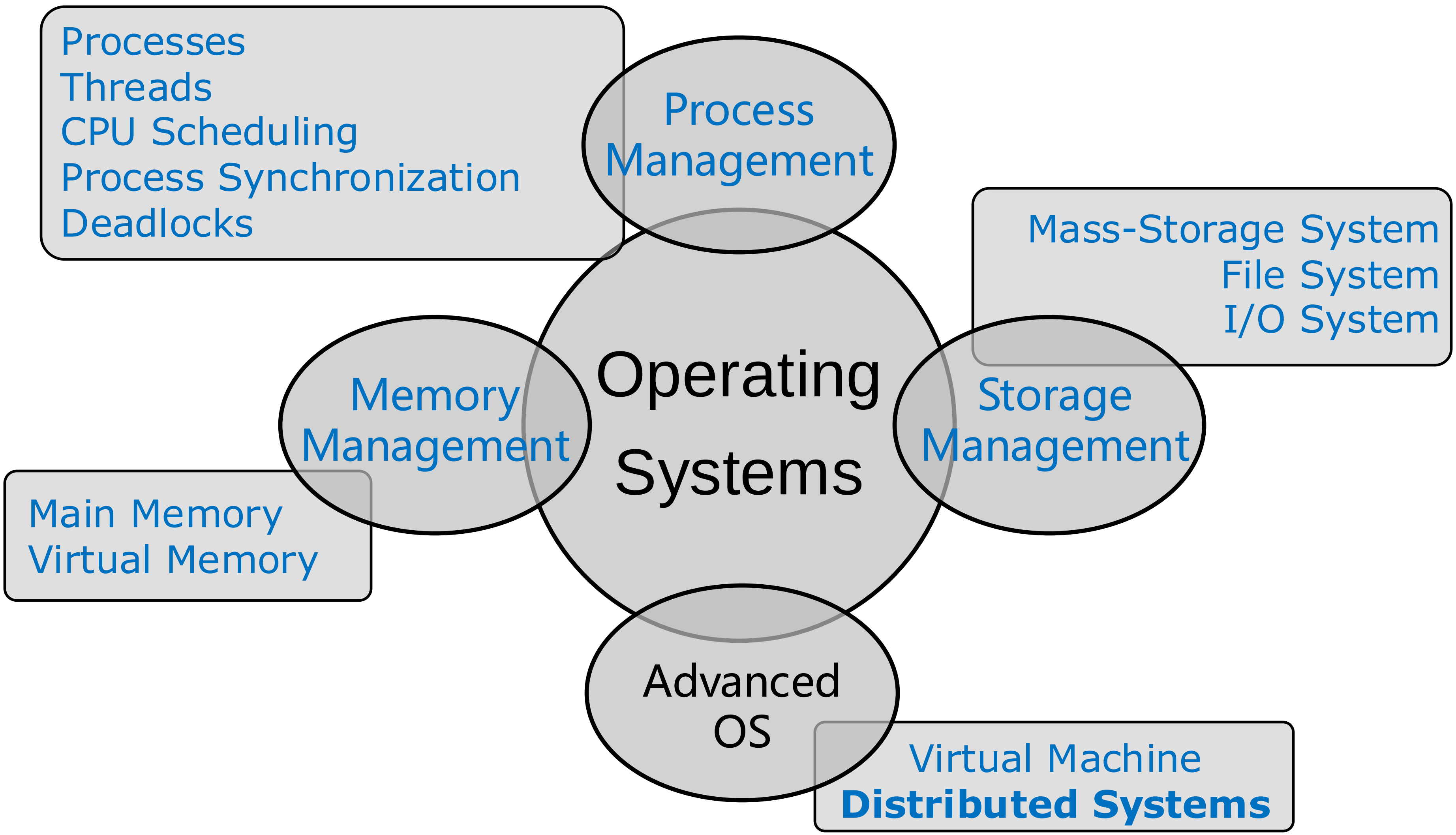
Distributed Systems

Shengzhong Liu
Department of Computer Science and Engineering
Shanghai Jiao Tong University

