ANJALAI AMMAL MAHALINGAM ENGINEERING COLLEGE,
KOVILVENNI – 614 403

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SYLLABUS

Subject Name : DISTRIBUTED SYSTEMS
Subject code : CS6601

UNIT I INTRODUCTION 7
Examples of Distributed Systems–Trends in Distributed Systems – Focus on resource sharing – Challenges. **Case study:** World Wide Web.

UNIT II COMMUNICATION IN DISTRIBUTED SYSTEM 10
System Model – Inter process Communication - the API for internet protocols – External data representation and Multicast communication. **Network virtualization:** Overlay networks. **Case study:** MPI **Remote Method Invocation And Objects:** Remote Invocation – Introduction - Request-reply protocols - Remote procedure call - Remote method invocation. **Case study:** Java RMI – Group communication - Publish-subscribe systems - Message queues - Shared memory approaches Distributed objects – Case study: Enterprise Java Beans –from objects to components.

UNIT III PEER TO PEER SERVICES AND FILE SYSTEM 10
Peer-to-peer Systems – Introduction - Napster and its legacy - Peer-to-peer – Middleware – Routing overlays. **Overlay case studies:** Pastry, Tapestry- Distributed File Systems – Introduction – File service architecture – Andrew File system. **File System:** Features-File model -File accessing models - File sharing semantics **Naming:** Identifiers, Addresses, Name Resolution – Name Space Implementation – Name Caches – LDAP.

UNIT IV SYNCHRONIZATION AND REPLICATION 9
Introduction - Clocks, events and process states - Synchronizing physical clocks- Logical time and logical clocks - Global states – Coordination and Agreement – Introduction – Distributed mutual exclusion – Elections – Transactions and Concurrency Control– Transactions -Nested transactions – Locks – Optimistic concurrency control - Timestamp ordering – Atomic Commit protocols –Distributed deadlocks – Replication – Case study – Coda.

UNIT V PROCESS & RESOURCE MANAGEMENT 9
Process Management: Process Migration: Features, Mechanism - Threads: Models, Issues, Implementation. **Resource Management:** Introduction- Features of Scheduling Algorithms –Task Assignment Approach – Load Balancing Approach – Load Sharing Approach**.**

TOTAL: 45 PERIODS

TEXT BOOKS

1. George Coulouris, Jean Dollimore and Tim Kindberg, “Distributed Systems Concepts and Design”, Fifth Edition, Pearson Education, 2012.

REFERENCES

1. Pradeep K Sinha, "Distributed Operating Systems: Concepts and Design", Prentice Hall of India, 2007.
2. Tanenbaum A.S., Van Steen M., “Distributed Systems: Principles and Paradigms”, Pearson Education, 2007.
3. Liu M.L., “Distributed Computing, Principles and Applications”, Pearson Education, 2004.
4. Nancy A Lynch, “Distributed Algorithms”, Morgan Kaufman Publishers, USA, 2003.

CS6601 - DISTRIBUTED SYSTEMS

UNIT I INTRODUCTION

Examples of Distributed Systems–Trends in Distributed Systems – Focus on resource sharing – Challenges. Case study: World Wide Web.

Introduction

- A distributed system is defined as one in which hardware or software components located at networked computers communicate and coordinate their actions only by passing messages.
- The above definition of distributed systems has the following significant consequences:

1. *Concurrency*:
   - In a network of computers, concurrent program execution is the norm. They share resources such as web pages or files when necessary.
   - The capacity of the system to handle shared resources can be increased by adding more resources (for example. computers) to the network.
   - The coordination of these concurrently executing programs that share resources is important.

2. *No global clock*:
   - When programs need to cooperate they coordinate their actions by exchanging messages.
   - 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.
   - This is a direct consequence of the fact that the *only* communication is by sending messages through a network.

3. Independent failures:
   - Faults in the network result in the isolation of the computers that are connected to it, but that doesn't mean that they stop running.
   - In fact, the programs on them may not be able to detect whether the network has failed or has become unusually slow.
   - Similarly, the failure of a computer, or the unexpected termination of a program somewhere in the system (a crash), is not immediately made known to the other components with which it communicates.
   - Each component of the system can fail independently, leaving the others still running.

- The prime motivation for constructing and using distributed systems stems from a desire to share resources.
- The term 'resource' 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 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.
- The Web consists of over 63 billion pages and one trillion unique web addresses.
- 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 has put significant effort into the design of a sophisticated distributed system infrastructure to support search.

- Highlights of this infrastructure include:
  1. an underlying physical infrastructure consisting of very large numbers of networked computers located at data centres all around the world;
  2. 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);
  3. an associated structured distributed storage system that offers fast access to very large datasets;
  4. a lock service that offers distributed system functions such as distributed locking and agreement;
  5. a programming model that supports the management of very large parallel and distributed computations across the underlying physical infrastructure.

2. 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 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 fast response times to preserve the user experience of the game.
- Other challenges include the real-time propagation of events to the many players and maintaining a consistent view of the shared world.
- A number of solutions have been proposed for the design of massively multiplayer online games:
  - *Client-server* architecture
    - 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.
    - The centralized architecture helps in the management of the virtual world and the single copy also eases consistency concerns.
    - The goal is to ensure fast response through optimizing network protocols and ensuring a rapid response to incoming events.
    - 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.
  - Distributed architectures
    - 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.
    - This style of architecture (adopted by EverQuest) is naturally extensible by adding new servers.
  - Researchers are now looking at more radical architectures that adopt completely decentralized approaches based on peer-to-peer technology where every participant contributes resources (storage and processing) to accommodate the game.

