Journal of
Applied
Business
Information
Systems

http://www.jabis.ro

Developing reliable distributed applications oriented on large datasets

Sorin PAVEL*

* Academy of Economic Studies, Dauphine Universite Paris

Abstract: *Distributed systems are defined and certain benefits and dangers of use are underlined. Software reliability, maintainability and availability is mathematically defined and methods of quantifying are specified. The development process for distributed software is presented and methods for increased reliability are proposed for each phase.*

Keywords: distributed systems, reliability, software development process

1. Distributed systems, organizational benefits and draw-backs

Computerization of modern society and large scale development of software products that work with large collections of data lead to wide spreading of distributed systems in any field of activity: economic, social and cultural. In addition, promulgation of new IT laws and disproportionate costs of processing raw very large data sets in contrast with processing already prepared data encourage the formation of large collections of data. However, these collections have to respect some quality standards because of the effects arising from errors in very large sized data sets and significant efforts to correct such errors.

Throughout time, distributed systems have been defined in several ways, such as:

- "you know you are using a distributed system when failure of a computer which you did not even know that existed retain you from doing your job" [LAMP87];
- a collection of computers that do not share common clock or common physical memory but which communicates through messages sent over a network; computers have their own memory and operating systems and work to solve a common problem [SISH94];
- a collection of computers that appears to users as a single coherent system [TAST03];
- a wide range of computers, from poorly connected systems, such as very large networks, to strong connected systems such as local area networks, and to strongly coupled systems such as multiprocessor systems [GOSC91];

Since there are many definitions of a distributed system [GHOS07] some properties are widely encountered:

- multiple autonomous processing entities, each with its own local memory [ANDR00], [DOLE00], [GHOS07], [LYNC96], [PELE00], [KSSI08], [VSTY09]; the system consists of several sequential processes that are independent and have their own resources; processes must have different address spaces for the system to be able to be called "distributed"; multiple processing units with shared memory are not distributed systems, but parallel systems (Figure 1.1);

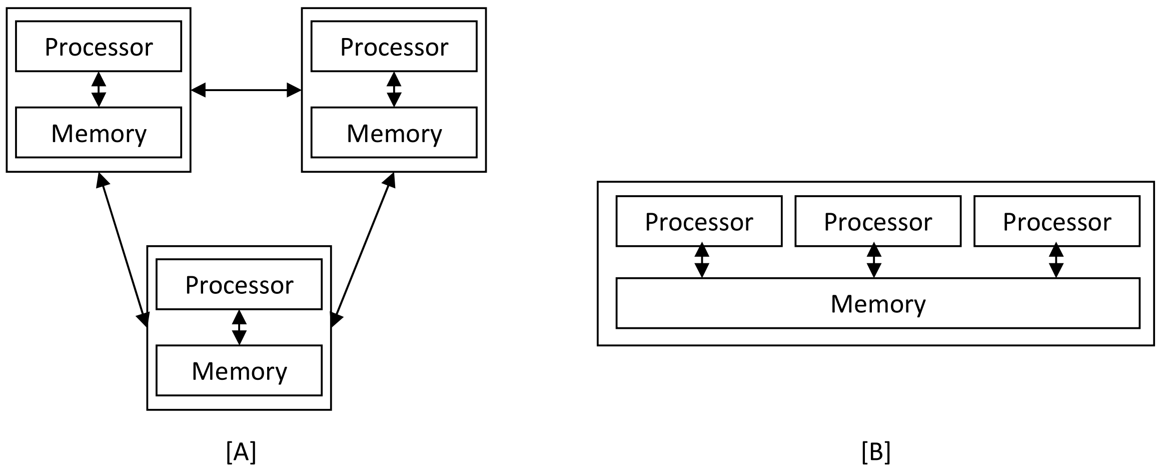

Figure 1.1 – Difference between distributed systems [A] and parallel systems [B]

- communication via messages between entities [ANDR00], [GHOS07], [PELE00], [KSSI08]; processes communicate with each other through messages which arrive in a finite time; the order of messages depends strictly on physical characteristics of the communication channels;
- common purpose of processes [GHSO07], [KSSI08], [PELE00]; processes must interact with each other to achieve a goal; if two processes P and Q are considered in a network of processes, P calculates f(x) = x2 for some values of x and Q a set of values multiplied by pi; the two processes do not form a distributed system, since there is no interaction between P and Q; if the two processes cooperate to calculate areas of circles of radius x, then P and Q is a good example of distributed system.

