Management of distributed systems

Christophe Pincemaille

November 10, 2008

University: Cork Institute of Technology
Department: Department of Computer Science
Module: Distributed objects
Lecturer: Jeanne Stynes

Contents

Abstract

This report has been made for the Distributed Objects module, in the Cork Institute of Technology. The goal is to see important aspects in distributed systems management. Actually, we focused here on three different aspects: fault tolerance, security and load balancing. Fault tolerance is how to make the system continue its processing in spite of errors and failures which may happen. Fault tolerance concepts include ways to detect errors, to react to errors, and to correct them. Security is the way to make that a system doesn't contain implementation faults which would enable a cracker to shut the system down, and is as well the confidence that information will not be intercepted by someone who doesn't have the right to do so. Finally, load balancing is the management of machine load.

Introduction

Currently, distributed systems are becoming strongly important in computing. Indeed, with the great networking tools that are hugely used, like the Internet, it becomes more and more important to make distributed applications. Moreover, for some particular uses, like computing huge amounts of data, distributing the load is one of the more obvious solutions that comes. Here, we will see some particular points in management of distributed systems.

There are some definitions of what a distributed system is [4]. According to *Distributed systems, concepts and design* [7], a distributed system "concists of a collection of autonomous computers linked by a computer network and equipped with distributed system software"[1].

According to Morris Sloman, "Management of a system is concerned with supervising and controlling the system so that it fulfills the requirements of both the owners and the users of the system"[2]. That implies that systems must have an acceptable reliability level, so that both the owner and the user of the system be satisfied of it. So the system must not crash due to overload, security attacks or faults. That also means that systems must "fulfill the requirements", that is to say provide the good result, so be protected against working faults and inserted errors from outside.

Then, we have here three very important domains of system management in systems, which are particularly critical in distributed systems: fault tolerence, security, and load balancing.

[1] Cf. [7], chapter 1, page 2.
[2] Cf. [15], part I, section 1.3, page 5

Chapter 1

Fault tolerence in distributed systems

In a distributed system, there can be a lot of errors and faults, by nature of these systems. Indeed, in order to be distributed, they are installed in several machines, so communicating via a network. And a network can be source of errors. Fault tolerance is the fact for machines to be tolerant to errors that can be made, generally by external sources, but also in internal [21]. The goal of fault tolerance is to be able to recover correctly after a failure, without a loss of data.

We will see some of the causes, and then view more precisely what is fault tolerance, and what are the existent systems to prevent faults.

1.1 Why fault tolerance ?

A network, more than a lot of other systems, can be source of faults. We can view the principle of some of the main causes that make faults so mainstream, and that makes fault tolerance so important in distributed systems.

1.1.1 Denial of Service

The Denial of Service (DoS) makes that a server is no longer able to process its clients' requests. It is generally due to accident or attack. The main DoS can be related to [6] :

a physical isolation unplug of a network cable for example ;

a saturation by sending a flood of messages to the server ;

a security fault exploiting which in general, enables its launcher to take control of the application ;

a logical bomb which aim is to saturate the server OS.

This can make the distant application very slow, or interrupt or even shut it down (lack of response).

1.1.2 Data corruption

DoS is more related to loss of data. But data can also be "corrupted", that is to say, transformed when traveling. So you obtain a wrong response, or more precisely, a response that doesn't correspond with the one sent by the correspondent. That could be very problematic, and here we can distinguish two cases. First, the response given to the client make it crash, throw an exception, become in error state, stop the processing This case is certainly annoying, but at least we know that there has been an error. Second, the client doesn't makes an error with the retrieved value. The client application can continue working, and eventually give a wrong result, as it has had a wrong input from the server. This is more problematic, because the mistake may be revealed very late (too late).

Generally speaking, the different kinds of data corruption can be classified by the parts of the system they affect [6]. Some examples of corruption types are the following one :

data storing when there are errors in electrical components, bits can be inverted ;

transmission on broadcasting, there can also be inversion of bits due to electrical and magnetic interferences ;

execution that are physical faults on the components (caches, ...) during the execution.

1.2 The theoretical concepts

We are going to see some technics that have been set up, in order to make systems hardy to faults. There are three main categories of techniques to make systems fault tolerant : code for errors correcting, redundancy, and resumption points.

Note that we are also other more complex systems which have been developed to make systems resistant to other kinds of faults [14].

