CMPU4021 Distributed Systems

Architectural Models for Distributed Systems

Introduction

- Architectural model of a distributed system

  - Concerned with hierarchy/placement of component parts and relationships, e.g., Client-Server (C-S) and Peer-Process models.

  – Define the layer/levels of components, placement, and manner of interactions

  – Well described mapping of components/functions onto underlying physical network architecture (of computers)

Introduction

- The lecture focuses on

  – Variants of architectural models – layered, C-S, aspects of DS for being open systems

  – Fundamental models - abstracting models to reveal

    - design issues,
    - properties,
    - difficulties,
    - correctness,
    - reliability,
    - security, and variations in performance requirements,
    - issues of heterogeneity (h'ware, s'ware, network types, clocks, data types/formats, integrity, QoS)

Architectural models

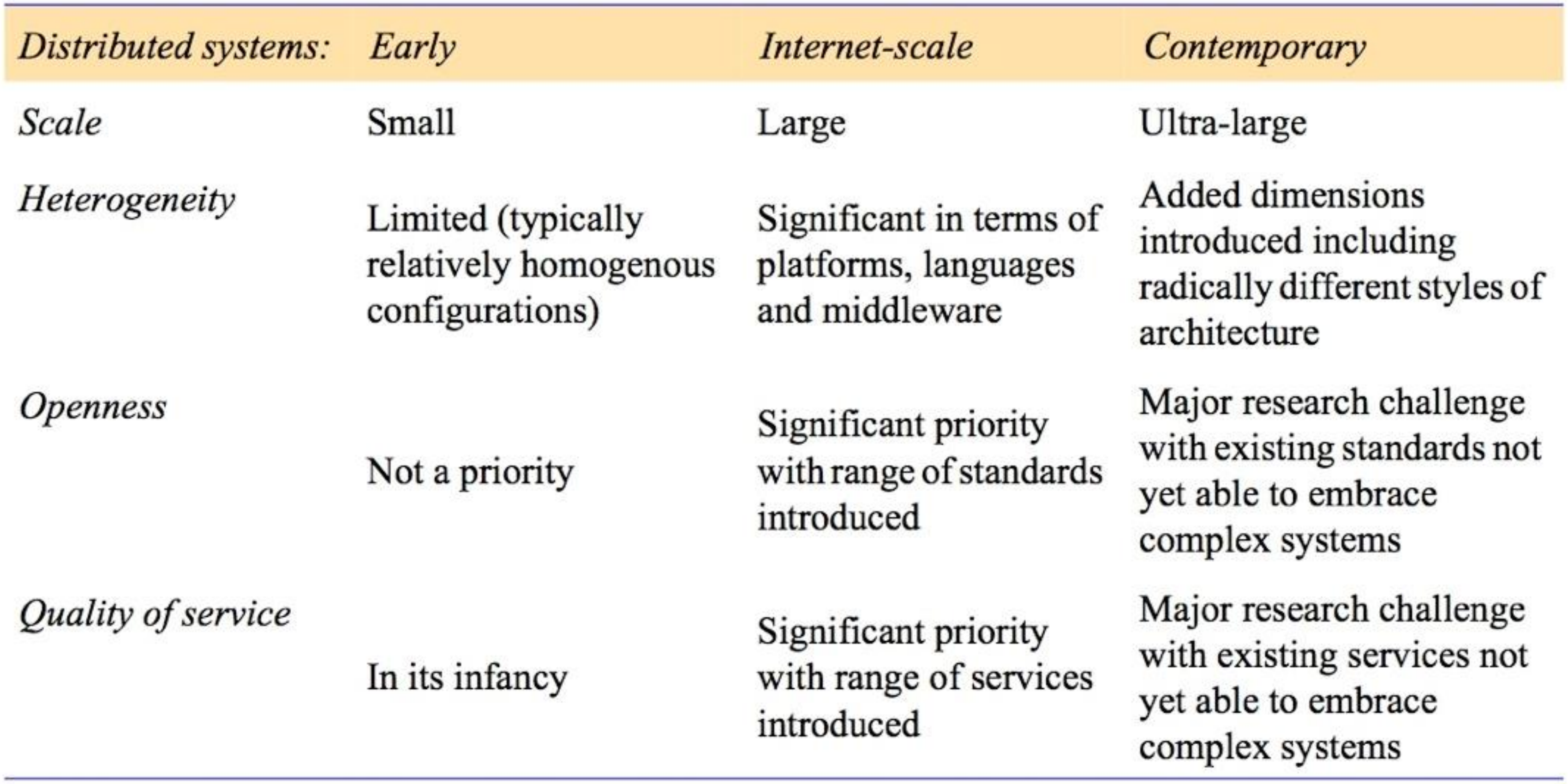
- *Architecture*
  - structure which defines the placement of system components, to guarantee reliability, manageability, adaptability, and cost-effectiveness
  - a consistent frame of reference

- An architectural model of a distributed system first simplifies and abstracts the functions of the individual components of a distributed system and then it considers:
  - the placement of the components across a network of computers
    - seeking to define useful patterns for the distribution of data and workload;
  - The interrelationships between the components
    - their functional roles and the patterns of communication between them.
- In most real-world distributed systems, many different styles are combined.

