0Network Stack

# Roles of OS in Network

* **Network abstractions**:
  + Reliable byte stream (TCP) vs unreliable datagram (UDP)
* **Communicate Network protocols**: IPv4, IPv6, UDP, TCP etc.
* **Network services**: packet I/O, routing, scheduling
* **Application Multiplexing**: sharing the NIC (Network Interface Card) among applications and protocols
* Design goals: **Generality**, **performance** and **extensibility**.

# Network Abstraction

* What do network applications do?
* Text

  Description automatically generatedWork similarly to file I/O APIs
* “File Descriptor” for each socket is unique – works similarly to file systems
* Lots of features abstracted away. When you send a request for a URL
  + DNS request must be sent
  + DNS response received
  + Connection to web server
  + Communication live
* When the user wants to communicate with another computer, the computer must register with the network stack (TCP/IP) that it wants to use the TCP/IP service on this port number.
* We **bind** the descriptor with the required service, the port number.
* Above we are acting as a **server** as we are listening for network packets.

# Asynchronous I/O

* When calling read(),

# Notes From Book

# Advantages of Distributed Systems

* **Definition**: a collection of loosely coupled nodes interconnected by a communication network.
* **Messages** are passed on lower level which can be encompassed by higher-level functionality, including file storage, application execution and remote procedure calls (RPCs)
* Three main reasons for building distributed systems:
  + **Resource Sharing**: provides mechanisms for sharing files at remote sites, processing information in distributed databases, or using remote specialised hardware devices such as a supercomputer
  + **Computation Speedup**: Partitioned sub-computations can be run concurrently amongst nodes in distributed system, providing **computation speedup**.
    - **Load balancing**: Movement of jobs between systems
  + **Reliability**: failure of one node (with enough redundancy) does not affect the operation of other nodes.

# Network Structure

* There are two types of network structures:
  + **Local-area network** (**LAN**): operates in small region such as building(s)
  + **Wide-area networks** (**WAN**): operates on large geographical areas such as countries
  + These are often interconnected

# Communication Structure

## Naming and Name Resolution

* Network processes stored as <**host name** (unique name within network), **process** (process ID for host)>
* Host name must be resolved into a **host-id** as numbers result in faster computation.
* Use **DNS** (**Domain Name System**) for this:
  + Specifies naming structure for hosts
  + Handles name-to-address resolution
  + Internet: involves converting host names to **IP addresses**
* Typically host names are made up of multiple domains such as ed.ac.uk, so **name-servers** operate for each domain to direct the user to the right resource.
* During domain name resolution, name servers will be returned until the 2nd last domain. This will return an address to the final server being searched for.
* This process is inefficient, so it is **cached**.
  + The OS is responsible for **accepting** for accepting messages destined for <host name, identifier> from its processes and transferring the message from the host.
  + The kernel from the receiving system is then responsible for transferring the message to the process named by the identifier.

## Communication Protocols

* Distributed layers are broken into layers, each with their own **communication protocol**.
* There are 8 layers:
  + **Physical layer**: handles mechanical and electrical components of physical bit stream transition.
  + **Data Link Layer**: Sends frames between physical addresses. Handles frames (fixed-length parts of packets). Includes error detection and recovery.
  + **Network Layer**: Routers works at this layer. Breaks messages into packets, provides communication between logical addresses and routing packets. This includes
    - Handling addresses of outgoing packets
    - Decoding addresses of incoming addresses
    - Maintaining routing information
  + **Transport Layer**: Transferring messages between nodes, maintaining packet order and flow.
  + **Session Layer**: Implements sessions or process-to-process communication protocols
  + **Presentation Layer**: resolves differences between sites
  + **Application Layer**: interacts directly with uses. Delas with:
    - File transfer
    - Remote-login protocols
    - E-mails
    - Schemas for distributed databases
* Modern protocol stack: **TCP/IP** model.
  + Combines several different functions in each layer. Difficult to implement but efficient
* **IP** (Internet Protocol) responsible for routing IP **Datagrams** (packets) through internet.
* Packets in form of **TCP** (Transmission Control Protocol) or **UDP** (User Datagram Protocol)