1.2.1 Codes to detect errors

First of all, systems can't be, by nature, fault-resistants. They have to detect that there is something wrong. Sometimes, it is easy to detect that there is a fault : for example, when the system stop due to a wrong value, for example. But the problem can be silent, making eventually a wrong result. That's why it is often necessary to set up a problem detection system.

One of the most used systems is a code to detect errors. Generally, it is called *checksum*. It is used in a lot of systems and protocols which assure a basis of reliability. For example, it is the case for X25 network protocol [2].

1.2.2 Errors correction: redundancy

Redundancy One of the main principles in order to correct errors is to put redundancy of information. Thanks to that, you can make comparisons, and eventually correct some errors. We will see an example of that, here: repetition coding.

The block code technique This uses the block coding technique. In that technique, the message is cut is blocks of n bits (cf. (1) on the figure 1.1). To each of this n blocks, k control bits are added (this is (2) on the figure 1.1). This couple of $n + k$ bits is named a *code word* (this is (3) on the figure 1.1). All the code words are called the *code*[1].

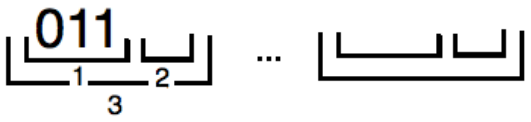

Figure 1.1: Block coding

Coding with repetitions The idea of this code is to add m times the same bit, for each transmitted bit. For example, with $m = 3$, and for the message `10`, we will transmit `111000`. Note that in that example, $n = 2$, and $k = 2$. With this, the probability that a message be seen as valid whereas it is wrong is $p \approx 0.0000003$ (cf. [10] p. 88), which is very little.

Other redundancy possibilities There are other systems which use redundancy principle, not making redundancy of data, but redundancy of machines which process to have the expected result, in order for the client to be enough sure that the given result is good. That make a group n of servers which perform the task, and then, the set of servers is said *t-fault tolerant* ($t \leq n$) if for one work, less than t systems fail for that work [7]. This will be illustrate in existing systems (see subsection 1.3.3 on page 6).

1.2.3 Resumption points

The principle of this method is to make *checkpoints*, to be able to make a restart when a fault comes up. According to Ramamurthy Badrinath and Christine Morin [5], there are three kinds of checkpoints policies :

- *Coordinated checkpoints* the principle is to make checkpoints in predefined points, for example a time interval ;

[1] Cf. [10], subsection 5.4.4, pages 84 – 88

- *Uncoordinated checkpoints* for which checkpoints are not made at predefined intervals, but when it is needed ;
- *Communication induced checkpoints* where checkpoints are made when needed, and other "forced" checkpoints are made at predefined intervals.

Then, it is possible to define links between the checkpoints, in order to restore the closest previous state, by restoring the subset of checkpoints needed, considering the relationships between checkpoints. Ramamurthy Badrinath and Christine Morin present in their paper an example of mechanism, with checkpoint dependancies, and all what is needed for the implementation of the mechanism [5].

1.3 Existing systems

1.3.1 Automated analysis of fault tolerance

US researchers have found a mean to make distributed systems hardier [18]. This method uses "a novel combination of stream-processing models of net-works of processes and abstract interpretation of programs".

This system considers a set of clients C_i and a set of servers S_i, each C_i talking to the correspondent S_i. The behavior of the system is represented by a *Message Flow Graph* (MFG), which nodes are systems and edges are possible communication channels. It considers three criteria (validity, integrity and agreement). The goal is, for failures scenarios, these three criteria must be validated. To do that, some notions are used, like the multiplicity of the messages (which will enable the agreement criteria to be satisfied), the message sequences (considering that systems are FIFO)... They present an algorithm which uses all these notions [18]. The result is a reliable distributed system.

1.3.2 Fault tolerance for clusters and grids

The classic techniques for fault tolerance consist in making a "checkpoint-recovery" on the application (or the server). But it seems that this technique is problematic, because it can lead to loss of data (order messages, sent data...). So some researchers have developed a new technique, to avoid the problems of classical approaches, applied to clusters and grids [13].

They present an algorithm, with a client side and a server side. Based on a "user-specified failover", the client will be able to make a retry mechanism, in function of the returns it gets. The return could be an error state given by a callback method, or the stdout/stderr which is not present at the good location. Thanks to a job ID, the client will be able to ask the server again for the work it knows that haven't been done properly.

