Distributed Systems

A 1st Year Revision Summary by James Bedford

Notice
Warning: I don't think this is complete.

Paradigmatic Examples:

- The Web:
  - Conceived as a means of sharing documents.
  - Independent hosts.
  - Exchange webpages/email etc via Internet protocols.
- Mobile Telephony:
  - Have been able to scale up and expand.
  - Roaming devices.
  - Exchange phone calls via a cellular network.
- Electronic Funds Transfer Systems:
  - LINK has 51 organisations, and 60,000 ATMS.
- Email
- Video Conferencing
- Multi-User Gaming
- Global Positioning Systems

Why distribute systems?

- Functional Reasons:
  - For making remote resources accessible for sharing.
  - For cooperative use.
- Non-Functional Reasons:
  - Can reduce costs.
  - For scaling - more is more!
  - For achieving stability.

What is a Distributed System?

- Independent, self-sufficient:
  - Each component has a processor, state (memory) and a resource control and management (e.g. operating system).

- Autonomous:
    - Each component may change on its own accord.
- Heterogeneous:
    - Different components may have different capabilities (e.g. different phones/hardware/software). These differences may cause components to drift apart over time. Failures may cause a component to not be fully aware of the current system state.
- Spatially-Seperated:
- Exchanging of Information
- A Common Interconnect
- To appear to its uses as a single coherent system.

Definition:

- Computation is *concurrent* (existing or done at the same time).
- There is *no global state or global clock*.
- Components *may fail independently*.

One Complex Consequence:

- If the distance between communications are significant, then communication events have a non-negligible duration.
- Communication costs may become more significant than processing time.
- Durations may also vary, and so may cause variable rates of information exchange.
- So **asynchrony** is inherent, and synchronisation requires special measures.
- Therefore components may exhibit variable rates of processing.
- Another consequence is that one computer's failure may lead to one not being able to use his or her computer at all.

The Basic Model:

- Machine = Computing resource (e.g. CPU) + Storage resource (e.g. memory)
- Resource usage controlled efficiently by the OS.
- A process is an executing instance of a program.
- An OS assigns a unique identity to each process and an address space.
- Address space is exclusive to a process , so processes can be sequential.

Multiprocessing:

- An OS multitasks by controlling how each process makes use of the resources allocated to it.
- The OS implements a scheduling policy, allocating time slices to processes.
- In this case the group of processes is **active concurrently**. This form of execution may only be apparent if the machine only has one form of computing resource.
- To the user its seems like things are running in parallel - but really this is

just an effect done through clever resource time allocation.

Forking:

- A form of multi-tasking.
- When a process forks it creates two copies of itself.
- The child process is executed first, and the parent waits.
- Safe - address spaces are distinct.
- Better to use Multiprocessing.
- Centralized - using the separate address space for each process, as opposed to concurrent.

Interaction: Goals and Modes

- Interaction:
    - Coordination and cooperation.
    - Relationships between events in the timeline.
    - There may be competition so management is needed.
    - Can be:
        - sharing an address space.
        - synchronous communication mechanisms.
        - message passing.
- An interconnect is needed if components are spatially apart.

- No interconnection models:
    - Have no access costs.
    - SISD (Single Instruction, single data).
- Interconnecting storage introduces access costs.
    - SIMD (Single Instruction, Multiple Data).
- Interconnecting processors:
    - To apply several algorithms to the same data.
    - MISD (Multiple instruction, single data).
- Inherently-distributed system.
    - Binds autonomous components.
    - MIMD (Multiple instruction, multiple data).
    - Also known as shared-nothing architectures.

- Interconnects at various scales:
    - Chip; bus connecting multiple cores.
    - Parallel Machines/Appliances; very fast interconnect (infiniband) within a machine binding components together.
    - Clusters of workstations (COW); now across racks.
    - Network of workstations (NOW); LAN, such as Ethernet.
    - Web/Grid; WAN, such as the internet.
- Networks as Interconnects:
    - Network exists as a physical fabric which must be mediated by complex software.
    - Machines at home connect to a server in the service provider (ISP) using DS0 or cable ISDN or DSL.
    - LANs are formed using Ethernet as a fabric connected via optical T1.