Operating System Topics



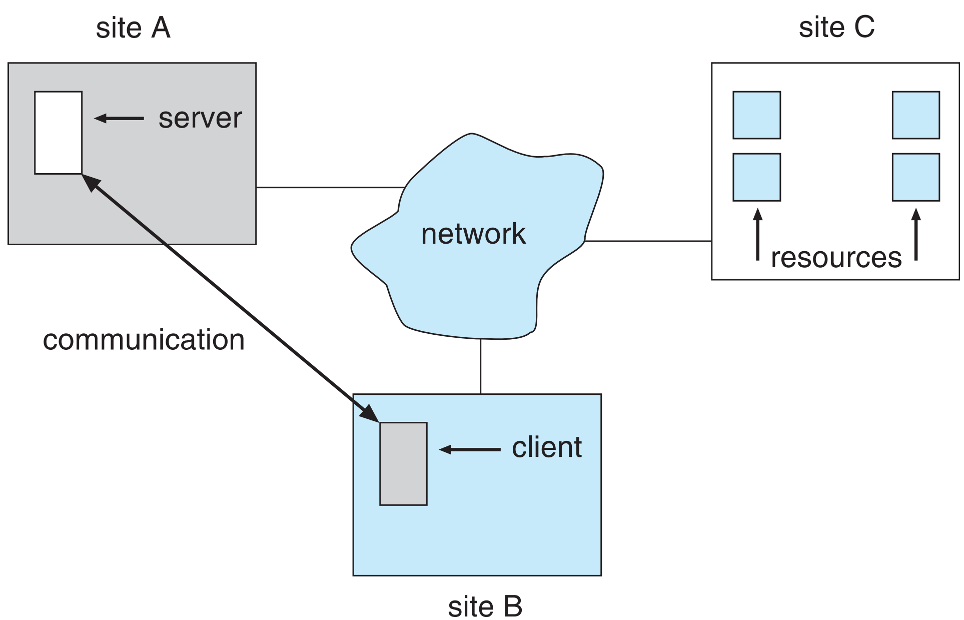
Outline

- Overview
- Network and Distributed Operating Systems
- Design Issues of Distributed Systems
- Distributed File Systems
- Distributed File System Challenges

Overview

Overview

- A **distributed system** is a collection of loosely coupled nodes interconnected by a communications network
- Nodes variously called **processors, computers, machines, hosts**
  - **Site** is location of the machine, **node** refers to specific system



Overview (Cont.)

- Nodes may exist in a **client-server**, **peer-to-peer**, or **hybrid** configuration.
  - In **client-server configuration**, server has a resource that a client would like to use
  - In **peer-to-peer configuration**, each node shares equal responsibilities and can act as both clients and servers
- Communication over a network occurs through **message passing**
  - All higher-level functions of a standalone system can be expanded to encompass a distributed system

Reasons for Distributed Systems

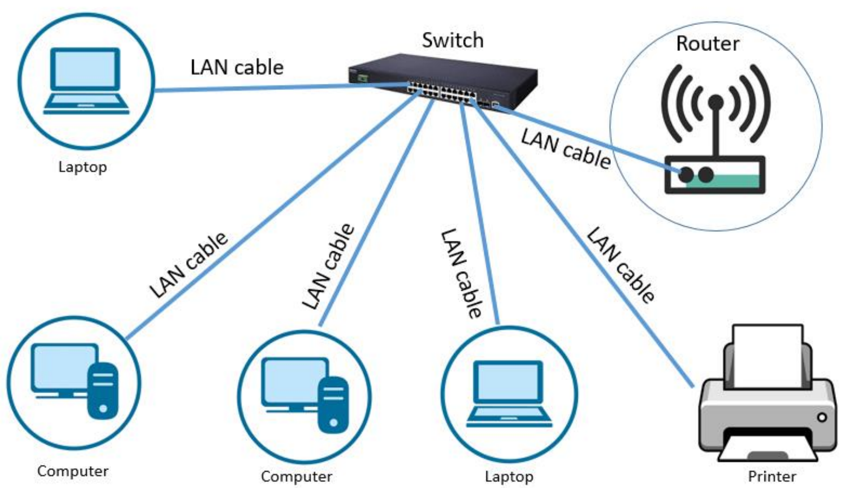
- **Resource sharing**
  - Sharing files or printing at remote sites
  - Processing information in a distributed database
  - Using remote specialized hardware devices such as **graphics processing units** (GPUs)
- **Computation speedup**
  - Distribute sub-computations among various sites to run concurrently
  - Load balancing – moving jobs to more lightly-loaded sites
- **Reliability**
  - Detect and recover from site failure, function transfer, reintegrate failed site

