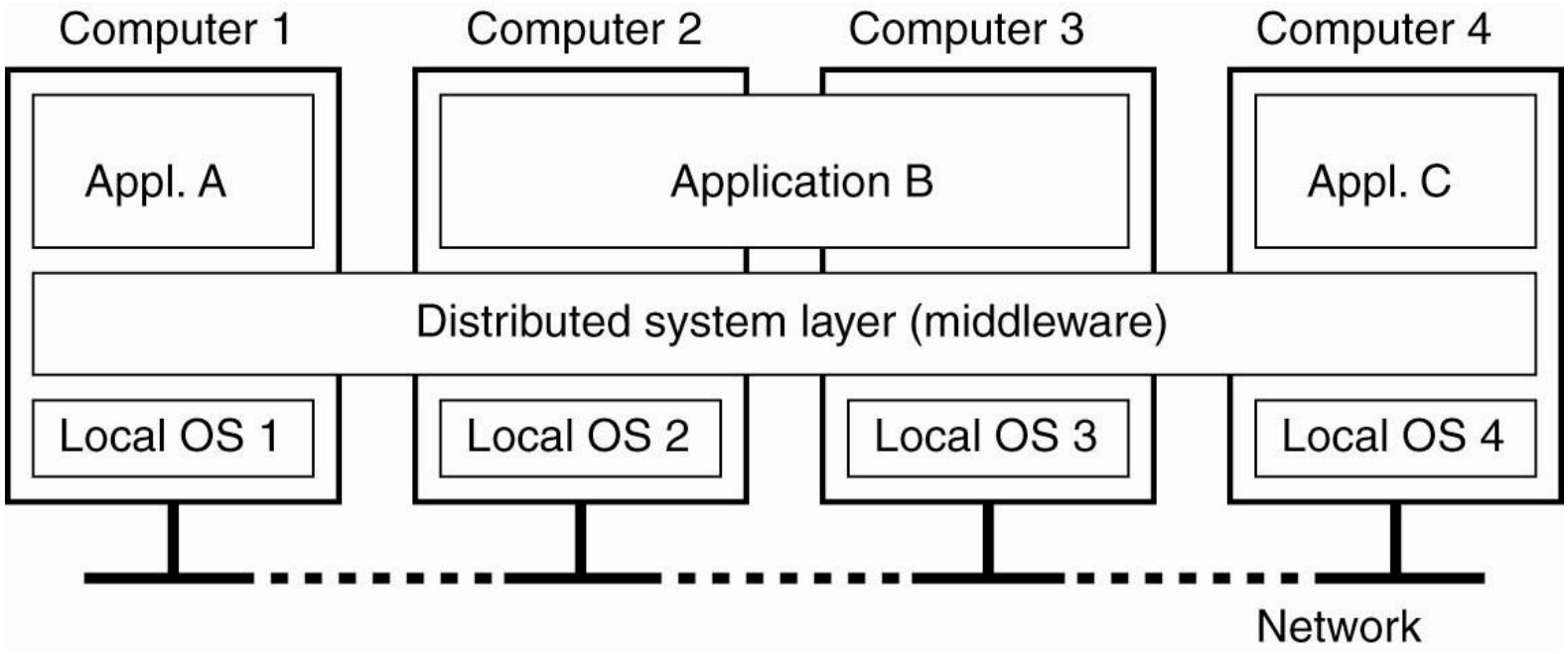
Distributed Computing

Challenges of Distributed Computing

Components of Distributed Software Systems



- Distributed systems
- Middleware
- Distributed applications

Challenges(Differences from Local Computing)

- Heterogeneity:
  - The Internet comprises an heterogeneous collection of computers (hardware and operating systems), networks, programming languages, databases, implementations by different vendors etc.
  - Although the Internet comprises different networks, their differences are masked by the fact that all computers attached to these networks use the IP protocol (and TCP or UDP at the transport layer) to communicate with each other.
  - Different programming languages have different representations for characters and other data structures.
  - The concept of a **virtual machine** provides a way of making code executable on any hardware. The compiler generates code for a virtual machine rather than for a specific processor. E.g. the Java compiler generates code for the Java Virtual Machine (JVM).

Challenges(Differences from Local Computing)

- Openness
  - The openness of the distributed system determines whether the system is extensible and re-implemented in various ways. Openness cannot be achieved until the key APIs of the components of a system are published and made available to software developers.
  - E.g. the DOM API for XML documents and other Java APIs such as JDBC, RMI, JNDI are all open source.