### TCP/IP

* In a TCP/IP network, every **host** has a **unique name** and an associated **unique IP address**.
* **Host-id** is split into **network number** and **host number.**
  + Sending system checks routing tables to find router to send a frame: entries are either updated manually or by routing protocols such as **Border Gateway Protocol** (**BGP**)
  + Routers use host-id to transfer frame to destination.
* On a local layer within a LAN, we need a mechanism for identifying local addresses.
* In order to physically address a **device** **within a network**, **Medium Access Control** (MAC) addresses are used. These are 48-bit number usually in hex. Tis works on the **data-link layer**.
* To obtain the MAC address, networking software generates an **address resolution protocol** (**ARP**) containing the destination system’s IP address, and **broadcasts** it on the LAN.
* The device with the corresponding IP address will then respond to the device with its MAC address. This entire process **does not use the router**.
* Note: on a LAN packets often come with a **checksum** to indicate packet damage.

## TCP and UDP

### UDP

* **User/Unreliable Datagram Protocol**: contains source port number, destination port number, length and checksum
* **Unreliable** because there is no guarantee of delivery. Also **connectionless** as there is no connection setup at beginning of transmission.

### TCP

* **Transmission Control Protocol**: **reliable** and **connection-oriented**.
* ACK packets are used to acknowledge that a package is received.
* Also includes **sequence numbers** in packets: this ensures the order of the packets and acknowledges if there are missing packets
* **Three-way-handshake** used to initiate communication, and control packets also used to close communication.
* In actual TCP, **cumulative ACK** packages are sent to acknowledge series of packets
* TCP can also communicate two properties in ACK packages:
  + **Flow Control**: aflowcontrol statecan be sent via ACK packets to slow or speed packet transmission
  + **Congestion Control**: sender monitors connection for dropped packets. Too many dropped packets: slow down transfer rate.

# Network and Distributed Operating Systems

## Network Operating System

* **Network OS** provides an environment in which users can access remote resources by implementing resource sharing.
* Accessed via **logging in** to appropriate machine, or transferring data from remote machine to their own machine.
* Network OSs are responsible for providing the following functionality:
  + Remote Login
  + Remote File Transfer
  + Cloud storage

### Remote Login

* Remote login achieved via ssh, providing username and password.

### Remote File Transfer

* Once logged in, user can upload and download files, as well as navigate the remote file system.
* Uploading/downloading achieved via (**Secure**) **File Transfer Protocol** ((**S**)**FTP**).
* Also various commands to change transfer modes and determine connection status

### Cloud Storage

* Upload to **cloud storage** via FTP.
* Can also share files with other cloud-service users via web links or sharing mechanism.

## Distributed OS

* Users can access remote resources the **same way** the access local resources.
* Can implement **data mitigation**, **computer migration** or/and **process migration** **.**

### Data Mitigation

* If site A wants a file from site B, there are two methods to **data mitigation**:
  + Whole file is downloaded and then uploaded, no matter the extent of the changes
  + Only portions of the file that are necessary are downloaded, changed, and then uploaded.
* Alsop involves data translation if sites are not directly compatible e.g. different character-code representation)

### Computer Migration

* Involves **transferring computations**  on system. Useful if source needs to run code on multiple files from destination.
* Two methods:
  + Use **Remote Procedure Call** (**RPC**) which uses network protocols to execute routine on remote system.
  + Message destination system requesting it to carry out the task. The destination system will then create its own process and return the results.

### Process Migration

* When a process’s execution is split among sites. May be used for:
  + Load Balancing
  + Computational Speedup
  + Hardware Preference
  + Software Preference
  + Data Access
* Two complementary techniques to achieve this:
  + Hide process migration from client. Usually used for load balancing and computational speedup among homogenous systems (systems with similar processor cores)
  + Allow client to specify how process should migrate to satisfy their hardware/software preferences.
* Essential for WWW.