1.3.3 Distributed voting

As we saw before (see section 1.2 on page 3), one of the main principles to be tolerant to faults is superfluousness. This has been applied into distributed voting [12].

1.3.3.1 Decentralized voting

As an example, we can take *decentralized voting*. This uses a special protocol : there are several voters, which exchange their votes. The winner result is given to the user, through one of the voters, chosen randomly. This highlights some major problems :

of transmission what if the chosen voter makes a mistake ?

of security what if an external system takes control of the committing voter ?

Algorithms have been proposed to prevent these problems. We can detail the principle of one of them. The client sends the request to a voter, which executes it, broadcasts its answer to the client, and broadcasts the request to other voters. And it is the client which receives several responses from the voters, and that makes a choice about the relevancy of the answer (for example, with n as maximum number of faults, the client can wait for $n + 1$ same answers of voters to decide it is an acceptable answer). But the problem of this approach is the processing time to compare the $n + 1$ answers, that can be long.

1.3.3.2 Air force research laboratory algorithm

The Air force research laboratory has produced an algorithm based on voting, to make systems fault tolerant [12]. It works on a network like a Local Area Network (LAN).

There are a set of independent voters (at least 3 voters), and another set containing the client and the “interface module”.

Chapter 2

Security in distributed systems

Security is very important in science computer, and it is particularly in distributed systems, because information can be intercepted when traveling. Moreover, critical systems often need to be distributed, so it makes this problematic particularly important.

2.1 Notions covered by *security*

2.1.1 Goals of security

Security in information technology, and more precisely in computer science, is generally viewed into two different ways. First, there is *computing security*, whose goal is to make the system strong enough not to be used as a wrong manner by externals. Then, there is *networking security*, whose aim is to make sure that nobody will be able to intercept and understand the content of the data that is traveling [16].

So, in order to reinforce systems about security, it is often necessary to build *security mechanisms*, which, according to William Stallings[1], can be defined as "a mechanism that is designed to detect, prevent or recover from a security attack".

2.1.2 Attacks

In distributed systems, attacks locate generally on the network. So, different kinds of attacks can be highlighted[2]:

interception equally named *eavesdropping*, consist in getting messages and data, without the right to do it ;

identity substitution consist in sending messages to a server, making the server believe it is another machine which sent that message ;

[1] Cf. [16], chapter 1, page 4
[2] cf. [7] p. 480.

message tampering consist in intercepting messages, stopping the messages with the interception, and make things with that message: adding or removing information, broadcast the message later to make the destination believe in a problem from the source...

