Distributed Systems(UNIT 1)

Syllabus: Distributed systems – Parallel computing architectures: Vector processing, Symmetric multiprocessing and Massively parallel processing systems -High performance Cluster computing – Grid computing – Service Oriented Architecture

overview-concept of virtualization- hypervisor types xen and vmware.

Definition: Distributed system is one in which hardware and software components are located at networked computers communicate and coordinate their actions only by passing messages.

Challenges—in Construction of distributed systems are

1. Heterogeneity of their components, openness (which allows components to be added or replaced).

2. Security

3. Scalability—the ability to work well when the load or the number of users increases.

4. Failure handling

5. Concurrency of components

6. Transparency and

7. Quality of service.

CFHSSQT (easy to remember)

The Distributed systems have the following significant consequences:

A *parallel* program is one that uses a multiplicity of computational hardware (e.g., several processor cores) to perform a computation more quickly. The aim is to arrive at the answer earlier, by delegating different parts of the computation to different processors that execute at the same time.

By contrast, *concurrency* is a program-structuring technique in which there are multiple *threads of control*. Conceptually, the threads of control execute “at the same time”; that is, the user sees their effects interleaved. Whether they actually execute at the same time or not is an implementation detail; a concurrent program can execute on a single processor through interleaved execution or on multiple physical processors.

While parallel programming is concerned only with efficiency, concurrent programming is concerned with structuring a program that needs to interact with multiple independent external agents (for example, the user, a database server, and some external clients). Concurrency allows such programs to be *modular*; the thread that interacts with the user is distinct from the thread that talks to the database. In the absence of concurrency, such programs have to be written with event loops and callbacks, which are typically more cumbersome and lack the modularity that threads offer.

1. Concurrency: In network of systems concurrent program execution is the norm. The capacity of the systems to add resources and share resources to the network. The coordination of concurrently executing programs that share resources is also an important and recurring topic.
2. No global clock: shared idea of time at which the program actions occur. Close coordination often depends on a shared idea of the time at which the programs’ actions occur. But it turns out that there are limits to the accuracy with which the computers in a network can synchronize their clocks – there is no single global notion of the correct time.

3.Independent Failures: All computer systems fail, and it is responsibility of system designers to plan the consequences of possible failures. Ex: faults in network result in isolation of computers that are connected to it. Each component of the system can fail independently, leaving the others still running.

Prime motivation for constructing and using distributed systems is from a desire to share Resources.

4.Resources: The term ‘resource’ is best characterizes the range of things that can usefully be shared in a networked computer system. It extends from hardware components such as disks and printers to software-defined entities such as files, databases and data objects of all kinds.

Examples of resources are

1. The stream of video frames that emerges from a digital video camera
2. The audio connection that a mobile phone call represents.

Examples of Distributed systems:

Broad Classification of Distributed System based on application domains.

Selected application domains and associated networked applications

1. *Finance and commerce:*  eCommerce as exemplified by companies such as Amazon and eBay, and underlying payments technologies such as PayPal; the associated emergence of online banking and trading and also complex information dissemination systems for financial markets.
2. *The information society—*The growth of the World Wide Web as a repository of information and knowledge; the development of web search engines such as Google and Yahoo to search this vast repository;
3. Legacy information sources such as books (for example, Google Books);
4. User-generated content through sites such as YouTube, Wikipedia and Flickr;
5. Social networking through services such as Facebook and MySpace.
6. *Creative industries and entertainment--*online gaming, Music and film in home through network media centers and YouTube.
7. *Healthcare --*The growth of1as a discipline with its emphasis on a)Online electronic patient records and related issues of privacy; b)Telemedicine in supporting remote diagnosis or more advanced services such as c)Remote surgery application of networking and embedded systems technology in assisted living, (ex:for monitoring the elderly in their own homes).
8. *Education* The emergence of e-learning through for example web-based tools such as virtual learning environments.
9. *Transport and logistics—*The use of location technologies such as GPS in route finding systems
10. General traffic management systems(alerts to users based on current traffic and suggesting alternate routes dynamically.
11. The modern car itself as an example of a complex distributed system
12. The development of web-based map services such as MapQuest, Google Maps and Google Earth.
13. *Science—*The emergence of the Grid as a fundamental technology for eScience, including the use of complex networks of computers to support the storage, analysis and processing of (often very large quantities of scientific data).
14. *Environmental management—*The use of (networked) sensor technology to both monitor and manage the natural environment,(example—to provide early warning of natural disasters such as earthquakes, floods or tsunamis) suggesting changes of world environment.