# Design Issues in Distributed Systems

* Issues are:
  + Robustness
  + Transparency
  + Scalability

## Robustness

* Failure of links, hosts or sites and losses of messages are common hardware issues.
* **Fault tolerance**: systems ability to tolerate failure of its nodes. A system is **fault-tolerant** if it can continue to function, even if it offers less functionality, to some extent.
* Difficult to implement:
  + Network layer: redundant communication paths and network devices needed to avoid communication failures.
  + Data link layer: Storage failures can break OS, applications, or corrupt data.
  + RAID systems can ensure access to data even in the event of storage device failure.
* Three main factors to consider:
  + Failure Detection
  + Reconfiguration
  + Recovery from failure.

### Failure Detection

* Generally cannot differentiate between failure of sites, hosts, links and message loss.
* **Link** and **site** failure detection: use **heartbeat** at fixed intervals. Using different links can help differentiate between either.
* Site A sends signal to site B requesting a response and gives its max waiting time. Failure to reply from site B indicates one of the following faults:
  + Site B is down
  + Direct link from A to B is down.
  + Alternative path from A to B is (also) down.
  + Message is lost (using TCP eliminates this concern).

### Reconfiguration

* Reconfiguration in response to a failure is necessary for a system to resume normal operation.
* If a **direct link** fails, an alert must be broadcast so routing tables of all devices are updated.
* If a **site** has failed, all other sites must be notified, including reallocation of roles such as **central coordination**. Undesirable: more than one central coordinator.

### Recovery from failure

* **Link recovery**: A and B must be notified if their link is repaired.
* **Site recovery**: If a site is made reachable/is repaired, it must notify all other sites **and** receive information about their sites in case things have changed since it went down.

## Transparency

* Challenge: **Transparency** of multiple processes and storage devices to user.
* **User Mobility**: ideally a transparent distributed system should facilitate user mobility by allowing user to access their own environment to any device they log into.
  + Protocols like LDAP provide authentication systems for local, remote and mobile users, and **desktop virtualisation** is then used to make the system accessible to the user.

### Scalability

* **Scalability**: adapting to **increased service load.**
* Two properties of scalable distributed system:
  + Performance degrades more moderately as load increases
  + Resources take longer to reach a saturated state.
* Having **spare resources** is essential for ensuring reliability and handling peak loads gracefully.
* Also relates to **efficient storage schemes**
* **Compression**: reducing file size for transmission and restoring original on destination site**.**
* **Deduplication**: lower storage requirements by removing redundant data from files.

# Distributed File Systems (DFS)

* **Service**: software entity running on **one or more machines** that provides a **particular type of function** to clients.
* **Server**: **service software** running on a single machine.
* **Client**: process invoking a service using a set of operations from its **client interface**.
* Lower-level interface for cross-machine interaction: **intermachine interface**.
* A **file system** provides **file services** to **clients**.
  + **Client interface**: create, delete, read, and write.
* **File server** primary hardware: **secondary storage devices**.
* Hence a **DFS** is a **file system** whose **clients**, **servers**,and **storage devices** are **distributed** among machines in a distributed system.
* System has multiple independent storage devices. Some systems have dedicated machines for servers, others run machines as client and servers.
* Client interface of a DFS should not distinguish between local and remote files: up to DFS.
* **Transparent DFS**: brings user’s environment to wherever the user logs in.

## Client-Server DFS Model

* **Server**: stores both files and metadata.
* **Clients**: connected to server via network and can request access to files in DFS via well known-protocols such as NFS version 3
* Serverresponsiblefor authentication of permissions.
* Client responsible for transmission of amended files to server.
* Protocols for file transmission:
  + **Network File System** (**NFS**):
    - Focus on simple, fast crash recovery.
    - Sends updates to file to server regularly stateless (no tracking of which client requests what file).
    - Client stores state and reissues operation if server crashes and comes back online.
  + **Andrew File System** (**OpenAFS**):
    - Minimises requests and traffic to server.
    - Updates only sent when file is closed.
* Client-Server DFS can suffer from single point of failure if server crashes.
* Solution: **computer clustering**: using redundant components and clustering methods.
* Susceptible to scalability and bandwidth problems as server presents bottleneck for all requests for both data and metadata.