Network Structure

- Distributed systems are powered by network connections
- There are two types of networks, differing in their geographical distribution:
  - **Local-area networks (LAN)**:
    - For a single building or several adjacent buildings
  - **Wide-area networks (WAN)**:
    - For a large geographical area, e.g., China

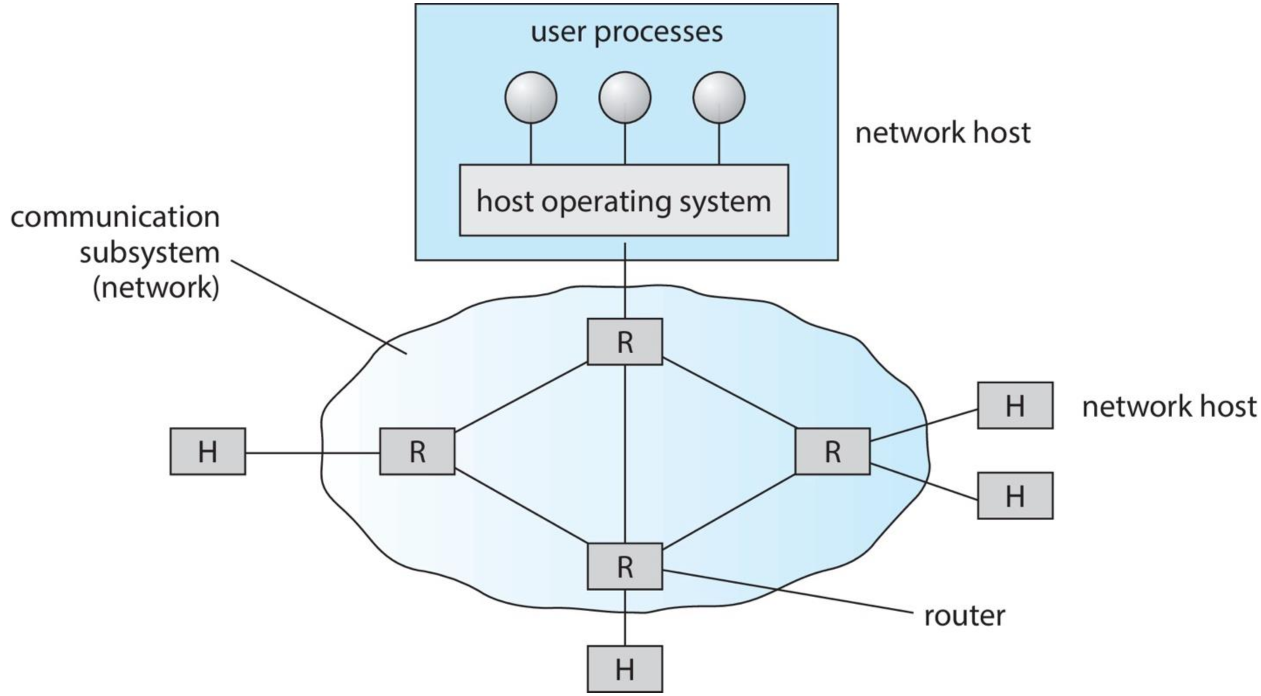
Network Structure - LAN

- **Local-Area Network (LAN)** – **designed to cover small geographical area**
  - Consists of
    - multiple computers (workstations, laptops, mobile devices),
    - peripherals (printers, storage arrays)
    - routers providing access to other networks
  - Ethernet and/or Wireless (**WiFi**) most common way to construct LANs
    - **Ethernet**: standard IEEE 802.3, speeds from 10Mbps to over 10Gbps
    - **WiFi**: standard IEEE 802.11, speeds from 11Mbps to over 400Mbps
    - Both standards constantly evolving