In [CUCO05], a computer system is distributed if it consists of hardware located at least two geographically distinct sites, connected electronically by telecommunications, where processing or data storage occurs at more than one site.

In a distributed system there are a number of important features to be considered [CUCO05], [BEGR92]:

- transformation, integration and distribution of data and processes between nods;
- the locations of processing, and the type of interaction between them;
- the locations of data storage and the way data is presented to users;
- the nature of communication links between the various locations;
- the users' access to data and its security

Since their appearance in 1970 the distributed systems have become increasingly used. This is partly because of the technological advances in telecommunications, distributed databases and communications software and partly because of the recognition of the organizational benefits by the users of such systems. These benefits are as follows:

- increased user satisfaction; as users can benefit from the application even thou they are remote and separated from the source of computing power; however, from a central organizational perspective, the sites are connected with one another and with the center in order to act congruently with the corporate goals;
- flexible system development; as the application is growing, more computing power can be added incrementally by purchase/installation/connection of new nods to the network; in a centralized system the growth is hardly incremental because any overload will lead to replacing the current system with a more powerful computer, which is expensive;
- low communications costs; there are many situations within a distributed system when computing takes place locally; the network is used only when data or processing from

elsewhere is needed; the communication costs are lower compared with a centralized system where every action demanded network traffic;
- increased fault resistance; the remaining machines in the network continue to function and take over the work of the failed node; what can be achieved depends on the particular network topology and the communications software;
- increased response times; the centralized systems have poor response time, especially at peak loading.

Though these organizational benefits may seem persuasive, there are potential dangers and costs associated with distributed systems:

- little centralized standard setting and control; in a distributed system where processing and data storage are located at many sites, there's a tendency of developing local practices, alterations and software patches in order to meet specific user needs; all these lead to lack of standardization and great heterogeneous data management which affects the quality of data, communication and security;
- complex networking software and hardware is needed; communication is dependent of the quality of network traffic; any failure of the network would freeze the activity;
- possibility of replicated common data at several sites; if the same partition of data is needed and used at several sites, it is common for the data to be held as copies at each of the several sites rather than be held once and accessed through the network when needed; this cuts communication costs and increases response times but may lead to inconsistencies when data is updated or changed.

The nowadays distributed systems are forced to manage very large collections of data, as the citizen-oriented software distribution increases and promulgation of new IT laws leds to developing applications that work with large data sets:

- telecommunications operators record each call or message within the network for a period of six months;
- internet and e-mail providers record accessed sites for each IP address in its administration, together with the exact date of access and data about each email message;
- government keep track of different payments for millions of people;
- national providers of utilities – gas, electricity etc. – process hundreds of millions of annual consumer bills;
- online search engines integrate content management of billions of sites.

The impact of such distributed applications is capital in the modern economic trades, therefore the quality of each process has to meet certain standards and regulations.

2. Software reliability, maintainability, availability

A study over software production shows that professional programmers have on average 6 software defects per 1,000 lines of code (CL). Given this rate, a typical commercial application of 350.000 CL contains over 2000 bugs, including memory leaks, language errors, library errors etc. As the project develops, the rate increases exponentially (Figure 2.1).