*Examples of Distributed Systems:*

1. Web Search—the task of a web search engine is to index the entire contents of the World Wide Web, encompassing a wide range of information styles including web pages, multimedia sources and (scanned) books. This is a very complex task, as current estimates state that the Web consists of over 63 billion pages and one trillion unique web addresses. Given that most search engines analyze the entire web content and then carry out sophisticated processing on this enormous database, this task itself represents a major challenge for distributed systems design.

Google, the market leader in web search technology, has put significant effort into the design of a sophisticated distributed system infrastructure to support search (and indeed other Google applications and services such as Google arth).

Highlights of this infrastructure include:

**•** An underlying physical infrastructure consisting of very large numbers of networked computers located at data centers all around the world.

**•** A distributed file system designed to support very large files and heavily optimized for the style of usage required by search and other Google applications (especially reading from files at high and sustained rates).

**• A**n associated structured distributed storage system that offers fast access to very large datasets.

**•** A lock service that offers distributed system functions such as distributed locking and agreement.

**•** A programming model that supports the management of very large parallel and distributed computations across the underlying physical infrastructure.

The key concepts that underpin the development of distributed systems, with an emphasis on addressing the main challenges of distributed systems, including heterogeneity, openness, security, scalability, failure handling, concurrency, transparency and quality of service.

The subsequent treatment focuses inevitably on the constituent parts of a distributed system,

including:

1. communication paradigms such as remote invocation and

its indirect alternatives; the programming abstractions offered by objects, components or web services;

1. specific distributed systems services for security, naming and file system support; and
2. algorithmic solutions such as coordination and agreement.

It is equally important, however, to consider the overall design of distributed systems and how the constituent parts come together, addressing the inevitable tradeoffs between the different challenges to derive an overall system architecture that meets the requirements of a large-scale application domain and operating environment.

A fuller treatment of distributed systems design methods would necessarily take us into the field of software engineering methodologies for distributed systems.

Massively multiplayer online games(MMOGs)

Massively multiplayer online games offer an immersive experience whereby very large numbers of users interact through the Internet with a persistent virtual world. Leading examples of such games include Sony’s EverQuest II and EVE Online from the Finnish company CCP Games. Such worlds have increased significantly in sophistication and now include, complex playing arenas (for example EVE, Online consists of a universe with over 5,000 star systems) and multifarious social and economic systems. The number of players is also rising, with systems able to support over 50,000 simultaneous online players.

The engineering of MMOGs represents a major challenge for distributed systems technologies, particularly because of the need for

1. Fast response times to preserve the user experience of the game.
2. Real-time propagation of events to the many players and
3. Maintaining a consistent view of the shared world.

This therefore provides an excellent example of the challenges facing modern distributed systems designers.

A number of solutions have been proposed for the design of massively multiplayer online games:

**•** Perhaps surprisingly, the largest online game, EVE Online, utilizes a *client-server* architecture where a single copy of the state of the world is maintained on a centralized server and accessed by client programs running on players’ consoles or other devices. To support large numbers of clients, the server is a complex entity in its own right consisting of a cluster architecture featuring hundreds of

computer nodes

**CLANS**

AETERNUS

AETERNUS s a video game by CCP Games. It is a player-driven, persistent-world MMORPG set in a science fiction space setting. Characters pilot customizable ships through a galaxy of over 7,500 star systems. Most star systems are connected to one or more other star systems by means of stargates. The star systems can contain moons, planets, stations, wormholes, asteroid belts and complexes.