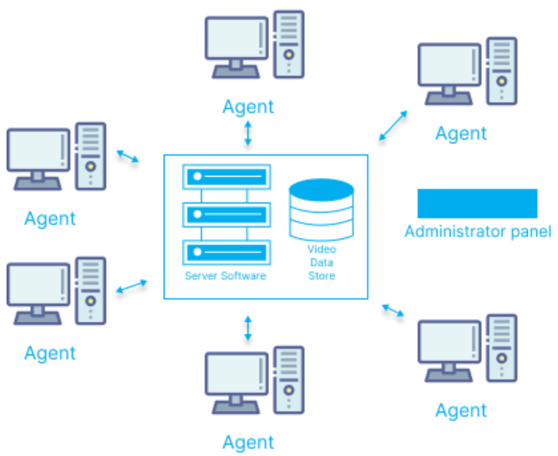
Network Structure - WAN

- **Wide-Area Network (WAN)** – links geographically separated sites
  - P2P connections via links, e.g., telephone lines, optical cable, radio waves
  - Implemented via **routers** to direct traffic from one network to another
  - Internet (World Wide Web) WAN enables hosts world wide to communicate
  - Speeds vary: Many backbone providers have speeds at 40-100Gbps

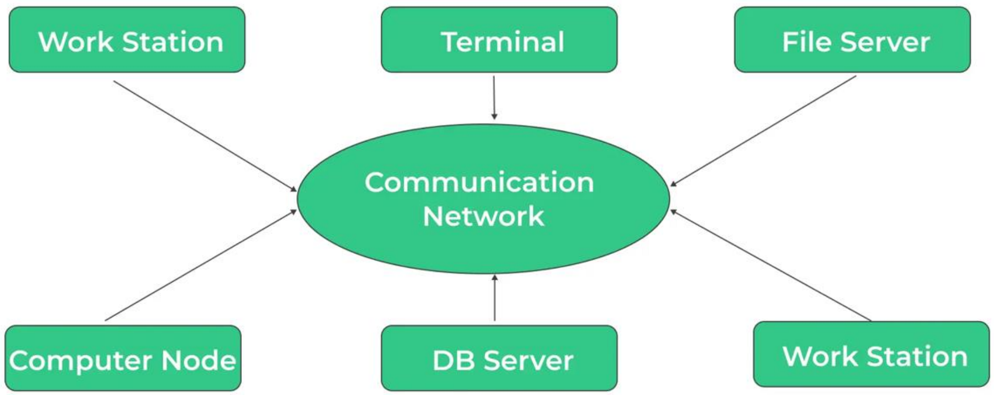


Network-oriented Operating Systems

- Two main types:
  - **Network Operating Systems**
    - Users are **aware** of multiplicity of machines
  - **Distributed Operating Systems**
    - Users are **not aware** of multiplicity of machines



Network OS



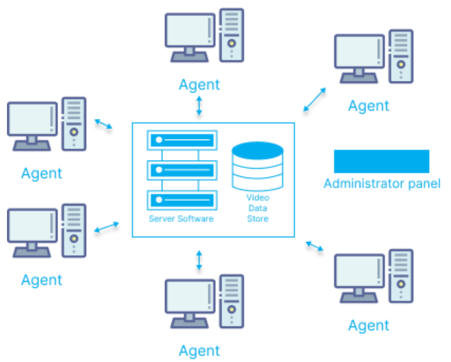
Distributed OS

Network Operating Systems

- Users are **aware of the multiplicity** of machines
- Access to resources of various machines is **done explicitly** by:
  1. **Remote logging** into the appropriate remote machine (ssh)

     ```
     ssh kristen.cs.yale.edu
     ```

  2. **Transferring data** from remote machines to local machines, via the File Transfer Protocol (FTP) mechanism
  3. **Upload, download, access, or share files** through cloud storage
- Users must change paradigms – establish a session, give network-based commands, use a web browser
  - More difficult for users



