Distributed Applications
TUM Summer Term 2014
Lecturer: Prof. Schlichter

Janosch Maier

5. Juni 2014

Inhaltsverzeichnis

1 Introduction

1.1 Background

- Production flows (manufacturing)
- Money flow (banking)
- Information flow

1.1.1 Internet Computing

- Shared Resources
- Information Communication
- Activity Coordination
- Examples: Online flight-reservation, ATMs, WWW, Grid Computing, ...

1.1.2 Enterprise Computing

- Applications ↔ Network ↔ Database Systems
- Close coupling of applications on heterogenous platforms over network
- Reliability: Consistency, Security / Privacy, Response time, Error tolerance, Autonomy of components

1.2 Key Characteristics of Distributed Systems

1.2.1 Motivation

- Cheaper processors, storage
- High bandwith
- Complex applications
- Cooperative applications (CSCW)

1.2.2 Definitions 'Distributed System'

- Tanenbaum: Independent computers appearing as single computer
- Lamport: Stops work if machine unknown to user crashes
- Working Definition: Hardware and software of network computers communicate & ccordinate actions (through messages)

1.2.3 Methods of Distribution

- Hardware components
- Load
- Data
- Control (e.g. distributes os)
- Processing (e.g. map-reduce-algorithm)

1.2.4 Properties of Distributed Systems

- Existence of multiple functional units (physical, logical)
- Distribution of functional units
- Independent breakdown of units
- Distributed component control
- Transparency (unit distribution hidden to user)
- Cooperative autonomy

1.2.5 Challenges of Distributed Systems

- Heterogenity (of networks, hardware, os, languages, ...) ⇒ Middleware needed
- Openness ⇒ Standardized interfaces
- Scalability
- Security & Privacy

1.2.6 Examples for Development Frameworks

- NFS – Network File System (SUN)
- ONC – Open Network Computing (SUN)
- ODP – Open Distributed Processing (ISO)
- CORBA – Common object Request Broker Architecture (OMG)
- J2EE – Java 2 Platform Enterprise Edition (SUN)
- .NET – Framework (Microsoft)

1.3 Distributed Applications

Set of cooperating, interacting functional units ⇒ Parallelism, Fault tolerance, Inherent Distribution

1.3.1 Definition

- Application A, split into components $A_1, ..., A_n; n \in \mathbb{N}, n > 1$. Each A_i has internal state (data) and operations
- Components A_i are autonomous, can be assigned to different machines F_i
- Components A_i exchange information via network

1.3.2 Interfaces

- Well-defined interaction points between components
- Specify component operations and commpunication
  - Parameters (+ types)
  - Results (+ type)
  - Side-Effects (e.g. data entry)
  - Effects on subsequent operations
  - Constraints

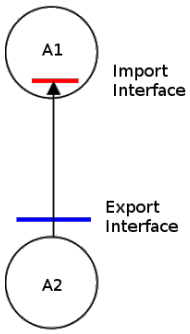

Abbildung 1: Interfaces between two applications

1.3.3 Distributed vs. parallel programs

- Distributed
  - Ganularity: Coarse
  - Data Space: Private
  - Failure Handling: In communication protocol
- Parallel
  - Ganularity: Fine
  - Data Space: Shared
  - Failure Handling: Not considered

1.4 Influential Distributed Systems

1.4.1 Mach

- OS from Carnegie-Mellon Univeristy for Multiprocessor / Distributed Applications
- Goals: Unix-Emulation, Transparency, Portability
- Architecture: Process with execution environment for secure resource access, Threads as distribution unit (Thread on single system), Shared-Memory-Objects possible (Copy-on-Write)
- Message Exchange through ports (protected by capabilities), Network messages possible

1.4.2 NFS

- Distributed File Management (e.g. on Unix)
- Characteristics: Server Exports (*/etc/exports*), Client Mounts (Host, Remote Path, Local Path), Automounter on access possible, Access Transparency
- Implementation: RPC calls between systems, UDP or TCP possible, stateful (used to be stateless)

1.4.3 J2EE

- Distributed application server environment
- Objectives: Standardized programming environment for enterprise applications, Java-based, Component-based, Network-oriented, Runtime infrastructure + Java extensions APIs
- Architecture: Application server (runtime environment), J2EE container, data storage
- Enterprise Java Beans (EJB): server-side managed infrastructure, bean offers business interfaces, 3-tier applications
- J2EE container: runtime environment for application, API access, e.g. JavaServlets within WebContainer
- J2EE application: Modules with application components e.g. jar-file
- Java Server Pages: XML-like-tags, e.g.:

```
<\% code fragments \%>
  <\% if (value.getName().length != 0) { \%>
    <H2>The value is: <\%= value.getName()\%></H2>
  <\%} else {\%>
  <H2>Value is empty</H2>
<\% }\%>
```

1.4.4 Google

- One of largest distrubited systems toda
- Physical infrastructure: Commodity pcs in racks in clusters in data centers
- Middleware: Buffers + publish/subscribe
- Distributed Computation: map-reduce-algorithms

2 Architecture of distributed systems

2.1 System models

Describtion of a distributed system

2.1.1 Architectural model

Interaction between components, mapping to network

- Software layers: applications / services, middleware, operating system, computer / network devices
- Middleware: Provide homogenity for programming: Hide (communication) complexity, communication, persistence, trinsaction, ..., Categories: distributed component, peer-to-peer
- System architecture: Client-Server, Proxy server, Peer process, Community of software agents

2.1.2 Interaction model

Performance and time-limits

- No single global time $\Rightarrow$ Logical clocks for synchronization
- Different order of messages at different recipients
- Consistent ordering

2.1.3 Failure model

How can failure occure + handling

- Crash fault: Hardware / Software failure
- Message lost: Buffer overflow, Router congestion
- Fail stop failures: System crashes, Partners are notified
- Timing failures: Clock not sychronized, Transmission timeout
- Arbitrary failures: Process steps not run / unintendetly run / wrong messages
- Malicious Byzantine failures: Messages replay, Program modification

2.1.4 Security model

Possible threats + handling

- Secure communication
- Unauthorized access
- Message authentication

2.2 Transparency

- Location Transparency: Location of object (resource or service) is in dstributed system
- Access Transparency: Access to object independent from their location
- Replication Transparency: User unaware if object is replicated (e.g. for fast access)
- Migration Transparency: Object's location might change without influencing application behavior
- Host Migration Transmission: Same environment for cumputer independent from subnetwork (offline/online migration possible)
- Language Transparency: Interaction between components independent from programming language
- Failure Transparency: Partial failures masked by system
- Concurrency Transparency: Shared access to objects possible
- Execution Transparency: Processes may be run on different runtime systems
- Performance Transparency: Dynamic reconfiguration for load balancing
- Scalability Transparency: Scaling possible without structural / algorithmic changes

2.2.1 Goals for distributed applications

- Realization of several transparency levels
- Problem CSCW: Not always group awareness, Selective transparency: location & access but NO strict concurrency transparency.