Architectural Models (I)

- An initial simplifications is achieved by classifying processes as
  - *server process*,
  - *client process* and
  - *peer processes*
    - processes that cooperate and communicate in a symmetrical manner

Generations of distributed systems



Distributed systems:	*Early*	*Internet-scale*	*Contemporary*
Scale	Small	Large	Ultra-large
Heterogeneity	Limited (typically relatively homogenous configurations)	Significant in terms of platforms, languages and middleware	Added dimensions introduced including radically different styles of architecture
Openness	Not a priority	Significant priority with range of standards introduced	Major research challenge with existing standards not yet able to embrace complex systems
Quality of service	In its infancy	Significant priority with range of services introduced	Major research challenge with existing services not yet able to embrace complex systems

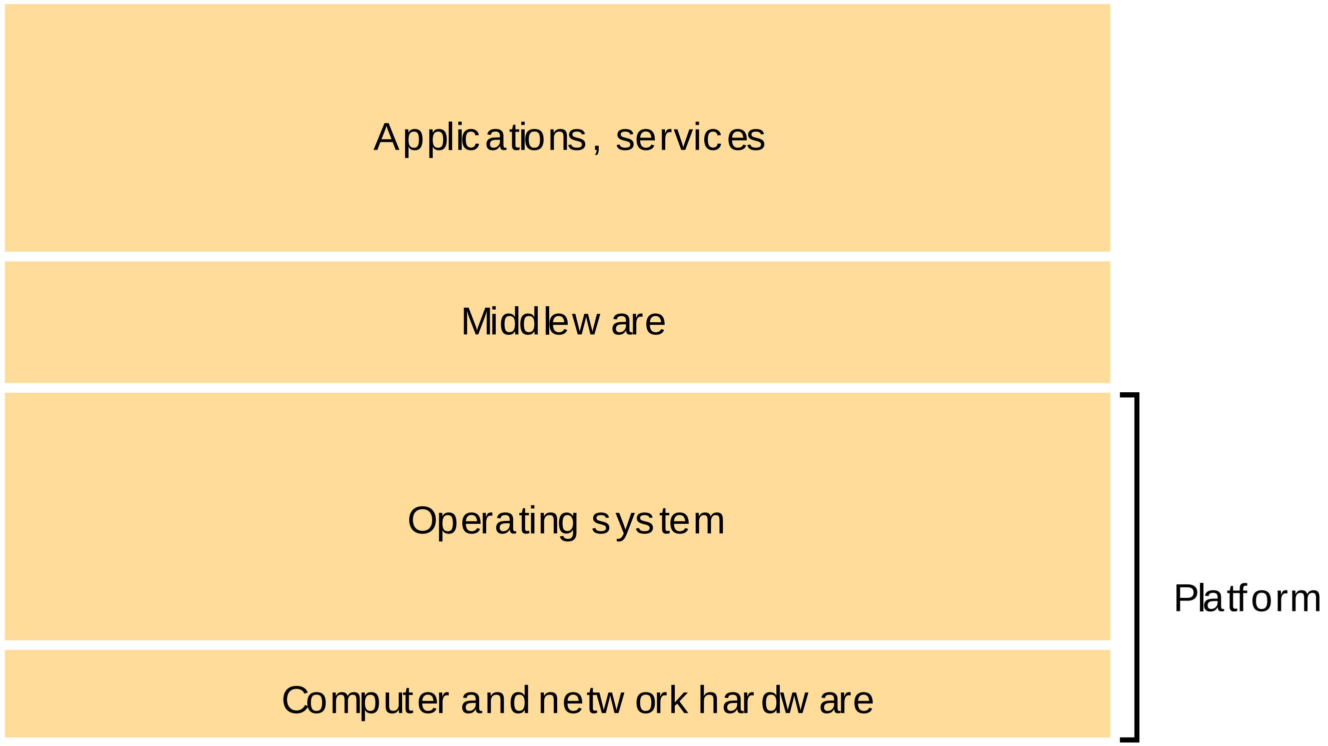
Software Architecture

- Layers of software modules in the same or different computers, and the manner of interactions among modules and services each provides and accessibility

- Each layer/module offers a kind of services.
  - E.g., the Internet has Network Time Protocol, used by server processes solely responsibility for reading and sending ‘LocalHostTime’ info to clients on request

- This process and service-oriented view can be expressed in terms of *service layers*.

Software and hardware service layers in distributed systems



Platform

- **Platform**
  - The lowest-level hardware and software layers, which provide services to the upper Middleware and Application Layers.

- The design and implementation of the platform is independent of the design and implementation of the upper layers
  - the ‘glue’ or interface between the Platform and ‘upper’ layers is realized through systems programming of Application Programming Interfaces (APIs) or Bundles.