Players can participate in a number of in-game professions and activities, including mining, piracy, manufacturing, trading, exploration, and combat (both player versus environment and player versus player). The character advancement system is based upon training skills in real time, even while not logged into the game.

Eve Online was released in North America and Europe in May 2003. It was published from May to December 2003 by Simon & Schuster Interactive, after which CCP purchased the rights and began to self-publish via a digital distribution scheme. On January 22, 2008, it was announced that Eve Online would be distributed via Steam. On March 10, 2009, the game was again made available in boxed form in stores, released by Atari. In February 2013, EVE Online reached over 500,000 subscribers. The current version of Eve Online is Crius, released July 22, 2014.

The centralized architecture helps significantly in terms of the management of the virtual world and the single copy also eases consistency concerns. The goal is then to ensure fast response through optimizing network protocols and ensuring a rapid response to incoming events. To support this, the load is partitioned by allocating individual ‘star systems’ to particular computers within the cluster, with highly loaded star systems having their own dedicated computer and others sharing a computer. Incoming events are directed to the right computers within the cluster by keeping track of movement of players between star systems.

* Other MMOGs adopt more distributed architectures where the universe is partitioned across a (potentially very large) number of servers that may also be geographically distributed. Users are then dynamically allocated a particular server based on current usage patterns and also the network delays to the server (based on geographical proximity for example). This style of architecture, which is adopted by EverQuest, is naturally extensible by adding new servers.
* More radical architectures that are not based on client-server principles but rather adopt completely decentralized approaches based on peer-to-peer technology where every participant contributes resources (storage and processing) to accommodate the game.

Financial Trading:

As a final example, we look at distributed systems support for financial trading markets. The financial industry has long been at the cutting edge of distributed systems technology with its need, in particular, for real-time access to a wide range of information sources (for example, current share prices and trends, economic and political developments). The industry employs automated monitoring and trading applications.

Note that the emphasis in such systems is on the communication and processing of items of interest, known as *events* in distributed systems, with the need also to deliver events reliably and in a timely manner to potentially very large numbers of clients who have a stated interest in such information items. Examples of such events include a drop in a share price, the release of the latest unemployment figures, and so on and such systems typically employ what are known as *distributed event-based systems*.

Trends in distributed systems

Distributed systems are undergoing a period of significant change and this can be traced back to a number of influential trends:

1. the emergence of pervasive networking technology;
2. the emergence of ubiquitous computing coupled with the desire to support user mobility in distributed systems;
3. the increasing demand for multimedia services;
4. the view of distributed systems as a utility.
5. Pervasive networking and the modern Internet

The modern Internet is a vast interconnected collection of computer networks of many different types, with the range of types increasing all the time and now including, for example, a wide range of wireless communication technologies such as WiFi, WiMAX, Bluetooth and third-generation mobile phone networks.

The Internet is also a very large distributed system. It enables users, wherever they are, to make use of services such as the World Wide Web, email and file transfer. The figure shows a collection of intranets – subnetworks operated by companies and other organizations and typically protected by firewalls. The role of a *firewall* is to protect an intranet by preventing unauthorized messages from leaving or entering. A firewall is implemented by filtering incoming and outgoing messages. Filtering might be done by source or destination, or a firewall might allow only those messages related to email and web access to pass into or out of the intranet that it protects.

Internet Service Providers (ISPs) are companies that provide broadband links and other types of connection to individual users and small organizations, enabling them to access services anywhere in the Internet as well as providing local services such as email and web hosting. The intranets are linked together by backbones. A *backbone* is a network link with a high transmission capacity, employing satellite connections, fiber optic cables and other high-bandwidth circuits.