2.3 Paradigms for distributed applications

2.3.1 Information Sharing

- Communication using shared, integrated information managements
- No direct communication (shared memory)

2.3.2 Message exchange

- Interprocess commpunication (IPC): Message between sender and receiver
- send(E: receiver, N: message);
- receive(S: sender, B: buffer);
- Sender / Receiver perspective possible
- Asynchronous message exchange (nonblocking)

  - S resumes processing, after N is put to message queue / message buffer (NP)
  - E repears receive operation until message arrives
  - Advantages: real-time, paralle execution, event signaling possible
  - Disadvantages: buffer management, notification of failures, design difficult

- Synchronous message exchange (blocking)
  - S blocked until E has received message
  - E is blocked until N is completely loaded to NP
  - Decoupling to avoid endless waiting: timeout, threads for message handling

- Remote-invocation send: S suspends execution until E has recieved and processed request which was part of sent message

2.3.3 Naming entities

- Names: unique character string refering an entity, invokation points bound to names (addresses, e.g. Apache bound to hostname)
- Name space: Organization of names, hierachical, labeled directed graph, absolute vs. relative patkh names
- name resolution: lookup of names

2.3.4 Bidirectional communication

- Request-answer scheme for message exchanges
- Sockets as low level abstraction: os-controlled interface for applications, identification: ip, port
- Call semantics: dealing with message loss, crash of S or E
  - at-least-once semantics: Operation processed once or several times
  - exaclty-once semantics: Processed once; result stored in case of lost answer
  - last semantics: Processed onece or several times; only last processing produces result
  - at-most-once semantics: Processed once or not at all. If processesd, similar to exaclty-once

2.3.5 Producer-consumer interaction

- Fire & forget interaction: Producer (S) directly resumes execution
- Special case: Piping (After sending, producer terminates)

2.3.6 Client-server model

- Central server provides service to requesting clients
- Request-Answer Interaction: Handshaking principle – Client suspends till server returns request response
- Service-oriented architecture (SOA): abstract archtectural approach – loose coupling / dinamic binding, modularized software, service manages its own data, 3 roles: service requestor, provider & registry, Web services moslty SOA
- E.g. webserver: stateless (session handlung on application layer)

2.3.7 Peer-to-peer model

- All processes similar role: cooperative interaction, no clients/servers
- client-server (server maintainance, client fewer resources) vs. p2p (similar resources for each peer, direct communication)
- Issues: Peer discovery, data location, file exchange, security/privacy
- Napster: Central server with music list
- Gnutella: no server directory – loose federation of gnutella computers, On connection at least one address needed, Find a file: send request to neighbours, propagate through network
- BitTorrent: recipients also propagate data to newer recipients
- eDonkey: Files identifieds compound md4 hash sums
- Gossip-based Approach: Information propagation similar to epidemic deseases: exponential rate: everybody tells one pearson in each step $\Rightarrow$ Spreadrate 1.8^k after k rounds. Combination of push and pull works best, robust & scalable algorithm

2.3.8 Group model

- Combine set of components into group (e.g. Service provided by group of servers)
- Important: Shared problem, information exchange, group awareness, corrdination
- Used in CSCW

2.3.9 Publish-Subscribe model

- Publish structured events
- Subscribe to particular events
- System matches subscribtions against events and sends notifications: heterogenous environment, asynchronous notifications)

2.3.10 Taxonomy of communication

- Message serialization: Messages to group are received in different order
- One sender:
  - Arrival time by receiver
  - Sequence number by sender
  - Own serialization criteria by receiver
- Several senders
  - No serialization
  - Loosely-synchronous: Loosely synchronized global time
  - Virtualle-synchronous: Order determinde by causal interdependencies (N send after M received, N might depend on M)
  - Totally ordered: By token / By coordinator

2.3.11 Levels of Abstraction

- Object Space, Collaborative applications
- Network services, Object request broker
- Remote procedure call, Remote Method invocation
- Client/Server, P2P
- Message Passing

2.4 Client-server model

Implements Handshaking principle

2.4.1 Terms and Definitions

- Sender/Receiver: Message exchanging vs Client/Server: Entities in specialized protocol
- Client: process (application) running on client machines, (typically) request for service operations (a priori unknown)
- Service: Software providing service operation running or one ore multiple machines
- Server: Subsystem providing service to clients. Executes software on server machine. Server machine can host multiple server subsystems
- Client-Server interfaces
  - Client interface / import interface: represents server within client
  - Server interface / export interface: represents all potential client within server
- Multitie architecture: One machine as client&server between actual client and server (e.g. webserver between browser and application server)

2.4.2 Concepts for client-server applications

- Remote data storage: NFS (Client: presentation & execution – Server: database)
- Remote presentation: X window system (Client: presentation – Server: presentation, execution & database)
- Distributed application: Cooperative processing (Client: presentation & execution – Server: execution & database)
- Distributed data storage: Data distributed between client and (Client: presentation, execution & database – Server: database)

2.4.3 Processing of service requests

- Different life spans for client/server. Server manages requests in queue
- Single dedicates server process: not parallel, no interruption when higher prioritized request, bottleneck
- Cloning of new server processes: expensive, synchronization, parallel processing possible
- Parallel request processing through threads: shared address space

2.4.4 File service

- Centralized data storage: Client for display, quick interaction times, caching for speed increases
- Stateless server: Client supplies al parameters to process request, cache refresh done by client, often: write-through, server crash not influencing client
- Stateful server: Server tracks clients and actions, cache owned by server, clients less complex, after server crash: abort message to server

2.4.5 Time service

provides synchronized time for network nodes

2.4.6 Name service

- Name management for clients (sometimes called directory service)
- Datastructure: {name, adress, access information, attributes}
- Example: DNS – hierachial, distributed databases across logical network of nameservers

2.4.7 Lightweight Directory Access Protocol (LDAP)

- Access and update of directory information
- Directory: List of objects in order, with meta-data – high volume of reads, no transactions, different query languages
- Directory service: name service containing object names and meta-data
- Queries
  - In directiories: Based on names & meta-data
  - White Pages: Object access based on name
  - Yello Pages: Object access based on meta-data
- LDAP models: information models (data structures), naming model (referencing objects), functional model (communication), security model (access control)
- LDAP architecture: Client/serer based on TCP/IP, string for data representation
  - Client initiates session with server (IP, port, username, password) – Binding
  - Client invoces operation (read, write, seek)
  - Client terminates – Unbinding
- Information model: Entry describes object with distinguished name (DN), set of attributes (meta-data) with type and value(s)
  - Attribute syntax: bin (binary), ces (case exact string), cis (case ignore string), tel (telephone number), dn (distinguished name), generalized time, postal address
  - Schemas for entries based on attributes: E.g. Person: entry for one person, attributes commonName (cn), surname (sn)
- Naming model: DN contains of relative distinguished names (RDN), hierarchically structured as Directory Information Tree (DIT), DIT supports aliases, distribution acress servers possible
- Functional model: Operatitons for access/modification. E.g. create, delete, update (e.g. move in DIT), compare, search (Find postal address for cn=John Smith,o=IBM,c=DE; Base object stparting point; scope: baseObeject, singleLevel, wholeSubtree; search filter possible)
- LDIF: LDAP Data Interchange Format: Import/Export directory information