3. Financial trading

- The financial industry has long been at the cutting edge of distributed systems technology with its need 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.
- 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.
- This requires a very different style of underlying architecture, and such systems typically employ what are known as *distributed event-based systems*.

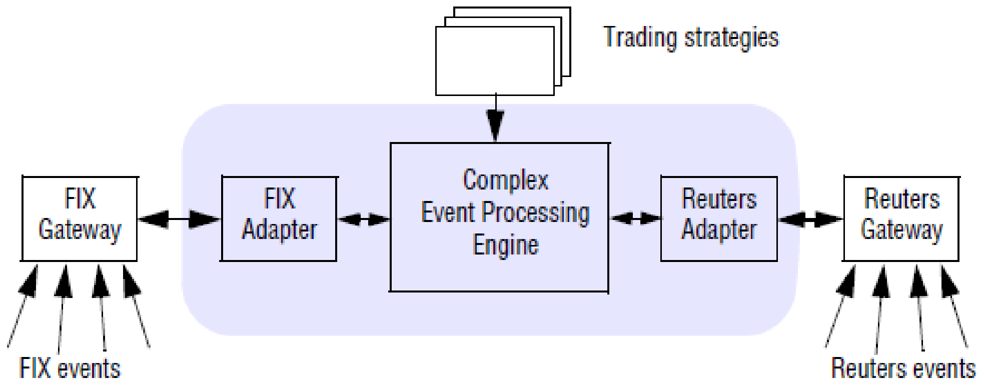


Fig. 1 An example financial trading system

- Fig. 1 illustrates a typical financial trading system. This shows a series of event feeds coming into a given financial institution.
- Such event feeds share the following characteristics.
    - Firstly, the sources are typically in a variety of formats, such as Reuters market data events and FIX events (events following the specific format of the Financial Information eXchange protocol), and indeed from different event technologies, thus illustrating the problem of heterogeneity
    - . The figure shows the use of adapters which translate heterogeneous formats into a common internal format.
    - Secondly, the trading system must deal with a variety of event streams, all arriving at rapid rates, and often requiring real-time processing to detect patterns that indicate trading opportunities.
    - This used to be a manual process but it is automated in terms of what is known as Complex Event Processing (CEP).
    - It offers a way of composing event occurrences together into logical, temporal or spatial patterns.
    - This approach is primarily used to develop customized algorithmic trading strategies covering both buying and selling of stocks and shares.
    - As an example, consider the following script:

```
WHEN
        MSFT price moves outside 2% of MSFT Moving Average
FOLLOWED-BY (
        MyBasket moves up by 0.5%
        AND
                HPQ’s price moves up by 5%
                OR
                MSFT’s price moves down by 2%
        )
    )
    ALL WITHIN
        any 2 minute time period
    THEN
        BUY MSFT
        SELL HPQ
```

- This script is based on the functionality provided by Apama, a commercial product in the financial world.
- The script detects a complex temporal sequence based on the share prices of Microsoft, HP and a basket of other share prices, resulting in decisions to buy or sell particular shares.

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:
  - the emergence of pervasive networking technology;
  - the emergence of ubiquitous computing coupled with the desire to support user mobility in distributed systems;
  - the increasing demand for multimedia services;
  - the view of distributed systems as a utility.

1. Pervasive networking and the modern Internet

- The modern Internet is a vast interconnected collection of computer networks of many different types, including, for example, a wide range of wireless communication technologies such as WiFi, WiMAX, Bluetooth and third-generation mobile phone networks.
- The net result is that networking has become a pervasive resource and devices can be connected (if desired) at any time and in any place.
- Fig. 2 illustrates a typical portion of the Internet. Programs running on the computers connected to it interact by passing messages, employing a common means of communication.
- The design and construction of the Internet communication mechanisms (the Internet protocols) enable a program running anywhere to address messages to programs anywhere else and abstracting over the myriad of technologies mentioned above.
- 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 set of services is open-ended – it can be extended by the addition of server computers and new types of service.

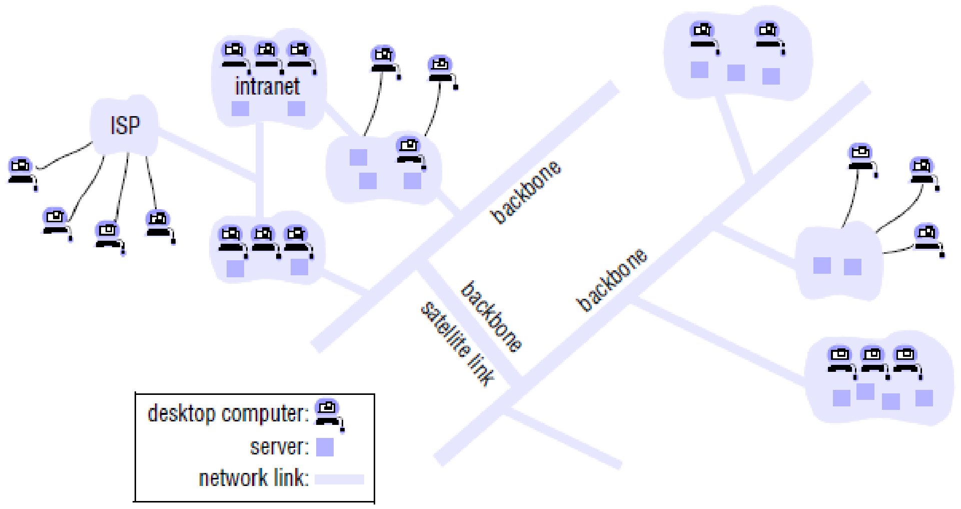


Fig. 2 A typical portion of the Internet

- 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, fibre optic cables and other high-bandwidth circuits.
- Some organizations may not wish to connect their internal networks to the Internet at all.
- Firewalls can also be problematic in distributed systems by impeding legitimate access to services when resource sharing between internal and external users is required. Hence, firewalls must often be complemented by more fine-grained mechanisms and policies.