Middleware

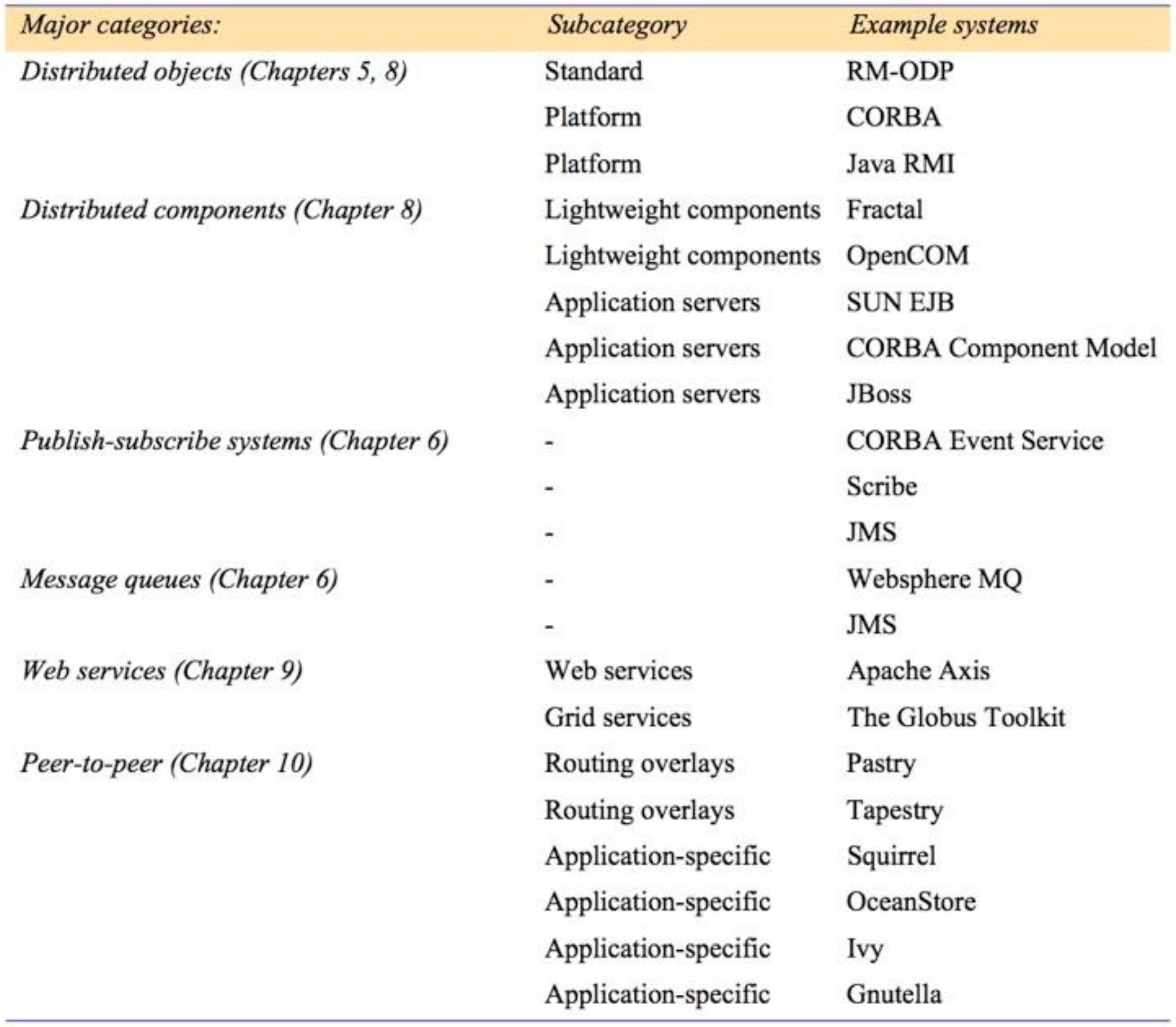
- Layer of software, which masks heterogeneity in DS, and facilitates API programming by application programmers

- It is represented by processes and objects on ‘service providing’ computers for communication and resource (files/data, code, device) sharing.

- Provides building blocks (classes, objects, interfaces, library modules) for constructing DS components

- It abstracts device-level communication requirements via, e.g., Remote Method Invocation (RMI), Remote Procedure Calling (RPC), request/send, notify, selective or group communication, broadcast, protection/security, data replication/integrity

Categories of middleware (Figure 2.12 in the book)



Major categories:	*Subcategory*	*Example systems*
Distributed objects (Chapters 5, 8)	Standard	RM-ODP
	Platform	CORBA
	Platform	Java RMI
Distributed components (Chapter 8)	Lightweight components	Fractal
	Lightweight components	OpenCOM
	Application servers	SUN EJB
	Application servers	CORBA Component Model
	Application servers	JBoss
Publish-subscribe systems (Chapter 6)	-	CORBA Event Service
	-	Scribe
	-	JMS
Message queues (Chapter 6)	-	Websphere MQ
	-	JMS
Web services (Chapter 9)	Web services	Apache Axis
	Grid services	The Globus Toolkit
Peer-to-peer (Chapter 10)	Routing overlays	Pastry
	Routing overlays	Tapestry
	Application-specific	Squirrel
	Application-specific	OceanStore
	Application-specific	Ivy
	Application-specific	Gnutella

