P

DISTRIBUTED SYSTEM-A TOOL FOR BUILDING DEPENDABLE DISTRIBUTED SYSTEM

Prof. Ravish Kumar Mishra
HOD-(Computer Science)
Poona School of Business, Pune
- Dr. JaiPrakash Sirur
Ph. D. Guide
JJT University Rajasthan

Abstract:

Physically, a Distributed System consists of a set of processors, with a collection of local storage mechanisms associated with each processor. A processor is able to execute programs that access and manipulate the local storage, where the term process denotes the locus of control of an executing program. In addition, an interconnection network connects the processors and allows them to communicate and share data via exchange of messages.These messages are encapsulated inside packets when transmitted on the network.

We conceptually view the underlying distributed system in terms of an object model in which the system is said to consist of a collection of objects. An object is either a physical resource, or an abstract resource. Objects are further characterized as being either passive or active, where passive objects correspond to stored data, and active objects correspond to processes that act on passive resources. For the purposes of this thesis, we use the term object to denote only passive objects.

There has been very little empirical data published on file system usage or performance. Obtaining trace data is difficult, typically requiring careful modifications to the operating system, and the resulting data is voluminous. The published studies tend to deal with older operating systems, and for this reason may not be applicable in planning future systems.

Our main concerns in gathering the data were the volume of the data and affecting the results by logging them through the cache system under measurement. We wished to gather data over several days to prevent temporary anomalies from biasing the data. The method settled upon used the local area network to send the trace data to another system, where it was written to magnetic tape. Logging routines inserted into the file system code placed the trace records in a memory buffer.

This article describes the practicalities in Distributed System concept and principles and its advancements from the earlier technologies.

Keywords :-

Comparison between Parallel and Distributed Computing, Architectures of Distributed System, Special Challenges for Distributed System, Solving Problems Using Distributed System, Current Trends in Software for Distributed System, Upcoming and Future Trends of Distributes System, Conclusion, Reference

I) INTRODUCTION

In computer science a Distributed System consists of multiple autonomous computers that communicate through a computer network. The computers interact with each other in order to achieve a common goal. A computer program that runs in a distributed system is called a Distributed Program, and Distributed Programming is the process of writing such programs. The principle of locality of reference is the observation that computer programs exhibit both spatial and temporal locality in referencing objects. Temporal locality means that objects to be referenced in the near future are likely to have been in use recently. Spatial locality means there is a high probability that objects needed in the near future can be located near the objects currently in use. Less expensive access to recently used objects increases program performance.

The idea that a computer should use a memory hierarchy dates back to at least the early portion of the 20th century; it is suggested in the pioneering paper of von Neumann *et al.* The motivation for a storage hierarchy in a processor is economic. The performance and

cost of various storage technologies varies widely. Usually, the fastest and most expensive technology is used for the registers in the processor. Ideally, one would like to execute programs as if all data existed in the processor registers.