2.4.8 Failure tolerant services

- Modular redundancy: Multiple redundant services, copies grouped into server/client groups
- Primary-standby-approach: One replica as master, At checkpoints, status is propageted to replicas, On error, master is replaced by replica, hot vs. cold standby

3 Remote Invocation (RPC/RMI)

3.1 Introduction

3.1.1 Local vs. remote procedure call

Local:

- Caller → request → procedure
- Caller ← answer ←

Distributed:

- RPC similar, Single thread responsible for data transfer
- Interface between remote systems

3.1.2 Definition

- Birrell & Nelson: Synchronous flow of control & data through pocedure calls between seperate adress spaceses via small channels
- Synchronous: Client blocked
- Procedure calls: format defined by signater of called procedure
- Different address spaces: No globac memory space; Pointer handling needed
- Smal channel: reduced bandwith

3.1.3 RPC properties

- Client/Server cannot assume that procedure call performed via network
- Control flow: S registers service, C binds to S, Request, Responce
- Differences betwen RPC and local procedure call
  - Caller & callee in different processes
  - No shared address space, No common runtime environment, Different lifetime
  - Errorhandling must consider communication failures
- Basic RPC characteristics
  - Uniform call semantics
  - Type-checking
  - Parameter functionality
  - Optimize response times
  - New error cases
- RPC and OSI

  - Application layer: client-server model
  - Presentation layer: RPC (hide communication details behind procedure call, bride heterogenous platforms)
  - Session layer: message exchange (OS interface)
  - Transport layer: transport protocols (transfer of data packets)
- RPC vs. message exchange
  - Synchronous – Asynchronous
  - 1 primitive operation – 2 primitive operations (send, receive)
  - Messages configured by RPC system – Messages configured by programmer
  - One open RPC – Several parallel messages possible
- RPC exchange protocols: Request (R), Request-Reply (RR), Request-Reply-Acknowledge (RRA)

3.2 Distributed applications based on RPC

3.2.1 Distributed application

- Stubs to make network interfaces transparent
  - Encapsulate distribution specific aspects
  - Represent interafces
  - Client stub: Proxy definiton of remote procedure P – Specification of remote service operation, assgning correct server, parameters in transmission format, decoding results, ublocking client
  - Server Stub: Proxy call for procedure P – Decoding paremeters, address of service operation, invoking operation, prepare and send response
- Implementing a distribution application
  - Manual stub implementation error-prone → RPC generator (declarative interface describtion)
  - Applying RPC generator: See picture on page 70

3.2.2 RPC language

- Declarative language specifing interface between component of a distributted application
- Interface attribute list: version of RPC system, fixed ports for invokation: interface identifier, constant declarations, type declarations, operation declarations

3.2.3 Phases of RPC based distributed applications

- 3 Phases: Design & implementation, Binding of components, Invocation
- Component Binding (Linking of components to enable RPC calls)
  - Static: Binding, when client generated; Server address hard-coded
  - Semistatic: Client determines server while initalization of client process (static for lifecycle); Address via database, broad-/multicast message, nameservice, mediation mechanism (broker/trader)
  - Dynamic: Server address determined when RPC is performed; Advantages: server migration, binding to alternative servers, dynamic server replacement.
- Mediation and brokering (registry, broker, trader)
  - Functionality: Server registers interfaces with broker (export interface), broker supplies client with interface information (import interface)
  - Broker information: Information about interfaces: names (white pages), types (yellow pages), behavioral/functional attributes (static: functionality, cost, bandwith / dynamic: server state)
  - Handling client requests: Direct – communication between C and S vs. Indirect – Communication between C and S only via broker V

3.3 Remote Method Invocation (RMI)

Communication between different Java virtual machines

3.3.1 Definitions

- Remote object: Can be called by object within another JVM (on another computer)
- Remote interface: Java interface specifying remote object
- Remote method invokation (RMI): objet-to-object communcation; Invocing a method of remote interface; Same syntax as local call

3.3.2 RMI characteristics

- location & access transparency
- localization
- communication with remote objects
- automated class loading
- Clients interact with remote interfaces (not classes)
- RMI workflow: S registers with RMI registry (nameservice), C looks S up in registry, C receives stub for S, C calls objects like a local object (communication via stubs & skeletons)

3.3.3 RMI architecture

- Application layer: client method invokation vs. remote object
- Presentation layer: stub (proxy object) vs. skeleton
  - Interception of client method calls; redirection to remote object
- Session layer: remote reference layer (client & server)
  - Connection via 1-to-1 link
  - Java Remote Method Protocol (JRMP) vic TCP/IP
  - Mapping of stub/skeleton to transport protocol
  - Method invoke

3.3.4 Locating remote objects

- Nameservice: RMI registry (mapping, stand-alone Java application, runs on all remote machines, standard port 1099, itself remote object)
- Access via java.rmi.Naming
- Naming interface methods: bind, rebind, Remote lookup, unbind, list
- Registry-Lookup: C invokes lookup for url (rmi://host:port/service) – socket connection, stub to remote registry returned, Registry.lookup() performed on stub, Stub for remote object returned, C interacts with remote object via stub

3.3.5 Developing RMI applications

- Define remote interface: public, extends java.rmi.Remote, each method throws java.rmi.RemoteException, remote object parametrs/returns musst be nterfaces
- Implement remote interface: Basics in java.rmi.server.RemoteServer, Subclasses: UnicastRemoteObject, Activatable
- Generate stubs and skeletons (using the tool rmic)
- Remote object registration: Register in registry on host of remote object, stub needed
- Client implementation: Client registry lookup to get reference to remote object, Interaction always with remote interface

3.3.6 Parameter Passing

- Primitive data types passed with values
- Local object parameter: Object passed (must implement java.io.Serializable or java.io.Externalizable)
- Remote object parameter: Stub of remote object transfered as reference to remote object

3.3.7 Distributed grabage collection

- Reference counter represents references which are alife
- Client access creates referenced message / No more reference → unreferenced message
- Lifetime limit of references, Afterwards connection to server must be renewed

3.4 Servlets

Programs invoked by client, execoted on server host to extend functionality of the server

3.4.1 Servlet Properties

- Execution by Servlet engine (e.g. Apache Tomcat)
- Methods specified within servlet object: init, shutdown, service (client request forwarded)
- Invoked via HTTP requests (e.g. http://myhost:8080/servflet/formServlet)

