Distributed Systems

Lecturer: Maysoon Isleem
Introduction to Distributed Systems

Why do we develop distributed systems?

• availability of powerful yet cheap microprocessors (PCs, workstations), continuing advances in communication technology,

What is a distributed system?

A distributed system is a collection of independent computers that appear to the users of the system as a single system.

Examples:

• Network of workstations
• Distributed manufacturing system (e.g., automated assembly line)
• Network of branch office computers
What is a Distributed Database System?

A distributed database (DDB) is a collection of multiple, logically interrelated databases distributed over a computer network.

A distributed database management system (D-DBMS) is the software that manages the DDB and provides an access mechanism that makes this distribution transparent to the users.

Distributed database system (DDBS) = DDB + D-DBMS
Homogeneous Distributed Databases

In a homogeneous distributed database

- All sites have identical software
- Are aware of each other and agree to cooperate in processing user requests.
- Each site surrenders part of its autonomy in terms of right to change schemas or software
- Appears to user as a single system

In a heterogeneous distributed database

- Different sites may use different schemas and software
  - Difference in schema is a major problem for query processing
  - Difference in software is a major problem for transaction processing
- Sites may not be aware of each other and may provide only limited facilities for cooperation in transaction processing
Applications

Manufacturing - especially multi-plant manufacturing
Military command and control
Electronic Funds Transfer
Corporate MIS
Airlines
Hotel chains
Any organization which has a decentralized organization structure
Distributed DBMS Promises

- Transparent management of distributed, fragmented, and replicated data

- Improved reliability/availability through distributed transactions

- Improved performance

- Easier and more economical system expansion
Distributed Database - User View
Distributed DBMS - Reality

Communication Subsystem
Centralised Systems:

System shared by users all the time
All resources accessible
Software runs in a single process
Single physical location
Single point of control
Single point of failure
Decentralised Systems:

Multiple autonomous components
Components shared by users
Some resources may not be accessible
Software can run in concurrent processes on different processors
Multiple physical locations
Multiple points of control
Multiple points of failure
No global time
No shared memory (in most cases)
Models of Distributed Systems
Consequences of Distributed Systems

- **Concurrency**
  - p and q are concurrent if either p can happen before q, or q can happen before p, thus having interleaving semantics
  - Synchronization and Coordination are handled by passing messages
  - Potential problems are deadlocks and starvation
- **No global clock, no global state**
  - Not possible to synchronize many computers on a network and guarantee synchronization over time, thus events are logically ordered.
  - Not possible to have a process that can be aware of a single global state.
- **Independent Failures**
  - Running processes may be unaware of other failures within context
  - Failed processes may go undetected
  - Both are due to processes running in isolation
Reasons for Distributed Systems

- Resource Sharing
  - Hardware
    - Distributing CPU power
    - Sharing storage drives
    - Sharing printers
  - Network
    - Sharing home DSL line across multiple computers
    - Computers on a MAN sharing an OC-48 to connect to a WAN
  - Software
    - Sharing a database server with multiple client applications
    - File Sharing
  - Video and Audio
    - Video On Demand (VOD)
    - Digital Telephony and Voice Mail
Reasons for Distributed Systems

• Functional Distribution
  - Separating clients from servers
    • Web browsers from web servers
    • Web servers from database servers

• Load Distribution
  - Load balancing clusters

• Physical Separation
  - Connecting multiple locations of an enterprise to single data points
  - Distributing a web infrastructure across multiple WAN segments for improved reliability

• Economics
  - Leveraging many cheaper systems provides better price to performance rather than buying large mainframes
  - Linux Beowulf Cluster vs. Super Computer
Design Challenges

• Heterogeneity of Resources
  - Networks (AppleTalk, Token Ring, Ethernet, etc.)
  - Hardware (RISC vs. CISC)
    • Example: if you compile a C++ program for Solaris Intel-based, it is not portable for Solaris SPARC-based
  - OS (UNIX – and its many flavors, Windows, Linux, BEOS, etc.)
    • IPC issues
    • File sharing issues
    • Printer sharing issues
  - Programming Languages (Java, C/C++/C#, Perl, Python, LISP, etc.)
  - Implementation Strategies
    • Integrating a .NET application to a J2EE application can pose its difficulties
    • Integrating a WebSphere based J2EE application to an iPlanet based J2EE application can also pose difficulties!
Design Challenges

• Openness
  - Published Applications Programming Interface (API)
  - Standards Compliance

• Security
  - Types
    • Confidentiality
    • Integrity
    • Availability
    • Non-Repudiation, prevalent in commerce applications
  - Authentication and Authorization
Design Challenges

• Scalability (Systems grow with time or become obsolete. Techniques that require resources linearly in terms of the size of the system are not scalable. e.g., broadcast based query won’t work for large distributed systems.)
  
  - Physical resources
  - Performance
  - Prevent resource loss
  - Bottlenecks
    o Centralized components: a single mail server
    o Centralized tables: a single URL address book
    o Centralized algorithms: routing based on complete information

• Caching/Replication
Design Challenges

- Failure Handling
  - Detection
    - e.g. checksums
    - Some errors are undetectable, such as in the case of Asynchronous messages
  - Masking failures
    - Retransmission
    - Striping
    - Tolerance by means of meaningful action
      » Transaction rollbacks
  - Redundancy
    - Multiple pathways
    - DNS example
    - Database replication
    - Web clustering
    - Linux Beowulf cluster
Design Challenges

- Concurrency
  - Session management
  - Resource management
  - Locking
- Transparency
  - Hiding it all from the users
  - ISO's Reference Model for Open Distributed Processing identifies the following 8 forms of transparency
    - Access (remote and local access are the same)
    - Location (don't care about where resources are)
    - Concurrency (share resources fairly)
    - Replication (don't care which replicated resource is being used)
    - Failure (hide faults from applications and users)
    - Mobility (move resources without affecting users)
    - Performance (adapt to load changes)
    - Scaling (expand capability, leave logic alone)
Modeling of distributed systems

abstraction:

- to capture properties that are common to a large range of systems so that it enables to distinguish the fundamental from the accessory
- to prevent reinvent the wheel for every minor variant of the problem

a model abstracts away the key components and the way they interact

purpose:

- to make explicit all relevant assumptions about the system
- to express behaviour through algorithms
- make impossibility observations etc through logical analysis including proofs
Modeling of distributed systems

abstracting the physical model: processes, links and failure detectors (latter an indirect measurement of time)
Modeling of distributed systems

component properties:

• channel (a communication resource) - message delays, message loss

• process (a computational resource, has only local state) - can incur process failure, be infinitely slow or corrupt

low level models of interaction: synchronous message passing, asynchronous message passing
Modeling of distributed systems

failure detector abstraction: a possible way to capture the notion of process and link failures based on their timing behaviour

incorporation of a failure detector, a specialized process in each process which emits a heartbeat to others

a failure detector can be considered as an indirect abstraction of time; simply a timeout is an indication of a failure, mostly unreliable with an outcome either suspected or unsuspected

a synchronous system => a ‘perfect failure detector’
Modeling of distributed systems

clock: physical and logical

abstracting a process: by the process failure model
Modeling of distributed systems

crashes: a faulty process as opposed to a correct process (which executes an infinite number of steps) does no further local computation or message generation or respond to messages

• a crash does not preclude a recovery later but this is considered another category

• also the correctness of any algorithm may depend on a maximally admissible number of faulty processes
Modeling of distributed systems

- arbitrary faults: a process that deviates arbitrarily from the algorithm assigned to it

  • also known as malicious or Byzantine faulty or in fact may be due to a bug in the program

  • under such conditions some algorithmic abstractions may be 'impossible'
Modeling of distributed systems

omission failure: due to network congestion or buffer overflow, resulting in process unable to send messages

Crash-recovery: a process simply crashes fail-stop or, crashes and recovers infinite times

- every process that recovers is assumed to have a stable storage (also called a log) accessible through some primitives, which stores the most recent local state with time stamps

- alternatively those which do never crash could also act as virtual stable storage
Modeling of distributed systems

abstracting communication: by loss or corruption of messages, also known as communication omission

usually resolved through end-to-end network protocol support unless of course there is a network partition

Desirable properties for ‘reliable’ delivery of messages

- **Liveness**: any message in the outgoing buffer of sender is ‘eventually’ delivered to the incoming message buffer of receiver

- **Safety**: the message received is identical to the one sent, and no messages are delivered twice
Modeling of distributed systems

Abstracting other higher level interactions
e.g., capturing recurring patterns of interaction in the form of

• distributed agreement (on an event, a sequence of events etc.,)

• atomic commitment (whether to take an irrevocable step or not)

• total order broadcast (i.e., agreeing on order of actions) leads to a wide range of algorithms
Modeling of distributed systems

Predicting impossibility results in higher level interactions

- due to in some cases indistinguishability of network failures from process failures or, a slow process from a network delay

- e.g., agreement in the presence of message loss, agreement in the presence of process failures in asynchronous situations

Impossibility of agreement in the presence of message loss

- leads to a widely used assumption in almost all models
- typical two army problem
- formal model described below