1. Mobile and ubiquitous computing: Device miniaturization and wireless networking 🡪 the integration of small and portable computing devices into distributed systems.

These devices include:

1. Laptop computers.
2. Handheld devices, including mobile phones, smart phones, GPS-enabled devices, pagers, personal digital assistants (PDAs), video cameras and digital cameras.
3. Wearable devices, such as smart watches with functionality similar to a PDA.(apple watches, Fitbit etc.)
4. Devices embedded in appliances such as washing machines, hi-fi systems, cars and refrigerators and Autonomous cars

Mobile computing 🡪mobility and connectivity on the move

Definition: Mobile computing is the performance of computing tasks while the user is on the move, or visiting places other than their usual environment. In mobile computing, users are still provided with access to resources via the devices they carry with them.

Mobility introduces a number of challenges for distributed systems, including the need to deal with variable connectivity and indeed disconnection, and the need to maintain operation in the face of device mobility.

*Ubiquitous computing* is the harnessing of many small, cheap computational devices that are present in users’ physical environments, including the home, office and even natural settings. The term ‘ubiquitous’ is intended to suggest that small computing devices will eventually become so pervasive in everyday objects that they are scarcely noticed. That is, their computational behavior will be transparently and intimately tied up with their physical function.

The presence of computers everywhere only becomes useful when they can communicate with one another. For example, now users can control their washing machine or their home air conditioning system from their mobile phone.

the user has access to three forms of wireless connection (host intranet, mobile phone, home intranet).

1.It connects to the rest of the host intranet via a gateway or access point.

2.The user also has a mobile (cellular) telephone, which is connected to the Internet. The phone gives access to the Web and other Internet services, constrained only by what can be presented on its small display, and may also provide location information via built-in GPS functionality.

Finally, 3. the user carries a digital camera, which can communicate over a personal area wireless network (with range up to about 10m) with a device such as a printer.

That means enabling the visitor’s device to communicate on the host network, and associating the device with suitable local services – a process called *service discovery*.

1. Distributed multimedia systems

Mobile and ubiquitous computing, ‘Ubiquitous’ means ‘to be found everywhere’,

wearable, handheld and context-aware computing.

*context-aware computing*, which is an important subfield of mobile and ubiquitous computing. This is where computer systems automatically adapt their behavior according to physical circumstances. Those circumstances can in principle be anything physically measurable or detectable, such as the presence of a user, the time of day or atmospheric conditions.

*volatile systems* that encompasses the essential distributed systems features of all of them. We call the systems 🡪‘volatile’ because,

The relevant forms of volatility include:

**•** failures of devices and communication links;

**•** changes in the characteristics of communication such as bandwidth;

**•** the creation and destruction of *associations* – logical communication relationships – between software components resident on the devices.

Multimedia support can usefully be defined as the ability to support a range of media types in an integrated manner. One can expect a distributed system to support the storage, transmission and presentation of what are often referred to as discrete media types, such as pictures or text messages. A distributed multimedia system should be able to perform the same functions for continuous media types such as audio and video; that is, it should be able to store and locate audio or video files, to transmit them across the network (possibly in real time as the streams emerge from a video camera), to support the presentation of the media types to the user and optionally also to share the media types across a group of users.

The crucial characteristic of continuous media types is that they include a temporal dimension, and indeed, the integrity of the media type is fundamentally dependent on preserving real-time relationships between elements of a media type.

The benefits of distributed multimedia computing are considerable in that a wide range of new (multimedia) services and applications can be provided on the desktop, including access to live or pre-recorded television broadcasts, access to film libraries offering video-on-demand services, access to music libraries, the provision of audio and video conferencing facilities and integrated telephony features including IP telephony or related technologies such as Skype, a peer-to-peer alternative to IP telephony.

