie Lamport*. See reference 8 for details. This algoritm has suffered modifications in time, to improve it, most notable being **vectorized logical clocks**.

(TP) time synchronisation

This is related to nodes physical clock synchronisation and there is a widely used a standard named **NTP** (Network Time Protocol) which is largely used by all known operating systems(see reference 6).

(EL) election services

This service must assure the election of a new master node. As mentioned, almost all distributed systems need a master node at least as arbiter. Even those decentralised need such a node.

The problem of master node is not to be(come) a single point of failure, and for this reason, modern distributed systems have an alghoritm to choose another master node if the current becomes unavailable or simply to change it periodically.

The basic algorithm is *paxos* (see reference 9), which assure consensus between (or of) nodes in choosing a new master.

Key issues in computers distribution

This topic makes the subject of a separated article which can be found at (http://tlp.renware.eu/articles) in Distributed Systems category.

Notes and biographical references

- 0. [About Andrew Tanenbaum] (https://en.m.wikipedia.org/wiki/Andrew_S._Tanenbaum (https://en.m.wikipedia.org/wiki/Andrew_S._Tanenbaum))
- 1. Andrew Tanenbaum, Distributed Systems
- 2. Andrew Tanenbaum, Operating System Concepts
- 3. Failure detection aspects (http://www.cs.yale.edu/homes/aspnes/pinewiki/FailureDetectors.html)

- 4. Alan S Willsky [Survey of failure detection methods] (http://www.cs.yale.edu/homes/aspnes/pinewiki/FailureDetectors.html (http://www.cs.yale.edu/homes/aspnes/pinewiki/FailureDetectors.html))
- 5. Web Services Definition Language (https://www.w3.org/TR/wsdl.html)
- 6. Network Time Protocol (https://www.google.com/amp/s/searchnetworking.techtarget.com/definition/Network-Time-Protocol%3famp=1)
- 7. All articles from section Distributed Systems by Petre Iordanescu (https://tlp.renware.eu/articles_directory)
- 8. Leslie Lamport, Lamport logical clocks (https://en.m.wikipedia.org/wiki/Lamport_timestamp)
- 9. paxos consensus alghoritm (https://en.m.wikipedia.org/wiki/Paxos_(computer_science))