Distributed Systems

Wouter Borremans
0461911

22nd November 2004

1 Preface

This document describes my personal view on *Distributed Systems*. In this report i will describe the meaning of several terms.

2 What is a distributed system?

Nowadays millions of computers are interconnected by a world-wide network, also called the Internet. The Internet is not the only network that is currently used, networks such as Mobile phone networks, corporate networks, factory networks, campus networks and in-car networks are widely used nowadays. All of these networks can be studied under die the heading *distributed system*. A distrubuted system is a system that can be seen as one in which hardware or software components located at networked computers communicate ad coordinate their actions only by passing messages.

3 Explanation of terms

In this chapter i will discuss every term briefly. All terms are directly in relation with a distibuted system and describe their characteristics.

3.1 Scalability

Distributed systems (DS) operate at many different scales, the range differs from an intranet to the Internet. In case of a DS, scalability means if it will remain effective when there is a significant increase of resources and the number of users.

The Internet is the perfect example of a DS, the number of users and hosts have increased dramatically. The design of scaleable DS presents the following challenges;

- *Controlling the cost of physical resources* When the demand of the availability of a DS is growing, it should be possible to extend the system at reasonable cost. It must be very easy to add new server systems to avoid performance bottlenecks.

- *Controlling the performance loss* Management of data resources which represent a huge amount of data and which importance of availability is very high. The availability of DNS (Domain Name Server) is of a huge importance. A great number of requests is sent to those servers. When a single system fails, another (slave) system takes over it's task. No or less performance loss is now achieved. A great importance designing a DS.

- *Preventing software resources running out* The perfect example in this case is the lack of free IP address space available globally. A new IP address space had to be designed; *IPv6*. It is very difficult to predict the future demand of a software resource, over-compensating for future growth may be worse than adapting to a change when we are forced.

- *Avoiding performance bottlenecks* In general, algorithms should be decentralized to avoid performance bottlenecks. To illustrate this point just think of the predecessor of the Domain Name System (HOSTS.TXT) in which the name table was kept in a single master file that could be downloaded by any computer that needed the file. This method works if the number of hosts is limited, if the number of hosts grows dramaticall this method isn't suitable anymore. The DNS system removed this bottleneck by partitioning the name table between servers throughout the internet and local networks.

3.2 Openness

The openess of a computer system can be defined as the way it can be extended and re-implemented in various ways. Looking at a DS, this means the degree to which new resources can be added and be made available for use.

Openness cannot be achieved unless the specification is documented properly. Developers need that to be able to cooperate with each other.

The publication of interfaces is only the starting point for adding and extending services in a DS. The real challenge for the designers is to tackle the complexity of DS consisting of many components engineered by different people. A RFC (Request For Comment) has set a solid base for documenting standards throughout the internet.

3.3 Heterogeneity

Nowadays the Internet enables users to access services and run applications over a hetrogeneous collection of computers and networks. This variety and difference applies to the following;

- Networks
- Computer Hardware
- Operating systems
- Programming languages
- Implementations by diffrent developers

The internet consits of many different hosts, alle running their own specific operating systems with their own standards. Though, their differences are masked by the protocol they are communicating with. Datatypes such as integer are respresented in different ways on different sorts of hardware.

To solve the problem regarding the different interpretation of datatypes *Middleware* has been introduced. Middleware applies to a software layer that provides a programming abstraction as well as masking the hetrogeneity of the underlying networks, hardware etc.

Well known middleware is *CORBA* (Common Object Request Brooker) and *JAVA RMI* (Remote Method Invocation). Middleware provides a uniform computational model for use by programmers of servers and distributed applications.

3.4 Resource sharing

Resource sharing means the ability to use any hardware or software or data anywhere in the system. A resource manager is needed to control access provides naming schemes and concurrency. Resource sharing models are used to describe how the resources are provided, how they are used and provided and how the user interacts with it.

3.5 Fault-Tolerance

Many computer systems sometimes fail. When this happens, there is a possibility that incorrect results are produced. This can cause that processes are stopped before they have completed the intended computation.

In DS, failures are partial. This means that some components fail while others continue to function properly. Below a few techniques for dealing with features;

- *Detecting failures* Some failures can be detected, for example the checksums of files. Sometimes it is very difficult or even impossible to detect of a remote server fails or has crashed. The challenge in this case is to manage in the presence of failures that cannot be detected but may be suspected.
- *Masking failures* Some failures that have been detected can be hidden or made less severe;
  - Messages can be retransmitted when they fail to arrive
  - File data can be written to a pair of disks so that if one is corrupted , they other one may still correct.
- *Tolerating failures* Most of the services running on the Internet exibit failures. Their clients can be designed to tolerate failures. In practise this means that the users have to tolerate them as well.
- *Recovery from failures* Recovery involves the design of software so that the state of permanent data can be recovered or rolled back after a server has crashed.
- *Redundancy* Services can be made to tolerate failures by the use of redundant compontents. (e.g. At least two routes to one destination between

any two routers on the internet or name tables of the DNS system are replicated over more than one server)

3.6 Transparency

Transparancy is defined as the concealment from the user and application programmer of the seperation of components in a distributed system, so that the system is perceived as a whole rather than as a collection of independent components.

The scope of transparency is listed below;

- *Access transparency* Enables local and remote resources to be accessed using identical operations. This means that the user isn't aware of differences in the definition of datatypes on a operating system.
- *Location transparency* Enables resources to be accessed without the knowledge of the location. In practise this means a user won't notice the difference viewing a website's mirror in the Netherlands or Germany.
- *Migration transparency* Enables resources to be moved without affecting the way they can be accessed througout a DS. In practise this can be a FTP server which is moved to another location but still has the same or another IP address. (It still has the same purpose no matter it's location or IP number.
- *Replication transparency* Enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers. Think of the DNS system, or a cluster of computers serving the same content.
- *Concurrency transparency* Enables several processes to operate concurrently using shared resources without interference beween them. For example: The SNB group using the makesite tool at Firenze at the same time, the users won't notice that other users are using the program at the same time.
- *Scalability transparency* Allow the system and applications to expand in scale without to change the system structure or application algorithms. A system like this is operating at SARA (Beowulf cluster) which is continously expanded with extra servers to increase CPU power.
- *Performance transparency* Makes users unaware of other (heavy load) processes are running too. The user always experiences the same performance. Think of QOS (Quality of Service)
- *Failure transparency* Any process, computer or network may fail independently of the others. Therefore each component of the system needs to be aware of possible ways in which the components its depends on may fail. For example multiple routes to each pair of two routers.

References

[1] Distributed Systems Concepts and Design, Addison Wesley, ISBN: 0-201-61918-0

[2] Distributed System principles, Wolfgang Emmerich , 1997, http://www.cs.ucl.ac.uk/staff/W.Emmerich/lectures/ds98-99/dsee3.pdf