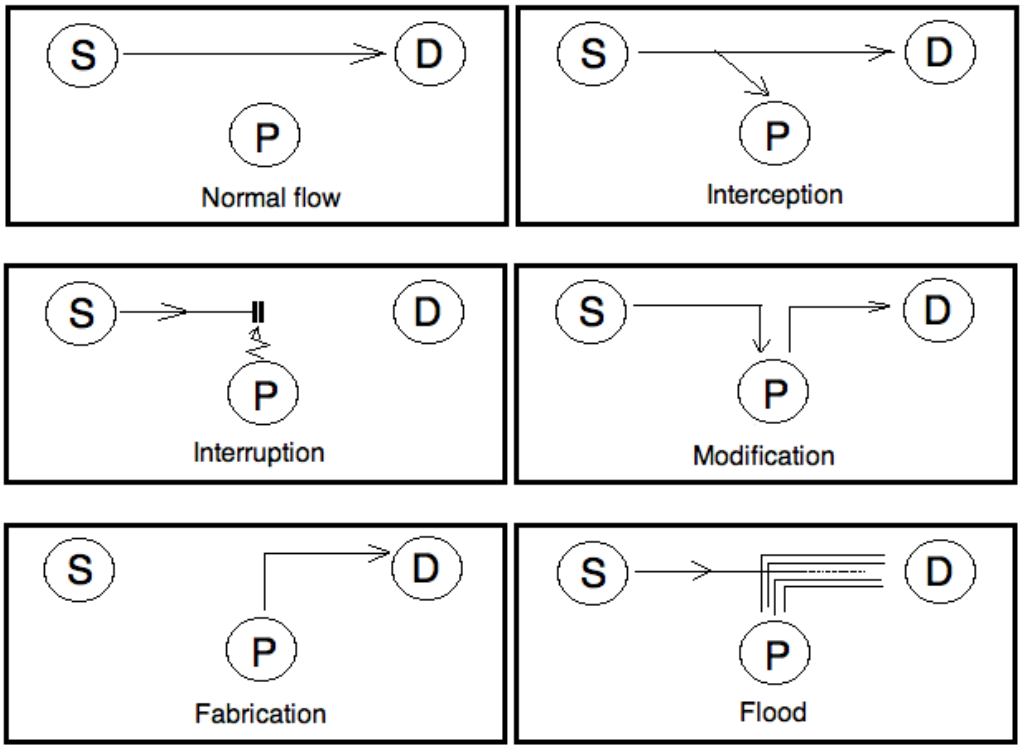

Figure 2.1: Schema of potential attacks, inspired from fig.1.1 in [16], chapter 1, page 7. $S = Source; D = Destination; P = Pirat$.

2.2 Security services

2.2.1 Introduction

To insure an acceptable level of security in a considerate system, *security services* can be built. These security services are software entities, composed by one or several security mechanisms, in the goal to provide strength to its related piece of software, for a given aspect.

There are many security aspects. William Stallings give some, through a

classification[3], but as security is a very open domain, we can't insure to have an exhaustive list of aspects. Nevertheless, we can see some of them, which are maybe the most important ; indeed, the objective is here to give an idea of what problems can be and what are the main principle of found solutions, not to fully cover this subject that is enormous.

2.2.2 Authentication

Authentication of a piece of data is make sure that this data has a reliable origin, that the source told to sent it is the source that actually sent it, and that this source is really allowed to send the data.

There are some means to ensure a good authentication, we will see here two of them, to give an idea of how to do.

2.2.2.1 Identification

Identification is knowing the user that orders the action, or transmits the message. That will enable to make controls, in order to know if that user is allowed to order, or if it is trustful. That need authentication, in order to be sure of the given identity.

In unconnected systems, the principle of authentication by identification is rather simple. Indeed, managing personal data encapsulation by sessions seems to be enough. So in undistributed systems, the user just has to connect himself, identified by his login, and authenticated thanks to his password. That, combined with a right-granting system ("user x has to right to do the action A for the resource r", like for example file rights in UNIX/Linux systems), insures the confidentiality of the data, and the security of orders that can be given.

For distributed systems, the identification, for a given system, of its correspondent, must be reliably established through the connexion, and maintained during the duration of the exchanges.

A mean to do that is to encrypt all the out-coming messages, and decrypt all incoming messages, with a encryption system agreed with the correspondent. So, for identification and authentication in distributed system, security is generally based on cryptography (see section 2.3 on page 12). The security service that makes the transformations (which encrypt and decrypt messages) is called *authentication service*[4].

2.2.2.2 e-signatures

We see here an example of authentication mean for remote (distributed) applications. This is the example of *e-signature*, which is seen here because of its important use.

[3]Cf. [16], chapter 1, page 5
[4]Cf. [7] page 484 – 485.

Principles An e-signature is similar to a hand-writing signature, and has basically the same properties: the e-signature must

1. authenticate the contents of the message ;
2. enable to reliably identify the message's author ;
3. be verifiable by third parties.

Consequently, we can infer that a signature must be unique for a given person, can't be reproducible by another person (so that there can't be identity thief), and must depend on the message (to insure that it is actually this content which has been sent by the signatory). Of course, there are also computing constraints, like for every piece of software, related to acceptable time of producing and recognizing[5].

How to implement it We saw that the signature had to depend on the message. So, generally, there is the use of a hash code, made from the message, because it makes a result of small size comparing to the message size, which is better for transmission (these will not be useful / information data), although hugely depending on the message content (see subsection 2.3.3 on page 14, about hash codes).

Besides, an identification login, depending on the author, must be added to authenticate the author. This identification login must be encrypted, in order for unauthorized people not to know it, so that no identity stealing could be done. Consequently, what is generally used is the public key encryption, which enable anyone to send a message to any other people, making sure that only this other person will be able to authenticate the signature[6] (see subsection 2.3.2 on page 13, for public key encryption).

The result: e-signature components Consequently, the e-signature must have at least two elements: the identification code of the author, and a hash code of the message. If we add a constraint of time authentication, in order for the receiver to know when the message has been sent, is becomes necessary to put a timestamp. These three elements are concatenated, and then are encrypted, through a public-key way. This makes the e-signature (see figure 2.2 on page 11).