3.4.2 Servlet lifecycle

Loaded, Created, Initialized, Served Destroyed

3.4.3 HttpServletInterface

- HttpServlet extends GenericServlet
- Functions: doGet, doPost, doDelet, doPut

4 Basic mechanisms for distributed applications

4.1 External data representation

- Heterogenous environment = different data presentations ⇒ data transformation needed
- Independence from hardware: External data representation

4.1.1 Marshalling & unmarshalling

- Marshal: Parameter serialization to data stream
- Unmarshal: Extraction from data stream and reassembly of arguments
- Either by RPC system or as software plugin

4.1.2 Centralized transformation

Only one node transforms data (send & received)

4.1.3 Decentralized transformation

All nodes transform data

- A transforms data send to B and vice versa
- A transforms data by B and vice versa
- A & B transform data into network-wide standard format; Recipient transforms into local format (→ Adding of components only need to know network standard)

4.1.4 Common external data representation

- Important aspects: Machine independent format, Description of complex data structures
- E.g. ASN.1
- For numbers: Little endian (lower part of numbers in lower memory area); Big endian (higher part of numbers in lower memory area) – Convention: Network transfer structure well-defined, such as big endian
- For strings: 4 bytes length n, n bytes data, r bytes 0s with: $(n+r) \mod 4 = 0$
- For arrays: 4 bytes length n, n elements (If variable number of elements: counted array)
- For pointers: Problem, no shared address space
  - Prohibit pointers in RPC

  - Dereference pointers in RPC: Serialize datastructure (marshal) and transfer whole data structure; booleans instead of null pointers; no function pointers in heterogenous environments (homogenous java possible)
  - Transfer pointer

4.1.5 XML as common data representation

- Complex datatypes mapped to XML schema types for network transfer
- Primitive Datatypes: XML Schema Definition (XSD) equivalent
- SOAP build-in array encoding support
- SOAP API for custom mapping
- Abstraction:
  - High: Application specific: XML
  - Middle: General encoding: ASN.1
  - Low: Network encoding: Sun XDR

4.1.6 Java Object Serialization

- Flattening object to store on disk or transmitting in messages
- Stored information: class information (name + version number); number, types & names of variables; values of instance variables
- Java Serialization: ObjectOutptutStream.writeObject(obj)
- Java Deserialization ObjectInputStream.readObject

4.2 Time

- Need to measure time accurately: Time of events on computer → Synchronized clocks for Concurrency control, Authentication (e.g. Kerberos)
- Notions of time:
  - Time seen by observer
  - Time seen by processes
  - Logical notion (A before B)

4.2.1 Introduction

- Each computer has own clock: Processes get time, Timestamp of events, Clocks drift from perfect time (clock drift rate = difference per unit of time since reference clock)
- Timestamp: At time t OS reads hardare wlock $H_i(t)$ and calculates time on software clock $C_i(t) = aH_i(t) + b$ (e.g. nanoseconds since base time)

- Skew between clocks: Disagreement between two clocks, Ordinary quartz clocks drift by ~1 sec in 11-12 days
- Coordinated Universal Time (UTC): International standard (atomic time, adjusted to astronomical time), Broadcasted land-based accurate about 0.1-10 ms, GPS about 1 microsecond

4.2.2 Synchronizing physical clocks

- Physical clocks to compute current time to timestamp events (file modification, transactions, ...)
- External Synchronization: With External authoritative clock S: $|S(t) - C_i(t) < D|$
- Internal Synchronization: Pair of computers: $|C_i(t) - C_j(t) < D|$ (might drift cellectively)
- Processes synchronized externally with bound $D \Rightarrow$ Synchronized internally with bound $2D$
- Clock correctness: H correct if drift rate within bound $q > 0$ (e.g. $10^{-6} \frac{secs}{sec}$)
  - Error of interval between t and t' bounded; No jumps
  - Weaker monotonicity: $t' > t \rightarrow C(t') > C(t)$ e.g. required by Unix make
  - Faulty clock is not correct: crash failure (clock stops ticking), arbitrary failure (anything else e.g. jumps)
- Synchronization in a synchronous system
  - Bounds in synchronous system: Time needed for process step has lower & upper bound, message transmission bounded, process clocks bounded drift rate
  - Process p_1 sends time t to p_2
  - p_2 sets clock to $t + \frac{(T_{trans_{max}} - T_{trans_{min}})}{2}$; Skew $\leq \frac{(T_{trans_{max}} - T_{trans_{min}})}{2}$
- Christian's method for asynchronous system
  - Observation: round trips reasonable short but unbounded; estimate possible, if round trip sufficiently short compared to accuracy needed
  - p requests time from S and sets clock to $t + \frac{T_{round}}{2}$; Accuracy: $\pm \frac{T_{round}}{2} - min$
  - Discussion: Only suitable in LAN/Intranet: Time server might fail, Faulty time servers, False clock reading
- Berkeley algorithm
  - Internal synchronization of group of computers (only intranet suitable)
  - Master (can be reelected on failure) polls to collect clock value; Round trip time used to compute slaves' clock values

  - Calculation of average (eliminating spikes)
  - Sends adjustments to slaves
- Network Time Protocol (NTP)
  - Time distribution over internet via hierarchical tree (Primary connected to UTC, secondary connected to primary)
  - Subnet can reconfigure on failures: Primary lost source becomes secondary, Secondary losing primary use another primary
  - Synchronization modes: Multicast (low accuracy), Procedure call (See Berkeley algorithm – middle accuracy), Symmetric (Pairs of servers symmetric – high accuracy)
  - Message exchange: UDP messages, timestamps of recent events (send/receive of previous message, send of current message), Non-negligible delay between messages possible; *See picture page 102*
  - NTP estimates offset o and round-trip delay d – $T_i = T_i - 1 + t' - o$, $d_i = t + t'$, $o = o_i + \frac{t'-t}{2}$
  - Offset estimation: o_i offset estimation, d_i measure of accuracy, NTP server pairs $< o_i, d_i >$, peer-selection for reliability estimate
  - Accuracy: Internet tens of ms, LAN ~1ms
- Precision Time Protocol (PTP)
  - Designed for LANs; accuracy < microseconds
  - Synchronization Message Exchange
    * Master-Slave hierarchy; Master sync message using UDP multicast + follow-up message with time, sync message left master
    * Slave initiates exchange to determine round-trip-delay
    * Calculation of offset *See picture + formulas on page 103*
    * Support to select best candidate clock

4.3 Distributed execution model

4.3.1 Events

- Messages causing events: Internal events, Message sending, Message receiving (s.t. + message delivery)
- send(m) $\rightarrow_{msg}$ receive(m): Causal relation. Sending before Receiving
- $||$ Concurrent events
- Happened Before (Lamport):
  - Ordered within one component
  - Send before receive
  - Transitivity
  - If $\neg(a \rightarrow b) \wedge \neg(b \rightarrow a) \Rightarrow a||b$
  - $C : E \rightarrow T$ (mapping events to timestamps): $a \rightarrow b \Rightarrow C(a) < C(b)$ (If $\Leftrightarrow$, then strictly consistent)