2. Mobile and ubiquitous computing

- Technological advances in device miniaturization and wireless networking have led increasingly to the integration of small and portable computing devices into distributed systems.
- These devices include:
    - *Laptop computers.*
    - *Handheld devices, including mobile phones, smart phones, GPS-enabled devices, pagers, personal digital assistants (PDAs), video cameras and digital cameras.*
    - *Wearable devices, such as smart watches with functionality similar to a PDA.*
    - *Devices embedded in appliances such as washing machines, hi-fi systems, cars and refrigerators.*
- The portability of many of these devices, together with their ability to connect conveniently to networks in different places, makes *mobile computing* possible.
- 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 who are away from their 'home' intranet (the intranet at work, or their residence) are still provided with access to resources via the devices they carry with them.
- They can continue to access the Internet; they can continue to access resources in their home intranet; and there is increasing provision for users to utilize resources such as printers or even sales points that are conveniently nearby as they move around.
- The latter is also known as *location-aware* or *context-aware computing.*
- 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.
- 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 behaviour 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, it may be convenient for users to control their washing machine or their entertainment system from their phone or a 'universal remote control' device in the home.*
    - *Equally, the washing machine could notify the user via a smart badge or phone when the washing is done.*

- Ubiquitous and mobile computing overlap, since the mobile user can in principle benefit from computers that are everywhere. But they are distinct, in general.
- Ubiquitous computing could benefit users while they remain in a single environment such as the home or a hospital.
- Similarly, mobile computing has advantages even if it involves only conventional, discrete computers and devices such as laptops and printers.

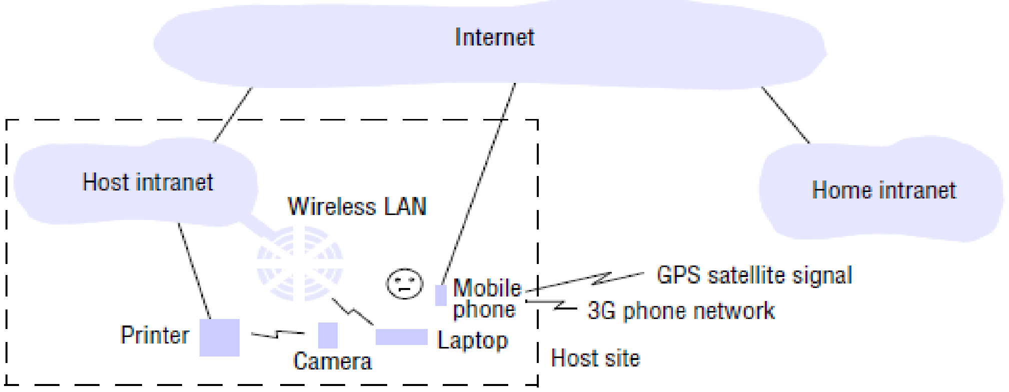


Fig. 3 Portable and handheld devices in a distributed system

- Fig. 3 shows a user who is visiting a host organization. The figure shows the user's home intranet and the host intranet at the site that the user is visiting.
- Both intranets are connected to the rest of the Internet.

- The user has access to *three forms of wireless connection*.
    - Their laptop has a means of connecting to the host's wireless LAN. This network provides coverage of a few hundred metres (a floor of a building, say).
    - It connects to the rest of the host intranet via a gateway or access point.
    - 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, 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.

- While journeying to the host site, the user can fetch the latest stock prices from a web server using the mobile phone and can also use the built-in GPS and route finding software to get directions to the site location.
- This scenario demonstrates the need to support *spontaneous interoperation*, whereby associations between devices are routinely created and destroyed.
- The main challenge applying to such situations is to make interoperation fast and convenient (that is, spontaneous) even though the user is in an environment they may never have visited before.
- 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*.

3. Distributed multimedia systems

- A distributed multimedia system 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.

- For example, in a video presentation it is necessary to preserve a given throughput in terms of frames per second and, for real-time streams, a given maximum delay or latency for the delivery of frames.
- 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

- It 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.
- Distributed multimedia applications such as webcasting place considerable demands on the underlying distributed infrastructure in terms of:
    - providing support for an (extensible) range of encoding and encryption formats, such as the MPEG series of standards (including for example the popular MP3 standard otherwise known as MPEG-1, Audio Layer 3) and HDTV;
    - providing a range of mechanisms to ensure that the desired quality of service can be met;
    - providing associated resource management strategies, including appropriate scheduling policies to support the desired quality of service;
    - providing adaptation strategies to deal with the inevitable situation in open systems where quality of service cannot be met or sustained.

4. Distributed computing as a utility

- With the increasing maturity of distributed systems infrastructure, a number of companies are promoting the view of distributed resources as a commodity or utility.
- 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
        - 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 (for example, for multimedia data such as photographs, music or video) and/or for backups.
        - Similarly, 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.
        - At the other end of the spectrum, users can access sophisticated *data centres* (networked facilities offering access to repositories of often large volumes of data to users or organizations) or indeed computational infrastructure using the sort of services now provided by companies such as Amazon and Google.
        - 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
        - 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.

Cloud computing

- 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 (Fig. 4 Cloud computing).
- 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.
- Cloud computing reduces requirements on users' devices, allowing very simple desktop or portable devices to access a potentially wide range of resources and services.
- Clouds are generally implemented on cluster computers to provide the necessary scale and performance required by such services.