2.2.3 Availability

2.2.3.1 About availability

Insurance of a system availability supposes this system to work properly an acceptable part of the time (95% of the time for example). This supposes the system to be enough strong, and not to include security faults, that could be exploited to restart, corrupt, or shut dow the system. There are a lot of known security faults,

[5]Cf. [16], chapter 10, page 300.
[6]Cf. [16], chapter 10, page 301.

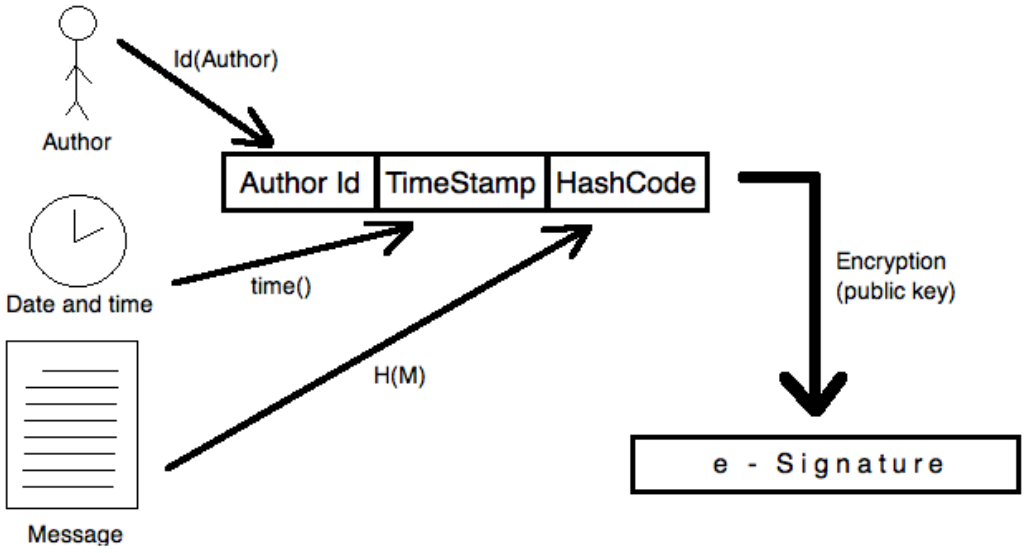

Figure 2.2: E-signature components

depending of the system, the context, even the implementation language... We can take an example to illustrate that.

2.2.3.2 SQL injection

We can take the example of an Internet application. Generally, data is stored in a SQL database (with MySQL for example), some PHP scripts doing the retrieving and processing of data, and ordering the display. Here, it is necessary to prevent some security faults. For example, the SQL injection. To load a new page, the client sends an HTTP request, that can be composed of the post of data. The PHP script can grab the data and use it to retrieve database information. Here, it is necessary to filtrate the posted data, in particular by inserting anti-slahes before apostrophes. Otherwise, the client can hijack the request, in order to make it do whatever he wants.

Indeed, from a request

```
select * from page_table where page_name = '$postedData';
```

where `$postedData` is a PHP variable containing the posted data, if the client posts a string like `' or 1 and XYZ`, the request, after string replacement, will be

```
select * from page_table where page_name = '' or 1 and XYZ;
```

This is very dangerous, as the client can replace XYZ with any SQL instruction (for example, `drop database the_database`). So, all the controls and filtering can be done by a security service, whose role would be here the integrity of the external data.

2.3 Cryptography

As we have seen before, cryptography is a very important element of security. One of the main reasons is that cryptography ensures (or at least, is supposed to ensure) privacy, confidentiality and even sometimes integrity of data.

We will see here an overview of what can be done in encryption, with three main means: private key coding, public key encryption, and hash codes. Each time, we will take one of the most used example.

2.3.1 Private key coding

2.3.1.1 Principle

A private key encryption is an encryption algorithm which uses a key to encrypt, and the same key to decrypt, by the inverse algorithm. So the key must be kept secrete, in order for non authorized people not to be able to decrypt the messages. This can cause problems, because the destination has to possess the private key, in order to decrypt. Moreover, this kind of encryption can cause problems, because the algorithm is symmetric ; that is to say, if the coding algorithm is a function f, then the function f^{-1} exists and it is the decoding function.