- One step up LANs are connected using T3 backbone network services (vBNS) with links like OC3.
- A network of networks is formed by routers that handle and send packets. See later notes.

Process Interaction:

- IPC - Inter process communication.
- Interaction event via IPC requires event synchronization.
- Distributed Computing (DC) Architectures:
  - Direct Message Exchange (DME):
    - A process must be in a position to receive a message from some other process.
    - A process may send a message to another process.
    - A process that receives a message may process it and send a response.
    - Event synchronization is unmeditated.
    - Most prevalent.
    - Two types of DME:
      - asymmetric DME:
        - one process (the server) can wait for requests and the other process (client) can just wait for the responses.
        - processes are assigned roles (server or client).
        - typically takes on heavier loads.
        - load balancing may prevent scaling.
        - the web and email are examples.
      - symmetric DME:
        - No roles assigned.
        - Processes are peers.
        - Event synchronization less simple because all the processes can send, process and receive.
        - lead naturally to balanced loads and graceful scaling.
        - Skype and torrent are examples.
      - mediated message exchange MME:
        - IPC is not direct.
        - Messages go through a middleware component instead - message-oriented middleware (MOM).
        - It mediates IPC.
        - Messages are sent to the MOM que which then sends them to the receiving process.
        - point to point.
        - publish-subscribe systems are examples.
      - remote-call approches:
        - remote procedure call (RPC):
          - IPC and event sync are triggered by a call to a remote process.
          - parameter passing.
          - object-orienteted RPC is known as remote

method invocation (RMI).
  - If the invoking process knows which remote object to envoke them early-binding (E-B) RMI is used. e.g. JRMP (Java remote method protocol).
  - If the invoking process delays the decision to which object to invoke. This is late-binding (L-B) RMI architecture. The invoking process may search a directory to select the remote object it wishes to invoke.
  - This is time-decoupling in the IPC approach.

Interprocess Communication (IPC):

- Basic IPC - each process acts as a sender or receiver.
- Operating System Support:
  - API is the most important, which relies on the OS's services:
    - Network buffers.
    - Synchronization mechanisms - which running process can be told that data has been sent or has arrived.
  - Most basic API for IPC:
    - Send - Sender process calls this, with arguments of the receiver process and the data to be sent.
    - Receive - the receiver process calls this, with arguments of the location where the data is to be placed and possibly the sender process to know where it's going to be coming from.
    - When this IPC is connection-oriented, the following are also used:
      - Accept - the receiver process calls this operation to specify its readiness to engage.
      - Connect - the sender process calls this operation to initiate the engagement.
      - Disconnect - either process (or both) must use this to cleanly disengage.
    - Each operation call causes an interaction event to occur between the two states of the two systems.
  - Synchronization/Blocking Primitives:
    - These in send and receive specify when control returns to the invoking process.
  - FOUR forms of send:
    1. Asynchronous Blocking Send:
       - Does not return until the the message is copied out of the buffer in the sender process. Does not wait for acknowledgement by the receiver process.
    2. Synchronous Blocking Send:
       - Does not return until the acknowledgement arrives back at the sender process that the message has been placed in the buffer at the receiver process.

3. Asynchronous Non-Blocking Send:
    - Different because it can return before the message has been copied out of the buffer in the sender process. WaitForAsynchronousSend is used to determine when the message has been copied out of the buffer.
4. Synchronous Non-Blocking Send:
    - Can return before acknowledgement arrives. Uses again the WaitForAsynchronousSend to notify when the receiver process has got the message into its buffer.

- TWO forms of receive:
- Receive has to be synchronous (sending some form of acknowledgment back) and its parameter is the buffer where it shall put the received message.
    1. Blocking Receive:
        - Does not return until the message has arrived and been copied into the buffer in the receiver process.
    2. Non-Blocking Receive:
        - Can return before the message has arrived.
        - Uses a WaitForReceive primitive which causes a signal to be passed to the latter that the message has been copied into its corresponding buffer.

Protocols:

- A protocol is a set of rules for IPC.
- Sets our the sequence of interactions.
- Unicast - from one single process to another single process.
- Multicast - from one single process to many other processes.
- t may be later in real time that t' but appear to be t' < t because of delay. To get round this you could use a synchronous Send or the web server could keep polling (looping the receive).