SCALABILITY

- A system is described as scalable if it will remain effective when there is a significant increase in the number of resources and users. e.g. the Internet is one such distributed system.

- Challenges in the designing of scalable distributed systems include:
  - *Controlling the cost of physical resources:* for a system with n users to be scalable, the quantity/cost of physical resources to support them must be at most O(n), i.e., proportional to n. So if a single file server can support 20 users, 2 such servers should support 40 users.
  - *Controlling the performance loss:* algorithms that use hierarchic storage structures (e.g. LDAP) scale better than those that use linear (e.g. flat file data tables) structures. Even with hierarchic structures an increase in size (of data) will result in some performance loss since the the time taken to access hierarchically structured data is O(log n), where n is the size of the data set.
  - *Preventing software resources running out*: lack of scalability is shown by the 32-bit IP addresses. The new version (IP v6) will use 128 bit addresses.
  - *Avoiding performance bottlenecks:* algorithms should be decentralized to avoid performance bottlenecks. E.g. the name table in DNS was originally kept in a single master file. This became a performance bottleneck. It was then partitioned between DNS servers located throughout the Internet and administered locally. Caching and replication can also improve performance of resources that are heavily used.

Challenges(Differences from Local Computing)

- Security
  - Security of information in distributed systems has three components:
    - **Confidentiality:** protection against disclosure to unauthorized individuals.
    - **Integrity:** protection against alteration or corruption.
    - **Availability:** protection against interference that prevents access to the resources.
  - E.g. sending credit card numbers over the Internet in an E-commerce application.
  - Security involves encryption and authentication. Two other security challenges include
  - **Denial of service attacks**.

Failure Handling

Failures in distributed systems are partial – some components fail while others continue to function. Hence failure handling can be difficult. Techniques can be used to:

- Detect failures: E.g. use checksums to detect corrupt data in a file/message. Other times failure can only be suspected (e.g. a remote server is down) but not detected and the challenge is to manage in the presence of such failures.
- Mask Failures: Some failures that have been detected can be masked/hidden or made less severe. E.g. messages can be retransmitted when then fail to be acknowledged. This might not help if the network is severely congested and in this case even the retransmission may not get through before timeout. Another e.g. File data can be written to a pair of disks so that if one is corrupted, the other may still be correct (redundancy to achieve fault-tolerance).
- Tolerate Failures: Most of the services on the Internet exhibit failures and it is not practical to detect or mask all the possible kinds of failures. In such cases, clients can be designed to tolerate failures. E..g. a web browser cannot reach a web server it does not make the user wait forever. It gives a message indicating that the server is unreachable and the user can try later.
- Recovery from failures: This involves the design of software so that the state of permanent data can be recovered or “rolled back” after a server has crashed. E.g. database servers have a transaction handling ability that enables them to roll back a transaction that was not completed.

Computers in the Internet

Date	*Computers*	*Web servers*
1979, Dec.	188	0
1989, July	130,000	0
1999, July	56,218,000	5,560,866
2003, Jan.	171,638,297	35,424,956

Computers vs. Web servers in the Internet

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		42,298,371	

Transparency in Distributed Systems

- ***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).

Access and location transparency are the most important, and are collectively called *network transparency.* E.g. of access transparency is using the same API to access both local and remote files (NFS). An e.g. of lack of this is using FTP to access files.

An email address such as name@server.com is an example of network transparency since it provides both access (address of another person is of same structure whether local or remote) and location transparency (don't know the other person's physical or network location).

Transparency in Distributed Systems

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

There is a possibility that several clients will attempt to access a shared resource at the same time. Servers in a distributed environment tend to be concurrent servers rather than iterative in order to increase throughput (clients serviced per sec). In many cases this involves having a new thread service each client request. It must be ensured that concurrent access to objects in a distributed application be safe by using appropriate synchronization techniques

- ***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.

Transparency in Distributed Systems

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

Failure transparency is exhibited by email which keeps attempting to deliver mail so the faults are masked.

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

Cell phones exhibit mobility transparency since the caller and called are not aware of the mobility of their phones.

Web URLs are location-transparent since the name refers to a computer domain name rather than an IP address. But URLs are not mobility transparent since your web page will not move to your new work place since all the links in other pages will still point to the original page. It also prevents replication transparency because even though DNS allows a domain name to refer to several computers, it picks one of them when it looks up a name (need to be able to access all of the participating computers).