Distributed Operating Systems

- Users **<u>not aware of multiplicity</u>** of machines
  - Access to remote resources similar to access to local resources
- **Data Migration** – transfer data by transferring entire file, or transferring only those portions of the file necessary for the immediate task
- **Computation Migration** – transfer the computation, rather than the data, across the system
  - Via remote procedure calls (RPCs)
  - Via messaging system
- **Process Migration** – execute an entire process, or parts of it, at different sites
  - **Load balancing** – distribute processes across network to even the workload
  - **Computation speedup** – subprocesses can run concurrently on different sites
  - **Hardware preference** – process execution may require specialized processor
  - **Software preference** – required software may be available at only a particular site
  - **Data access** – run process remotely, rather than transfer all data locally

Distributed System Design Issues

Design Issues of Distributed Systems

- We investigate three design questions:
  - **Robustness**
    - Can the distributed system withstand failures?
  - **Transparency**
    - Can the distributed system be transparent to the user both in terms of where files are stored and user mobility?
  - **Scalability**
    - Can the distributed system be scalable to allow the addition of more computation power, storage, or users?

Design Issues: (1) Robustness

- Hardware failures can include failure of a link, failure of a site, and loss of a message
- A **fault-tolerant system** can tolerate a certain level of failure
  - Degree of fault tolerance depends on design of system and the specific fault
  - The more fault tolerance, the better!
- Fault tolerance involves:
  - **Failure detection**
  - **Reconfiguration and recovery**

Robustness – (1) Failure Detection

- Detecting hardware failure is difficult
  - To detect a link failure, a **heartbeat protocol** can be used
- Assume **<u>Site A</u>** and **<u>Site B</u>** have established a link
  - At fixed intervals, each site will exchange an **<u>I-am-up message</u>** indicating that they are up and running
  - If Site A does not receive a message within the fixed interval, it assumes
    1. the other site is not up, or
    2. the message was lost
  - Site A can now send an **<u>Are-you-up? message</u>** to Site B
    - If Site A does not receive a reply, it can repeat the message or try an different route to Site B
  - If Site A does not ultimately receive a reply from Site B, it concludes **<u>some type of failure has occurred</u>**

Robustness – Failure Detection (Cont.)

- Types of failures:
  - Site B is down
  - The direct link between A and B is down
  - The alternative link from A to B is down
  - The message has been lost
- However, Site A cannot determine exactly **why the failure has occurred**

Robustness – Reconfiguration and Recovery

- When Site A determines **a failure has occurred**, it must reconfigure the system:
  - If **<u>the link from A to B has failed</u>**, this must be **broadcast** (**广播**) to every site in the system
  - If **<u>a site has failed</u>**, every other site must also be notified indicating that the services by the failed site are no longer available

- When **the link or the site becomes available** again, this information must again be **broadcast** to all other sites
  - Suppose the A-B link has failed:
    - When it is repaired, both A and B must be notified.
  - Suppose site B has failed:
    - When it recovers, it must notify all other sites that it is up again.

Design Issues: (2) Transparency

- **Requirement**: The distributed system should appear as a conventional, centralized system to the user
- User interface should not distinguish between local and remote resources
  - **Example**: Network File system (NFS)
- User mobility allows users to log into any machine in the environment and see his/her environment
  - **Example**: Lightweight Directory Access Protocol (LDAP) plus desktop virtualization