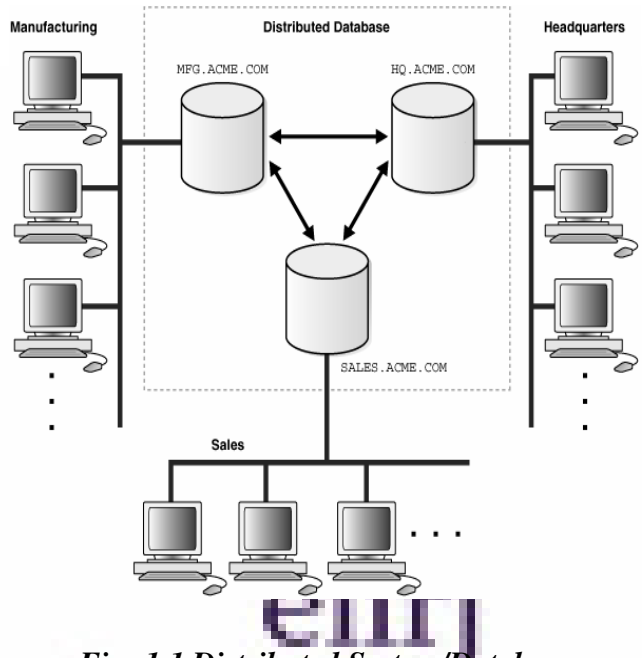

Fig: 1.1 Distributed System/Database

When more data are required, larger, lower-cost storage technologies are used for data and instruction storage, proceeding from fast semiconductor memory, to slower semiconductor memory, to magnetic disk storage, and finally to magnetic tape or other archival storage media.

Characteristics of Distributed System:

A Distributed System can be much larger and more powerful given the combined capabilities of the distributed components, than combinations of stand-alone systems. But it's not easy - for a distributed system to be useful, it must be reliable. This is a difficult goal to achieve because of the complexity of the interactions between simultaneously

running components. To be truly reliable, a distributed system must have the following characteristics:

- ❖ Fault-Tolerant: It can recover from component failures without performing incorrect actions.
- ❖ Highly Available: It can restore operations, permitting it to resume providing services even when some components have failed.
- ❖ Recoverable: Failed components can restart themselves and rejoin the system, after the cause of failure has been repaired.
- ❖ Consistent: The system can coordinate actions by multiple components often in the presence of concurrency and failure. This underlies the ability of a distributed system to act like a non-distributed system.
- ❖ Scalable: It can operate correctly even as some aspect of the system is scaled to a larger size. For example, we might increase the size of the network on which the system is running. This increases the frequency of network outages and could degrade a "non-scalable" system. Similarly, we might increase the number of users or servers, or overall load on the system. In a scalable system, this should not have a significant effect.
- ❖ Predictable Performance: The ability to provide desired responsiveness in a timely manner.
- ❖ Secure: The system authenticates access to data and services.

Main Features and Benefits of a Distributed System:

A common misconception among people when discussing distributed systems is that it is just another name for a network of computers. However, this overlooks an important distinction. A distributed system is built on top of a network and tries to hide the existence of multiple autonomous computers. It appears as a single entity providing the user with whatever services are required. A network is a medium for interconnecting

entities enabling the exchange of messages based on well-known protocols between these entities, which are explicitly addressable.

There are various types of distributed systems, such as Clusters, Grids, P2P (Peer-to-Peer) networks, distributed storage systems and so on. A cluster is a dedicated group of interconnected computers that appears as a single super-computer, generally used in high performance scientific engineering and business applications. A grid is a type of distributed system that enables coordinated sharing and aggregation of distributed, autonomous, heterogeneous resources based on users' QoS (Quality of Service) requirements. Grids are commonly used to support applications emerging in the areas of e-Science and e-Business, which commonly involve geographically distributed communities of people who engage in collaborative activities to solve large scale problems and require sharing of various resources such as computers, data, applications and scientific instruments. P2P networks are decentralized distributed systems, which enable applications such as file-sharing, instant messaging, online multiuser gaming and content distribution over public networks. Distributed storage systems such as NFS (Network File System) provide users with a unified view of data stored on different file systems and computers which may be on the same or different networks.

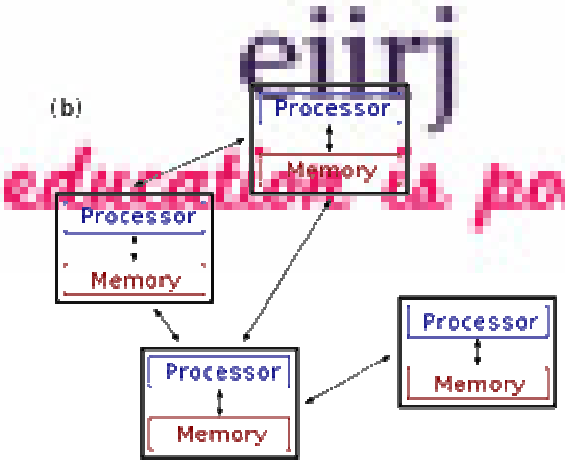

Fig:1.2 Distributed System with Processor

The main features of a distributed system include:

- ❖ Functional Separation Based on the functionality / services provided capability and purpose of each entity in the system.
- ❖ Inherent distribution Entities such as information, people, and systems are inherently distributed. For example, different information is created and maintained by different people. This information could be generated, stored, analyzed and used by different systems or applications which may or may not be aware of the existence of the other entities in the system.
- ❖ Reliability Long term data preservation and backup at different locations.
- ❖ Scalability Addition of more resources to increase performance or availability.
- ❖ Economy sharing of resources by many entities to help reduce the cost of ownership.

II. COMPARISON BETWEEN PARALLEL AND DISTRIBUTED COMPUTING:

Distributed systems are networked computers operating with same processors. The terms “Concurrent Computing”, “Parallel Computing” and "Distributed Computing" have a lot of overlap, and no clear distinction exists between them. The same system may be characterized both as "parallel" and "distributed"; the processors in a typical distributed system run concurrently in parallel. Parallel computing may be seen as a particular tightly-coupled form of distributed computing, and distributed computing may be seen as a loosely-coupled form of parallel computing. Nevertheless, it is possible to roughly classify concurrent systems as "parallel" or "distributed" using the following criteria:

- ❖ In parallel computing, all processors have access to a Shared Memory. Shared memory can be used to exchange information between processors.
- ❖ In distributed computing, each processor has its own private memory i.e., Distributed Memory. Information is exchanged by passing messages between the processors.

III. ARCHITECTURES OF DISTRIBUTED SYSTEM :

Various hardware and software architectures are used for distributed computing. At a lower level, it is necessary to interconnect multiple CPUs with some sort of network, regardless of whether that network is printed onto a circuit board or made up of loosely-coupled devices and cables. At a higher level, it is necessary to interconnect processes running on those CPUs with some sort of communication system.

Distributed programming typically falls into one of several basic architectures or categories: such as Client-Server, 3-tier Architecture, n-tier Architecture, Distributed Objects, Loose Coupling, or Tight Coupling.

- Client-Server: Smart client code contacts the server for data then formats and displays it to the user. Input at the client is committed back to the server when it represents a permanent change.
- 3-tier Architecture: Three tier systems move the client intelligence to a middle tier so that stateless clients can be used. This simplifies application deployment. Most web applications are 3-Tier.
- N-tier Architecture: *n*-tier refers typically to web applications which further forward their requests to other enterprise services. This type of application is the one most responsible for the success of application servers.
- Tightly coupled clustered: refers typically to a cluster of machines that closely work together, running a shared process in parallel. The task is subdivided in parts that are made individually by each one and then put back together to make the final result.
- Peer-to-Peer: an architecture where there is no special machine or machines that provide a service or manage the network resources. Instead all responsibilities are uniformly divided among all machines, known as peers. Peers can serve both as clients and servers.
- Space Based: refers to an infrastructure that creates the illusion of one single address-space. Data are transparently replicated according to application needs. Decoupling in time, space and reference is achieved.

Another basic aspect of distributed computing architecture is the method of communicating and coordinating work among concurrent processes. Through various message passing protocols, processes may communicate directly with one another, typically in a Master/Slave relationship. Alternatively, a "database-centric" architecture can enable distributed computing to be done without any form of direct inter-process communication, by utilizing a shared database.

IV. SPECIAL CHALLENGES FOR DISTRIBUTED SYSTEM :

We are now moving from having the number of computers working on a problem being small enough to fit in a building or two to having tens of thousands of computers working on single problems. This brings many special challenges that mark distributed computing as being a vastly more complex enterprise than what has gone before.

The first of these special problems is security. The first aspect of security is the security of the computers themselves, since few people feel like giving some wannabe digital mobster a free pass to misuse their computers. The second aspect is the security of the data being processed, much of which may be highly confidential or a trade secret. The third aspect of security is the security of the messages used to control the other computers, which are often important in them and could be used to conduct a wide range of other mischief if intercepted.

