DISTRIBUTED COMPUTING

Shaveta
Assistant Professor, Department of Computer Science & Applications, Guru Nanak College, Ferozepur, Punjab, India

Received: July 13, 2018 Accepted: August 25, 2018

ABSTRACT ***Distributed computing*** *is a field of computer science that studies distributed systems. A distributed system is a system whose components are located on different networked computers, which then communicate and coordinate their actions by passing messages to one other. The components interact with one other in order to achieve a common goal. Three significant characteristics of distributed systems are: concurrency of components, lack of a global clock, and independent failure of components. Examples of distributed systems vary from SOA-based systems to massively multiplayer online games to peer-to-peer applications.*

Keywords:

I. INTRODUCTION

The word *parallel* in terms such as "parallel system", "distributed programming", and "distributed algorithm" originally referred to computer networks where individual computers were physically distributed within some geographical area. The terms are nowadays used in a much wider sense, even referring to autonomous processes that run on the same physical computer and interact with each other by message passing.

While there is no single definition of a distributed system, the following defining properties are commonly used:

- There are several autonomous computational entities (*computers* or nodes), each of which has its own local memory.
- The entities communicate with each other by message passing.

A distributed system may have a common goal, such as solving a large computational problem; The user then perceives the collection of autonomous processors as a unit. Alternatively, each computer may have its own user with individual needs, and the purpose of the distributed system is to coordinate the use of shared resources or provide communication services to the users.

II. HISTORY

1. The use of concurrent processes that communicate by message-passing has its roots in operating system architectures studied in the 1960s.\ The first widespread distributed systems were local-area networks such as Ethernet, which was invented in the 1970s.\
2. ARPANET, the predecessor of the Internet, was introduced in the late 1960s, and ARPANET e-mail was invented in the early 1970s. E-mail became the most successful application of ARPANET, and it is probably the earliest example of a large-scale distributed application. In addition to ARPANET, and its successor, the Internet, other early worldwide computer networks included Usenet and FidoNet from the 1980s, both of which were used to support distributed discussion systems.
3. The study of distributed computing became its own branch of computer science in the late 1970s and early 1980s. The first conference in the field, Symposium on Principles of Distributed Computing (PODC), dates back to 1982, and its counterpart International Symposium on Distributed Computing (DISC) was first held in Ottawa in 1985 as the International Workshop on Distributed Algorithms on Graphs

III. EVOLUTION OF DISTRIBUTED COMPUTING

- The first was the development of powerful microprocessors. Initially, these were 8-bit machines, but soon 16-, 32-, and 64-bit CPUs became common. Many of these had the computing power of a mainframe (i.e., large) computer, but for a fraction of the price. The amount of improvement that has occurred in computer technology in the past half century is truly staggering and totally unprecedented in other industries. From a machine that cost 10 million dollars and executed 1 instruction per second. We have come to machines that cost 1000 dollars and are able to execute 1 billion instructions per second, a price/performance gain of 1013. Network-infrastructure attacks.

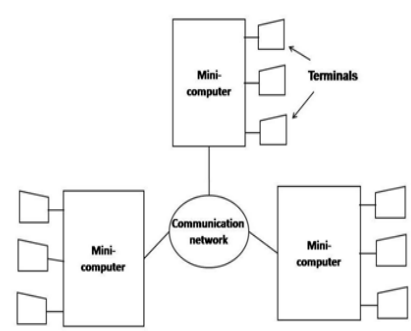
Hacker attacks against network infrastructures can be easy, because many networks can be reached from anywhere in the world via the Internet.

- The second development was the invention of high-speed computer networks. Local-area networks or LANs allow hundreds of machines within a building to be connected in such a way that small amounts of information can be transferred between machines in a few microseconds or so. Larger amounts of data can be Distributed Computing become popular with the difficulties of centralized processing in mainframe use.
- With mainframe software architectures all components are within a central host computer. Users interact with the host through a terminal that captures keystrokes and sends that information to the host. In the last decade however, mainframes have found a new use as a server in distributed 6 client/server architectures (Edelstein 1994). The original PC networks (which have largely superseded mainframes) were based on file sharing architectures, where the server transfers files from a shared location to a desktop environment/
- The requested user job is then run (including logic and data) in the desktop environment. File sharing architectures work well if shared usage is low, update contention is low, and the volume of data to be transferred is low. In the 1990s, PC LAN (local area network) computing changed because the capacity of the file sharing was strained as the number of online users grew and graphical user interfaces (GUIs) became popular (making mainframe and terminal displays appear out of date).
- With two tier client-server architectures, the GUI is usually located in the user's desktop environment and the database management services are usually in a server that is a more powerful machine that services many clients. Processing management is split between the user system interface environment and the database management server environment. The two tier client/server architecture is a good solution for locally distributed computing when work groups are defined as a dozen to 100 people interacting on a LAN simultaneously. However, when the number of users exceeds 100, performance begins to deteriorate and the architecture is also difficult to scale. The three tier architecture (also referred to as the multi-tier architecture) emerged to overcome the limitations of the two tier architecture. In the three tier architecture, a middle tier was added between the user system interface client environment and the database management server environment.
- As an interface for software to use it is a set of rules for marshalling and un-marshalling parameters and results, a set of rules for encoding and decoding information transmitted between two processes; a few primitive operations to invoke an individual call, to return its results, and to cancel it; provides provision in the operating system and process structure to maintain and reference state that is shared by the participating processes. RPC requires a communications infrastructure to set up the path between the processes and provide a framework for naming and addressing.