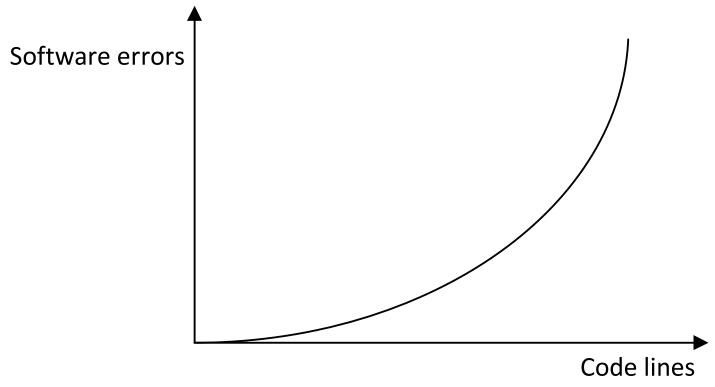

Figure 2.1 – The rate of software errors by lines of code

Another study conducted by Microsoft and cited in [PHAM00] shows that for a single location and correction of a programming error it takes about 12 hours. At this rate, for the same application of 350.000 CL, over 24.000 hours or 11.4 man-years are needed to debug the application, at a cost of over $1 million. To overcome such situations there's an increasing need of reliable software.

Reliability is defined as the probability of success or likelihood of the system to function properly within certain design limits. Mathematically, reliability *R(t)* is the probability that the system will operate successfully in the interval from time 0 to time *t*:

$$R(t) = P(T > t), \quad t \geq 0$$

where *T* is a random variable representing failure time.

Unreliability is a measure of failure, defined as the probability that the system has an error up to time *t*.

$$F(t) = P(T \leq t), \quad t \geq 0$$

F(t) is the failure distribution function. If the variable *T* has density function *f(t)* then

$$R(t) = \int_t^\infty f(s)ds$$

or

$$f(t) = -\frac{d}{dt}[R(t)]$$

If it is considered a successful tested system that works well when placed in use at time *t = 0*. The system is increasingly exposed to error as the time increases. The probability of success for an infinite time interval is zero. We must therefore take into account the time when it comes to reliability. It can be said about a system that has the reliability of 0.995 for a period of 24 hours. But a statement such as system reliability is 0.995 has no meaning because the period is unspecified.

If a system reliability function is given by *R(t)*, the time to failure when the component is working well, or the average time of failure *(ATF)* is given by:

$$ATF = \int_0^\infty tf(t)dt$$

Subtituting the last two equations and integrating by parts, we obtain:

$$ATF = -\int_0^\infty td[R(t)] = [-tR(t)]|_0^\infty + \int_0^\infty R(t)dt$$

The first term of the equation is 0 at both limits, since the system must fail after a finite period of time. Thus, *ATF* is:

$$ATF = \int_0^\infty R(t)dt$$

ATF is to be used when the distribution function for failure is known because the level of reliability determined by *ATF* depends on it. When a system fails to function satisfactorily, repair is carried out to locate and correct the error.

Maintainability is defined as the probability of a failed system to be restored to the conditions specified within a certain timeframe in which maintenance is carried out according to procedures. In other words, maintainability is the probability of isolating and repairing a system error in a given time.

Let *T* be the variable describing the time of repair or the total downtime. If the repair time *T* has a density function *g(t)*, then maintainability *V(t)* is defined as the probability that the failed system becomes operational by the time *t*.

$$V(t) = P(T \le t) = \int_0^t g(s)ds$$

Average repair time *(ART)* or downtime is the expected value of the variable time for repair and is given by:

$$ART = \int_0^\infty tg(t)dt$$

The repair time of the system consists of two separate intervals: active repair time and passive repair time. The passive time is determined by time elapsed from the occurrence of failure to the start of the repair, given by the overall travel time at the customer site. Active time is the time in which the system is actually repaired:

- the time the error is located and isolated;
- the time the erroneous component is replaced;
- time to test the new components.

Active time is optimized by designing the system so that errors are detected and isolated in a short time. As the system becomes complex, it becomes increasingly difficult for errors to be isolated.

Reliability is a measure of the success of a system for a specified period of time. There are situations where errors or repairs of the system are not accepted (space missions, aeronautics).

Availability is defined as the probability that the system is operational at time *t*. Mathematically,

$$Availability = \frac{System\ up\ time}{System\ up\ time + System\ down\ time} = \frac{ATF}{ATF + ART}$$

Availability is a measure of success used mostly for repairable systems. For non-repairable systems, availability is the same as reliability. In repairable systems availability is greater than or equal with reliability.