*Webcasting* is an application of distributed multimedia technology. Webcasting is the ability to broadcast continuous media, typically audio or video, over the Internet. It is now commonplace for major sporting or music events to be broadcast in this way, often attracting large numbers of viewers (for example, the Live8 concert in 2005 attracted around 170,000 simultaneous users at its peak).

*Webcasting* is an application of distributed multimedia technology. Webcasting is the ability to broadcast continuous media, typically audio or video, over the Internet. It is now common place for major sporting or music events to be broadcast in this way.

1. Distributed computing as a utility: A number of companies are promoting the view of distributed resources as a commodity or utility, drawing the analogy between distributed resources and other utilities such as water or electricity. With this model, resources are provided by appropriate service suppliers and effectively rented rather than owned by the end user.

This model applies to both physical resources and more logical services: Physical resources such as storage and processing can be made available to networked computers, removing the need to own such resources on their own. At one end of the spectrum, a user may opt for a remote storage facility for file storage requirements.

This approach would enable a user to rent one or more computational nodes, either to meet their basic computing needs or indeed to perform distributed computation. Operating system virtualization is a key enabling technology for this approach, implying that users may actually be provided with services by a virtual rather than a physical node. This offers greater flexibility to the service supplier in terms of resource management.

Software services can also be made available across the global Internet using this approach. Indeed, many companies now offer a comprehensive range of services for effective rental, including services such as email and distributed calendars. Google, for example, bundles a range of business services under the banner Google Apps.

The term *cloud computing* is used to capture this vision of computing as a utility. A cloud is defined as a set of Internet-based application, storage and computing services sufficient to support most users’ needs, thus enabling them to largely or totally dispense with local data storage and application software.

The term also promotes a view of everything as a service, from physical or virtual infrastructure through to software, often paid for on a per-usage basis rather than purchased. Note that cloud computing reduces requirements on users’ devices, allowing very simple desktop or portable devices to access a potentially wide range of resources and services.

A *cluster computer* is a set of interconnected computers that cooperate closely to provide a single, integrated high performance computing capability. Building on projects such as the NOW (Network of Workstations) Project at Berkeley.

*Blade servers* are minimal computational elements containing for example processing and (main memory) storage capabilities. A blade system consists of a potentially large number of blade servers contained within a blade enclosure. Other elements such as power, cooling, persistent storage (disks), networking and displays, are provided either by the enclosure or through virtualized solutions.

The overall goal of cluster computers is to provide a range of cloud services, including high-performance computing capabilities, mass storage (for example through data centres), and richer application services such as web search).

*Grid computing* can also be viewed as a form of cloud computing. The terms are largely synonymous and at times ill-defined, but Grid computing can generally be viewed as a precursor to the more general paradigm of cloud computing with a bias towards support for scientific applications.

Focus on resource sharing

Users are so accustomed to the benefits of resource sharing that they may easily overlook their significance. We routinely share hardware resources such as printers, data resources such as files, and resources with more specific functionality such as search engines. But of far greater significance to users is the sharing of the higher-level resources that play a part in their applications and in their everyday work and social activities. For example, users are concerned with sharing data in the form of a shared database or a set of web pages – not the disks and processors on which they are implemented. Similarly, users think in terms of shared resources such as a search engine or a currency converter, without regard for the server or servers that provide these

At one extreme, a search engine on the Web provides a facility to users throughout the world, users who need never come into contact with one another directly. At the other extreme, in *computer-supported cooperative working* (CSCW), a group of users who cooperate directly share resources such as documents in a small, closed group. The pattern of sharing and the geographic distribution of particular users determine what mechanisms the system must supply to coordinate

users’ actions.

We use the term *service* for a distinct part of a computer system that manages a collection of related resources and presents their functionality to users and applications. For example, we access shared files through a file service; we send documents to printers through a printing service; we buy goods through an electronic payment service.

The only access we have to the service is via the set of operations that it exports. For example, a file service provides *read*, *write* and *delete* operations on files.