2.3.1.2 DES and AES

Lucifer is a encryption algorithm, born in 1971, from IBM labs. The Data Encryption Standard (DES)[7] is the US National Bureau of Standards algorithm, which is born from Lucifer. Now, DES is not enough secure to protect state secrets. One of the reasons is the key, which is 56 bits long, that is too few. Now, even books are produced explaining how to crack DES [3]. So it has been replaced [1] by AES (Advanced Encryption Standard) in 2001, made from the Rijndael algorithm, which is faster and more secure than DES.

According to the scientific community, AES is secure, the only usable way being brute force. Indeed, it is resistant to the newest mathematical attacks, like differential cryptanalysis (that DES is not[8]). This is the way AES works [20]:

1. **Initialization**: the 16 bytes input is split and inserted into a 4x4 matrix.
2. **rounds**: there are a number of rounds, depending of the key size (see table 2.1 on page 13). For each round:
   (a) `SubBytes` step: substitution of the bits according to the *S-box*, a matrix constructed thanks to an invertible affine transformation.
   (b) `ShiftRows` step: cyclically shift bits in each row, by a certain offset (0 for the first line, 1 for the second...).

[7] Cf. [16], chapter 3, pages 49 – 56.
[8] Cf. [17], section 3.6, pages 89 to 109.

(c) `MixColumns` step: linear transformations of each column (each column is viewed as a polynomial, and is then multiplied modulo $x^4 + 1$ with the fixed polynomial $c(x) = 3x^3 + x^2 + x + 2$).

(d) `AddRoundKey` step: For each iteration, a subkey is calculated from the key, thanks to "Rijndael's key schedule"[9]. The subkey is added by a `XOR`.

3. **end**: the final round is the same as the other, except that `MixedColumn` step is not done.

Key size (bits)	Number of rounds
128	10
192	12
256	14

Table 2.1: AES: number of rounds

2.3.2 Public key coding

2.3.2.1 Principle

The principle of public key cryptography is to code using a key, and to decode using another key. Decoding with the coding-key is not possible, which insures the security of the algorithm. Therefore, public key encryption functions are not invertible (that is to say, if the encryption function is f, it must be proved that it doesn't exist a f^{-1} function)[10].

This approach is interesting, because only the receiver of the message has to possess the decryption key, and the encryption key can be diffused (generally in index, catalogs...). So there is no need to a previous agreement of a key, or need to transmit a key. Anybody can write to anyone, without necessarily knowing him, and the source is ensured that only the target will be able to read the message.

2.3.2.2 RSA

This public-key algorithm has been officially born in 1977. Its name comes from the name of its inventors: Ron Rivest, Adi Shamir, and Leonard Adleman[11]. We are going to see quickly who does that work, that will enable to see strengths and weaknesses.

[9] Cf. `http://en.wikipedia.org/wiki/Rijndael_key_schedule`

[10] Cf. `http://en.wikipedia.org/wiki/Public-key_cryptography`

[11] Cf. [16], chapter 6, section 2 (pages 173 – 182), and [17], chapter 4, pages 114 to 161.

Obtaining the keys To create keys, you have to chose two huge prime numbers, p and q. Then, calculate $n := p \times q$, and $\phi(n) := (p-1)(q-1)$. Then chose an integer e, so that $1 < e < \phi(n)$, and e and $\phi(n)$ are coprime. Then compute the relationship $de \equiv 1 \ (mod\ \phi(n))$. The private key is then (n, d) and the public key is (n, e).

Encryption / descryption The receiver calculates a private and public keys, and publishes the public key. To encrypt a message M, the sender has to convert it into a number m, by an agreed-upon reversible algorithm. Then, the sender calculates $c := m^e \ (mod\ n)$, and sends c. The only way to decrypt is to calculate $c^d \ (mod\ n)$, which gives m, so using the private key [22].

Discussion about RSA Therefore, the difficulty for RSA is to find enough large p and q prime integers. Indeed, the security of RSA depends on the diffculty to factorize big integers, given that n is public and that when you have p and q, you can calculate d (the private key). Statistical methods exist to find enough large p and q in a reasonable time. Then, mathematical theorems exist to calculate numbers as $a \equiv b \ (mod\ c)$.

2.3.3 Hashcodes

2.3.3.1 Principle

The principle of a hashcode is to convert a huge quantity of data into a small piece of data of a pre-defined size. It is non invertible because there is loss of information. The hashcode function must have some characteristics: low cost, determinism (a same entry give always the same result), it must be difficult to find the original message from the hashcode, it must difficult to find two random messages which have the same hascode...