## Cluster-Based DFS Model

* Fault Tolerance and scalability increasingly important as data, I/O workload and processing expands. Solved by **cluster-based DFS models.**
* **Chunks** of files are **copied** and spread amongst **different servers**.
* To access a file, client contacts a **metadata server**
  + It maps the chunks to the servers, as well as a traditional hierarchical mapping of directories and files.
  + Responsible for balancing file chunks among designated servers, uses polling to detect server failures.
* Server returns **servers** that hold **chunks** of the file, and client contacts **nearest** servers to receive file information.
* Chunk copies are also updated: not handled by metadata server.
* Example system: Google File System (GFS)
* **MapReduce** developed to carry out large-scare parallel computations using GFS.

# DFS Naming and Transparency

* **Naming**: **mapping** between **logical** (user) and **physical** objects (system).
* **Abstracted** for user so specific locations of files and their copies are hidden.

## Naming Structures

* **Location Transparency**: Name of a file does not reveal file’s physical storage location
* **Location Independence**: No name change when physical storage location changes.
* **Location**-**Independent Naming Scheme**: dynamic mapping by mapping same file name to different physical locations.
* Some support **file migration**: **changing location** of a file **automatically** (location independence).
* Most DFSs provide **static**, **location-transparent** mapping for user-level names.
* Differences between static location transparency and location independence:
  + Divorce of data from location: LI provides better abstraction for files, as they are not attached to the physical location. SLT still denotes a specific set of physical disk blocks.
  + SLT provides users with convenient method for sharing data by making data accessible to user as if it were local. LI promotes sharing the storage space itself, making it look like a single virtual resource. A possible benefit of this is balancing storage utilisation across the system.
  + LI separates naming hierarchy from storage-device hierarchy and from intercomputer structure. Correspondence between component units and machines easily exposed in SLT,
* LI can allow clients to be **diskless** and rely on servers to provide all files, including the kernel. Local boot protocol however needed by client.
  + Advantageous as it lowers cost and makes updates to OS more convenient.
  + Disadvantageous as boot protocols are more complex and performance is weaker as data is transmitted over network, rather than a physical disk.

## Naming Schemes

* Three approaches:
  + File identified by combination of host name and local name, guaranteeing system-wide name.
  + Attach remote directories to local directories. Popularised by **NFS**.
    - **Automount**: mounts on demand based on table of mount points and file-structure names.
    - Integration is limited and not uniform, as each machine may attach different remote directories to its tree, this is versatile.
  + Global name structure that spans all files of the system.
    - Difficult due to special device files and machine-specific binary directories.

### Implementation Techniques

* Aggregate sets of files into component units and provide mapping on component-unit basis rather than single-file basis.
* Replication and/or local caching enhances availability of mapping information.
* For location independence replicating is impossible however as mapping changes over time.
  + Solution: low-level **location-independent file identifiers** mapping textual file names to component units.
  + Can still be replicated and cached freely without being invalidated by migration of component units.
  + Price: need for second level of mapping between component units and locations, with simple update mechanism.
  + Common implementation uses bit string **structured names** with two parts:
    - Component unit file belongs to
    - File itself

# Remote File Access

* Transferring data between server and user: **remote-server mechanism**
* Requests for accesses delivered to server, server machine perform accesses, results delivered back to user.
* Can use **caching** to reduce network traffic and disk I/O,

## Basic Caching Scheme

* Client keeps a copy of the data, and accesses are performed on cached copy.
* Replacement policy keeps cache bounded.
* **Cache consistency problem**: local cache must update master file.
* **NFS** mounts remotely, implementing this **network virtual memory**, with performance cost.
* Both **hit ratio** and **miss penalty increased** if caching unit is increased.
* Consistency problems also an issue as large units must be disassembled for transmission and reassembled on reception.
* The lower the cache size, the lower the block size/cache unit should be, otherwise there are fewer blocks in the cache hence lower hit ratio

## Cache location

* Either on disk or in main memory. If on disk:
  + Reliable as disk is not volatile, so no loss of data on system crash.
  + Cache data kept on recovery and no need to fetch them from server
* However if on memory:
  + permits workstations to be diskless
  + Faster access times
  + Server caches in their main memory. If user system also uses main-memory cache, single caching mechanism for both server and user can be built.