Resources in a distributed system are physically encapsulated within computers and can only be accessed from other computers by means of communication. For effective sharing, each resource must be managed by a program that offers a communication interface enabling the resource to be accessed and updated reliably and consistently.

The term *server* is probably familiar to most readers. It refers to a running program (a *process*) on a networked computer that accepts requests from programs running on other computers to perform a service and responds appropriately. The requesting processes are referred to as *clients,* and the overall approach is known as *client-server computing*. In this approach, requests are sent in messages from clients to a server and replies are sent in messages from the server to the clients.

When the client sends a request for an operation to be carried out, we say that the client *invokes an operation* upon the server. A complete interaction between a client and a server, from the point when the client sends its request to when it receives the server’s response, is called a *remote invocation*.

In a distributed system written in an object-oriented language, resources may be encapsulated as objects and accessed by client objects, in which case we speak of a *client object* invoking a method upon a *server object*.

1. Challenges
2. Heterogeneity: The Internet enables users to access services and run applications over a heterogeneous collection of computers and networks. Heterogeneity (that is, variety and difference) applies to all of the following:

* Networks: systems connected by ethernet have different protocol where as systems connected with other optical fiber may employ different protocol.
* computer hardware: Processor, Byte ordering
* operating systems:
* programming languages;
* implementations by different developers.

Computer Hardware: Data types such as integers may be represented in different ways on different sorts of hardware – for example, there are two alternatives for the byte ordering of integers.

ABCD-----65666768(ASCII format)

0100 0001100 0010 10000011 10000100

lsb msb big endian

DCBA

0100 0001

31 0 –little endian

Big Endian and Little Endian

Diagram

Description automatically generated

(big endian)A<---🡪B(little)

Windows o/s linux o/s

These differences in representation must be dealt with if messages are to be exchanged between programs running on different hardware.

1. Operating Systems: Although the operating systems of all computers on the Internet need to include an implementation of the Internet protocols, they do not necessarily all provide the same application programming interface to these protocols. For example, the calls for exchanging messages in UNIX are different from the calls in Windows.
2. Programming Languages: Different programming languages use different representations for characters and data structures such as arrays and records. Java uses Unicode for character representation different from ASCII character representation.

Middleware • The term middleware applies to a software layer that provides a programming abstraction as well as masking the heterogeneity of the underlying networks, hardware, operating systems and programming languages.

Examples of middleware are: the Common Object Request Broker (CORBA), Some middleware, such as Java Remote Method Invocation (RMI) 🡪 supports only a single programming language. Most middleware is implemented over the Internet protocols, which themselves mask the differences of the underlying networks, but all middleware deals with the differences in operating systems and hardware

In addition to solving the problems of heterogeneity, middleware provides a uniform computational model for use by the programmers of servers and distributed applications. Possible models include remote object invocation, remote event notification, remote SQL access and distributed transaction processing. For example, CORBA provides remote object invocation, which allows an object in a program running on one computer to invoke a method of an object in a program running on another computer. Its implementation hides the fact that messages are passed over a network in order to send the invocation request and its reply.

Heterogeneity and mobile code: The term mobile code is used to refer to program code that can be transferred from one computer to another and run at the destination – Java applets are an example. Code suitable for running on one computer is not necessarily suitable for running on another because executable programs are normally specific both to the instruction set and to the host operating system.

The virtual machine approach provides a way of making code executable on a variety of host computers: the compiler for a particular language generates code for a virtual machine instead of a particular hardware order code.

For example, the Java compiler produces code for a Java virtual machine, which executes it by interpretation. The Java virtual machine needs to be implemented once for each type of computer to enable Java programs to run.

1. Openness: The openness of a computer system is the characteristic that determines whether the system can be extended and re-implemented in various ways. The openness of distributed systems is determined primarily by the degree to which new resource-sharing services can be added and be made available for use by a variety of client programs.