2.3.3.2 MD5

The MD5 is probably one of the most used hashcode function[12]. It transforms a text into a 32 characters string. It is very used on the Internet, for example in authentication in websites. When registering on the website, the user gives a password, and $code := md5(password)$ is stored in the database. This enables not to store in clear the user's password. When the users is connecting, the website computes $md5(givenPassword)$, and checks that it is the same as the *code* value stored in database. So, the only possible attacks of that is: brute force and tables containing the hash codes of statistically frequent strings. The security of MD5 is based on the hashcode functions criteria that it must be very difficult to find two random strings or texts that have the same hashcode.

[12] Cf. [16], chapter 9, pages 271 – 280.

Chapter 3

Load balancing

3.1 What is load balancing?

In distributed systems, there are several machines which work together trough a network. Each of these machines have a certain computational power. There are some distributed system which are used to take advantage of the total computational power of a set of machines working together. This is interesting, because avoids to make huge machines to do very complicated and long calculus, that could be very expensive, and quickly limited by the technology.

So, for that kind of use, it is mandatory to balance the load between the processors. Indeed, it must be ensured that the work is done properly, that no processor is inactive, that no processor is overloaded... So load balancing regroups all the techniques and methods to balance the work on all the different machines available in the distributed system [19].

3.2 Example of load balancing systems

We see here some examples of load balancing problematics in distributed systems, to illustrate how the concept exists in some systems.

3.2.1 Load balancing on grids and clusters: SGE

What are clusters? Clusters are set of machines which are connected and work together, in order to provide huge computation services[1].

Sun Grid Engine SGE is an open-source batch-queuing system, developed by Sun Microsystems[2]. SGE is capable of balance given works on several machines, each containing one or more processors. It maintains queues of work, which are treated by processors. SGE dequeues the work as the processors become no longer

[1] For more information, you can go to `http://en.wikipedia.org/wiki/Cluster_(computing)`
[2] Website: `http://gridengine.sunsource.net/`

busy. It supports muti-clustering, XML reporting, scheduler fault tolerance. It can run in a lot of platforms, like BSD, Linux, Mac OS X, Solaris, Windows...

3.2.2 Load balancing in Grid'5000

Grid'5000 Grid'5000 is a wide network of processors, in France. For the moment, Grid'5000 groups 1,300 bi-processors, distributed in 9 sites in France : Bordeaux, Grenoble, Lille, Lyon, Nancy, Orsay, Rennes, Sophia-Antipolis and Toulouse. This huge grid is very useful for researchers, to make very complicated computing work (because of its computing power), to make studies and tests related to distributed computing, and large scale experiments[3].

Load balancing For load balancing, experimentation have been conducted on Grid'5000. We can take an example of an experimentation led in Grid'5000. It is to experiment a load balancing algorithm, by researchers of Franche-Compté university (France). This algorithm, after theoretical study and practical tests, reveals an optimization of execution time up to 74% [8].

[3]For more information, you can go to the website: `http://www.grid5000.fr`

Conclusion

So, we have seen in this report, three major components of managements of distributed systems. To manage properly distributed systems, it is necessary to have a minimum level of fault tolerance, of security, that is mainly based on cryptography, and some concepts of load balancing, to take advantage of the distributed system.

Fault tolerance is very important to have strong and reliable systems, as network-based systems can be very sensible to errors (due to transmission for example) and attacks. Some systems already exist (distributed voting for example), based on improved theoretical concepts (redundancy, resumption points...). But this is a domain where research and technique improvement is very important; the future here is thus improving existing methods, and maybe try to find other concepts.

Security is another very important topic, because there can't be important applications set up without a minimum insured level of security. In distributed systems, this is provided by software components names *security services*. These services rely a lot on cryptography, to be able to securely transmit messages.

In cryptography, current coding algorithms are considered as sure, like AES or RSA. Indeed, it is impossible currently to decode messages coded by this algorithms (with sufficient large key for RSA), in spite of all the mathematical attacks (differential cryptanalysis for example) which are being developed. So for cryptography, the short and mid-term future is ensured by current encryption methods. For the long-term, the future of that domain is to extend key sizes, and also work in parallel in new encryption method which will resist to the most advanced attacks.