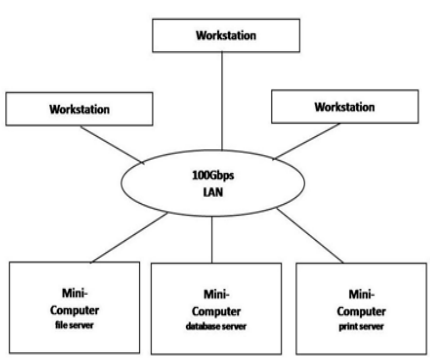
IV. DISTRIBUTED COMPUTING MODELS

Various models are used for building distributed computing systems. These models can be broadly classified into five categories – minicomputer, workstation, workstation-server, processor pool, and hybrid. They are briefly described below.

- **Minicomputer Model**: The minicomputer model is a simple extension of the centralized time sharing system as shown in Figure 1.2, a distributed computing system based on this model consists of a few minicomputers (they may be large supercomputers as well) interconnected by a communication network. Each minicomputer usually has multiple users simultaneously logged on to it. For this, several interactive terminals are connected to each minicomputer. Each user is logged on to one specific minicomputer, with remote access to other minicomputers. The network allows a user to access remote resources that are available on some machine other than the one on to which the user is currently logged.
- The minicomputer model may be used when resource sharing (Such as sharing of information databases of different types, with each type of database located on a different machine) with remote users is desired.



- **Workstation Model :**
  - As shown above, a distributed computing system based on the workstation model consists of several workstations 9 interconnected by a communication network. A company's office or a university department may have several workstations scattered throughout a building or campus, each workstation equipped with its own disk and serving as a single-user computer.
  - It has been often found that in such an environment, at any one time (especially at night), a significant proportion of the workstations are idle (not being used), resulting in the waste of large amounts of CPU time. Therefore, the idea of the workstation model is to interconnect all these workstations by a high speed LAN so that idle workstations may be used to process jobs of users who are logged onto other workstations and do not have sufficient processing power at their own workstations to get their jobs processed efficiently.
  - In this model, a user logs onto one of the workstations called his or her "home" workstation and submits jobs for execution. When the system finds that the user's workstation does not have sufficient processing power for executing the processes of the submitted jobs efficiently, it transfers one or more of the process from the user's workstation to some other workstation that is currently idle and gets the process executed there, and finally the result of execution is returned to the user's workstation.



- **Processor Pool Model :**
  - The processor – pool model is based on the observation that most of the time a user does not need any computing power but once in a while he or she may need a very large amount of computing power for a short time. (e.g., when recompiling a program consisting of a large number of files after changing a basic shared declaration). Therefore, unlike the workstation – server model in which a processor is allocated to each user, in the processor-pool model the processors are pooled together to be shared by the users as needed. The pool of processors consists of a large number of microcomputers and minicomputers attached to the network. Each processor in the pool has its own memory to load and run a system program or an application program of the distributed computing system.
- **Hybrid Model :**Out of the four models described above, the workstationserver model, is the most widely used model for building distributed computing systems. This is because a large number of computer users only perform simple interactive tasks such as editing jobs, sending electronic mails, and executing small programs. The workstation-server model is ideal for such simple usage. However, in a working environment that has groups of users who often perform jobs needing massive computation, the processor-pool model is more attractive and suitable.

V. FAULT DETECTION AND RECOVERY

The faulty detection and recovery method of improving reliability deals with the use of hardware and software mechanisms to determine the occurrence of a failure and then to correct the system to a state acceptable for continued operation. Some of the commonly used techniques for implementing.

- **Atomic transactions :** An atomic transaction (or just transaction for shore) is a computation consisting of a collection of operation that take place indivisibly in the presence of failures and concurrent computations. That is, either all of the operations are performed successfully or none of their effects prevails, other processes executing concurrently cannot modify or observe intermediate states of the computation. Transactions help to preserve the consistency of a set of shared date objects (e.g. files) in the face of failures and concurrent access. They make crash recovery much easier, because transactions can only end in two states : Either all the operations of the transaction are performed or none of the operations of the transaction is performed.
- **Stateless servers:** The client-server model is frequently used in distributed systems to service user requests. In this model, a

server may be implemented by using any one of the following two service paradigms – stateful or stateless. The two paradigms are distinguished by one aspect of the client – server relationship, whether or not the history of the serviced requests between a client and a server affects the execution of the next service request. The stateful approach does depend on the history of the serviced requests, but the stateless approach does not depend on it. Stateless servers have a distinct advantage over stateful servers in the event of a failure. That is, the stateless service paradigm makes crash recovery very easy because no client state information is maintained by the server. On the other hand, the stateful service paradigm requires complex crash recovery procedures. Both the client and server need to reliably detect crashes. The server needs to detect client crashes so that it can discard any state it is holding for the client, and the client must detect server crashes so that it can perform necessary error – handling activities. Although stateful service becomes necessary in some cases, to simplify the failure detection and recovery actions, the stateless service paradigm must be used, wherever possible.