Openness cannot be achieved unless the specification and documentation of the key software interfaces of the components of a system are made available to software developers. In a word, the key interfaces are published.

To summarize:

1. Open systems are characterized by the fact that their key interfaces are published.
2. Open distributed systems are based on the provision of a uniform communication mechanism and published interfaces for access to shared resources.
3. Open distributed systems can be constructed from heterogeneous hardware and software, possibly from different vendors. But the conformance of each component to the published standard must be carefully tested and verified if the system is to work correctly.
4. Security: Security for information resources has three components: confidentiality (protection against disclosure to unauthorized individuals), integrity (protection against alteration or corruption), and availability (protection against interference with means to access resources).

Denial of service attacks: Another security problem is that a user may wish to disrupt a service for some reason. This can be achieved by bombarding the service with such a large number of pointless requests that the serious users are unable to use it. This is called a denial of service attack. There have been several denial of service attacks.

Security of mobile code: Mobile code needs to be handled with care. Consider someone who receives an executable program as an electronic mail attachment: the possible effects of running the program are unpredictable; for example, it may seem to display an interesting picture but in reality it may access local resources, or perhaps be part of a denial of service attack.

1. Scalability: A system is described as scalable if it will remain effective when there is a significant increase in the number of resources and the number of users. The number of computers and servers in the Internet has increased dramatically.

The design of scalable distributed systems presents the following challenges: Controlling the cost of physical resources: As the demand for a resource grows, it should be possible to extend the system, at reasonable cost, to meet it. For example, the frequency with which files are accessed in an intranet is likely to grow as the number of users and computers increases. It must be possible to add server computers to avoid the performance bottleneck that would arise if a single file server had to handle all file access requests.

Controlling the performance loss: Consider the management of a set of data whose size is proportional to the number of users or resources in the system. Example: the table with the correspondence between the domain names of computers and their Internet addresses held by the Domain Name System, which is used mainly to look up DNS names such as www.amazon.com. Algorithms that use hierarchic structures scale better than those that use linear structures. But even with hierarchic structures an increase in size will result in some loss in performance: the time taken to access hierarchically structured data is O(log n), where n is the size of the set of data. For a system to be scalable, the maximum performance loss should be no worse than this.

Preventing software resources running out: An example of lack of scalability is shown by the numbers used as Internet (IP) addresses (computer addresses in the Internet). In the late 1970s, it was decided to use 32 bits for this purpose, but the supply of available Internet addresses is running out. For this reason, a new version of the protocol with 128-bit Internet addresses is being adopted and this will require modifications to many software components.

Avoiding performance bottlenecks: In general, algorithms should be decentralized to avoid having performance bottlenecks. With reference to the Domain Name System, in which the name table was kept in a single master file that could be downloaded to any computers that needed it. That was fine, later this was distributed by partitioning the main table to be stored among different servers of internet for ease of access and performance improvement.

The issue of scale is a dominant theme in the development of distributed systems. The techniques that have been successful are discussed extensively in this book. They include the use of replicated data, the associated technique of caching and the deployment of multiple servers to handle commonly performed tasks, enabling several similar tasks to be performed concurrently.

1. Failure handling: Computer systems sometimes fail. When faults occur in hardware or software, programs may produce incorrect results or may stop before they have completed the intended computation.

Failures in a distributed system are partial – that is, some components fail while others continue to function. Therefore the handling of failures is particularly difficult.

Detecting failures: Some failures can be detected. For example, checksums can be used to detect corrupted data in a message or a file. It is difficult or even impossible to detect some other failures, such as a remote crashed server in the Internet. The challenge 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.

Two examples of hiding failures:

1. Messages can be retransmitted when they fail to arrive.
2. File data can be written to a pair of disks so that if one is corrupted, the other may still be correct.