4.3.2 Ordering by logical clocks

- Component manages: Logical clock (lc), View on global clock (gc)
- Update Rule 1: Update lc, when events occur
- Update Rule 2: Update gc: Attach lc to sending messages; Update view on gc when receiving messages

4.3.3 Logical clocks with scalar values

- Clock value is positive integer. lc and view on gc represented by counter C.
- R1: Prior to event execution: $C := C + d$
- R2: Receiving messages with timestamp C_{msg}: $C := max(C, C_{msg})$, execute R1, deliver message
- Partial ordering; Not strictly consistent

4.3.4 Logical clocks with vectors

- $n-$dimensinoal (number of components) vector with positive integers. Component TK_i manages vector $vt_i[1...n]$.
- $vt_i[i]$ logical clock of TK_i
- $vt_i[k]$ view of TK_i on logical clock of TK_k
- R1: $vt_i[i] := vt_i[i] + d$
- R2: $1 \leq k \leq n : vt_i[k] := \max(vt_i[k], vt[k])$, Execute R1, deliver message
- Timestamp comparison
  - $vh \leq vk \Leftrightarrow \forall x : vh[x] \leq vk[x]$
  - $vh < vk \Leftrightarrow vh \leq vk \wedge \exists x : vh[x] < vk[x]$
  - $vh||vk \Leftrightarrow \neg(vh < vk) \wedge \neg(vk < vh)$
- a and b are events with timestamps
  - $a \rightarrow b \Leftrightarrow va < vb$
  - $a||b \Leftrightarrow va||vb$
- a of TK_i and b of TK_j triggered:
  - $a \rightarrow b \Leftrightarrow va[i] < vb[i] \wedge va[j] < vb[j]$
  - $a||b \Leftrightarrow va[i] < vb[i] \wedge va[j] < vb[j]$

4.4 Failure Handling in distributed applications

- Local applications: Exception Handling
- Distributed: Communication failuse, system crashes, byzantine failure (erratc behavior), ...

4.4.1 Testing distributed applications

- Testing without communication ⇒ Component functaonality
- Testing with local communication ⇒ Prediction abount components (no transport times)
- Testing with network communication ⇒ Identification of time dependencies, execution ordering, multiple clients

4.4.2 Debugging of distributed applications

- Server breakpoints can cause client timeouts
- Communication between components (Message flow)
- Snapshots (No shared memory, No strict clock synchronisation; State of system)
- Breakpoints
- Nondeterminism (Message transmission)
- Interference between debugger and application (delay)

4.4.3 Approaches of distributed debugging

- Focus and send/receive
- Monitoring communication between components. Components as blackboxes (tested before locally)
- Global breakpoints: Events are partially ordered; Causally distributed breakpoint: Remote components rolled back to earliest state after last event in a before-relationship with triggering event

4.5 Distributed transactions

- Important to design reliable, fault tolerant distributed applications
- Several requests bundled in transaction
- Distributed transation if more than one server involved
- ACID properties: Atomicity (All or nothing), Consistency (before/afterwards consistent state), Isolation (No effect before commit), Durability (Results persistent)

4.5.1 Isolation

- Serializability: Same sequence on each server
- Timestamps: Timestamp (local timestamp + server id) issued to transaction on start. $if(t_{trans} < t_{obj})$ then abort else access obj.

- Locking: Server has locks for local objects. TA locks before access, exclusive locks (or r/r), all locks removed before termination; 2-Phase-Locking: No locks requested after first release
- Optimistic: Check for conflicts if commit ready

4.5.2 Atomicity and persistence

- Intention list: All modifications in intention list (log file). Each server S performs $AL_S(trans)$ to update local objects. Then delete AL
- New version: On access new version of object obj_{trans} is created. Overwrite old element on commit.

4.5.3 Two-phase commit Protocol (2PC)

- Voting protocol to determine commit: Voting Phase, Completion phase
- Coordinator (Client / First server) contacts all servers S_i (canCommit?)
- If one server votes no: Abort to all servers which voted yes (doAbort)
- All servers vote yes: Commit message to all servers (doCommit)
- Acknowledgement (haveCommited)
- getDecision: Yes/No call from participant to coordinator
- Problems: Failures on crashes (server, coordinator)

4.5.4 Extended 2PC

- Coordinator has Write-Ahead-Loging, Send Outcome, for pending trancations in outcomes table
- Server sends acknowledgement when asked for finished commits, asks for outcome of uncommited transactions
- 3PC also possible

4.5.5 Distributed Deadlock

- Deadlock detection schemes try to find cyles in wait-for graph. Problems: Single point of failure, Communication time
- Edge Chasing: No global wait-for graph, server has knowledge about edges, find cycles by forwarding messages; TA starts at coordinator C (records if TA active / waiting), Lock Manager informs C, when TA starts waiting / aquires lock
  - Initiation: Server X notes, that W is waiting for transaction U. $W \rightarrow U$ send via C
  - Detection Server Y receives $W \rightarrow U$. It notes, $U \rightarrow V$. It forwards $W \rightarrow U \rightarrow V$.

  - Resolution: If Cycle detected one TA in cycle is aborted

- Transaction Priorities: Every TA can initate deadlock detection; If parallel, several TAs might be aborted. TAs totally ordered by Priorities. Abort TA with lowes priority

4.6 Group communication

- Traditional 1:1 communication
- Distributed environments: 1:n for fault-tolerance, object locaization, conferencing/groupware, syncronization
- Functional components composed to group
- Group membership: Structural characteristics, composition, management
- Support of group communication: addressing, delivery
- Communication: unicast, broadcast, multicast (fault tolerance, location objects, multiple update of distributed data)
- Synchronization: consistent sequence of actions
- Group addressing: Central server which knows group composition / Decentralized (members know composition)
- Communcation services: Datagrams (UDP) / Reliable streams (TCP)
- Consistent behavior: ISIS / Horus

4.6.1 Groups of components

- Closed (no external messages) vs. open group (external messages broadcasted to group members)
- Flat (peer) vs. hierachical group
- Implicit (anonymous, group address implicity expanded) vs. explicit group

4.6.2 Group Management

- Operations: Query names, groupCreate, groupDelete, groupJoun, groupLeave, reading/modifying attributes, read member information
- Management architecture: Centralized (group server), Decentralized (all components perform management tasks – syncronization), Hybrid (group manager within lan clusters)

4.6.3 Message dissemination

- Unicast to group members
- Group multicast to whole group
- Inter-group multicast to several groups
- Broadcast to all components (filtering required)

