Transparency in Distributed Systems

By

Sudheer R Mantena

Abstract

The present day network architectures are becoming more and more complicated due to heterogeneity of the network components and mainly due to the extensive use of the Internet services. For instance a company may have many branches operating at different geographical locations and may need to interact with all other branches for day-to-day activities. This gives rise to Distributed computing environment where the resources are located at different locations and sharing of the resources such as printers, files, web pages or database records. The design and implementation of such systems poses greater challenges. The user should not be worried about the intrinsic details of the distributed system being used, how it is implemented and handles different situations. This points to the characteristic of the distributed system, being transparent. The different transparencies, which come across Distributed systems and how they are taken care of, are discussed.

1. Introduction

Distributed system architecture consists of a collection of workstations and servers connected by a local area network and distribution middle ware. The clients or the users can access these workstations at the same time and doing the computation at there own end. They can also share the remote resources provided by the servers and the workstations [1]. This kind of distributed computing environment has many advantages like heterogeneity of the network components, availability of the resources etc. The characteristics of the distributed systems are, multiple autonomous components i.e. the components by themselves have the computational power and access the shared resources for further operations, the components may not be shared by all the existing components in the system, resources sometimes may not be accessible due to some kind of failures, concurrent processing of tasks is allowed on different processors, multiple points of access control, the middle ware need to provide security by some kind of encryption and

transparency i.e. the aspects of distribution are made invisible to the client or the application user to provide a single centralized view of the system without worrying about the design and implementation details of the system [2].

Our focus is on understanding the types of transparencies involved in Distributed systems and implementation with the help of a case study mentioned in later sections. The paper is organized as follows: Section 2 introduces the distributed system characteristics and defines the transparency issues Section 3 defines the different types of transparencies involved Section 4 deals with some particular transparencies along with a case study and Section 5 gives the conclusion with references.

2. Distributed System Features

As we have seen distributed system is a collection of autonomous systems, which are connected together by means of a local area network. Even though distributed systems are found in many applications designing them is a difficult task, as many issues have to be considered during its implementation. Any ideal distributed system should have all the features discussed below. But it may not be possible to incorporate all of them, so depending on the application requirements the features required are considered. The following issues have to be taken care while designing. [2]

- **Heterogeneity**
- **Openness**
- **Security**
- **Scalability**
- **Fault Tolerance**
- **Concurrency**
- **Transparency:** The distributed systems should be perceived as a single entity by the users or the application programmers rather than as a collection of autonomous systems, which are cooperating. The users should be unaware of where the services are located and also the transferring from a local machine to a remote one should also be transparent.

3. Types of Transparencies

The implementation of the distributed system is very complex, as a number of issues have to be considered to achieve its final objective. The complexities should not worry the user of the distributed system from using it i.e., the complexities should be hidden from the user who uses the distributed system. This property of the distributed system is called its transparency. There are different kinds of transparencies that the distributed system has to incorporate. The following are the different transparencies encountered in the distributed systems [3] [2].

1. **Access Transparency:** Clients should be unaware of the distribution of the files. The files could be present on a totally different set of servers which are physically distant apart and a single set of operations should be provided to access these remote as well as the local files. Applications written for the local file should be able to be executed even for the remote files. The examples illustrating this property are the File system in Network File System (NFS), SQL queries, and Navigation of the web.
2. **Location Transparency:** Clients should see a uniform file name space. Files or groups of files may be relocated without changing their pathnames. A location transparent name contains no information about the named object's physical location. This property is important to support the movement of the resources and the availability of services. The location and access transparencies together are sometimes referred as Network transparency. The examples are File system in NFS and the pages of the web.
3. **Concurrency Transparency:** Users and Applications should be able to access shared data or objects without interference between each other. This requires very complex mechanisms in a distributed system, since there exists true concurrency rather than the simulated concurrency of a central system. The shared objects are accessed simultaneously. The concurrency control and its implementation is a hard task. The examples are NFS, Automatic Teller machine (ATM) network.