Tolerating failures: Most of the services on the Internet do exhibit failures – it would not be practical for them to attempt to detect and hide all of the failures that might occur in such a large network with so many components. Their clients can be designed to tolerate failures, which generally involves the users tolerating 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. In general, the computations performed by some programs will be incomplete when a fault occurs, and the permanent data that they update (files and other material stored in permanent storage) may not be in a consistent state.

Redundancy: Services can be made to tolerate failures by the use of redundant components. Consider the following examples:

1. There should always be at least two different routes between any two routers in the Internet.
2. In the Domain Name System, every name table is replicated in at least two different servers.
3. A database may be replicated in several servers to ensure that the data remains accessible after the failure of any single server; the servers can be designed to detect faults in their peers; when a fault is detected in one server, clients are redirected to the remaining servers.

The design of effective techniques for keeping replicas of rapidly changing data up to date without excessive loss of data without excessive

1. Concurrency: Both services and applications provide resources that can be shared by clients in a distributed system. There is therefore a possibility that several clients will attempt to access a shared resource at the same time.

For example, a data structure that records bids for an auction may be accessed very frequently when it gets close to the deadline time. The process that manages a shared resource could take one client request at a time. But that approach limits throughput. Therefore services and applications generally allow multiple client requests to be processed concurrently. To make this more concrete, suppose that each resource is encapsulated as an object and that invocations are executed in concurrent threads.

1. Transparency: Transparency is defined as the concealment from the user and the application programmer of the separation of components in a distributed system, so that the system is perceived as a whole rather than as a collection of independent components. The implications of transparency are a major influence on the design of the system software.

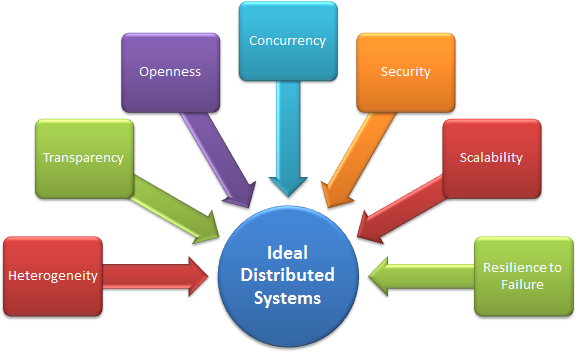
International Organization for Standardization’s Reference Model for Open Distributed Processing (RM-ODP) [ISO 1992] identify eight forms of transparency. We have paraphrased the original ANSA definitions, replacing their migration transparency with our own mobility transparency, whose scope is broader:

1. Access transparency enables local and remote resources to be accessed using identical operations.
2. Location transparency enables resources to be accessed without knowledge of their physical or network location (for example, which building or IP address).
3. Concurrency transparency enables several processes to operate concurrently using shared resources without interference between them.
4. Replication transparency: transparency enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers.
5. Failure transparency enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components.
6. Mobility transparency allows the movement of resources and clients within a system without affecting the operation of users or programs.
7. Performance transparency allows the system to be reconfigured to improve performance as loads vary.
8. Scaling transparency allows the system and applications to expand in scale without change to the system structure or the application algorithms.

Network transparency=Location +Access Transparency

The two most important transparencies are access and location transparency; their presence or absence most strongly affects the utilization of distributed resources. They are sometimes referred to together as network transparency.

1. Quality of service: The main nonfunctional properties of systems that affect the quality of the service experienced by clients and users are reliability, security and performance. Adaptability to meet changing system configurations and resource availability has been recognized as a further important aspect of service quality. Reliability and security issues are critical in the design of most computer systems.

QoS has effectively been commandeered to refer to the ability of systems to meet such deadlines. Its achievement depends upon the availability of the necessary computing and network resources at the appropriate times. This implies a requirement for the system to provide guaranteed computing and communication resources that are sufficient to enable applications to complete each task on time (for example, the task of displaying a frame of video. 

**Key characteristics of distributed systems**

* Resource sharing.
* Openess.
* Concurrency.
* Scalability.
* Fault Tolerance.
* Transparency.