Tiered architecture

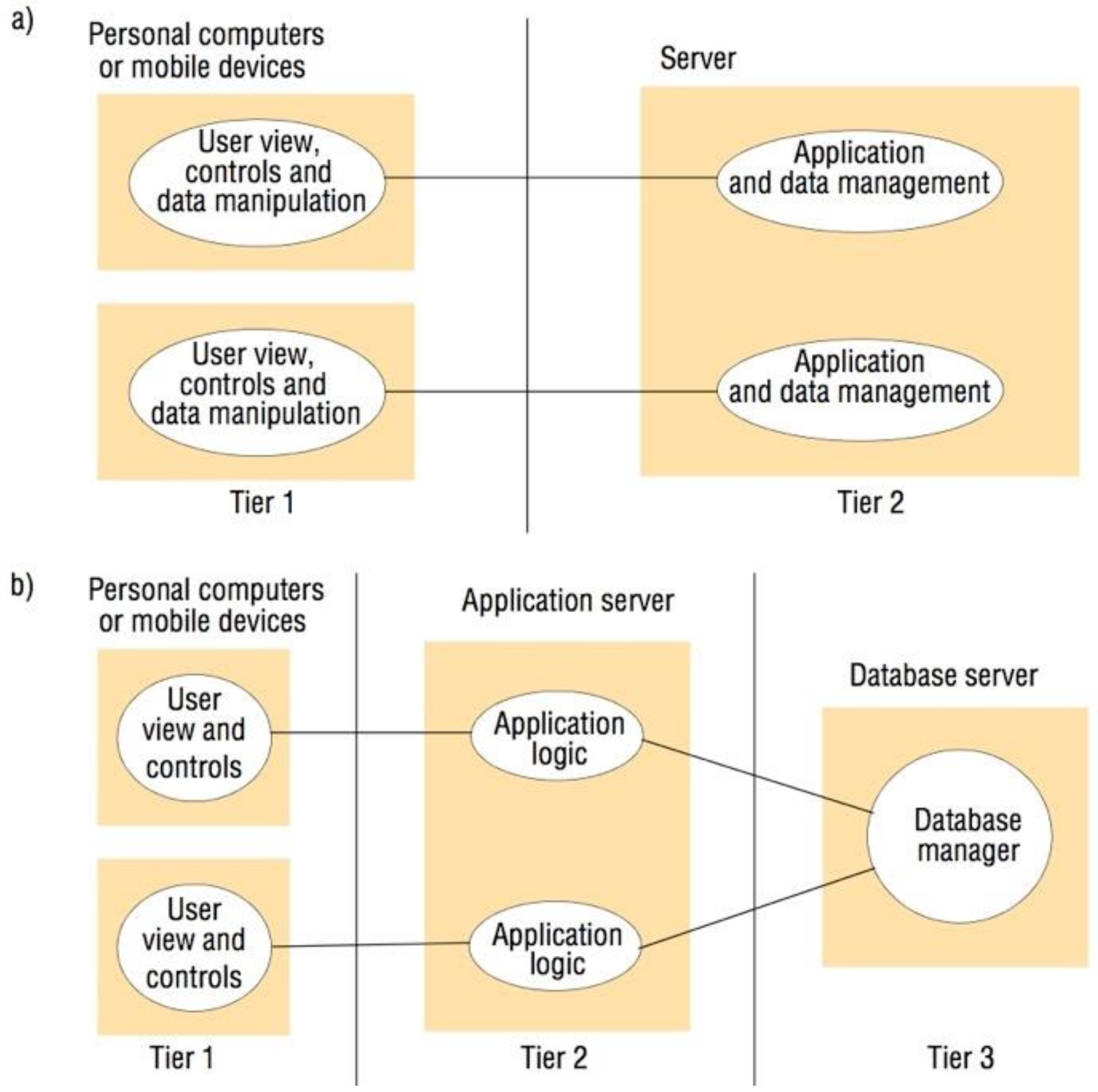
- Tiered architectures are complementary to layering.
- Layering deals with the vertical organization of services into layers of abstraction
- Tiering is a technique to organize functionality of a given layer and place this functionality into
  - appropriate servers and, as a secondary consideration, on to
  - physical nodes.
- This technique is most commonly associated with the organization of applications but it also applies to all layers of a distributed systems architecture.

Tiered architecture

Consider the functional decomposition of a given application, as follows:

- The presentation logic
  - concerned with handling user interaction and updating the view of the application as presented to the user
- The application logic
  - concerned with the detailed application-specific processing associated with the application (also referred to as the business logic, although the concept is not limited only to business applications)
- The data logic
  - concerned with the persistent storage of the application, typically in a database management system

Two-tier and three-tier architectures


a)
Personal computers or mobile devices
Server
User view, controls and data manipulation
Application and data management
User view, controls and data manipulation
Application and data management
Tier 1
Tier 2
b)
Personal computers or mobile devices
Application server
Database server
User view and controls
Application logic
Database manager
User view and controls
Application logic
Tier 1
Tier 2
Tier 3

Multi-tier architectures

- n-tiered (or multi-tier) solutions
- Application domain is partitioned into n logical elements, each mapped to a given server element.
- Example: Wikipedia, the web-based publicly editable encyclopaedia, adopts a multi-tier architecture to deal with the high volume of web requests
  - up to 60,000 page requests per second.

System architectures

- Architectural design has a major impact on performance, reliability, and security. In a distributed system, processes with well-defined responsibilities interact with each other to perform a *useful* activity.