4.6.4 Message delivery

- Who gets message? / When is message delivered?
- Atomicity (who?): Exaclty-Once to all recipients; All-or-Nothing to all group members or none
- Sequence of message delivery: Same sequence for all group members (otherwise Nondeterminism possible)
  - Ordering: synchronous (system-wide global time ordering), loosely synchronous (consistent time, but no global absolute time)
  - Sequencer (total ordering): Sequencer serializes all messages send to group and determines sequence number, e.g. Apache Zookeeper
  - Virtually synchronous ordering: based on before relation
  - Sync-ordering: Synchronization points. Synchronously ordered messages delivered to group members in-sync. Ordering method to synchronize local states

4.6.5 Taxonomy of multicast

See picture page 128

- Unreliable Multicast: No acknowledgement, At-Most-Once semantics, No ordering
- Reliable Multicast: "best-effort" (at-least-once), B-multicast primitive process delivers, if multicaster not crashing, B-deliver primitive similar when receiveed
- Atomic Multicast: Reliable with atomic gurantee (all-or-nothing)
- Serialized Multicast: Consistent sequence totally vs causually ordered (e.g. virtually synchronous)
- Atomic, Serialized Multicast: Atomic + Serialized

4.6.6 Group communication in ISIS

- Toolkit for group management, ordered multicast abcast (totally ordered) & cbcast (causually ordered)
- abcast (atomic broadcast)
  - Phase 1: Sender S sends message N with logical timestamp $T_S(N)$, Receivers determine new timestamp $T_r(N)$ and return to S
  - Phase 2: S creates new timestamp $T_{S,new}(N) = \max(T_r(N)) + \frac{j}{|R|}$ (j unique identifier of S), S send commit to all r. r deliveres message according to new timestamp
- cbcast (causual broadcast)
  - Vector timestamps

  - Vector specifies number of messages received in sequencece from particular group members
  - Sending appends incremented state vector
  - Two conditions for delivery: No message from sender missing, No other depending message not yet received

4.6.7 JGroups

- Group communication toolkit for Java
- Reliable, atomic ordering
- Group membersihp managment
- Groups identified in channels: channel.connect("MyGreup");
- Channel connected to protocol stack (e.g. Sequencer, GMS, Frag, UDP)

4.7 Distributed Concensus

- Distributed processes agree on value (even in case of failure desireable)

4.7.1 Consensus problem

- p_i is undecided, value v_i proposed
- Processes communicate
- p_i sets decision variable d_i and is decided then
- Properties: termination (algorithm ends), agreement (same value of d_i), integrity
- Algorithm (in failure-free environment): Reliable multicast of all processes. $d_i = \text{majority}(p_1, p_2, ...)$.
- Properties: Termination/Integrity depending on multicast

4.7.2 Byzantine Generals Problem

- Generals issue commands to lieutnants
- Lieutnants have to agree to attack or to retreat
- Difference to Consensus problem: General supplies value, lieutnants have to agree on
- Properties: Termination, agreement, Integrity (if general correct, all decide as he suggests)

4.7.3 Interactive Consistency Problem

- Process suggests single value
- creation of decision vector
- Properties: Termination, Agreement

4.7.4 Consensus in synchronous networks

- Assumtion $f < n$ processes crash
- Algorithm proceeds in $f + 1$ round to reach concensus

4.8 Authentacation service Kerberos

- Based on Needham Schröder Protocol
- Client C, Server S, Key distribution center KDC, Ticket granting service TGS
- C requests service from S. KDC and TGS gurantee secrecy & authenticity requirements
- TGS ticket issued by KDC to C; Authentifier of C to gurantee valid communication with S, Session key between C and S
- Problem: Synchronization of clocks

4.8.1 Authentication process

- $C \rightarrow KDC$: Request TGS ticket
- $KDC \rightarrow C$: TGS ticket
- $C \rightarrow TGS$: Request server ticket
- $TGS \rightarrow C$: Server ticket
- $C \rightarrow S$: Authentifier
- $S \rightarrow C$: Authentifier

5 Web Services

Standard means of communication among distributed applications

5.1 Service Oriented Architecture (SOA)

- Loose coupling and dynamic binding between services (find/publish in service registry)
- Service well defined, self-contained
- Focus on interface design
- SOA vs. Component Based
  - loose integration vs. tight integration
  - process-oriented programming vs. code-oriented
  - interoperable architecture vs. technical complexity
  - build to change vs. buidl to last

5.1.1 Layered Approach

- Mapping of business processes to services
- Application layer, Process layer, Service layer, Component layer, Object Layer

5.1.2 Adpoting SOA

+ Interopable, Easy data exchange, Easy access, Availability of external services, ...

– Different formats, Security issues

- Enterprise Services Bus (ESB): Software architecture / software class for SOA: Interopability via XML, Web Services interfaces, ... e.g. Mule

5.2 Web Services – Characteristics

- Web Services
  - Live somewhere in the network
  - Are described using a service description Language (XML)
  - Are published to service registry
  - Are available through declared API
  - Provide entry point accessing local/remote services
- Allow integration of functionality (within/between organizations)
- Features: Programmable, Self descriptive, Encapsulated, Loosely copuled, Location transparent, Protoco transparent, Composition, Document-Centric

- Webservices vs. Distributed Objects: Description language (operation, returns, ...), Client stub / Server skeleton, network interations
  - Web Services: Stateless, Internet
  - Distributed Objects: Stateful, Intranet

5.3 Web Services Architecture

- W3C: Web service is software system identified by URI, interfaces/bindings described using XML, discoverable & interaction possible using XML messages
- XML: tag data, SOAP: transfer, WSDL: describe services, UDDI: list services
- Simpilied: RPC over internet using XML

5.3.1 Interoperability Stack

- Compositional (WS-notification)
- Quality of Experience (WS-Security/Transactions)
- Description (WSDL, UDDI)
- Messaging (XML, SOAP)
- Transport (HTTP, SMTP)

5.3.2 Basic architecture

- Interaction between components as message exchange
- Functions: message exchange, description, publishing/finding
- Web services is interface, service provided by implementation
- Service description: Details of interface / implementation

5.3.3 Roles

- Service Provider
- Service Discovery Agency
- Service Requestor

5.3.4 Operation

- Publish
- Find
- Interact

5.3.5 Basic Standard Technologies

- WSDL: Simple Object Access Protocol
- UDDI: Web Services Description Language
- SOAP: Universal Description, Discovery and Integration
- Providing & Consuming Service
  - Provider describes service in WSDL and publishes to agency
  - Requestor queries agency to locate service/communication methods
  - Agency sends service description
  - Requestor sends request based on WSDL
  - Provider sends request based on WSDL

5.3.6 Message Exchange Patterns

- eg. one-way, request/response, broadcast
- Peer-to-Peer: Each web service acts as requestor and provider
- Direct interaction: Requestor & discovery agency fulfilled by the client
- Intermediary (web server between requestor & provider): Additional functions such as routing, security management

5.4 Simple Object Access Protocol (SOAP)

- simple, lightweight XML messaging
- no specific protocol
- RPC or document transfer

5.4.1 Parts

- Envelope
- Encoding Rules
- Convention for RPCs and responses
- SOAP message: Envelope (XML namespaces), SOAP header (optional), SOAP body (payload e.g. method name & arguments)

5.4.2 Exchange Model

- One-way transmission. Interaction is combination of SOAP messages.
- Processing messages: Interpret message for application and "SOAP actor"; Verify mandatory parts; (Remove parts from step one and forward message)

5.4.3 SOAP in HTTP

- HTTP request & response used for SOAP request & response
- Media type “text/xml”
- Interpretation of request by webserver/servlet/...