4. **Replication Transparency:** This kind of transparency should be mainly incorporated for the distributed file systems, which replicate the data at two or more sites for more reliability. The client generally should not be aware that a replicated copy of the data exists. The clients should also expect operations to return only one set of values. The examples are Distributed DBMS and Mirroring of Web pages.
5. **Failure Transparency:** [4] Enables the concealment of faults, allowing user and application programs to complete their tasks despite the failure of hardware or software components. Fault tolerance is provided by the mechanisms that relate to access transparency. The distributed system are more prone to failures as any of the component may fail which may lead to degraded service or the total absence of that service. As the intricacies are hidden the distinction between a failed and a slow running process is difficult. Examples are Database Management Systems.
6. **Migration Transparency:** This transparency allows the user to be unaware of the movement of information or processes within a system without affecting the operations of the users and the applications that are running. This mechanism allows for the load balancing of any particular client, which might be overloaded. The systems that implement this transparency are NFS and Web pages.
7. **Performance Transparency:** Allows the system to be reconfigured to improve the performance as the load varies.
8. **Scaling Transparency:** A system should be able to grow without affecting application algorithms. Graceful growth and evolution is an important requirement for most enterprises. A system should also be capable of scaling down to small environments where required, and be space and/or time efficient as required. The best-distributed system example implementing this transparency is the World Wide Web.

4. Case Study: Andrew File System

An ideal distributed system, which provides all the above-mentioned transparencies, is not always possible and all these transparencies may not be required by all the distributed systems. The need for any particular transparency mainly depends on the application of the distributed system. For example the replication transparency is more pronounced in case of Distributed file systems. The process migration transparency is more relevant in case of distributed systems which are more computational centric as they have to balance the load by executing the process at different or less loaded sites. The transparencies could be defined and implemented according to the distributed system under development. If we take the example of Distributed File System (DFS), the sharing of the storage devices and the data is very important. For the purpose of case study let us consider **Andrew File System (AFS)**, which is developed at the **Carnegie-Mellon university (CMU)**. The AFS is a Distributed File System, which was developed to meet the campus wide requirement of connecting all the workstations of the CMU and also to enable the sharing of information between the workstations.

The main design task of the AFS is to scale to a large number of workstations [13]. The high speed Local Area Network (LAN) technology was used to connect all the workstations on the campus. As the campus is wide spread, making the entire system physically secure is not possible. Selected numbers of workstations are physically secured. As the main function of AFS is to share the information across all the workstations the Distributed File System architecture was chosen for its implementation. The following are the file system goals, Location transparency, User Mobility, Security, Performance, Scalability, Availability, Integrity, and Heterogeneity. All these are discussed in detail in the later half of this section.

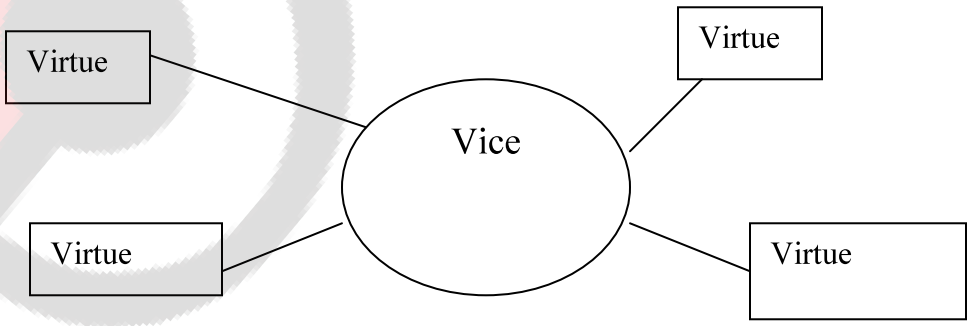

Fig 5.1: Vice and Virtue

Figure 5.1 gives the high level view of the system. Vice is a collection of all the communication as well as the computational resources in the network. The Virtues are the workstations connected in the network. The Virtue accesses the file or the resources that are present in the Vice and the software in the Virtue hides the remoteness of the Vice. The user programs are executed on the virtue, which is the local workstation, rather than on the Vice and the communication between the vice and the virtue is by means of the encrypted messages. The finer view of the Vice is shown in Fig 5.2.