The second special problem of distributed computing is due to the use of systems owned by others, either other people or other organizations. The issues here are to do with the fact that people ultimately retain control over their own systems; they do not like to cede it to others. This behaviour could be considered just a matter of human nature, but it does mean that it is extremely difficult to trust others in this space. The key worries relate to either the computer owner lying about what actions were taken on their systems or the distributed computer user using the system for purposes other than those that the computer owner wants to permit.

Interoperability presents the third challenge for distributed computing. Because the systems that people use to provide large-scale computing capabilities have grown over many years, the ways in which they are accessed are quite diverse. This does mean that a lot of effort has been put into finding out access methods that balance the need for efficiency with those of security and flexibility, but it also means that frequently it is extremely difficult to make these systems work together as one larger system. Past attempts by hardware and software vendors to lock people in to specific solutions have not been helpful here either; researchers and practitioners want to solve a far more diverse collection of challenges than the vendors have imagined there to be. After all, the number of things that people wish to do is limited only by the human imagination.

These challenges can be surmounted though, even if the final form of the solutions is not yet clear. We know that the demands of security can be met through a combination of encryption, digital signatures, firewalls, and placing careful constraints on what can be done by any program. The second challenge is being met through the use of techniques from digital commerce like formal contracts, service level agreements, and appropriate audit and provenance trails. The third, which will become ever more important as the size of problems people wish to tackle expands, is primarily dealt with through standardization of both the access mechanisms and the formal understanding of the systems being accessed by common models, lexicons and ontology.

The final major challenge of distributed computing is managing the fact that neither the data nor the computations are open to relocation without bounds. Many datasets are highly restricted in where they can be placed, whether this is through legal constraints or because of the sheer size of the data; moving a terabyte of data across the world can take a long time, and in no time the most efficient technique becomes sending disks by courier, despite the large quantity of very high capacity networks that exist out there.

This would seem to indicate that it makes sense to move the computations to the location of the data, but that is not wholly practical either. Many applications are not easy to relocate: they require particular system environments or direct access to other data

artifacts or are dependent on highly restricted software licenses. This problem does not go away even when the users themselves develop the software — it is all too easy for them to include details of their development environment in the program so that it only works on their own computer or in their own institution — writing truly portable software is a special and rare talent.

V. SOLVING PROBLEMS USING DISTRIBUTED SYSTEM

Because of these fundamental restrictions that will not go away any time soon, it is important for someone tackling a problem with distributed computing to take them into account when working out what they wish to do. In particular, they need to bear in mind the restrictions on where their data can be, where their applications can be, and what sort of computational patterns they are using in their overall workflow.

As a case in point, when performing drug discovery in relation to a disease, the first stage is to discover a set of potential candidate receptors for the drug to bind to in or on the cell. This is then followed by a search for candidate substances that might bind to the receptor in a useful way, first coarsely and then in depth. Once these candidates have been identified, they then have to be screened to see if there are warning areas associated with them that might make their use inadvisable. If we look at the data-flow between these steps, we see that the amount of data actually moved around is kept relatively small; the databases being searched are mostly not relocated, despite their massive size. However, once these steps are completed, the scientist can have a much higher level of confidence that their in silicon experiments will mean that follow-up clinical trials of the winning candidate will succeed, and in many cases it may be possible to skip some parts of the trials.