VI. APPLICATIONS OF DISTRIBUTED COMPUTING SYSTEM

1. The very nature of an application may require the use of a communication network that connects several computers: for example, data produced in one physical location and required in another location.
2. There are many cases in which the use of a single computer would be possible in principle, but the use of a distributed system is beneficial for practical reasons. For example, it may be more cost-efficient to obtain the desired level of performance by using a cluster of several low-end computers, in comparison with a single high-end computer. A distributed system can provide more reliability than a non-distributed system, as there is no single point of failure. Moreover, a distributed system may be easier to expand and manage than a monolithic uniprocessor system.
3. Acknowledgments and timeout-based retransmission of messages. In a distributed system, events such as a node crash or a communication link failure may interrupt a communication that was in progress between two processes, resulting in the loss of a message. Therefore, a reliable interprocess communication mechanism must have ways to detect lost messages so that they can be retransmitted. Handling of lost messages usually involves return of acknowledgment messages and retransmissions on the basis of timeouts. That is, the receiver must return an acknowledgment message for every message received, and if the sender does not receive any acknowledgement for a message within a fixed timeout period, it assumes that the message was lost and 22 retransmits the message. A problem associated with this approach is that of duplicate message. Duplicates messages may be sent in the event of failures or because of timeouts. Therefore, a reliable interprocess communication mechanism should also be capable of detecting and handling duplicate messages. Handling of duplicate messages usually involves a mechanism for automatically generating and assigning appropriate sequence numbers to messages. Use of acknowledgement messages, timeout-based retransmissions of messages, and handling of duplicate request messages for reliable communication.

VII. DESIGN PRINCIPLES

Based on his experience with the AFS and other distributed file systems, Satyanarayanan [1992] has stated the following general principles for designing distributed file systems :

TABLE I. Clients have cycles to burn : This principle says that, if possible, it is always preferable to perform an operation on a client's own machine rather than performing it on a server machine. This is because server is a common resource for all clients, and hence cycles of a server machine are more precious than the cycles of client machines. This principle aims at enhancing the scalability of the design, since it lessens the need to increase centralized (commonly used) resources and allows graceful degradation of system performance as the system grows in size.

TABLE II. Cache whenever possible : Better performance, scalability, user

mobility, etc autonomy motivate this principle. Caching of data at clients' sites frequently to improve overall system performance because it makes data available wherever it is being currently used, thus saving a large amount of computing time and network bandwidth. Caching also enhances scalability because it reduces contention on centralized resources.

TABLE III. Exploit usage properties : This principle says that, depending on usage properties (access and modification patterns), files should be grouped into a small number of easily identifiable classes, and then class specific properties should be exploited for independent optimization for improved performance. For example, files known to be frequently read and modified only once in a while can be treated as immutable files for read only replication. Files containing the object code of system programs are good candidates for this class.

TABLE IV. Minimize system-wide knowledge and change : This principle is aimed at enhancing the scalability of design. The larger is a distributed system, the more difficult it is to be aware at all times of the entire state of the system and to update distributed or replicated data structures in consistent manner. Therefore monitoring or automatically updating of global information should be avoided as far as practicable. The 102 callback approach for cache validation and the use of negative rights in an access control list (ACL) based access control mechanism are two instances of the application of this principle. The use of hierarchical system structure is also an application of this principle.

- **Trust the fewest possible entities** : This principle is aimed at changing the security of the system. For example, it is much simpler to ensure security based on the integrity of the much smaller number of servers rather than trusting thousands of clients. In this case, it is sufficient to only ensure the physical security of these servers and the software they run.
- **Batch if possible :** Parching often helps in improving performance greatly. For example, grouping operation together can improve throughput, although it is often at the cost of latency. Similarly transfer of data across the network in large chunks rather than as individual pages in much more efficient. The full file transfer protocol is an instance of the application of this principle.

VIII. KEY POINTS TO REMEMBER DISTRIBUTED COMPUTING

1. Distributed computing is a field of computer science that studies distributed systems. A distributed system is a system whose components are located on different networked computers, which then communicate and coordinate their actions by passing messages to one other.
2. The components interact with one other in order to achieve a common goal. Three significant characteristics of distributed systems are: concurrency of components, lack of a global clock, and independent failure of components
3. A computer program that runs within a distributed system is called a distributed program (and distributed programming is the process of writing such programs).

IX. CONCLUSION

The concept of distributed computing is most efficient way to achieve the optimization. Distributed computing is anywhere : intranet, Internet and mobile ubiquitous computing. Main motivation factor is resource sharing like printers.

X. REFRENCES

1. Wikipedia
2. Inder Science
3. Andrew S. Tanenbaum and Maarten Van Steen Distributed Systems : Principles and Paradigms