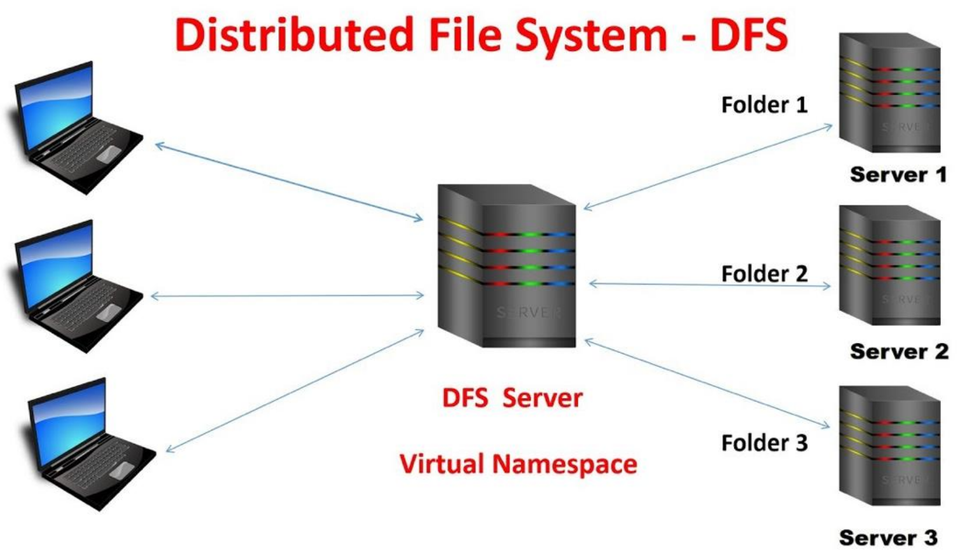
Design Issues: (3) Scalability

- As demands increase, the system should **easily accept new resources** to accommodate the **increased demand**
  - Reacts gracefully to increased load
  - Adding more resources may generate additional indirect load on other resources
    - e.g., adding machines to a distributed system can clog the network and increase service loads
- Scalability and fault tolerance
  - Having **spare resources** is essential for
    - ensuring reliability
    - handling peak loads gracefully
- Scalability and efficient storage
  - Data **compression**（压缩）or **deduplication**（**去重**）can cut down on storage and network resources used

Distributed File System

Distributed File System (DFS)

- **Distributed file system (DFS)** – a file system whose clients, servers, and storage devices are dispersed among the machines of a distributed system
  - Should appear to its clients as a conventional, centralized file system
- Key distinguishing feature is
  - **Management of distributed storage devices**

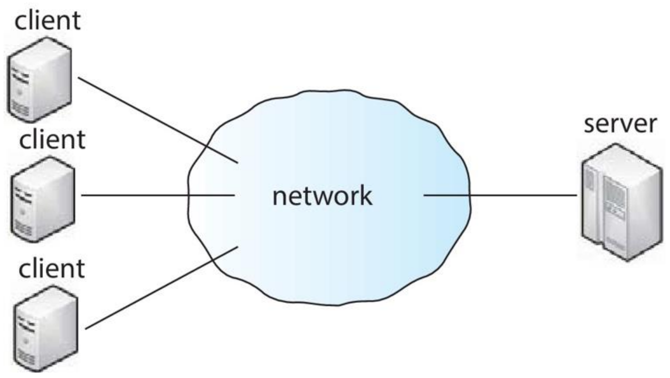


Distributed File System (Cont.)

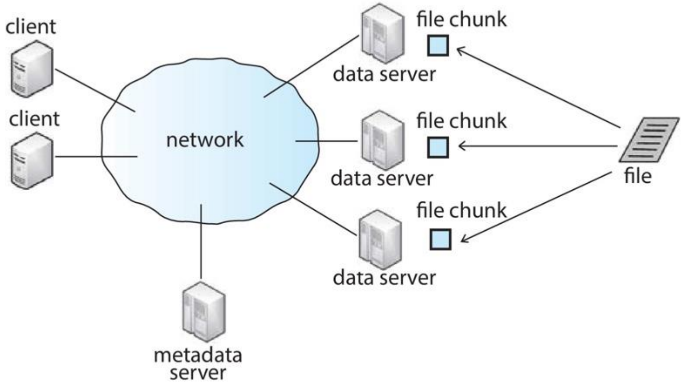
- Concepts:
  - **Service** – software entity running on one or more machines and providing a particular type of function to a priori unknown clients
  - **Server** – service software running on a single machine
  - **Client** – process that can invoke a service using a set of operations in its client interface
- Interface:
  - A client interface is formed by a set of primitive file operations
    - create, delete, read, write
  - Client interface of a DFS should be transparent;
    - i.e., not distinguish between local and remote files
  - Sometimes lower level **inter-machine** interface need for cross-machine interaction

Distributed File System (Cont.)