This use-case has other interesting aspects in terms of distributed computing, in that it involves the blending of both public and private information to produce a result. The initial searches for binding receptors relating to a particular disease will often involve mainly public data — archives of scientific papers — and much of the coarse fit checking

that identifies potential small molecules for analysis will benefit from being farmed out across such large numbers of computers that the use of public cycle scavengers makes sense; it would be difficult to backtrack from the pair of molecules being matched to exactly what was being searched for. On the other hand, there are strong reasons for being very careful with the later stages of the discovery process; scientists are looking at that stage in great depth at a small number of molecules, making it relatively easy for a competitor to act pre-emotively. Moreover, the use of detailed patient data means that care has to be taken to avoid breaches of privacy. In addition, the applications for the detailed analysis steps are often costly commercial products. This means that the overall workflow has both public and private parts, both in the data and computational domains, and so there are inherent complexities. On the other hand, this is also an application area that was impossible to tackle until very recently, and distributed computing has opened it up.

It should also be noted that distributed computing has many benefits at the smaller scale. For example, it is a key component of providing acceleration for many more commonplace problems, such as recalculating a complex spreadsheet or compiling a complex program. These tasks also use distributed computing, though only within the scope of a workgroup or enterprise. And yet there is truly a continuum between them and the very large research Grids and commercial Clouds, and that continuum is founded upon the fact that bringing more computational power together with larger amounts of data allows the discovery of finer levels of detail about the area being studied. The major differences involve how they respond to the problems of security, complex ownership of the parts, and interoperability. This is because those smaller scale solutions can avoid most of the security complexity, only needing at most SSL-encrypted communications, and they work within a single organization. Of course, as time goes by this gap may be closed from both sides; from the lower end as the needs for more computation combine with the availability of virtual-machines-for-hire and from the upper end as the benefits of simplified security and widespread standardized software stacks make adoption of scalable solutions easier.

VI. CURRENT TRENDS IN SOFTWARE FOR DISTRIBUTED SYSTEMS

Software infrastructures for distributed systems such as CORBA or Jini provide basic communication facilities to application components and handle issues such as platform heterogeneity that are due to differing hardware systems, operating systems, or programming languages. Furthermore, they provide a set of standard services that are typically needed by distributed applications such as directory service or cryptographic security.

One of the most widely used infrastructures for distributed systems today is CORBA, the Common Object Request Broker Architecture, which is supported by a large industry consortium. In contrast to DCE it is based on an object-oriented model. The first CORBA standard was introduced in 1991, and it has undergone continual and significant revisions ever since. Part of the specification describes IDL (Interface Definition Language) that all CORBA implementations must support. IDL is based on C++ and is used by CORBA applications to define the externally visible parts of object methods that can be invoked by other objects. It has a similar function than the IDL of RPC toolkits.

The central component of a CORBA system is the so-called Object Request Broker (ORB). The ORB provides a mechanism for transparently communicating client requests to target object implementations. It simplifies distributed programming by decoupling the client from the details of the method invocations: When a client invokes an operation, the ORB is responsible for finding the object implementation, transparently activating it if necessary, delivering the request to the object, and returning any response to the caller.

Furthermore, CORBA was conceived for static distributed systems, requires considerable resources at run time, and uses the traditional client-server model as the basic

metaphor. It is therefore not well suited for small devices, highly dynamic systems, and services that are spontaneously integrated into a federation. This, however, is a major trend in distributed systems, to which Jini and similar systems are better adapted.

VII. UPCOMING AND FUTURE TRENDS OF DISTRIBUTED SYSTEM

The cornerstone of the networked home is a bridge, a so-called "home gateway" or "residential gateway", between the internal home network and the public Internet. Ideally, this will be an open and vendor-neutral small server that makes Internet-connected services available to any household and gives all device manufacturers and service providers a common software interface to ensure interoperability. It may also simplify IP telephony because it functions much like a PBX.

The most important technologies to date for service discovery are Jini, Salutation, Universal Plug and Play (UPnP) from Microsoft, E-speak from Hewlett-Packard, and the Service Location Protocol (SLP), which has been jointly developed by researchers both from academia and industry as a widely accepted and usable Internet standard for service discovery.

The emergence of information appliances and new types of connectivity is spurring a new form of networking: unmanaged, dynamic networks of devices, especially mobile devices that spontaneously and unpredictably join and leave the network. Underlying network technologies already exist, for example Bluetooth. Consumers will expect these ad hoc, peer to peer networks to automatically form within the home, in networked vehicles, in office buildings, and in various arbitrary environments.