Cluster computers

- A *cluster computer* is a set of interconnected computers that cooperate closely to provide a single, integrated high-performance computing capability.
- Most clusters consist of commodity PCs running a standard (sometimes cut-down) version of an operating system such as Linux, interconnected by a local area network.
- Companies such as HP, Sun and IBM offer blade solutions. *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.
- Through this solution, individual blade servers can be much smaller and also cheaper to produce than commodity PCs.
- 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.

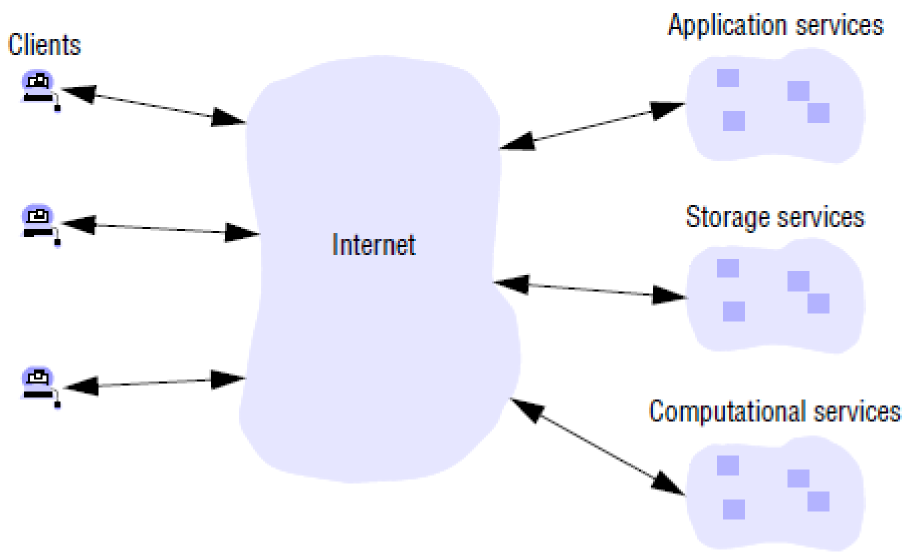


Fig. 4 Cloud computing

Focus on resource sharing

- Hardware resources such as printers, data resources such as files, and resources with more specific functionality such as search engines are routinely shared.
- Looked at from the point of view of hardware provision, equipment such as printers and disks are shared to reduce costs.
- 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.

- In practice, patterns of resource sharing vary widely in their scope and in how closely users work together.
- 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 determines what mechanisms the system must supply to coordinate users' actions.

- The term *service* is used 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, accessing shared files through a file service; sending documents to printers through a printing service; buying goods through an electronic payment service.
- The only access 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.

- The fact that services restrict resource access to a well-defined set of operations is in part standard software engineering practice. But it also reflects the physical organization of distributed systems.
- 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.

Client-server computing

- The term *server* 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.
- The client *invokes an operation* upon the server by sending a request for an operation to be carried out.
- 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*.
- The same process may be both a client and a server, since servers sometimes invoke operations on other servers.
- The terms 'client' and 'server' apply only to the roles played in a single request.

- Clients are active (making requests) and servers are passive (only waking up when they receive requests); servers run continuously, whereas clients last only as long as the applications of which they form a part.
- Many, but certainly not all, distributed systems can be constructed entirely in the form of interacting clients and servers.
- Example: The World Wide Web, email and networked printers all fit this model.
- An executing web browser is an example of a client. The web browser communicates with a web server, to request web pages from it.

Challenges

The main challenges of distributed systems are:

1. Heterogeneity

- The Internet enables users to access services and run applications over a heterogeneous collection of computers and networks. Heterogeneity applies to all of the following:
  - networks;
  - computer hardware;
  - operating systems;
  - programming languages;
  - implementations by different developers.
- Although the Internet consists of many different sorts of network, their differences are masked by the fact that all of the computers attached to them use the Internet protocols to communicate with one another.
- Data types such as integers may be represented in different ways on different sorts of hardware.
- These differences in representation must be dealt with if messages are to be exchanged between programs running on different hardware.
- 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.

- Different programming languages use different representations for characters and data structures such as arrays and records.
- These differences must be addressed if programs written in different languages are to be able to communicate with one another.
- Programs written by different developers cannot communicate with one another unless they use common standards, for example, for network communication and the representation of primitive data items and data structures in messages.
- For this to happen, standards need to be agreed and adopted.

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.
- The Common Object Request Broker (CORBA) is an example.
- 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.
- 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.

2. Openness

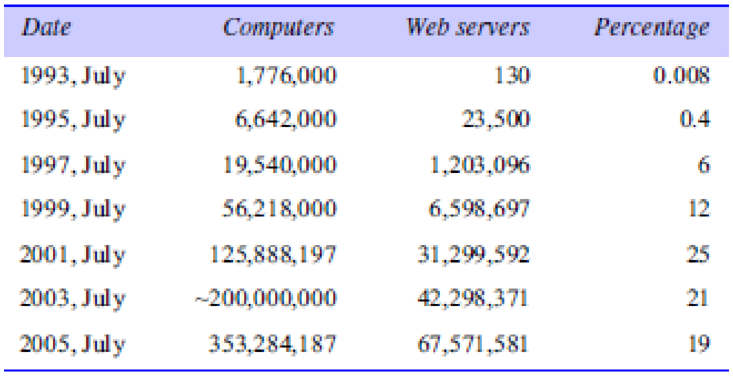
- ➢ The openness of a computer system is the characteristic that determines whether the system can be extended and re-implemented in various ways.
- ➢ The openness of distributed systems is determined primarily by the degree to which new resource-sharing services can be added and be made available for use by a variety of client programs.
- ➢ To achieve openness the specification and documentation of the key software interfaces of the components of a system are made available to software developers (that is the key interfaces are *published*).
- ➢ This process is akin to the standardization of interfaces, but it often bypasses official standardization procedures.
- ➢ Systems that are designed to support resource sharing in this way are termed *open distributed systems* to emphasize the fact that they are extensible.
- ➢ They may be extended at the hardware level by the addition of computers to the network and at the software level by the introduction of new services and the reimplementation of old ones, enabling application programs to share resources.
- ➢ A further benefit is their independence from individual vendors.