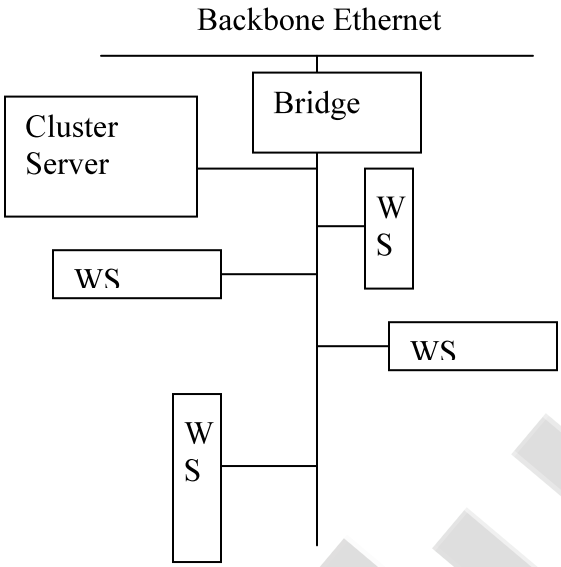

Fig 5.2: Vice Topology.

Each WS is the virtue workstation. The Cluster Server is a representative of the Vice. These are connected to the LAN by means of a bridge/router. These bridges provide a uniform network address space for all the nodes. Each workstation has its own logical disk to implement its programs and the shared space is on the Vice, which is accessible to all the connected workstations. The main implementation details are discussed in the later paragraphs.

Naming and Location:

The space is partitioned into Local name space and the shared name space. The local name space is the name, which the user at particular workstations uses and is restricted to that user only. All the personal files or programs could be stored in that. The shared name space is the one at the Vice and is shared by all the workstations. Virtue is

responsible for mapping the structure of the files in shared space to the structure of files at any particular workstation. As the AFS is built on the UNIX platform it has the same file structure as that of UNIX. The local name space is taken as the root directory and the shared space is mounted on it. The mounting is done as "/vice", this mounts the vice on the local space. The workstations could be of different types. This is handled by the virtue using Symbolic link. For instance, if the workstation is a Sun workstation then the local directory "/bin" is symbolic link to the remote directory "/vice/unix/sun/bin". These **Symbolic links** greatly support the **Heterogeneous Environment**.

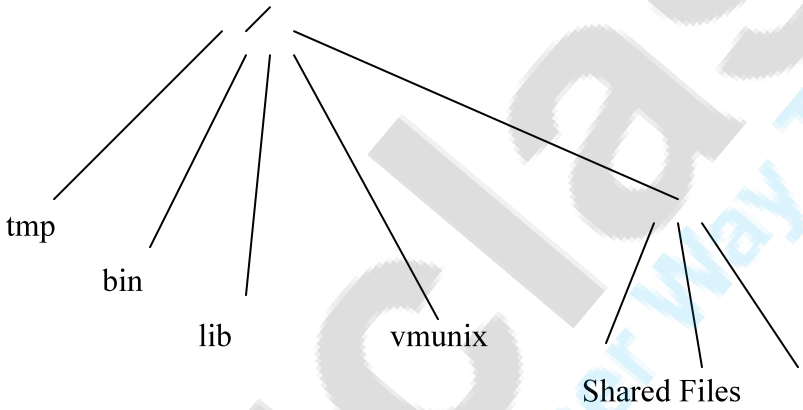

Fig 5.3: AFS Workstation's view.

The virtue cannot distinguish the server from which the file has been shared from as the **Location Transparency** is implemented by the Vice. Each cluster server has a location database which maps the files to their custodians. If all the files in the subtree have the same custodian then the location database has a single entry. The workstation has to find the custodian of the file by searching the location database. If any particular server is not the custodian then that request is passed on to the other servers. All the work that is done by the vice and virtue to locate the file and making it accessible to the user is hidden. This is how the Location transparency is dealt in AFS.