- Two widely-used architectural models include
  - **Client-Server model**
  - **Cluster-based model**



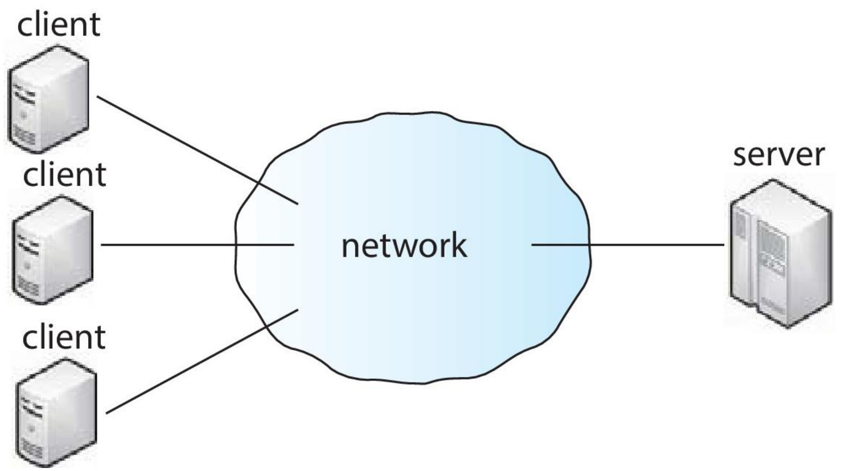
Client-Server Model



Cluster-based Model

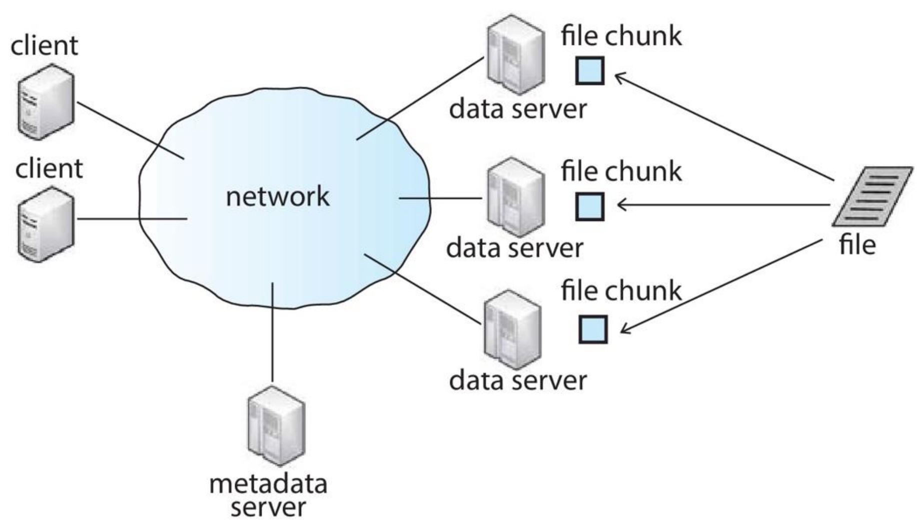
DFS: Client-Server Model

- Server(s) store **both files and metadata** on attached storage
  - **Clients**: Contact the server to request files
  - **Server**: Authentication, checking file permissions, and delivering the file
  - Changes client makes to file must be propagated back to the server
- Popular examples include **NFS** and **OpenAFS**
- Limitations:
  - Design suffers from **single point of failure** if server crashes
  - Server presents a bottleneck for all requests of data and metadata
    - Could pose problems with scalability and bandwidth



DFS: Cluster-based Model

- Built to be **more fault-tolerant and scalable** than client-server DFS
- E.g., **Google File System (GFS)** and **Hadoop Distributed File System (HDFS)**
  - Clients connected to
    - **Master metadata server**
    - **Several data servers** that hold "chunks" (portions) of files
  - Metadata server keeps mapping of data servers to file chunks
    - As well as hierarchical mapping of directories and files
  - File chunks replicated **n** times to protect against failures and for fast access