Sockets:

- Low level, logical level IPC.
- In unix they can be viewed as input/output streams.
- Client-side sockets are often short-lived, e.g. in a web browser one is created to send a request and receive a response, and then discarded:
    - Create a socket for the given transport (e.g. TCP) and internet protocol (e.g. IPv4).
    - Connect to the server using the port linked to the protocol (e.g. HTTP is 80).
    - Then the client can send the request. Through the same socket a response is waited for and then received in chunks.
    - Socket is discarded.
    - Gopher is a great example.
- Server-Side Sockets:
    - Acts more like a dispatcher.
    - It listens for connections on the server socket and ports them.
    - When it gets a new connection it gets a new socket in response.

    - It then spawns a handler process, that actually uses the new socket to exchange messages with the client-connected socket.
    - It then goes back to listening for more connections.
    - Server-Side Socket set up:
        - Create a server socket for the given transport (e.g. TCP) and internet protocol (e.g. IPv4).
        - Set the socket options.
        - Bind the server socket to a host address and a port.
        - Start listening to connections on the server socket and set the max number that could be left waiting.

Voice Over IP:

- Microphones convert sound vibrations into a variation in voltage.
- Humans can hear up to 20 kHz.
- CDs vary from 50Hz to 20kHz.
- Telephone quality varies from 300Hz to 3.4kHz.

- Plain Old Fashioned Telephone Systems (POTS)
    - Analogue.
    - 1 ms per 100 miles.
    - Circuit set up between two users and voltages are exchanged.
    - Circuit switch and was *connection orientated*.
    - This is still used for the first and last mile in telephone communications.
- Digitisation of Speech and Music:
    - Done by taking voltage waveforms and converting them into a binary numbers of an appropriate word length.
    - 'Sampling Theorem' :
        - If the signal bandwidth is 0 to B Hz, we must sample more than 2B samples/second (Hz) to obtain a faithful representation of it.
    - Narrow-band telephone speech is often sampled at 8 kHz with 8 bits per sample to obtain 64 kb/s log-PCM (ITU-G711) - the famous standard.
    - This is the basis of 'pulse code modulation (PCM)' with 'logarithmic compression' (which means that small samples are digitised more accurately than larger ones.
- Other standards:
    - The 64k/s required by the ITU-G711 is too high for mobile telephones. Other ITU standards exist:
    - G76 (32 kb/s), G728 (16 kb/s), G729 (8 kb/s) and G723.1 (5.3 kb/s).
    - Mobile phones use a 9.6 kb/s standard.
- Buffers
    - Sound cards control their sampling rates using independent crystal controlled clocks (0.01% accurate).
    - They must have *buffers* (an array or block of storage) to store sections of their inputs and outputs.
    - An output buffer would be filled periodically by the CPU and emptied at a regular rate of say 8000 samples per second to feed the

analogue to digital converter.
  - An input buffer would be filled by the analogue to digital converter and emptied by the CPU.
  - Leaky bucket idea.
  - The output buffer should never be empty, nor should either buffer ever get full.
- Problems caused by lack of synchronization:
  - Host 1 is sampling at 8001 Hz, and Host 2 is sampling at 7999 Hz. (We wouldn't hear this difference but because of the way the communication works via buffers it's bloody important.)
  - Host with the slower clock receives two extra samples per second.
  - These extra samples fill up the output buffer and cause it to overflow.
  - After ten minutes, 1200 extra samples will have been received that cannot be sent to the D to A converter. Also, as the buffer fills up there will be an increasing delay going up to 150 ms. Also, the it won't be sending out as many samples so it will get a buffer underflow.
  - This problem is solved by monitoring buffer levels and discarding single samples, or creating extra ones. Doing this during quiet sound is a good idea.
- Connect via a network:
  - Computer networks convey data in 'packets'.
  - Delays and imperfections are expected.
  - Protocol Layers:
    - '7-layer' OSI reference model:
      1) Physical Layer (lowest) e.g. Ethernet for wired LANs, IEEE802.11 for wireless LANs, PPP for modems etc.
      2) Data Link Layer e.g. Ethernet, IEEE802.11.
      3) Network Layer e.g. IP (Versions 4 & 6).
      4) Transport Layer e.g. TCP, UDP, RTP, RTCP.
      5) Session Layer
      6) Presentation Layer
      7) Application Layer (highest) e.g http, POP3, SMTP, DHCP, DNS, IMAP4, TELNET, FTP.
    - These layers are found on both sides of the communication, with the physical layer connection the two.
    - In the TCP/IP Reference model, layers 7-5 are known as the 'Application Layer', Layer 4 is the same, Layer 3, the Network Layer, is known as the Internet (IP) Layer, and Layers 2-1 are known as the 'Host-to-Network' or 'Link Layer'.
  - Network Layer: IP
    - This layer deals with routing and addressing IP packets (datagrams).
    - Header (about 20 bytes) holds information about the IP version number, header length, overall datagram length, 'time to live' (8 bits), a check-sum (16 bit). Source and destination IP address (32 bits each).
    - The routers can change the header (which means the checksum also has to be changed), every time packet is read by a router it's 'time to live' is decremented by one.