Replication:

The performance or the reliability in AFS is obtained by doing replication of the files. The replication in AFS is implemented in terms of the caching. The whole-files are cached along the custodianship information. This helps in meeting the design objectives of performance, mobility and scalability. The disk at the workstation is partially used to store the local files and the rest to cache the file from the vice. When the workstation requests a file from vice the virtue locates it and fetches it. The fetch is avoided if the file is present in the cache. Once it is fetched the read and write are done on the cached copy bypassing the vice and thus reducing the communication traffic between the vice and virtue. When the file is closed it is written back to the vice. All the interactions with the vive are transparent to the user. If the file is being modified then the cache validation could be initiated by the vice or the virtue. In AFS it is vice initiated and it is done when the file is closed. This gives a better performance results. Another reason for good performance of AFS is that the files are cached in whole blocks rather than in parts as the networks overheads are reduced. The whole-file caching also supports the heterogeneous workstations.

Authentication:

In AFS, kerberos authentication is used to establish the user identity. When the user logs in the password is not transmitted but a key is derived from the password, which is used to encrypt and decrypt the messages between the vice and the virtue. A kerberos server ticket is issued. The file server trusts this ticket and assumes the user is safe. This ticket expires after a particular amount of time and can be re-issued as per the requirement. In this manner a secured communication is obtained.

Scalability:

To incorporate the scalability and the scaling transparency in the AFS, the client-server architecture is chosen. The communication between the vice and virtue is done using the **Remote Procedure Call (RPC)** mechanism. The mutual client/server and the end-to-end encryption are provided by the RPC package itself. Each client can initiate a connection to the server. In order to handles a large number of clients; the multiple requests for the server resources are handled using the Lightweight Processes (LWP) or Threads. A thread is a process that has a very little nonshared state. A group of threads

share code, address space and the operating system resources. All the I/O requests could be placed in request queue and the context switching using LWP is fast as compared to the traditional heavyweight processes.

A distributed system is said to be scalable if the cost of adding a user is a constant amount in terms of the resources that must be added. The algorithms used to access shared data should avoid performance bottlenecks and the data should be structured hierarchically to get the best access times. Frequently accessed data can be replicated [2]. Distributed systems operate effectively and efficiently at many different scales, ranging from a small intranet to a large internet. The internet provides the best example illustrating the scalability of a distributed system. The design of scalable distributed systems presents many challenges like controlling the cost of physical resources, controlling the performance loss, preventing the software resources running out, and avoiding performance bottlenecks. Scaling transparency allows the system and the applications to expand in scale without change to the system structure or the application algorithms. **Andrew File System** is the best know **highly scalable distributed file system** [10]. The *Vice* presents a location-transparent file name space to all the clients. The process called *Venus,* caches the files from vice and stores them back to the server to which it belongs after the modifications are done. Venus accesses the files only when opening and while closing, the updates are done on the cached copy of the file. This kind of file improves the scalability of the system as the file changes are done bypassing the Venus without putting much strain on the system. The highly dynamic nature of the Andrew file system is responsible for its high scalability. A simple workstation with a small disk can access the files in Andrew file system. Adding or removing the workstations is quite an easy task. Adding a workstation is like connecting it to the network and assigning to it an address. Backup is needed only for the server as the updates are only done on the cached copies near the client. Another file system that scales well is the Coda file system [8].

5. Discussions:

The Andrew File System discussed so far is the best Distributed File System. The main highlights/advantages of the AFS are:

- *Security:* AFS uses the kerberos authentication protocol for the client authentication. All the sensitive data on the network is encrypted.
- *Reliability:* AFS supports the replication of the read-only files too. As the server fails, only he non-replicated data is not accessible and the replicated data is automatically fetched from one of the remaining servers. The clients automatically do the load balancing.
- *Name Space and Service models:* AFS distinguish the local and the remote spaces and also the client and the server are distinct.
- *Performance:* AFS caches the whole file on to the client and all the read or write operations are done on this cached copy avoiding the network traffic, thus reducing the network latencies and improving the performance time.
- *Scalability:* AFS is distinguished by its scalability. The policy of caching the whole file helps in scaling. The server has not much interaction with the client once the file is opened. Thus reducing the load on the sever and many clients can access the server for resources. The server initiated call back mechanism for cache invalidation also reduces the network traffic while checking for the cache consistency.

5.1 Comparison:

In this section we try to compare the different aspects of AFS and Locus file system, even though they are designed for different purposes [6].