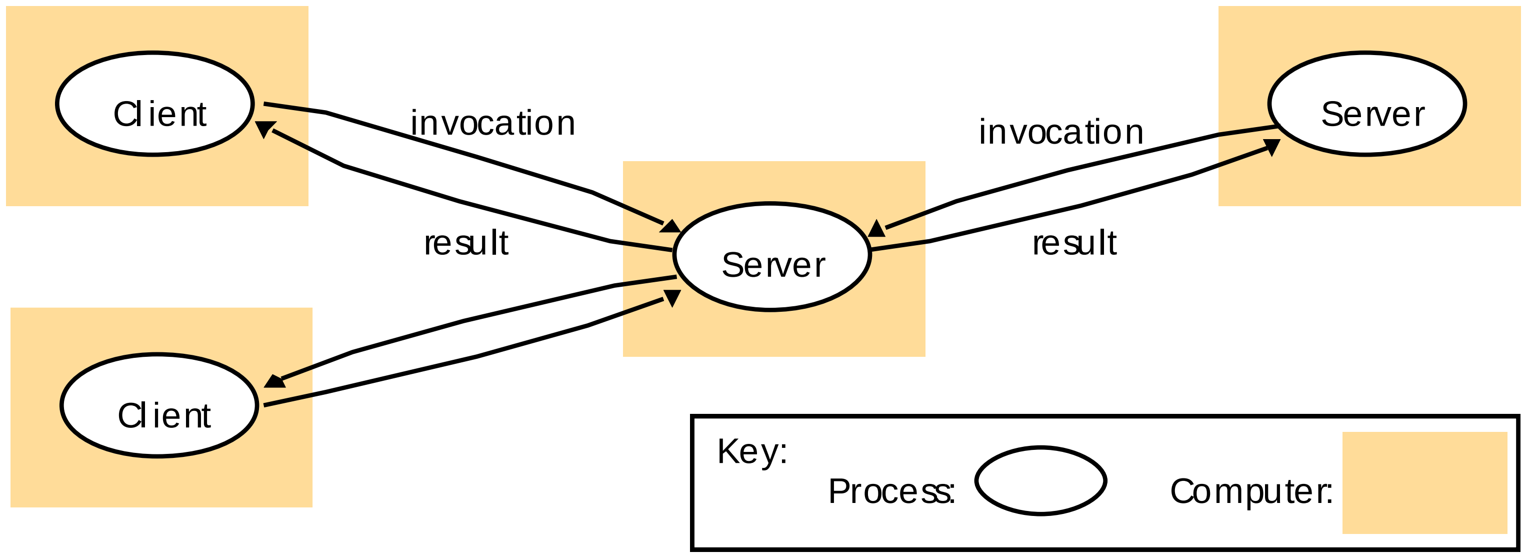
- Main types of architectural models:
  1. The C-S model
  2. The Multi-Server model
  3. The Proxy Servers and Caches model
  4. The Peer Process model

- Variations on the C-S model:
  5. Mobile agents model
  6. Thin client model

The Client-Server (C-S) model

- Widely used, servers/clients on different computers provide services to clients/servers on different computers via request/reply messaging.

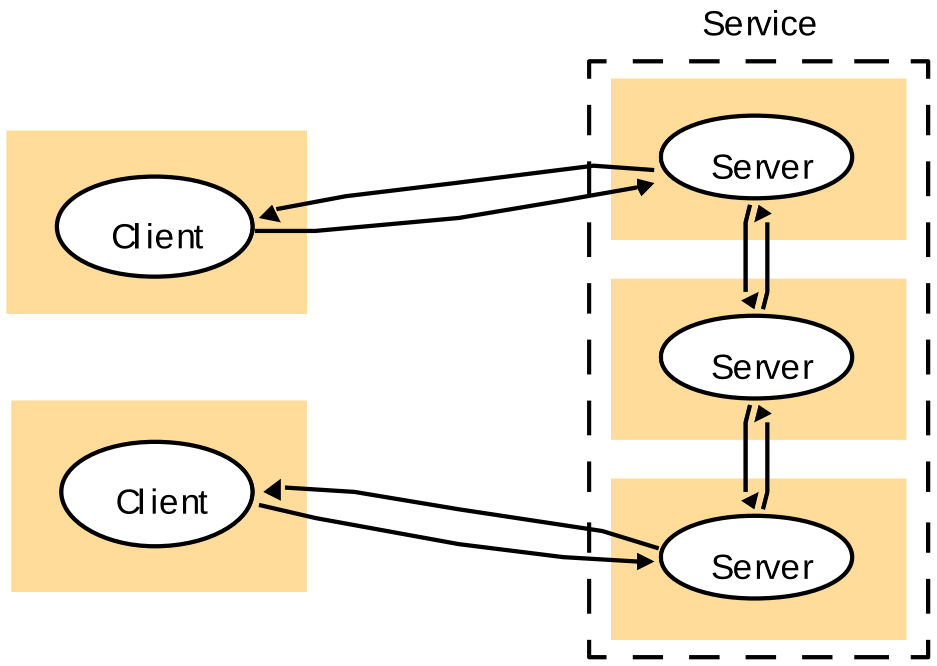
- Servers could also become clients in some services, and vice-versa, e.g., for web servers and web pages retrievals, DNS resolution, search engine-servers and web ‘crawlers’, which are all independent, concurrent and asynchronous (synchronous?) processes.