3. Security

- ➢ Security for information resources has three components:
    - o Confidentiality (protection against disclosure to unauthorized individuals),
    - o integrity (protection against alteration or corruption),
    - o availability (protection against interference with the means to access the resources).
- ➢ In a distributed system, clients send requests to access data managed by servers, which involves sending information in messages over a network.
- ➢ But security is not just a matter of concealing the contents of messages – it also involves knowing for sure the identity of the user or other agent on whose behalf a message was sent.
- ➢ The second challenge here is to identify a remote user or other agent correctly. Both of these challenges can be met by the use of encryption techniques developed for this purpose.
- ➢ However, the following two security challenges have not yet been fully met:
    - • *Denial of service attacks*:
        - o A user may wish to disrupt a service for some reason.
        - o 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.
    - • *Security of mobile code*:
        - o Consider someone who receives an executable program as an electronic mail attachment: the possible effects of running the program are unpredictable;
        - o 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.

4. 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.
- Fig. 5 shows the increasing number of computers and web servers during the 12-year history of the Web up to 2005.



Date	Computers	Web servers	Percentage
1993, July	1,776,000	130	0.008
1995, July	6,642,000	23,500	0.4
1997, July	19,540,000	1,203,096	6
1999, July	56,218,000	6,598,697	12
2001, July	125,888,197	31,299,592	25
2003, July	~200,000,000	42,298,371	21
2005, July	353,284,187	67,571,581	19

Fig. 5 Growth of the Internet (computers and web servers)

- 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.
- In general, for a system with *n* users to be scalable, the quantity of physical resources required to support them should be at most O(*n*) – that is, proportional to *n*.

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.
- For 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, 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. There is no correct solution to this problem.
- Overcompensating for future growth may be worse than adapting to a change when we are forced to – larger Internet addresses will occupy extra space in messages and in computer storage.

Avoiding performance bottlenecks:

- In general, algorithms should be decentralized to avoid having performance bottlenecks.
- In the predecessor of the Domain Name System, the name table was kept in a single master file that could be downloaded to any computers that needed it.
- That was fine when there were only a few hundred computers in the Internet, but it soon became a serious performance and administrative bottleneck.
- The Domain Name System removed this bottleneck by partitioning the name table between servers located throughout the Internet and administered locally.
- Some shared resources are accessed very frequently; for example, many users may access the same web page, causing a decline in performance.
- Ideally, the system and application software should not need to change when the scale of the system increases, but this is difficult to achieve.
- The issue of scale is a dominant theme in the development of distributed systems. The techniques that have been successful 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.

5. Failure handling

- 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. The following are the techniques for dealing with failures:

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 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.
- Just dropping a message that is corrupted is an example of making a fault less severe – it could be retransmitted.
- The techniques described for hiding failures are not guaranteed to work in the worst cases; for example, the data on the second disk may be corrupted too, or the message may not get through in a reasonable time however often it is retransmitted.

Tolerating failures:

- Most of the services in 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 involve the users tolerating them as well.
- For example, when a web browser cannot contact a web server, it does not make the user wait for ever while it keeps on trying – it informs the user about the problem, leaving them free to try again later.

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 upto- date without excessive loss of performance is a challenge.
- Distributed systems provide a high degree of availability in the face of hardware faults. The *availability* of a system is a measure of the proportion of time that it is available for use.
- When one of the components in a distributed system fails, only the work that was using the failed component is affected.
- A user may move to another computer if the one that they were using fails; a server process can be started on another computer.

6. Concurrency

- ➢ Both services and applications provide resources that can be shared by clients in a distributed system.
- ➢ There is a possibility that several clients will attempt to access a shared resource at the same 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.
- ➢ In this case it is possible that several threads may be executing concurrently within an object, in which case their operations on the object may conflict with one another and produce inconsistent results.
- ➢ Any object that represents a shared resource in a distributed system must be responsible for ensuring that it operates correctly in a concurrent environment.

7. 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.
- ➢ The ANSA Reference Manual and the International Organization for Standardization's Reference Model for Open Distributed Processing (RM-ODP) identify eight forms of transparency.

Access transparency enables local and remote resources to be accessed using identical operations.

Location transparency enables resources to be accessed without knowledge of their physical or network location (for example, which building or IP address).

Concurrency transparency enables several processes to operate concurrently using shared resources without interference between them.

Replication transparency enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers.

Failure transparency enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components.

Mobility transparency allows the movement of resources and clients within a system without affecting the operation of users or programs.

Performance transparency allows the system to be reconfigured to improve performance as loads vary.

Scaling transparency allows the system and applications to expand in scale without change to the system structure or the application algorithms.

The two transparencies, access and location transparency are sometimes referred to together as *network transparency*.

8. Quality of service

- Once users are provided with the functionality that they require of a service, such as the file service in a distributed system, then the question is about the quality of the service provided.
- 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 important aspect of service quality.
- **Reliability and security** issues are critical in the design of most computer systems.

- The **performance** aspect of quality of service was originally defined in terms of responsiveness and computational throughput, but it has been redefined in terms of ability to meet timeliness guarantees.
- Some applications, including multimedia applications, handle *time-critical data* – streams of data that are required to be processed or transferred from one process to another at a fixed rate.
- For example, a movie service might consist of a client program that is retrieving a film from a video server and presenting it on the user's screen.