Mobile devices in the form of portable telephones, pagers, and notebook computers are now commonplace. Technologies such as WAP, GSM, and in particular UMTS and similar so-called third generation cellular communication standards will soon give rise to new mobile devices that provide fast and immediate access to the Internet.

Mobile code is an important programming paradigm and opens up new possibilities for structuring distributed software systems in an open and dynamically changing environment. It can improve speed, flexibility, structure, or ability to handle disconnections and it is particularly well-suited if adaptability and flexibility are among the main

application requirements. It has applications in many areas, such as mobile computing, active networks, network management, resource discovery, software dissemination and configuration, electronic commerce, and information harvesting.

Networked embedded processors, which form the heart of all smart devices, will become an important research and development field. New types of low-power processors will be developed specifically for networked embedded applications. Of primary interest are also advances in networking technology that could allow large numbers of embedded computers to be interconnected so they can share information and work together as part of a larger system. Reliability is crucial in embedded computing systems since such systems will increasingly control critical functions where safety is a factor and, in some applications, may be given authority to take some actions with little or no human intervention. Ensuring the reliability of networked embedded computing systems could be difficult since large, interconnected information systems are notorious for becoming unstable. Such problems could be magnified by the presence of millions of interconnected embedded systems.

VIII. CONCLUSION

These results from a single processor system allow us to draw several important conclusions concerning the design of a distributed file system. On the average, users demand fairly low data rates from the file system. Thus, the bandwidth available in a conventional 10 Mbit/second local area network should be sufficient to support several hundred active users, including the burst high traffic levels sometimes experienced.

Since much of the file system activity is associated with management of the on-disk structures of the file system, a distributed file system which provides high-level file system access by clients will greatly reduce the amount of network traffic. If the server is solely

responsible for management and access of these structures, network traffic can be cut by as much as 50%, compared to a distributed file system in which each client reads and writes directories, and reads, writes, locks, and unlocks inodes across the network.

There are two ways to further increase the file system performance of a client workstation adding local disk storage or greatly increasing the size of the cache memory. The current economy of memory costs *vs.* disk costs indicates that adding more memory is the less expensive way to increase performance.

The intelligent interface implements a network protocol to maintain consistency among the caches of clients sharing an object and to minimize the amount of data lost if a client fails. The simulations show that the Caching Ring hardware, with a sufficiently large cache at each

CRI can provide clients with access to a remote disk at performance levels similar to those of a locally attached disk with a small cache.

Distributed systems are developing rapidly as the take up of sophisticated theory by commercial concerns increases, and as the Internet and the World Wide Web drive the requirements of users for more accurate information and higher-performance services. Especially important is the application of Internet technology within corporations, leading to company-wide Intranets being developed.

IX.REFERENCES

Book Reference*:*

1. *Distributed Systems: Principles and Paradigms,* Prentice Hall, Pearson Education, USA, 2002, A. Tannenbaum and M. Van Steen.

2. *Peer-to-Peer Computing: Evolution of a Disruptive Technology,* Idea Group Inc., Hershey, PA, USA, 2005, R. Subramanian and B. Goodman.

3. P. Thomas, H.W. Gellersen (Ed.): Proc. Second Int. Symp. Handheld and Ubiquitous Computing, Springer-Verlag, ISBN 3-540-41093-7, 2000

4. Operating System, Vision Publication, Pune, Ravish Kumar Mishra.

5. Software Engineering and Object-Oriented Analysis, Vision Publication, Ravish Kumar Mishra

Web Reference:-

1. www.sun.com/jini/factsheet/
2. http://www.cdtltd.co.uk/
3. www.cs.caltech.edu/~adam/phd/why-events.html
4. http://www.cwt.vt.edu/faq/rfid.htm
5. http://www.ncsa.uiuc.edu/People/mcgrath/Discovery/dp.html