The Multi-Server model

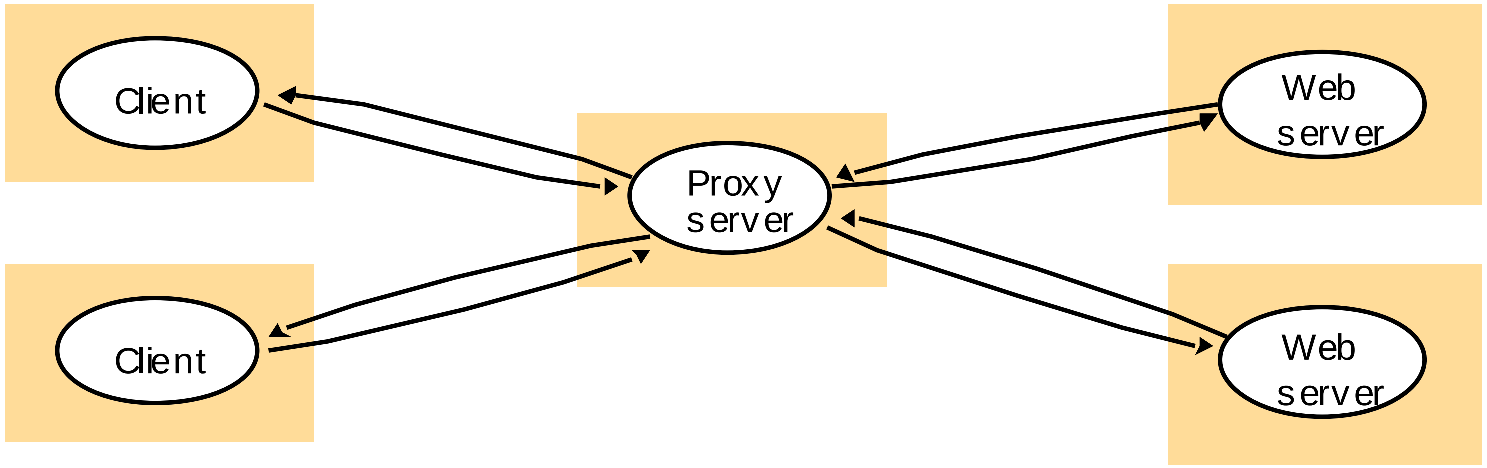
- A DS with multiple, interacting servers responding to parts of a given request in a cooperative manner.

- Service provision is via the partitioning and distributing of object sets, data replication, (or code migration).
  - E.g., A browser request targeting multiple servers depending on location of resource/data OR replication of data at several servers to speed up request/reply turnaround time, and guarantee availability and fault tolerance – consider the Network Information Service (NIS) replication of network login-files for user authorization.
  - Replication is used to increase performance and availability and to improve fault tolerance. It provides multiple consistent copies of data in processes running in different computers.



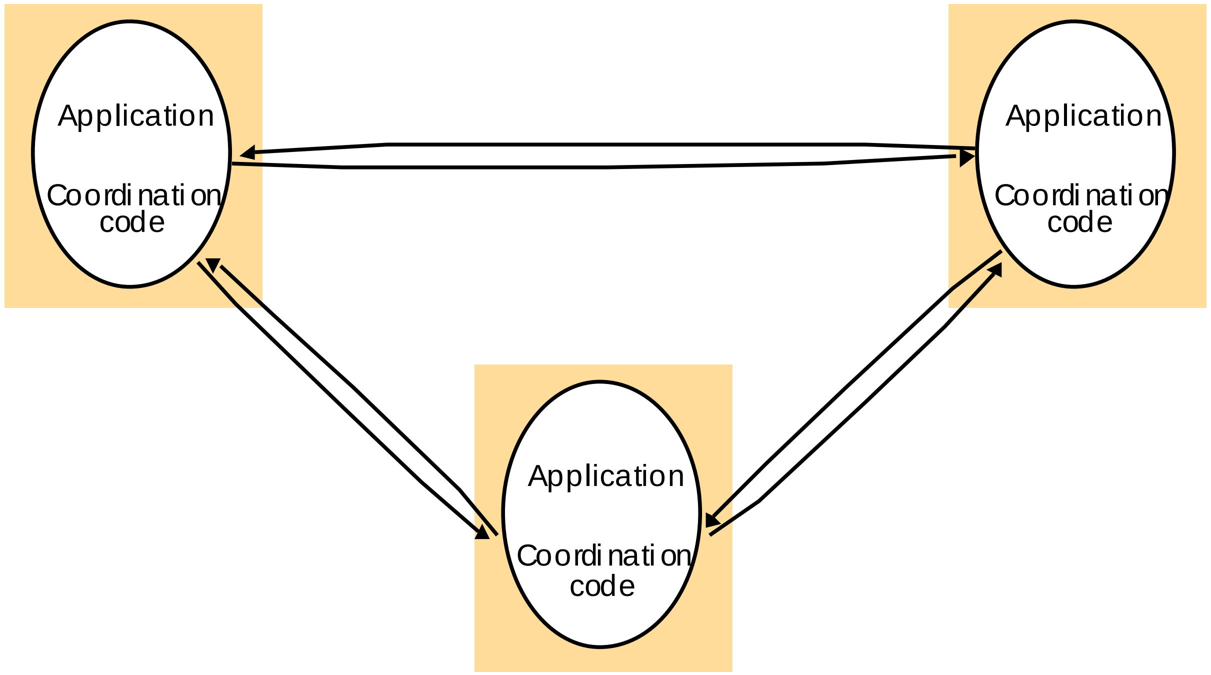
The Proxy Servers and Caches model

- A *cache* is a store of recently used data objects that is closer than the objects themselves.
- Caching frequently used: objects/data/code, which can be collocated at all clients, or located at a single/multiple 'proxy' server(s) and accessed/shared by all clients.
- When requested object/data/code is not in cache is it fetched or, sometimes, updated. E.g., clients caching of recent web pages.
- Web proxy servers provide a shared cache of web resources for the client machines at a site or across several sites. The purpose of proxy servers is to increase availability and performance of the service by reducing the load on the wide-area network and web servers. Proxy servers can take on other roles: e.g., they may be used to access remote web servers through a firewall.