3. Increasing reliability through the development lifecycle

A software development process or lifecycle is a structural approach to the development of a software product. It covers the entire time span of an application or project from the rising of an idea to the "retired" software.

As any other software, the distributed applications oriented on very large datasets are developed following five successive phases shown in Figure 3.1:

1. analysis (requirements and functional specifications)
2. design
3. coding
4. testing
5. operating and maintaining

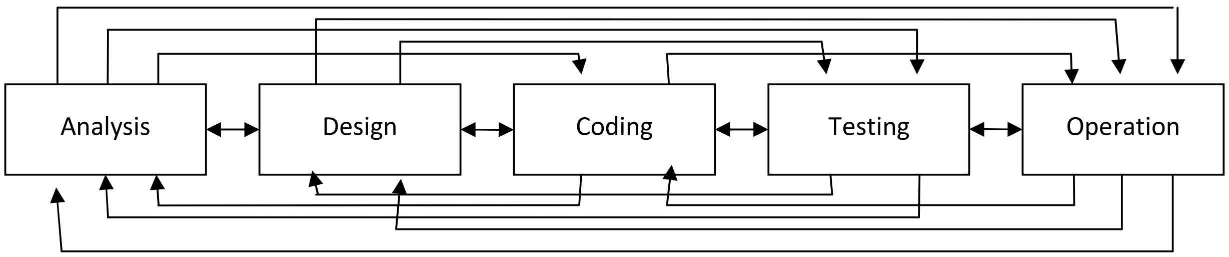

Figure 3.1 - Software development process

Increasing reliability through development process is aiming at implementing ways to reduce error appearances and building software easy to debug and correct. Table X shows the amount of errors introduced and detected in a software lifecycle of a distributed application oriented on very large datasets. In the early stages of software development, predictive models are used because no failure data is available.

Table 3.1 – Percentage of errors introduced and detected in development phases

Development stage	*Errors introduced (%)*	*Errors detected (%)*
Analysis	50	15
Design	30	10
Coding and testing	20	55
Operating	10	20

At each level, a model for determining reliability level is needed. Reliability estimation is based on the analysis of failure data. For each phase, methods for increasing reliability are given, having considered the particularities of very large distributed applications.

Analysis phase's purpose [PHAM00] is to define requirements and provide specifications for the subsequent phases and activities. Research indicates that increased effort and care during this stage will generate significant rewards in terms of reliability and general software quality. We will discuss the distributed application special reliability issues using the stage model with three main activities described in [PHAM00]: problem definition, requirements, specifications.

The problem definition for large distributed applications includes the problem statement and the scope of the project. The issues discussed are:

- what is the problem trying to be solved (from the users perspective)?

- why is it necessary to have a software solution?
- what is the reason for using large collections of datasets?
- what are the advantages of using distributed computing for the software solution?

Within this activity all the reasons are clearly specified and sustained because this can be the starting point or the end of a distributed application. A comparative study it's a must between solving the problem in a centralized system and doing it distributed.

The requirement activity consists of collecting and analyzing requirements. These are descriptions of the capabilities and constraints of the software product. For large distributed applications, the issues approached have to be:

- who will use the application; the target group has to be well defined, including geographical, hardware and knowledge context of the general user;
- what are the users' expectations in the behavior of the distributed application; detailed description of each use case has to be given from users working on different OS, with different resources;
- what are the users' reliability needs; the users limits concerning eventual errors or situations of unavailability.

Requirement analysis for distributed applications includes a feasibility study and documentation including costs estimation, benefit estimation, schedule and risk analysis. The documentation for requirements is the project plan that indicates the budget, the procedures and the schedule of the project. Until this point, there are no significant differences between classic projects and distributed applications because from the user's perspective, it has little importance. From the following activity (the building of specifications) the details refer to the software engineer needs and specify:

- how to process input information into expected results;
- how the software interacts with the distributed environment and with diverse other systems;
- the need of technical support for implementing (hardware, software, people);
- quality standards and measurements.