DFS: Cluster-based Model

- Google file system (GFS) design influenced by the following observations:
  - Hardware failures are normal and should be routinely expected
  - Large files:
    - Files stored on such a system are very large.
  - Append dominated:
    - Most changes are appending new data rather than overwriting existing data.
  - Redesign **applications** and **file system APIs** to increase flexibility
    - Requires applications to be programmed with new API
- Modularized software layer **MapReduce** can sit on top of GFS to carry out large-scale parallel computations while utilizing GFS advantages
  - **Hadoop** framework also stackable and modularized

DFS Challenges

DFS Challenges

- Challenges include:
  - Challenge 1: Naming and transparency
  - Challenge 2: Remote file access
  - Challenge 3: Caching and cache consistency

DFS Challenge: (1) Naming and Transparency

- **Naming** – mapping between logical and physical objects
- **Multilevel mapping** – abstraction of a file that hides the details of how and where on the disk the file is actually stored
- A **transparent** DFS hides the location in the network where the file is stored
- For a file being replicated in several sites, the mapping returns a set of the locations of this file's replicas
  - Both the **existence of multiple copies** and **their locations** are hidden

Naming Structures

- **Location transparency**
  - File name does not reveal the file's physical storage location
- **Location independence**
  - File name does not need to be changed when the file's physical storage location changes

- In practice most DFSs use **static, location-transparent mapping** for user-level names
  - Some support file migration (e.g. OpenAFS)
  - **Hadoop** supports file migration but without following POSIX standards; hides information from clients
  - **Amazon S3** provides blocks of storage on demand via APIs, placing storage dynamically and moving data as necessary

Naming Schemes

- Three file naming approaches:
  1. Files named by a combination of their **host name** and **local name**
     - This scheme is neither location transparent nor location independent.
  2. **Attach remote directories to local directories**, giving the appearance of a coherent directory tree
     - Only previously mounted remote directories can be accessed transparently
  3. **Single global name structures span all files in the system**
     - If a server is unavailable, some arbitrary set of directories on different machines also becomes unavailable

DFS Challenge: (2) Remote File Access

- Consider a user who requests access to a remote file.
  - The server storing the file has been located by the naming scheme
  - Now the actual data transfer must take place.
- **Remote-service mechanism** is one transfer approach.
  - Requests for access are delivered to the server
  - The server machine performs the access
  - Their results are forwarded back to the user
- One of the most common ways of implementing remote service is the **RPC paradigm**

DFS Challenge: (2) Remote File Access

- Reduce network traffic by retaining recently accessed disk blocks in a **cache**, so repeated accesses to the same data can be handled locally
  - If needed data not already cached, a copy of data is brought from the server to the user
  - Accesses are performed on the cached copy
  - Files identified with one master copy residing at the server machine, but copies of the file are scattered in different caches

- **Cache-consistency problem** – keeping the cached copies consistent with the master file
  - Could be called **network virtual memory**

DFS Challenge: (3) Cache Consistency

- Is locally cached copy of the data consistent with the master copy?
  - **Client-initiated approach**
    - Client initiates a validity check
    - Server checks whether the local data are consistent with the master copy
  - **Server-initiated approach**
    - Server records, for each client, the (parts of) files it caches
    - When server detects a potential inconsistency, it must react
- In cluster-based DFS, cache-consistency issue is more complicated due to presence of **metadata server** and **replicated file data chunks**
  - HDFS allows append-only write operations (no random writes) and a single file writer
  - GFS allows random writes with concurrent writers

Summary

- A distributed system is a collection of processors that do not share memory or a clock
- A distributed system provides the user with access to all system resources.
  - Access to a shared resource can be provided by data migration, computation migration, or process migration
- Many challenges to overcome for a distributed system to work correctly. Issues include naming of nodes and processes in the system, fault tolerance, error recovery, and scalability
- DFS is a file-service system with service activity carried out across the network
  - Two main types: the **client-server model** and the **cluster-based model**
  - DFS should look to its clients like a conventional, centralized file system
- DFS challenges:
  - Naming and transparency
  - Remote file access
  - Cache consistency

Homework

- Reading
  - Chapter 19

- We meet at **Dongshang Yard 415 (东上院415)** this Friday!