The Peer Process model

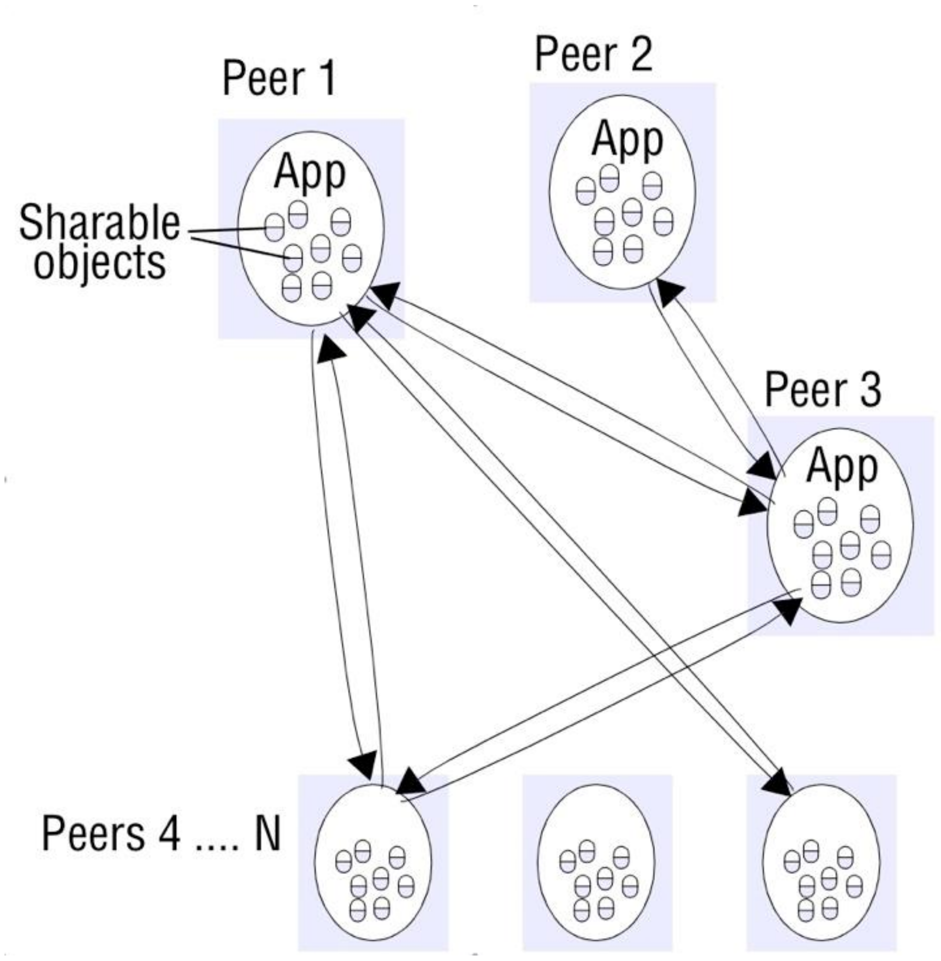
- Without any distinction between servers and clients.
- All processes interact and cooperate in servicing requests.
- Processes are able to maintain consistency and needed synchronization of actions; and pattern of communication depends on the application.
- E.g., consider a 'whiteboard' application where multiple peer processes interact to modify a shared picture file – interactions and synch done via middleware layer for notification/group comm.



Peer-to-peer systems characteristics

- Their design ensures that each user contributes resources to the system.
- They may differ in the resources that they contribute, but all the nodes in a peer-to-peer system have the same functional capabilities and responsibilities.
- Their correct operation does not depend on the existence of any centrally administered systems.
- They can be designed to offer a limited degree of anonymity to the providers and users of resources.
- A key issue for their efficient operation
  - the choice of an algorithm for the placement of data across many hosts and subsequent access to it in a manner that balances the workload and ensures availability without adding undue overheads.

Peer-to-peer architecture



Peer-to-peer: issues to handle

- Connectivity
  - how to find and connect other P2P nodes that are running in the network
    - unlike traditional servers, they don't have a fixed, known IP address
- Instability
  - nodes may always be joining and leaving the network
- Message routing
  - how messages should be routed to get from one node to another
    - the two nodes may not directly know about each other
- Searching
  - how to find desired information from the nodes connected to the network
- Security - extra issues including
  - nodes being able to trust other nodes,
  - preventing malicious nodes from doing corrupting the P2P network or the individual nodes,
  - being able to send and receive data anonymously, etc.

Peer-to-peer systems

Three generations of peer-to-peer system and application development can be identified:

- The *first generation* was launched by the Napster music exchange service
- A *second generation* of files sharing applications offering greater scalability, anonymity and fault tolerance followed:
  - Freenet, Gnutella, Kazaa and BitTorrent
- The *third generation* - middleware layers for the application-independent management of distributed resources on a global scale.
  - Designed to place resources (data objects, files) on a set of computers that are widely distributed throughout the Internet and to route messages to them on behalf of clients,
    - relieving clients of any need to make decisions about placing resources and to hold information about the whereabouts of the resources they require.
    - Examples include: Pastry, Tapestry, CAN, Chord and Kademlia

Mobile agents model

- Both code and associated data are migrated to a number of computers to carry out specified functions/tasks, and eventually returning results
  - A variant of the C-S model

- It tends to minimize delays due to communication (vis-à-vis static clients making multiple requests to servers).
  - E.g., a software installation-agent installing applications on different computers for given hardware-configs; or price compare-agent checking variations in prices for a commodity; or a worm agent that looks for idle CPU cycles in cluster-computing.