- In fact, the abbreviation 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).

- The networks commonly used today have high performance – for example, BBC iPlayer generally performs acceptably – but when networks are heavily loaded their performance can deteriorate, and no guarantees are provided.
- QoS applies to operating systems as well as networks. Each critical resource must be reserved by the applications that require QoS, and there must be resource managers that provide guarantees. Reservation requests that cannot be met are rejected.

Case study: The World Wide Web

- The World Wide Web is an evolving system for publishing and accessing resources and services across the Internet.
- Through commonly available web browsers, users retrieve and view documents of many types, listen to audio streams and view video streams, and interact with an unlimited set of services.

- A key feature of the Web is that it provides a hypertext structure among the documents that it stores, reflecting the users' requirement to organize their knowledge.
- This means that documents contain links (or hyperlinks) – references to other documents and resources that are also stored in the Web.

- The Web is an *open* system: it can be extended and implemented in new ways without disturbing its existing functionality.
    - First, its operation is based on communication standards and document or content standards that are freely published and widely implemented.
    - For example, there are many types of browser, each in many cases implemented on several platforms; and there are many implementations of web servers.
    - Second, the Web is open with respect to the types of resource that can be published and shared on it.
    - At its simplest, a resource on the Web is a web page or some other type of *content* that can be presented to the user, such as media files and documents in Portable Document Format
- The Web has moved beyond these simple data resources to encompass services, such as electronic purchasing of goods. It has evolved without changing its basic architecture.
- The Web is based on three main standard technological components:
    - the **HyperText Markup Language (HTML)**, a language for specifying the contents and layout of pages as they are displayed by web browsers;
    - **Uniform Resource Locators (URLs)**, also known as Uniform Resource Identifiers (URIs), which identify documents and other resources stored as part of the Web;
    - a **client-server system architecture**, with standard rules for interaction (the HyperText Transfer Protocol – **HTTP**) by which browsers and other clients fetch documents and other resources from web servers.
    - Fig. 6 shows some web servers, and browsers making requests to them. It is an important feature that users may locate and manage their own web servers anywhere on the Internet.

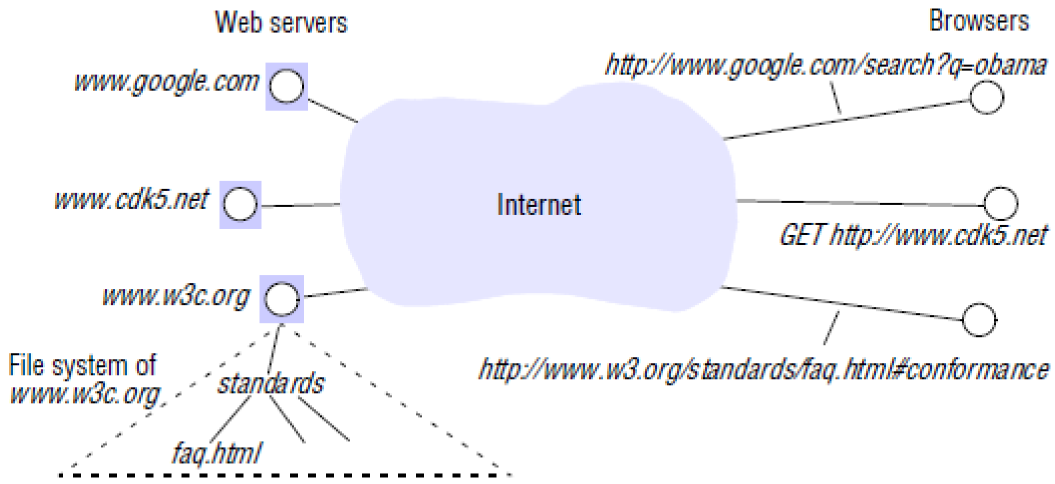

Web servers
Browsers
www.google.com
http://www.google.com/search?q=obama
www.cdk5.net
Internet
GET http://www.cdk5.net
www.w3c.org
http://www.w3.org/standards/faq.html#conformance
File system of www.w3c.org
standards
faq.html

Fig. 6 Web servers and web browsers

HTML

- The HyperText Markup Language is used to specify the text and images that make up the contents of a web page, and to specify how they are laid out and formatted for presentation to the user.
- A web page contains such structured items as headings, paragraphs, tables and images.
- HTML is also used to specify links and which resources are associated with them.
- Users may produce HTML by hand, using a standard text editor, but they more commonly use an HTML-aware 'wysiwyg' editor that generates HTML from a layout that they create graphically.

- A typical piece of HTML text follows:
  <IMG SRC = "http://www.cdk5.net/WebExample/Images/earth.jpg"> 1
  <P> 2
  Welcome to Earth! Visitors may also be interested in taking a look at the 3
  <A HREF = "http://www.cdk5.net/WebExample/moon.html">Moon</A>. 4
  </P> 5
- This HTML text is stored in a file that a web server can access – let us say the file *earth.html*.
- A browser retrieves the contents of this file from a web server – in this case a server on a computer called www.cdk5.net.
- The browser reads the content returned by the server and renders it into formatted text and images laid out on a web page in the familiar fashion.
- Only the browser – not the server – interprets the HTML text.
- But the server does inform the browser of the type of content it is returning, to distinguish it from, say, a document in Portable Document Format. The server can infer the content type from the filename extension '.html'.