It is necessary to let the user review the output of this phase and even run some prototypes of the application, before going to the next step. The reliability of the distributed application depends on how well the users and software team communicate and collaborate at the analysis phase.

The design phase covers the issues concerning the building of the system to perform as required. The two main activities of this phase refer to software architecture design and detailed design. For distributed applications, the system structure design is capital because it partitions the software system into smaller part that will be scattered on the network. Before subdividing the system there's a need of further specification analysis concerning the performance and security requirements, assumptions and constraints and the need for software and hardware on the network. System distribution includes subsystem process control and interface relationship: the internal interfaces which control the interaction of distributed subsystems and the external interfaces which facilitate interaction with environment. All of these issues are integrated in the system architecture document.

Detailed design is about designing the program and algorithmic details. The activities within detailed design are program structure, program language and tools, validation and verification, test planning and design documentation. For distributed application, a distributed architecture is implemented which is one of the following:

- *client–server*: client code contacts the server for data which is formatted and displayed it to the user; any permanent input of the user is committed back to the server.
- *3-tier architecture*: the presentation tier, the application tier and the data tier work together in a distributed environment for a common goal;
- *n-tier architecture*: some web applications further forward their requests to other enterprise services; this type of application is the one used in application servers;
- *tightly coupled*: a cluster of machines work together, running a shared process in parallel; the task is subdivided in parts that are processed individually by each one and then assembled together to make the final result;
- *peer-to-peer*: there is no machine or machines that provide a service or manage the network resources; all responsibilities are uniformly divided among all machines, known as peers, which can serve both as clients and servers;
- *space based*: infrastructure that creates the illusion (virtualization) of one single address-space; data are transparently replicated according to application needs;

To provide reliability in software architecture of distributed applications oriented on large collections, there are two important parts that have to be cleared: error detection and error isolation. In order to meet reliability objectives several methods are recommended:

- building checkpoints - places of restarting the execution in case of failure;
- using redundant software elements – because in the distributed systems the physical space is becoming of less importance, redundant elements increase their positive impact by offering pieces of quality data in case of need;
- identifying high-risk areas – using different models (like the fault tree analysis) the impact of the errors are approximated and critical areas are located, focusing extra-care from the developers.

During detailed design, the selected data sets structures and algorithms are implemented in a particular programming language on particular machines of the distributed network. Choosing the appropriate program language and tools is essential.

Coding involves implementing the design into code of a programming language. The main activities of this phase include: identifying reusable modules, editing code, inspecting code and test planning.

Coding for reliable, distributed software oriented on large datasets involves:

- practicing a development methodology; it facilitates good communication between project team members, helping reduce introduction of faults into software;
- constructing modular, independent parts (procedures, functions) that are distributed through the nodes of the network;
- identify and update reusable modules from other reliable similar systems;
- inspecting written code which includes code reviews and quality verification;
- controlling changes and updates and maintaining an orderly procedure for submitting, tracking and completing requested changes to items;
- measuring reliability of acquired software especially of the reusable modules, before developing them.

The test planning should provide details of what needs to be tested, testing strategies and methods, testing schedules and all necessary resources.

Testing is the activity of verification and validation for the newly written software product. The goals of the testing phase are:

- to find and eliminate software errors or faults;
- to check for all specified functionality of the product;
- to estimate the operational reliability of the software.

During the test phase of distributed software, issues that have to be taken into account refer to:

- testing the units (software models) by themselves for functionality, performance and security;
- testing the subsystems, focusing on interfaces and interdependences of the modules scattered through the distributed system;
- testing the communication between subsystems through the network;
- testing the system as a whole for functionality and performance;
- testing the system's behavior in diverse hardware (wireless, intra/extranet) and software environments (SO, frameworks);
- testing the system with a group of common users.

Testing the software by an independent, specialized group – a different one than the developers – provides assurance that the system satisfies the original requirements. The costs of error correcting should also be taken into account. The costs increase rapidly during the latter parts of the development cycle because the impact gets stronger. The probability of fixing incorrectly a known problem also arises rapidly during the latter stages. Sometimes an incorrect fix to a problem causes more harm than the original problem.

