- Author: Petre Iordanescu
- Publisher: RENware Systems
- Edition: 2021 August 19
- Tags: distributed systems
- Language: EN

About brief series. These materials are derived from author teaining 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. Key issues

Issues treated in this article (in reverse order):

- Security
- Clustering and high availability
- Addresing in multiple network domains
- Information correlation from multiple modes
- Shared resources between nodes
- Notifying events and exchanging messages. Process synchronisation
- Sending parameters and intercepting results
- Mapping objects ID with their address
- Objects IDentifier

Objects IDentifier

First must say that objects refer to any element / component / artefact from a distributed system, being it a node, a process, a user (!), and so on. Everything must have an IDentifier. And this **must be unique** in order to identify.

Of course, a unique identifier (UUID) seems to be the universal solution. And in many cases there is no other option.

But this is not necessarily the best solution. Why? Because to be able to address a mode, by role for example, ee need to search for the role and then to find the corresponding address in what is called a **mapping table**.

Seems ok and reasonable, and it is so. What about the found node is not available? Need to search for other that have the necessary role (ie, can assure a desired service). That means another search in mapping table. Sure, we can optimize this searching, but...

If we do not use a random ID, but a hashed one (for example, and see *reference 1* for details) the system will have the chance to find out a right node in an **mathematical way** and not by scanning a table to search for one... In case of hashing roles, the nodes that can assure a role will be in *collision*, but certainly the collision table will be significantly shorter than a table with cartesian product of all roles and nodes.

So, as ***conclusion***, do not treat too simple the IDs. And for sure not any type of object needs the same type of ID like the others.

Mapping objects ID with their address

This aspect was explained in section ref ID.

Just a little more about alghoritms and methods. Hashing is not necessarily the only one possible. For example there are systems that determine "next node" (next having meaning and making sense in context of and for that distributed system) by adding a fixed value to the current (ie current being those that need to address next). This is perfectly working in a logical ring topology and approach.

Other systems make use of a predefined own schema of using IP address to code rack (for one router hop),data center location, recovery location, and so on. Of course, such a scheme impose restrictions in using and allocating IP addresses and probably a DHCP service will be not usable anymore, but it works and there are real implementations that use it.

Sending parameters and intercepting results

This issue appears when you need to design a distributed systems in which some processes needed to be launched by a node (the node who need results of execution of that process and let's name it **requester**) ftom various reasons wants that the process to be executed bu other node (let's name it **executor**). Or, more simple, the system is designed to execute any process on best node available (bu best meaning a useful reason, such as a node with a stronger GPU or a FPU.

In this case there must be happening the follows:

- requester must give to the process a unique ID.
- requester must prepare the code to be executed.
- requester must prepare the parameters that intended to be sent to the process.
- executor must be able to execute the code that receives, meaning that has a compiler for that language, has necessary libraries required by that process, and has the indicated resources (GPU in above example).
- after executing the code, executor must send back the results (any would be them, including stdin, stdout, stderr and any logs).
- the requester receive all of thats and consider job done.
- otherwise, requestor, needs to find out another node to execute the process or gives up.

Asynchronous execution

In any case, the execution is an asynchronous one. If the requester wats to busy wait for execution of process is its business.

If executor did not want to busy wait, then it must supply to executor a callback (callable) routine address to be called when execution is finished.

Notifying events and exchanging messages. Process synchronisation

This issue is (mainly but not only) from asynchronous execution, where the executor must notify requester that job was finished.

But sometimes (often) are situations when processes (executed on different nodes) need to exchange messages between them. Nowadays for these reason are frequently used *message queues*, these having some advantages:

- can be used as they are; You not need to write new software for that
- the ordering of messages stored in queue is guaranteed as being the order in which was written on queue; you must not deal with any clock to benefit of this ordering sequence
- they are standardised, so dirrent nodes (as written software) can use them without compatibility issues
- most of these systems are themself distributed, therefore being able to be addressed over the network / remotely, and have already own mechanisms for fail safe
- automatic & right encoding and serialisation of messages in order to be sent over the network

Examples of these states (messaging queues can be found in "Distributed systems. Brief. Overview" article.

Another dor of messaging is the "pub / sub" paradigm which allow for notifications on receiving messages, so without the need for periodic polling of message queue. Also, most of actual messaging queues systems can allow for this paradigm without probles.

Not in the last, the current message queuing systems allow for internal messages encrypting and signing (JWT style - see reference 3), this not being not at all a minor feature. Think only that the code needs to be sent over the network and an intruder inject its own code to be executed by a node... You do not want to happen something like that.

Shared resources between nodes

This issue can be (and is) accomplished almost like in operating systems running on multiple processors machine. The only difference is that communication is not s local one (ie, inside the same physical equipment) but over the network.

The issues are are same and are solved in the same manner. Let's enumerate the most relevant:

- two or more processes wants to write concurrently on the same device - solution is to treat them FCFS (first come first served)
- two or more processes need the same resource concurrently, some to write and some to read abd need the same variable to keep the evidence of where to write vs where to read (consumer producer classic problem) - this is accomplished by using mutexes (semaphores) to gain access to resource

Generally speaking, all issues derived from shared resources can be accomplished by using **mutex or critical sections**. The real problem in distributed systems is who takes care of mutex semaphores and critical sections.

In centralised systems this job is done by master node. In decentralised systems is needed a master node (but is not a hard coded one, and established by an election alghoritm).

Another modern approach is to use a specialised database like system with role of logically locking and releasing resources. This system must be "fast" and not a single point of failure. A **distributed database** is perfect for this role (more about distributed data systems can be found in a dedicated article on the same platform).

Information correlation from multiple modes

This issue becomes relevant in case of *massive parallel execution* when a process is launched in tens, thousands copies to solve distinct parts of a bigger problem.

There are different alghoritms specialised to this kind of problems frequently encountered in artificial intelligence (training of the neural network) and in arrays / matrix processing..The most known alghoritm is **map and reduce**, but does not makes the object of this article (for details see reference 2).

Addresing in multiple network domains (LANs)

That is referring to ability to address systems placed in different LANs, and is an issue for ipv4 based networks (in ipv6 being enough addresses space to solve in other ways the LANs advantage).

Here different mechanisms csnbe and are used being more as part of network infrastructure. Most of them are proxies, or hardware proxy enabled network equipments.

There are also specialised software ti address these kind of issues, most known being HA-PROXY or NGINX, but there are many more other proxy software systems for different complexity of systems.

Clustering and high availability

This issue is a "response" to failure protection, or to be more precisely to effects of failure of systems.

The clustering is itself a form of distributed systems having as main role the **availability** in case of systems failure.

Typically the clustering being about systems failure, by default the modes from a cluster gave a *passive role except one system* which execute the job at a moment. If this system fails, other one from cluster will continue the work exactly from the pount (moment) where main system has down becoming unavailable.

This kind of stuff is assured by a common database which keeps a log of everything done, so in case of failure, any otger system will be able to continue the work from the point of failure. This database (or repository) is called **quorum**.

Any cluster has at least one quorum database. The quorum itself is subject of single point of failure, so for quorum there are used tge best quality equipments and components.

Security

This issue is too large to be treated in this article and makes the object of a dedicated article.

Notes and biographical references

- 0. All articles by Petre Iordanescu, from section Distributed Systems (http://tlp.renware.eu)
- 1. Hashing algorithm explained (https://en.m.wikipedia.org/wiki/Hash_function)
- 2. MapReduce model (https://en.m.wikipedia.org/wiki/MapReduce)
- 3. JSON Web Token (https://jwt.io/)