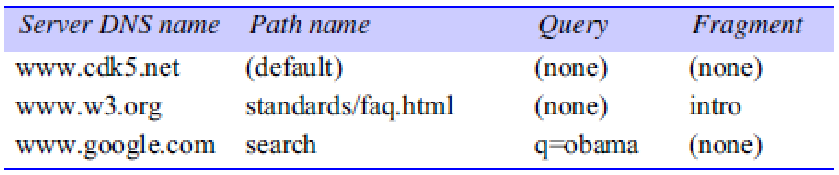
- HTML directives, known as *tags*, are enclosed by angle brackets, such as *<P>*.
- Line 1 of the example identifies a file containing an image for presentation. Its URL is *http://www.cdk5.net/WebExample/Images/earth.jpg*.
- Lines 2 and 5 are directives to begin and end a paragraph, respectively. Lines 3 and 4 contain text to be displayed on the web page in the standard paragraph format.
- Line 4 specifies a link in the web page. It contains the word 'Moon' surrounded by two related HTML tags, *<A HREF...>* and *</A>*.
- The text between these tags is what appears in the link as it is presented on the web page. Most browsers are configured to show the text of links underlined by default, so what the user will see in that paragraph is:

- Welcome to Earth! Visitors may also be interested in taking a look at the Moon. The browser records the association between the link's displayed text and the URL contained in the *<A HREF...>* tag – in this case:
  - *http://www.cdk5.net/WebExample/moon.html*
- When the user clicks on the text, the browser retrieves the resource identified by the corresponding URL and presents it to the user. In the example, the resource is an HTML file specifying a web page about the Moon.

URLs
- The purpose of a Uniform Resource Locator is to identify a resource.
- Browsers examine URLs in order to access the corresponding resources. Sometimes the user types a URL into the browser.
- More commonly, the browser looks up the corresponding URL when the user clicks on a link or selects one of their 'bookmarks'; or when the browser fetches a resource embedded in a web page, such as an image.
- Every URL, in its full, absolute form, has two top-level components:
  scheme : scheme-specific-identifier
- The first component, the 'scheme', declares which type of URL this is. URLs are required to identify a variety of resources.
- For example, *mailto:joe@anISP.net* identifies a user's email address;
- The Web is open with respect to the types of resources it can be used to access, by virtue of the scheme designators in URLs.
- HTTP URLs are the most widely used, for accessing resources using the standard HTTP protocol.
- An HTTP URL has two main jobs: to identify which web server maintains the resource, and to identify which of the resources at that server is required.

- In general, HTTP URLs are of the following form:
  http:// servername [:port] [/pathName] [?query] [#fragment]
  where items in square brackets are optional.
- A full HTTP URL always begins with the string 'http://' followed by a server name, expressed as a Domain Name System (DNS) name.
- The server's DNS name is optionally followed by the number of the 'port' on which the server listens for requests, which is 80 by default.
- Then comes an optional path name of the server's resource. If this is absent then the server's default web page is required.
- Finally, the URL optionally ends in a query component – for example, when a user submits the entries in a form such as a search engine's query page – and/or a fragment identifier, which identifies a component of the resource.
- Consider the URLs:
  http://www.cdk5.net
  http://www.w3.org/standards/faq.html#conformance
  http://www.google.com/search?q=obama

These can be broken down as follows:



Server DNS name	*Path name*	*Query*	*Fragment*
www.cdk5.net	(default)	(none)	(none)
www.w3.org	standards/faq.html	(none)	intro
www.google.com	search	q=obama	(none)

- The first URL designates the default page supplied by *www.cdk5.net*.
- The next identifies a fragment of an HTML file whose path name is *standards/faq.html* relative to the server *www.w3.org*.
- The fragment's identifier (specified after the '#' character in the URL) is *intro*, and a browser will search for that fragment identifier within the HTML text after it has downloaded the whole file.
- The third URL specifies a query to a search engine. The path identifies a program called 'search', and the string after the '?' character encodes a query string supplied as arguments to this program.

Publishing a resource:

- While the Web has a clearly defined model for accessing a resource from its URL, the exact methods for publishing resources on the Web are dependent upon the web server implementation.
- In terms of low-level mechanisms, the simplest method of publishing a resource on the Web is to place the corresponding file in a directory that the web server can access.
- Knowing the name of the server *S* and a path name for the file *P* that the server can recognize, the user then constructs the URL as *http://S/P*.
- The user puts this URL in a link from an existing document or distributes the URL to other users, for example by email.
- Huang provide a model for inserting content into the Web with minimal human intervention.
- This is particularly relevant where users need to extract content from a variety of devices, such as cameras, for publication in web pages.

HTTP

- The HyperText Transfer Protocol defines the ways in which browsers and other types of client interact with web servers.

Request-reply interactions:

- HTTP is a 'request-reply' protocol. The client sends a request message to the server containing the URL of the required resource.

- The server looks up the path name and, if it exists, sends back the resource’s content in a reply message to the client.
- Otherwise, it sends back an error response such as the familiar ‘404 Not Found’.
- HTTP defines a small set of operations or *methods* that can be performed on a resource. Example: GET, to retrieve data from the resource, and POST, to provide data to the resource.

Content types:
- Browsers are not necessarily capable of handling every type of content.
- When a browser makes a request, it includes a list of the types of content it prefers – for example, in principle it may be able to display images in ‘GIF’ format but not ‘JPEG’ format.
- The server may be able to take this into account when it returns content to the browser.
- The server includes the content type in the reply message so that the browser will know how to process it.
- The strings that denote the type of content are called MIME types.
- For example, if the content is of type ‘text/html’ then a browser will interpret the text as HTML and display it; if the content type is ‘application/zip’ then it is data compressed in ‘zip’ format, and the browser will launch an external helper application to decompress it.
- The set of actions that a browser will take for a given type of content is configurable, and readers may care to check these settings for their own browsers.