The operating phase usually contains activities such as installation, training, support and maintenance. In most cases, the installation for distributed software is unnecessary, because the application is online and already accessible. However signing-up or downloading some resources or tools might be necessary [OCEA10].

The reliable distributed software will have at operating phase:

- online instructions for each functionality of the software;
- 24/7 online support for users question and answers;
- tools for monitoring users processes and the distribution of tasks among the network;
- tools for balancing the workloads between nodes of processing;
- available machines to scale the system in case of an great computing need;
- presence of maintenance activities that refer to: correction of errors, adaptation to other changes, perfection of acceptable functions and prevention of future errors.

The systems improvement phases are similar to those of the system development lifecycle: analysis, design, coding and testing.

4. Conclusions

Distributed systems have an increasing presence in the current software environment giving the widespread of networks and online computing. The organizational benefits of using distributed applications overcome the potential dangers and costs associated with them. However, running bad distributed systems could ruin an entire organization, especially if the nature of the application is related to financial status.

Reliability is a quality metric that refers to the capacity of a system to function properly within certain design limits. The methods of increasing reliability are strictly related to the software development lifecycle phases. Each of them has certain aspects to take into account and to focus on. To improve the general design progress, great attention must be given to the way the tasks are distributed through the nodes of the distributed systems, especially in the case of scaled system. The methods given in this paper have to be detailed and further studied, but certain aspects are already practiced in real life.

Bibliography

[CUCO05] Graham Curtis, David Cobham – Business information Systems, 5th edition, Pretince Hall, 2005, ISBN 0273687921

[PHAM00] Hoang Pham – Software Reliability, Springer-Verlag Singapore, 2000, ISBN 9813083840, pp. 339

[LYNC96] Nancy A. Lynch – Distributed Algorithms, Morgan Kaufmann, 1996, ISBN 1-55860-348-4

[KEID08] Idit Keidar – Distributed computing column, The year in review, ACM SIGACT News 39 (4): 53–54, 2008

[GHOS07] Sukumar Ghosh – Distributed Systems, An Algorithmic Approach, 2007, Chapman & Hall/CRC, ISBN 978-1-58488-564-1

[ANDR00] Gregory R. Andrews – Foundations of Multithreaded, Parallel, and Distributed Programming, 2000, Addison–Wesley, ISBN 0-201-35752-6

[DOLE00] Shlomi Dolev – Self-Stabilization, 2000, MIT Press, ISBN 0-262-04178-2

[PELE00] David Peleg – Distributed Computing: A Locality-Sensitive Approach, SIAM, 2000 ISBN 0-89871-464-8

[KSSI08] Ajay D. Kshemkalyani, Mukesh Singhal – Distributed Computing, Principles, Algorithms and Systems, Cambridge University Press, 2008, ISBN-13 978-0-511-39341-9, 756 pg

[VSTY09] Deo Prakash Vidyarthi, Biplab Kumer Sarker, Anil Kumar Tripathi, Laurence Tianruo Yang – Scheduling in distributed computing systems, Analysis, Design and Models, Springer Science-Business Media, 2009, ISBN 978-0-387-74483-4

[LAMP87] L. Lamport – Distribution email, 1987, disponibil: http://research.microsoft.com/users/lamport/pubs/distributed_systems.txt.

[SISH94] M. Singhal, N. Shivaratri – Advanced Concepts in Operating Systems, New York, McGraw Hill, 1994.

[TAST03] A. Tanenbaum, M. Van Steen – Distributed Systems: Principles and Paradigms, Upper Saddle River, NJ, Prentice-Hall, 2003.

[GOSC91] A. Goscinski – Distributed Operating Systems: The Logical Design, Reading, MA, Addison-Wesley, 1991.

[OCEA10] http://oceanstore.cs.berkeley.edu/ accessed december 3rd 2010

[BEGR92] David Bell, Jane Grimson – Distributed database systems, Addison-Wesley Publishers Ltd, 1992, ISBN 0201544008, pp. 410