- Caution: security threat due to potential ‘Trojan Horse’ problem in migrated code, incomplete exec or ‘hanging’ of agents.

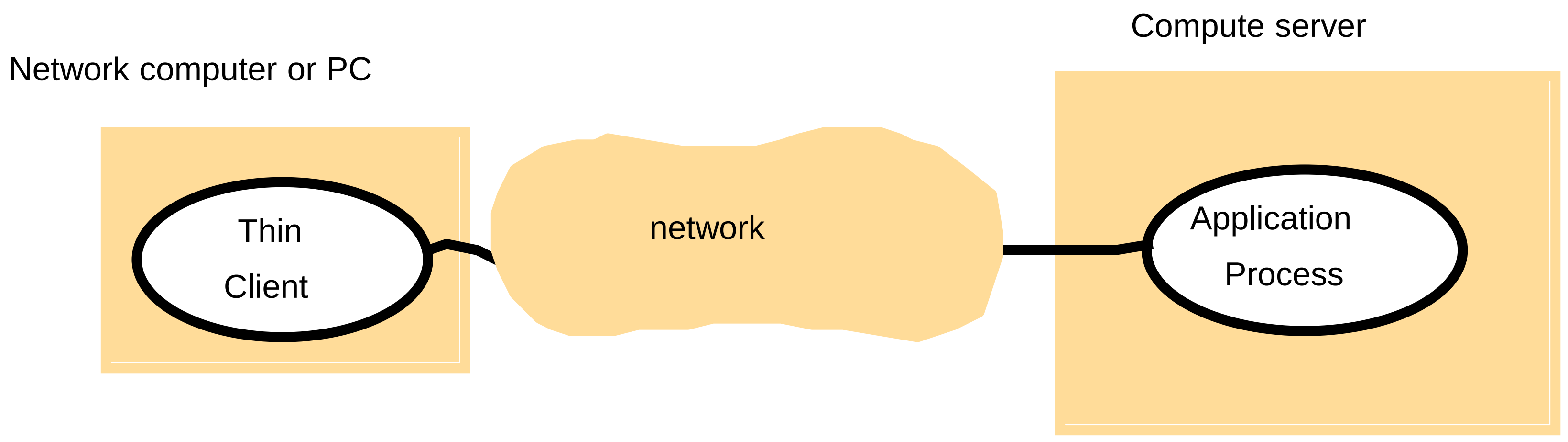
- The mobile agents may not be able to complete their tasks if they are refused access to the information they need.

Thin Client model

- Each client computer supports a layer of software which invokes a remote *compute server* for computational services.
  - A variant of the C-S/Network computer model,

- The compute server will typically be a multiprocessor or cluster computer.

- If the application is interactive and results are due back to client-user, delays and communication can eclipse any advantages.



Thin Client model

- Drawbacks:
  - delays experienced by users can be increased to unacceptable levels by the need to transfer images and vector information between the thin client and the application process,
    - due to both network and operating system latencies.

- Thin client concept has led to the emergence of virtual network computing (VNC)
  - providing remote access to graphical user interfaces
  - a VNC client (or viewer) interacts with a VNC server through a VNC protocol.
  - Examples:
    - RealVNC, Apple Remote Desktop, TightVNC, Aqua Connect

Resource-based architectures: RESTful architectures

- View a distributed system as a collection of resources, individually managed by components.

- Resources may be added, removed, retrieved, and modified by (remote) applications.

Resource-based architectures

1. Resources are identified through a single naming scheme
2. All services offer the same interface
3. Messages sent to or from a service are fully self-described
4. After executing an operation at a service, that component forgets everything about the caller

Resource-based architectures

Basic operations:

- PUT - Create a new resource
- GET - Retrieve the state of a resource in some representation
- DELETE - Delete a resource
- POST - Modify a resource by transferring a new state

Resource-based architectures

Example: Amazon's Simple Storage Service (S3):

- Objects (i.e., files) are placed into buckets (i.e., directories). Buckets cannot be placed into buckets. Operations on `ObjectName` in bucket `BucketName` require the following identifier:

  `http://BucketName.s3.amazonaws.com/ObjectName`

- Typical operations:
  - All operations are carried out by sending HTTP requests:
  - Create a bucket/object: PUT, along with the URI
  - Listing objects: GET on a bucket name
  - Reading an object: GET on a full URI

Publish-subscribe architectures

- Dependencies between processes as loose as possible
- Separation between processing and coordination
- View a system as a collection of autonomously operating processes.
- **Coordination** encompasses the communication and cooperation between processes.

Summary

- Most distributed systems are arranged according to one of a variety of architectural models.
- The client-server model is prevalent – the Web and other Internet services such as ftp, news and mail , DNS are based on C-S model.
- Services such as DNS that have large numbers of users and manage a vast amount of information are based on multiple servers and use data partition and replication to enhance availability and fault tolerance.
- Caching by clients and proxy servers is widely used to enhance the performance of service.

References

- Chapter 2: Coulouris, Dollimore and Kindberg, Distributed Systems: Concepts and Design, 5/E

- Chapter 2: Maarten van Steen, Andrew S. TanenbaumDistributed Systems, 3rd edition (2017)