5.4.4 SOAP RPC Conventions

- RPC interactions mapped to SOAP (Converted through middleware)
- e.g. (Simpilied): <nameSpace:functionName ...><arg ...>value</arg></nameSpace:functionName>

5.4.5 SOAP-Router

- Deliver through series of nodes; Move messages between networks
- May provide: logging, auditing, security enforcement
- WS_Routing protocol

5.5 Web Services Description Language (WSDL)

- Defines service as colletion of network endpoints / ports (compare IDL)
- Describes: Functionality of a service (arguments), Accessability of a service (protocols), Location of a service (URI)

5.5.1 WSDL Information model

- Types: Container for non build-in types
- Message: Definition of transferred data
- Port Type: Set of operations per endpoint
- Operation: Supported actions (input/output message)
- Binding: Protocol, data format, port type
- Port: Binding + network address
- Service: collection of related endpoints

5.5.2 Parts

- Abstract (What is offered?): Types, message, operation, port types
- Concrete (Where/How is it offered?): Bindings, services, ports
- *See picture page 159*
- Relationship: XML definitions; Operations supported by WebService; Bindings connect port types to port

5.5.3 Generate code from WSDL

- WSDL compiler can create e.g. Java interface
- WSDL documents from API / Stubs & Skeletons from WSDL document

5.5.4 Bad Practices

- Bad names and comments
- Port Types tied to protocols
- Unrelated operations placed in single port type
- Overload output messages

5.6 Universal Description, Discovery and Integration (UDDI)

- Services for description/discovery of businesses, services, interfaces
- UDDI is web services itself (can be described by UDDI)

5.6.1 UDDI Business Registry System

- Wite Pages: Basic information (Name, ... of company & its services)
- Yellow Pages: Detailed business data & web services
- Green Pages: Information how web service can be invoked

5.6.2 UDDI Entities

- UDDI can store & manipulate four main types of entities
- businessEntity: Owner of web service (name, key, services, ...)
- businessService: Group of Web Service(s) (name, key, binding, ...)
- bindingTemplate: Single WebService (key, access point)
- TModel WSDL interface types (name, key, URI to data)

5.6.3 UDDI Registry API

- 3 main user types: Providers, requesters, other registries
- Inquiry API: find_service, get_serviceDetail
- Publishers API: save_service, delete_service
- Security API: get/discard authentication tokens
- Ownership Transfer API
- Subscription API: Monitoring changes in registry
- Replication API: Replication between registries

5.7 Representational State Transfer (REST)

- Principles of using standards as HTTP, URIs and Mime Types
- Resource has ID, URIs to identify item of interest
- Link resources together
- Standard methods get/post/put/delete
- Stateless communication
- Resources with multiple representation: client chooses

5.8 Web Service Composition

- Choice of granularity
- Composition of complex services from smaller ones

5.8.1 Dimensions to handle complexity

- Component model: Sub-services
- Orchestration model: Order of sub-services (e.g. WS-Coordination)
- Data access model: Data Exchange
- Transactional models: Transactional semantics (WS-Transactions)
- Exception Handling: Handle errors

5.8.2 Web Service Orchestration

- Transparent Chaining: Client determines usage
- Translucent Chaining: Worklow services invokes services in order (Status propagation to client)
- Opaque chaining: Aggregate service invokes services (no client awareness)

5.9 Adopting Web Services

5.9.1 Example Web Services

- Amazon E-Commerce Service (ECS): Amazon product database, SOAP/-REST, search/similarity lookup, remote shopping
- XMethods: Clearinghous for web serives

5.9.2 Apache Axis

- Environment to implement web services
- APIs for invoking SOAP & manipulating SOAP objects
- WSDL compiler and data bindings for Java classes
- Hosting mechanisms & transport framework
- Axis2: Java based implementation + REST

5.9.3 Web Services & Java

- Several Java APIS for web services
- SAAJ, JAX-WS, JJWSDL, JAXR, JAXP, XWSS

5.9.4 Distributed Process Architecture

- Client ↔ adapter/application server ↔ application

5.9.5 Semantic Web Services

- Semantic meta-data to automate discovery / interaction with web services
- Map-Service: Input (int, int), Output gif – (x, y) is what? Kind of map?
- Candidate: OWL-S (Ontology Web Language for Web Services)

5.10 Mashups

Create new applications by combining existing ones

5.10.1 Mashup Techniques

- Mashing on the Web Server: Browser just waits for response, Browser decoupled from supply pages, Web server as proxy serves entire page, Scalability problems
- Mashing using Ajax: Work divided between server and browser, Complex, Browser navigation bypassed, Browser doing most work, All data routed through server
- Mashing with JSON: Browser communicates with source, handling of pre-made JSON objects, no data consolidation on server

5.10.2 Development Support

- Component model: Characteristics of mashup components give interface. Properties: type (data, logic, ui), interface (CRUD, API, IDL/WSDL), extensibility (user may extend component model?)
- Composition model: How components ordered – flow-based vs. event-based
- Example-tool: Yahoo Pipes

6 Design of distributed applications

- Specification of software structure: small, distributed components (local vs. remote), testing
- Name resolution: remote services
- Communication: client-server vs. peer-to-peer, network errors
- Consistency: replicated data, cache, interface Consistency
- User requirements: functionality, non-functional requirements, security, client errors, heterogenity

6.1 Steps in design

- Identify repositories
- Data assigment to modules
- Define module interface
- Define network interface
- Classify module as client/server
- Registration of servers (which are available)
- Strategy for binding process

6.2 Development environment

6.2.1 Open Distributed Processing (ODP)

- Standards for distributed systems (e.g. ISO/OSI reference model)
- Complexity reduction using abstraction levels (viewpoints)
  - Enterprise: overall goals
  - Information: structure, controll/access of information
  - Computation: logical distribution
  - Engineering: physical distribution
  - Technology: different systems (network, hardware)

6.2.2 Model Driven Architecture (MDA)

- Object Management Group (OMG) Standard
- Model: Description of system (part) in well-defined (syntax, semantics) language (automatic interpretation possible)
- MDA concept:
  - Development of platform independent models (PIMs) – business function, components, classes, conditions, semantics – UML diagrams (use cases, class, sequence, ...)