One resource per request:
- Clients specify one resource per HTTP request. If a web page contains nine images, say, then the browser will issue a total of ten separate requests to obtain the entire contents of the page.
- Browsers typically make several requests concurrently, to reduce the overall delay to the user.

Simple access control:
- By default, any user with network connectivity to a web server can access any of its published resources.
- If users wish to restrict access to a resource, then they can configure the server to issue a ‘challenge’ to any client that requests it.
- The corresponding user then has to prove that they have the right to access the resource, for example, by typing in a password.

Dynamic pages
- Much of the users’ experience of the Web is that of interacting with services rather than retrieving data.
- For example, when purchasing an item at an online store, the user often fills out a *web form* to provide personal details or to specify exactly what they wish to purchase.
- A web form is a web page containing instructions for the user and input widgets such as text fields and check boxes.
- When the user submits the form (usually by pressing a button or the ‘return’ key), the browser sends an HTTP request to a web server, containing the values that the user has entered.
- Since the result of the request depends upon the user’s input, the server has to *process* the user’s input.
- Therefore the URL or its initial component designates a *program* on the server, not a file.
- If the user’s input is a reasonably small set of parameters it is often sent as the *query* component of the URL, using the GET method; alternatively, it is sent as additional data in the request using the POST method.
- For example, a request containing the following URL invokes a program called ‘search’ at www.google.com and specifies a query string of ‘obama’: *http://www.google.com/search?q=obama.*
- That ‘search’ program produces HTML text as its output, and the user will see a listing of pages that contain the word ‘obama’.
- The server returns the HTML text that the program generates just as though it had retrieved it from a file.
- The difference between static content fetched from a file and content that is dynamically generated is transparent to the browser.

- A program that web servers run to generate content for their clients is referred to as a Common Gateway Interface (CGI) program.
- A CGI program may have any application-specific functionality, as long as it can parse the arguments that the client provides to it and produce content of the required type (usually HTML text).
- The program will often consult or update a database in processing the request.

Downloaded code:

- A CGI program runs at the server. Sometimes the designers of web services require some service-related code to run inside the browser, at the user's computer.
- In particular, code written in Javascript is often downloaded with a web page containing a form, in order to provide better-quality interaction with the user than that supported by HTML's standard widgets.
- A Javascript enhanced page can give the user immediate feedback on invalid entries, instead of forcing the user to check the values at the server, which would take much longer.
- Javascript can also be used to update parts of a web page's contents without fetching an entirely new version of the page and re-rendering it.
- These dynamic updates occur either due to a user action (such as clicking on a link or a radio button), or when the browser acquires new data from the server that supplied the web page.
- In the latter case, since the timing of the data's arrival is unconnected with any user action at the browser itself, it is termed *asynchronous*.
- A technique known as *AJAX* (Asynchronous Javascript And XML) is used in such cases.
- An alternative to a Javascript program is an *applet*: an application written in the Java language, which the browser automatically downloads and runs when it fetches a corresponding web page.
- Applets may access the network and provide customized user interfaces.

Web services

- Programs other than browsers can be clients of the Web, too; indeed, programmatic access to web resources is commonplace. HTML is inadequate for programmatic interoperation.
- The Extensible Markup Language (XML) has been designed as a way of representing data in standard, structured, application-specific forms.
- In principle, data expressed in XML is portable between applications since it is *self-describing*: it contains the names, types and structure of the data elements within it.
- For example, XML may be used to describe products or information about users, for many different services or applications.
- In the HTTP protocol, XML data can be transmitted by the POST and GET operations. In AJAX it can be used to provide data to Javascript programs in browsers.
- Web resources provide service-specific operations. For example, in the store at amazon.com, web service operations include one to order a book and another to check the current status of an order.
- HTTP provides a small set of operations that are applicable to any resource. These include principally the GET and POST methods on existing resources, and the PUT and DELETE operations, respectively for creating and deleting web resources.
- Any operation on a resource can be invoked using one of the GET or POST methods, with structured content used to specify the operation's parameters, results and error responses.
- The REST (REpresentational State Transfer) architecture for web services adopts this approach on the basis of its extensibility: every resource on the Web has a URL and responds to the same set of operations, although the processing of the operations can vary widely from resource to resource.
- The flip-side of that extensibility can be a lack of robustness in how software operates

Discussion of the Web

- The Web's phenomenal success rests upon the relative ease with which many individual and organizational sources can publish resources, the suitability of its hypertext structure for organizing many types of information, and the openness of its system architecture.
- The standards upon which its architecture is based are simple and they were widely published at an early stage. They have enabled many new types of resources and services to be integrated.

- The Web's success belies some design problems. First, its hypertext model is lacking in some respects. If a resource is deleted or moved, so-called 'dangling' links to that resource may still remain, causing frustration for users.
- And there is the familiar problem of users getting 'lost in hyperspace'. Users often find themselves confused, following many disparate links, referencing pages from a disparate collection of sources, and of dubious reliability in some cases.

- Search engines are a highly popular alternative to following links as a means of finding information on the Web, but these are imperfect at producing what the user specifically intends.
- One approach to this problem, exemplified in the Resource Description Framework, is to produce standard vocabularies, syntax and semantics for expressing metadata about the things in our world, and to encapsulate that metadata in corresponding web resources for programmatic access.
- Rather than searching for words that occur in web pages, programs can then, in principle, perform searches against the metadata to compile lists of related links based on semantic matching.
- Collectively, the web of linked metadata resources is what is meant by the *semantic web.*

- As a system architecture the Web faces problems of scale. Popular web servers may experience many 'hits' per second, and as a result the response to users can be slow.