## Cache Update Policy

* Master doc must be updated.
* **Write through-policy**: write modifications as soon as they are put in cache.
  + Advantage: high reliability so little data is lost in system failure
  + Disadvantage: Poor write performance as each write access must wait until information is sent to the server.
* **Delayed-write policy/write-back caching**: Delay updates to master copy.
  + Advantages: Write accesses quicker and only last update of file is written.
  + Disadvantage: Introduces reliability problems
  + Variations for flushing blocks back to the server:
    - Flush a block just before ejection from cache: good performance but blocks may reside on cache for long time
    - Scan cache at regular intervals and flush blocks that have been modified since last scan. Used by NFS, but update must reach server’s disk before it is completed.
      * NFS metadata changes are issues synchronously to server, minimising file-structure and directory-structure information loss.
    - **Write-on-close**: Used by OpenAFS: Does not optimise if files open for short space of time, and delays the closing process of files. Optimal for when files are open for long time.

## Consistency

* Cached data must be validated to ensure that it matches master copy. If it does not:
  + **Client-initiated approach**: clients initiates validity check with server
  + **Server-initiated approach**:server records files that client caches, and reacts to potential inconsistency. This occurs when two different clients in conflicting modes cache a file.
    - Server can disable caching if file has been opened simultaneously in conflicting modes for the file.
* More difficult issue in cluster-based DFSs due to metadata server and file chunk replication.
  + HDFS allows append-only write operations whereas GFS allows random writes with concurrent writers.
  + Hence maintaining consistency more complicated for HDFS than GFS

# Summary

* **Distributed system**: collection of **processors** that **do not share** a **memory** or a **clock**. Instead, processors **communicate** **through** **various communication lines**, such as high-speed buses and the internet.
* The **function** of a distributed system is to provide the user with access to all system resources across a network. Access to shared resources can be provided by **data migration**, **computation migration** and **process migration**. Access can be specified by the user or implicitly supplied by OS and applications.
* **Protocol stacks** add information to a message to ensure it reaches its destination. Specified by **network layering models**.
* A **naming system** (such as DNS) is required to **translate** a **hostname** to a **network address**. A protocol (e.g. **ARP**) my be needed to translate the network number to a network device address, such as a MAC Address.
* If systems are located on **separate networks**, **routers** are used to pass packets from the source to the destination network.
* UDP and TCP direct packets to waiting processes via system-wide port numbers. TCP also allows flows of packets to be a reliable, connection-oriented byte stream.
* Many issues for distributed system, including **naming nodes and processes** in the system**, fault tolerance**, **error recovery** and **scalability**. **Scalability issues** include **handing increased load**, being **fault tolerant** and using **efficient storage schemes** such as **compression** and/or **deduplication**.
* Two main types of DFS models: Client-server mode and cluster-based model. Client-server allows transparent file sharing amongst one or more clients. Cluster-based distributes files among one or more data servers, and is built for large-scale parallel processing.
* Ideally, a DFS should look like a conventional, centralised file system. The multiplicity and dispersion of servers and storage devices should be transparent. A **transparent DFS** facilitates client mobility by bringing client’s environment to any site where a user logs in.
* Several approaches to **naming** schemes in DFS. Simplest approach involves a **combination of the host name and local name**, guaranteeing a unique system-wide name. Another approach, popularised by NFS, provides a means to **attach remote directories to local directories**, giving the appearance of a coherent directory tree.
* Requests to access a remote file are handled by two complementary methods. **Remote service**: **requests for accesses** are delivered to the server, which **performs the accesses** and **forwards the results** back to the **client**. Data can be **cached** by client and accesses are performed on cache, which brings the **cache-consistency problem**.