  - Mapping to platform dependent models (PSMs) – Realization of software in UML
  - Implementation, Integration & Testing – Code generation (Use of tools possible)
  - AutoFocus: Tool to specify distributed systems – hierachical description, platform independent development, requirement Analyses, Desing modelling, interactive simulation, code generation

6.3 Service-Oriented Modeling

- Transfer service approach to design/modeling of software systems
- Service-oriented modeling (SOM): model SOA systems
- Service-oriented modeling framework (SOMF): development life cycle methodolgy, universal language

6.3.1 Service Evolution

- Conceptual service: idea / concept
- Analysis service: unit of analysis
- Design service: design entity
- Solution service: physical solution (to be deployed)

6.3.2 Life Cycle Structure

- Elements for service development / operations
- Timeline: life span of service
- Events: predicted / unexpected events during life span (begining + duration)
- Seasons: design-time / run-time
- Disciplines: Identify best practices – season disciplines (service orinted conceptualization) vs. continuous disciplines (service portfolio management)

6.3.3 Life Cycle Modeling

See Picture page 182

- Conceptual: Identify concepts
- Discovery & analysis: Granularity, reusability, coupling, ...
- Business integration: Integration in business (organization, IT, ...)
- Logical design: Service relationships, message exchange, ...
- Conceptual architecture: SOA design, environment, technological stack
- Logical architecture: Integrate SOA assets, depentencies, service reuse, ...

6.3.4 SOM Framework

See image on page 183

7 Distributed file service

7.1 Introduction

- Replication & concurrency control
- Distributed file system: logical colletion of files on different computers into common file system & storage computers connected through network
- Distributed file service: set of services supported bi distributed file systems
- File server: executon of file service software on computer
- Allocation: placement of files on different computers
- Relocation: changes of file allocation
- Replication: multiple copies of file on several computers (Relication degree REP_d of file d is total numbers of copies)
- Motivation: Network traffic / response times / availability / fault tolerance / parallel processing $\Rightarrow$ Transparency

7.1.1 Consistency types

- Internal Consistency: Single file copy consistent (2-phase commit)
- Mutual Consistency: All copies identical (multiple copy update protocol) – Strict (All copies same state), Loose (All copies converge to same consistent state)

7.1.2 Replica placement

- Permanent replicas: decided in advance (e.g. mirroring)
- Server-initiated replicas: enhance server performance (reduce server load, migrate to server near clients)
- Client-initiated replicas: caches (improve access time, placed on client, limited time)

7.2 Layers of a distributed file service

- Naming / Directory service: placement / relocation of files, server localization
- Replication service: response times / availability / consistency / multiple copy update
- Transaction service: group operations to transaction / concurrency control / error reboot
- file service: read / write operations
- block service: access / allocate disk blocks

7.3 Update of replicated files

7.3.1 Optimistic concurrency control

- No user constraints, access to inconsistent data
- Available copy: read local / best-available file copy, write all file copies
- Example: Coda file system (Carnegie-Mellon University)

7.3.2 Pessimistic concurrency control

- Allways access consistent data (data-critical applications)
- Multiple copy update
  - nonvoting
    - primary site: primary site serializes/synchronizes operations
    - token passing: access possible if client has token
  - voting: negotiation result determines access (global consent)
    - majority voting
    - weighted voting

7.3.3 Voting schemes

- REP_d replicas of file d
- $sg(r)$ weight of computer $r \in K$
- Sum of weights $SUM = \sum_{r \in K} sg(r)$
- Votum: sum of votes voted for access
- Quorum (R, W): lower bound where acces is granted
- Multiple-reader-single-writer: $R + W > SUM$, $W + W > SUM$
- Write-All-Read-Any: $W = n$, $R = 1$
- Majority consenus: $W = R = \frac{REP}{2} + 1$ if REP even; $W = R = \frac{REP+1}{2}$ if REP odd
- Weighted voting: $W = R = \frac{SUM}{2} + 1$ if SUM even; $W = R = \frac{SUM+1}{2}$ if SUM odd

7.4 Coda file system

- scalable, secure, available distributed file system
- mobile use, organization in (replicated) volumes

7.4.1 Architecture

- *Picture page 192*
- Venus processes provide access to remote files (comparable to NFS client)
- Allows to continue if access is impossible

7.4.2 Naming

- Each file exactly in one volume, physical vs. logical (all volume replicas)
- Replicated Volume Identifier (RVID) for logical volumes
- Volume Identifier (VID) for phyical volumes
- File identifier (96-bit)
- *See picture page 193*

7.4.3 Replication strategy

- Client caching: cache complete file when opened, server records callback promise for client, update on client $\Rightarrow$ server notification $\Rightarrow$ Invalidation to other clients
- Server replication: Volume Storage Group (VSG): servers that have copy of volume, Accessible VSG (AVSG): servers available for client, read-one, write-all update protocol
- Coda version vector (CVV): optimistic strategy, CVV vector timestamp initalized to $[1, ..., 1]$, On file close Venus breadcasts update messages to servers in AVSG, if for two CVVs neither $v1 \leq v2$ nor $v2 \leq v1 \Rightarrow$ conflict

7.4.4 Disconnected operation

- Client resorts to local copy, priority list for cache (hoarding possible)
- AVSG = {}
- Reintegration: Send update operations to AVSG servers for updated files

8 Distributed Shared Memory (DSM)

- Abstraction for processes who do not share physical memory
- DSM appears as memory in processes' address space

8.1 Programming model

- Direct access to variables (no marshalling)
- DSM possible if non overlapping lifetimes
- Implementation: Hardware (Shared memory multiprocessor architecture e.g. NUMA) vs. Software (e.g. Linda Tuple Spaces / JavaSpaces)

8.2 Consistency model

- Local caching possible $\Rightarrow$ Consistency?
- Write-Update: Local updates multicasted
- Write-Invalidate: Send invalidate, acknowledgement (block all other access), update, send updated copy

8.3 Tuple space

- Originally for Linda language
- Set of tuples interpreted as list of typed fields
- Based on shared memory, tuple stores information

8.3.1 Atomic operations

- out(t) creates new tuple
- in(t) reads tuple and deletes
- read(t) reads tuple
- eval(p) generates new process
- Synchronous in/read, Asynchronous inp/readp

8.3.2 Tuple space implementation

- Central tuple space
- Replicated tuple space (each computer has complete replica)
- Distributed tuple space / subspaces (out operations performed locally)

8.3.3 Exapmle program

Client out, Server in, Server out, Client in

8.4 Object Space

- Shared, network-accessible object repository

8.4.1 Features of JavaSpaces

- Objects passive: objects not manipulated / run in space
- Shared spaces: network-accessible memory, many remote processes interact concurrently
- