Definition of a DS

„A collection of independent computers that appear to its user as a single coherent system."

Challenges in DS

- Transparency, Heterogeneity, Failure Handling, Openness, Scalability and Openness

Hardware Architecture

- Multiprocessors: multiple processors share a pool of memory
- Multicomputers: multiple processors with private memory are interconnected

Software Architecture

- Distributed OS
  - Multiprocessors
  - Multicomputers
- Network OS
- Middleware-based OS

Types of Network Interaction

- Peer processes
- Cluster of servers
- Web proxy server
- Code mobility → e.g. Applet
- Thin clients

ISO/OSI Layers

- Layer 1 (physical layer): electrical/mechanical/physical signaling interfaces
- Layer 2 (data link layer): groups bits into frames and adds some extra information (starting and ending bits pattern, sequence number, checksum)
- Layer 3 (network layer): routes packets towards the destination (normally IP)
- Layer 4 (transport layer): provides end-to-end functionality and splits application messages into packets (message fragmentation) (normally over TCP or UDP)
- Layer 5 (session layer): used for synchronization (not used in practice)
- Layer 6 (presentation layer): deals with the meaning of bits
- Layer 7 (application layer): all distributed systems are here (protocols: HTTP, FTP, SSH, SMTP, ...)

Type of Connections

- Connection-oriented: negotiation before communication
- Connectionless: no setup & no termination

Sockets

- UDP & Sockets: multiple clients can be accessing the server immediately
- TCP & Sockets: we need threads to support multiple concurrent clients

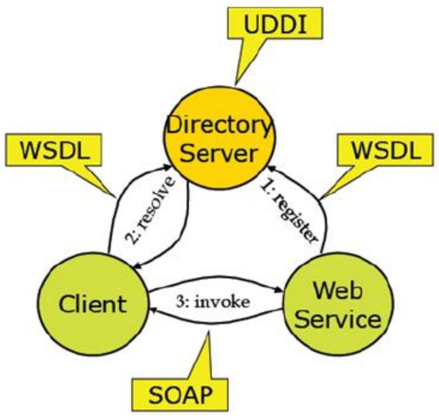
RPC

- Stub on the client machine
- Implementation of the stub with a local call on the server
- Problems
  - Data representation (different encodings)
  - Passing arguments (pass-by-reference) → Solution: use copy/restore

Distributed Objects

- Client: Proxy object with same interface as Object
- Server: Object and Skeleton which handles access to Object

Web Services



SOAP

Defines how a client can invoke a remote procedure by sending a SOAP message and how the server can reply by sending another SOAP message back.

WSDL

XLM syntax for formally describing how to invoke a web service and to communicate with it

UDDI

Provides a standard and flexible way to discover where a web service is located and where to fine more information about what the web service does

Web Services vs. CORBA

- CORBA is used in organizations, Web Services are used in the Internet
- CORBA identifiers aren't generally understood
- HTTP/XML is easy to learn, CORBA has a steeply learning curve
- CORBA has transactions, concurrency control, security access control and persistent objects

Naming Types

- Hierarchical Naming
- Flat Naming
- Attribute-based Naming

Hierarchical Naming

- Each name is a path in the naming graph
- Hard Links: a node may have multiple names within the same name space → multiple paths that lead to the same leaf node
- Symbolic Links: a special node may contain the absolute (or relative) name of another node
- Mounting: a symbolic link may refer to a remote name space
- Merging name spaces
    - Problem: absolute names of all name spaces are changed
    - Solution: at root node cache the original top level names

DNS (Distributed Name System)

- Iterative Name Resolution: 2 * N messages

- Recursive Name Resolution: N + 1 up to 2 * N messages

Flat Naming

Very common in centralized systems (e.g. memory addressing) but very complicated in DS (DHT necessary)

Location Service

- Pointer caches: caching a reference to a directory node at the lowest-level domain in which an entity will reside most of the time
- Problem of unreferenced objects: → Solution: reference counting

Time Synchronization

- Synchronization with a time server
    - Problem: transmission delay of messages
    - Solution: Cristian's algorithm → transmission delay is estimated
- Logical clocks
    - Useful if absolute synchronization is no needed but ordering is necessary
- Lamport Timestamps
    - Wenn ein Prozess eine Nachricht empfängt, die logisch später abgesendet als empfangen wurde, korrigiert der Empfänger seine locale Uhr, indem er den Zähler um mindestens 1 weitersetzt, als der Zeitstempel der Nachricht

Mutual Exclusion

- Requirements: safety, liveness, ordering
- Centralized Approach (1 Coordinator)
    - Advantages: easy to implement, few messages necessary, fair (FIFO), no starvation
    - Drawbacks: single point of failure, processes can't distinguish between dead coordinator or busy resource
- Distributed Approach
    1) Interested node sends CS name to all other nodes
    2) Receiving nodes reply:
        a. Ok if not interested
        b. OK if interested but newer timestamp
        c. If older timestamp, put sender in queue and don't reply
        d. If already in CS, put sender in queue and don't reply
    3) Enter CS if all nodes send OK
    4) When node exits CS send OK to all nodes in queue
        - Drawbacks: more messages: 2 * (n – 1) and no single point of failure but n points of failure
- Token-Ring Approach
    - Node can enter CS if he owns the token
    - Advantages: very simple and no starvation
    - Drawbacks: messages: 1 until ∞, problem if the token is lost

Leader Election

- Bully algorithm
    1. Node notice that coordinator is dead → start election process
    2. Node sends message to all nodes with higher numbers → if no response → node is elected
    3. If node gets election message and has higher ID → sends OK and starts election process
    4. Process with highest ID sends Coordinator message to all other nodes
- Ring algorithm
    1. Node notice that coordinator is dead and starts election
    2. Sends election message to its successor with a list containing only its own ID

3. When a node gets an election message that originated at a different node, the node appends its ID and forwards message to successor node
4. When a node gets back its own election message, it picks the highest ID in the list and announce this node as coordinator to all other nodes

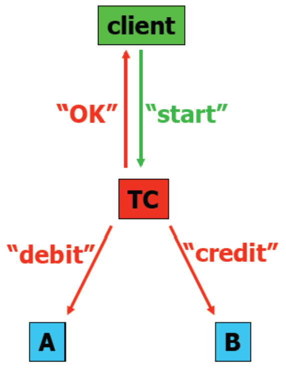
Vector Clocks

- Lamport's timestamp can be used for total ordering, but dependencies (causality) is lost
- Solution: each node has a vector of N logical clocks
- Algorithm:
  1. All vector clocks are initialized with zero
  2. When event happens on a node → increase own clock by 1
  3. When sending a message include whole vector
  4. When receiving a message the node updates each element in his vector with the maximum from both vectors
- An event A is considered to happen before B, if all elements from A's vector are less or equal than the elements of B's vector

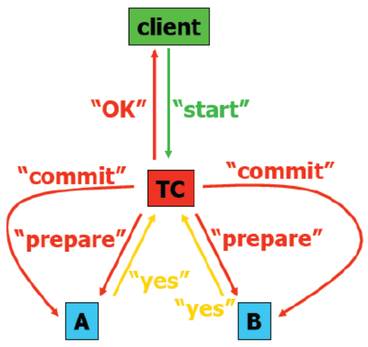
Atomicity

- Serializability → complete order
- Recoverability → each operation executes completely or not at all

One-Phase-Commit (1PC)

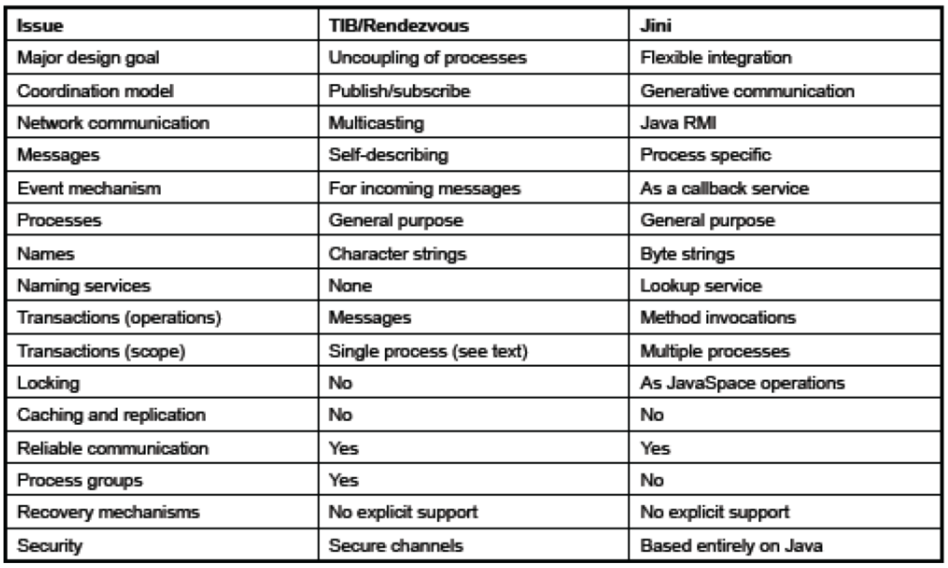


Two-Phase-Commit (2PC)



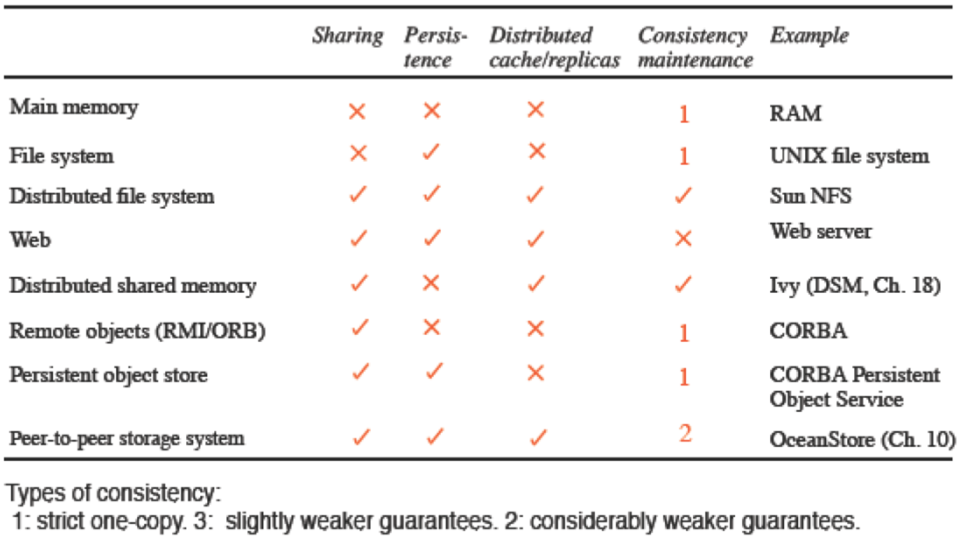
Advantage: fullfills correctness property
Drawback: doesn't obey liveness property
Solution: introduce timeouts

Middleware Systems



Issue	TIB/Rendezvous	Jini
Major design goal	Uncoupling of processes	Flexible integration
Coordination model	Publish/subscribe	Generative communication
Network communication	Multicasting	Java RMI
Messages	Self-describing	Process specific
Event mechanism	For incoming messages	As a callback service
Processes	General purpose	General purpose
Names	Character strings	Byte strings
Naming services	None	Lookup service
Transactions (operations)	Messages	Method invocations
Transactions (scope)	Single process (see text)	Multiple processes
Locking	No	As JavaSpace operations
Caching and replication	No	No
Reliable communication	Yes	Yes
Process groups	Yes	No
Recovery mechanisms	No explicit support	No explicit support
Security	Secure channels	Based entirely on Java

Distributed File Systems



	Sharing	*Persis-tence*	*Distributed cache/replicas*	*Consistency maintenance*	*Example*
Main memory	×	×	×	1	RAM
File system	×	✓	×	1	UNIX file system
Distributed file system	✓	✓	✓	✓	Sun NFS
Web	✓	✓	✓	×	Web server
Distributed shared memory	✓	×	✓	✓	Ivy (DSM, Ch. 18)
Remote objects (RMI/ORB)	✓	×	×	1	CORBA
Persistent object store	✓	✓	×	1	CORBA Persistent Object Service
Peer-to-peer storage system	✓	✓	✓	2	OceanStore (Ch. 10)

Types of consistency:
1: strict one-copy. 3: slightly weaker guarantees. 2: considerably weaker guarantees.

- Requirements
  - Transparency (access, location, mobility, performance and scaling)
  - Concurrent file updates
  - File replication
  - Hardware & software heterogeneity
  - Fault tolerance
  - Consistency
  - Security
  - Efficiency
- Examples: Network File System (NFS), Andrew File System (AFS) and Google File System

P2P Definition

P2P is a class of systems where:

- Resources available at the edges of the internet are utilized
- Service is carried out collectively
- Irregularities and dynamicity are treated as the norm

Main Advantages of P2P

- Inherently scalable (higher demand → higher contribution)

- Increased (massive) aggregated capacity
- Utilization of otherwise wasted resources
- Distribution of load and administration
- Designed to be fault tolerant
- Inherently handles dynamic conditions

Overlay Types

- Unstructured P2P (centralized, pure or hybrid)
    - Any two nodes can establish a link
    - Topology evolves at random
    - Topology reflects desired property of linked nodes
- Structured P2P
    - Topology strictly determined by node ID

P2P Characteristics

- Overlay maintenance
    - Bootstrapping (how to join the system)
    - Continuous maintenance (how to handle changes and faults)
- Scalability: avoid central server, distribute load on multiple peers and limit load per peer
- Fairness: → user behavior: give incentives for fair play
- Dynamicity and Adaptability: changing topology, data, profiles or load
- Fault tolerance: robustness of the topology and resistance to node and link crashes
- Self-organization: no one keeps full state → nodes take local decisions
- Performance: reduce network latency → locality
- Privacy: anonymity, reputation and resistance to censorship
- Security: defend against attack against the network (e.g. DDOS)
- Legal issues: copyright violations
- Simplicity: provide abstraction to hide complexity

Application Areas of P2P

- Content Sharing (e.g. File Sharing)
- Distributed Storage (e.g. PAST)
- Shared Bandwidth during Content Distribution (e.g. WOW Patch)
- Collaboration (e.g. Instant Messaging)
- Sharing CPU (e.g SETI@home)

File Sharing Applications

- Napster: central index
- Gnutella: pure P2P and search by flooding
- KaZaa: hybrid P2P

BitTorrent's Peer Set

- Tracker picks peer at random on its list
- Once a peer is incorporated in the BitTorrent session, it can also be picked to be in the peer set of another peer → a peer knows both older peers and newcomers
- A peer communicates with its initial peer set and the other peers that contacted it but not with other peer sets

BitTorrent's Chunk Selection Policy

- Which missing chunk should be requested first?

    - Simple: random selection
    - Biased strategy: rarest-first policy
- Reasons
    - Rare chunks can more easily be traded with others
    - Maximize the minimum number of copies in any given chunk in each peer set
- BitTorrent uses rarest-first policy, except for newcomers

BitTorrent's Peer Selection Policy

- Seeders' policy: the ones that offer the best upload rate
- Leechers' policy: the ones that also serve us: tit for tat

➔ to find better hosts: unchocke peers randomly (newcomers with higher priority)

Distributed Hash Tables (DHT)

- Store particular content on particular node
- In a network with N nodes, each node is expected to hold 1/N of the items

Pastry

- Circular m-bit space → addresses have m/b digits
- A key is mapped to the node whose ID is numerically-closest to the key
- Routing can be done in O(log(n)) hops

Pastry Routing Algorithm

1. Directly to correct node if this node is in leaf set
2. A node in the routing tale with a longer prefix
3. A node in the routing table with the same prefix length but numerically closer

Pastry Routing Table

- Leaf Set
- Routing Table
- Neighborhood Set

Chord

- m-bit ID space
- Each key is mapped to its successor node
- Each node is responsible for O(K/N) keys

Lookup in Basic Chord

- Each node knows successor and predecessor
- Forwarding requests around the ring trough successor pointers

➔ Routing is possible in O(N) hops

Lookup in Complete Chord

- Each node has m fingers → finger(i) points to node on or after 2^i steps ahead
- O(log(N)) states per Node
- Lookup is achieved by following longest preceding finger, then successor pointer
- O(log(N)) hops

Node Joining in Chord

1) Initialize predecessor and all fingers of new node j
    a. Locate any node n in the ring

   b. Ask n to lookup the peers at j + 2^0, j + 2^1, j + 2^2, …
   c. Use result to populate finger table of j
2) Update predecessor and fingers of existing nodes
   a. New node j calls update function on existing nodes that must point to j (nodes in the range [j-2^i, pred(j)-2^i+1]) → O(log(N)) nodes
3) Transfer some keys to the new node
   a. Connect to successor
   b. Copy keys from successor to new node
   c. Update successor pointer and remove keys

Node failure/leaving in Chord

After failure n contacts first alive successor and updates successor list

Map Reduce

- Framework für nebenläufige Berechnungen über grosse Datenmengen auf Computercluster
- Examples: Hadoop and Google MapReduce