In load balancing, the future is to try to limit at its maximum the number of delays and costs of algorithms, and to find methods to make load balancing algorithms be executed as few as possible [11]. Besides, there is also research to select "the best predicting methods for processor speeds" [9]. This has to be done with progress in networking too.

So there are a lot of research fields for the future, to improve the management of distributed systems.

Bibliography

[1] Advisory memorandum on the retirement of data encryption standard (des) based cryptography to protect national security systems. `http://www.cnss.gov/Assets/pdf/cnssam-ia-2-04.pdf`, March 2005. Official document.

[2] Cisco. X.25 overview. `http://www.cisco.com/en/US/docs/internetworking/technology/handbook/X25.html`, 2008.

[3] Electronic Frontier Foundation, editor. *Cracking DES*. O'Reilly, May 1998. isbn: 1-56592-520-3.

[4] Sape and Mullender Second edition, editors. *Distributed systems*. -, 1993. isbn: 0-201-62427-3.

[5] Ramamurthy Badrinath and Christine Morin. Common mechanisms for supporting fault tolerance in dsm and message passing systems. `http://www.inria.fr/rrrt/rr-4613.html`, November 2002. Research report.

[6] Lucas Baire, Thomas Ernest, Emilien Girault, and Léo Terziman. Projet safescale, présentation de l'état de l'art. Report of a research project at INSA of Rennes (France), 2008.

[7] Georges Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed systems, concepts and design – Second edition*. Addison - Wesley, 1994. isbn: 0-201-62433-8.

[8] Christophe Denis, Raphael Couturier, and Fabienne Jézéquel. Load balancing of the direct linear multisplitting method in a grid computing environment. `http://info.iut-bm.univ-fcomte.fr/gremlins/gremlins_lb.pdf`, February 2008. Research report.

[9] Menno Dobber, Ger Koole, and Rob van der Mei. Dynamic load balancing for a grid application. `http://www.math.vu.nl/en/publications/stochastics/2004/WS2004-15.pdf`, Vrije Universiteit, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands, *(No date)*.

[10] Pascal Garcia. Introduction à larchitecture des ordinateurs. Lecture notes - INSA of Rennes (France), February 2008.

[11] J. Ghanem, S. Dhakal, M. M. Hayat, H. Jérez, C. T. Abdallah, and J. Chiasson. Load balancing in distributed systems with large time delays: Theory and experiment. http://www.eece.unm.edu/faculty/hayat/med_04_ghanem.pdf, *(No date)*. Research report.

[12] Ben Hardekopf, Kevin Kwiat, and Shambhu Upadhyaya. Secure and fault-tolerant voting in distributed systems. publication from Air Force Research Laboratory.

[13] Kshitij Limaye, Box Leangsuksun, Zeno Greenwood, Stephen L. Scott, Christian Engelmann, Richard Libby, and Kasidit Chanchio. Job-site level fault tolerance for cluster and grid environments.

[14] Nasra Mahamoud and Steve Attelly. La tolérance aux fautes dans les systèmes informatiques. http://www.univ-angers.fr/docs/etudquassi/Tolerance_aux_Fautes.pdf, February 2002. slide show.

[15] Morris Sloman. *Network and distributed systems management.* Addison-Wesley, 1994. isbn: 0-201-62745-0.

[16] William Stallings. *Cryptography and network security, principles and practices – second edition.* Prentice Hall, 1999. isbn: 0-13-869017-0.

[17] Douglas R. Stinson. *Cryptography – theory and practice.* CRC Press, 1995. isbn: 0-8493-8521-0.

[18] Scott D. Stoller and Fred Schneider. Automated analysis of fault-tolerance in distributed systems, 2005.

[19] Taieb F. Znatit, Rami G. Melhem, and Kirk R. Pruhs. Dilation based biddin schemes for dynamic load balancing on distributed processing system. http://www.cs.pitt.edu/~melhem/vita/doc/00633104.pdf, *(No date)*. Study from university of Pittsburgh.

[20] Wikipedia-en. Advanced encryption standard. http://en.wikipedia.org/wiki/Advanced_Encryption_Standard, November 2008.

[21] Wikipedia-en. Fault-tolerant system. http://en.wikipedia.org/wiki/Fault-tolerant_system, September 2008.

[22] Wikipedia-en. Rsa. http://en.wikipedia.org/wiki/RSA, November 2008.