- Checksum:
  - If an error is found using the checksum the packet is discarded.
  - The exact IP checksum is made negative and put into the checksum field. The receiver then adds up all the 16 bit words (and with the now negative checksum) should get the answer of 0.
  - Cyclic redundancy check (CRC), is more powerful. It uses the remainder of a division (normally a high number like 16-bit or 32-bit), which is performed at both ends.
- This is a 'connectionless' service.

- Data-Link and Physical Layers:
  - Physical layer sends pulses representing 1's and 0's. This is the source of the errors.
  - The Data-Link layer is responsible for detecting and possibly correcting bit-errors. Ethernet can share one link with many users using a 'carrier sensing multiple access' (CSMA) mechanism.
- Transport Layer:
  - Gets packetised data transfer in a way that's suitable for application level using the IP layer below.
  - TCP - Transmission Control Protocol:
    - Connection oriented.
    - 'Reliable'.
    - Suitable for data that cannot tolerate any bit errors but can tolerate delay.
    - 'Port numbers' distinguish between data streams. Source and destination (16 bits).
    - Sequence numbers (32 bits):
      - Each byte is numbered. The header is the number of the first byte in the payload.
    - Acknowledgement Number (32 bits).
      - When the destination host receives the packet, it sends a packet back with an acknowledgement number which tells the original host which bytes its got up to.
    - Payload check-sum (16 bit).
    - SYN and Ack (1 bit each) are used to synchronise a connection between the two hosts.
    - FIN and AckF used similarly to end the communication.
  - UDP - User Datagram Protocol:
    - Connectionless.
    - 'Unreliable' - not such a problem for VoIP because it doesn't matter if the odd bit is wrong, or, if the packet is missing it could be replaced because of predictability (PLC).
    - 'Fire and forget'.
  - RTP - Real Time Transfer Protocol:
    - Adds a 'time stamp' and a 'sequence number' to the

data.
        - The receiver then knows how to order the data if it is received in the wrong order. Also 'packet-loss concealment' can be put into place. If the same packet it received twice it can also be deleted.
        - The RTP 'time stamp' can synchronise voice and video.
    - RTCP - Real Time Control Protocol:
        - Is used to determine how many packets are actually getting through and with what delay variation.
        - Expense = generating more datagrams.
        - Sends reports every 5 seconds or so, giving, the current 'quality of service', a measurement of the round trip delay (one way delay is very hard to measure), the percentage of lost packets and the average jitter (delay variation) of the time-span between reports.
    - (RTP and RTCP are sometimes considered application level.)
- Proxy and Location Servers:
    - Location server is the same as a DNS server.
    - Proxy Server is software that acts like both the server and the client to make requests on behalf of other clients.
    - Useful for admission control, preventing congestion, and having efficient access to the location server.
    - Can provide access from the network to a public switched telephone network (PSTN).
    - Encryption could also be determined by the proxy server.
    - Can set up P2P connections.
- Quality of Service in VoIP:
    - 'Jitter' buffer.
    - Jitter-buffer of size 0.1 seconds (i.e. 800 bytes with a 64 kb/s G711 speech) used to prevent data underflow.
    - Jitter buffered is used to increase the delay but reduce the number of lost packets.
- H323 Application Layer Protocol Suite
    - A set of International Telecommunication Union (ITU) protocols for setting up, maintaining & closing VoIP calls.
    - RTP/RTCP is used.
    - P2P is used.Typically 64 kb/s speech samples may be conveyed in blocks of 240 samples (30 ms). Video can also be sent simultaneously but synchronised.
    - Compression may be used and negotiated via H323.
    - Has 'gatekeepers' & 'gateways'.
- SIP (Session Initialisation Protocol) as defined by the IETF (Internet Engineering Task Force) to provide a simpler alternative to H323.
    - Addresses are URLs.
    - Uses RTP & RTCP and P2P.
    - Has proxy servers acting as gatekeepers.
    - Uses ports 5060 & 5061 for TCP and UDP.
- Mobile IP: Voice over Wi-Fi
    - Bit errors are more frequent, bandwidth more limited and congestion likely.

        - Forward Error Correction (FEC).
        - More bits placed into the packet to help with error detection and correction.
        - BIT OF CONFUSION HERE.
    - Packet Loss Concealment (PLC) Strategies:
        - Zero stuffing - just add no sound.
        - Copy what comes next from what's already happened, useful in vowels.
        - A more complex algorithm is needed to predict varying speech patterns.
    - IntServ and DiffServ
        - IntServ - reserve link capacity between routers for VoIP.
        - DiffServ - allows prioritisation of VoIP.
    - Transparency
        - This means for VoIP that it should:
            - Accommodate differences in data representation.
            - Not to worry about where resources are allocated.
            - Allow resources to move whilst in use.
            - Allow resources to be replicated.
            - Allow processing to be distributed.
            - Be robust to failure of components.

Axioms of Distributed Computing:

1. The network is reliable.
    - The physical fabric can fail in many ways.
    - Security flaws could cause the network to fail.
    - Message may not be sent properly.
2. Latency is greater than zero.
    - Measures the round trip time it takes to get somewhere.
    - Has been reduced 10 times in 10 years.
    - One should send messages of importance only.
3. Bandwidth is less than infinite.
    - The amount of information that can be transmitted within a given time.
4. The network is not secure.
    - Can have defences breached.
    - Spyware on the rise.
5. Topology does change.
    - Servers and clients are added and removed constantly.
    - Endpoints should provide location transparency or be discoverable.
6. There is more than one administrator.
    - Decisions regarding changes to the network shouldn't be made without ensuring that's fine with other administrators or the network might go down.
7. Transport cost is greater than zero.
    - The trip from the application layer down to the transport layer incurs a resource cost and adds to the latency. It must be marshalled and unmarshalled.
    - Interconnect infrastructure incurs significant costs.

8. The network is not homogeneous.
    - Communication technologies (e.g. wired/wireless) vary.
    - Protocols.
    - Representations (e.g. ASCI, Unicode).

Types of Transparency in a Distributed System:

- Access:
    - Hide differences in representation and access paths.
    - Compensate using wrappers, mappers or mediator middleware.
- Location:
    - Hide the fact that a resources location may not be known.
    - DNS is used for this purpose.
- Migration:
    - A resource may move from one node to another. Hide this.
    - Facilitated again by DNS.
- Relocation:
    - Same as migration only WHILST BEING ACCESSED OMG.
    - E.g. mobile clients.
    - The relocation us assumed to take place between accesses.
    - Websites with heavy traffic use replication strategies.
- Replication:
    - Hide that a resource may be in more than one node.
    - Also many online tools are expected to synchronise with that in desktop applications.
    - Keep the replicas consistent.
- Concurrency:
    - Hide that the same resource may be shared by different users.
    - E.g. wiki pages - hard, particularly when multiple users are changing the page.
- Failure:
    - Hide it.
    - e.g. if a web browser comes up with a timed out it doesn't mean the web server is down.
    - It's impossible to determine between a resource that's failed and an indefinitely slow one.

Distribution for Processing Power:

- Master and Worker split.
-