Design Issue: The AFS is designed to be highly scalable where as the Locus is designed for high reliability and for greater transparency.

Naming Scheme: In AFS the names spaces for the local and remote sites are different and the shared tree descends from the local name space. In Locus singletree structure is used hiding both replication and location.

Remote-access method: In AFS the whole file is cached in local disks in a single access reducing the network latencies. In Locus the access are served by caching.

Availability: In AFS the client has to have a connection with the server and each component in the path should be present. In case of Locus the Stored Site and the Centralized Synchronous Site should be available. The primary copy must be present for writing.

Special Feature: The special feature of AFS is the authentication and encryption built in the RPC. Access list is also used for protection. Where as in Locus replication of the primary copy is done and the atomic updates are done using the shadow paging.

Scalability: AFS is a highly scalable system. This is mainly achieved by reducing the server load and also the clustering of the workstations. In Locus the replicated mount table makes it non scalable.

6. Conclusions

As we have seen so far in this paper, the design and implementation of the distributed system is a very complex job. The definition of an ideal distributed system as proposed by Tanenbaum and van Renesse [1]: "*A distributed operating system is one that looks to it users like an ordinary centralized operating system, but runs on multiple independent CPUs. The key concept here is transparency, in other words, the use of multiple processors should be visible to user. Another way of expressing the same idea is to say that the user views the system as a virtual processor, not as a collection of distinct machines.*" An ideal distributed system, which provides a transparent access to all its resources does not exist but the sub systems built on the distributed architectures do provide transparencies for particular resources like files, disks and memory.

Thus a lot of research is being done in the field of distributed systems, which have good scope in the future and also the need for these kind systems would be high. Development of the transparent distributed system assumes greater significance as we have seen the complexities of the design and implementation of such system which the general user need be not be worried about.

References

[1] Michel Banatre, "*Hiding Distribution in Distributed Systems*", Proceedings of the 13th international conference on Software engineering May 1991

[2] George Coulouris, Jean Dollimore, Tim Kindberg, "*Distributed Systems Concepts and Design"* 3rd edition, Addison-Wesley.

[3] http://www.cs.ucl.ac.uk/staff/W.Emmerich/lectures/DS97-98/dsee3.pdf.

[4]http://www.cs.ucl.ac.uk/staff/J.Crowcroft/ods/node18.html - SECTION00530000000000_0000000

[5] http://www.cs.ucl.ac.uk/staff/J.Crowcroft/ods/ods.html

[6] *"Distributed File Systems: Concepts and Examples*". E Levy and A Silberschatz, ACM computing Surveys, Vol. 22, No.4, Dec. 1990.

[7] "*Name transparency in very large scale Distributed file systems",* Richard G. Guy, Thomas W.Page Jr., John S.Heidemann, Gerald J.Popek, Experimental Distributed Systems, 1990. Proceedings., IEEE Workshop on , 1990 Page(s): 20 -25

[8] "*Coda: A highly available file system for a distributed workstation environment"*, M.Satyanarayanan, James kistler, Punnet Kumar, Maria Okasaki, Ellen Siegel and David Steere. Computers, IEEE Transactions on , Volume: 39 Issue: 4 , April 1990

[9] "*Failure Transparency in Remote Procedure calls"* K.Ravindran, Samuel Chanson, Computers, IEEE Transactions on, Volume: 38 Issue: 8, Aug. 1989 Page(s): 1173 -1187.

[10] "*Scale and performance in a Distributed File System*" John Howard, Michael Kazar, Sherri Menees, David Nichols, M.Satyanarayanan, Robert Sidebotham, and Michael West. ACM Transactions on Computer Systems (TOCS) February 1988 Volume 6 Issue 1

[11] http://research.compaq.com/SRC/personal/thekkath/frangipani/home.html

[12]http://homepages.ihug.com.au/~ijoyner/Transparencies.html - 2.4.%20Migration%20Transparency

[13] Satyanarayanan, M [1985] "*The ITC Distributed File System: Principles and Design.*" In proceedings of the 10th ACM Symposium on Operating System Principles. (Dec 1985), ACM, 35-49.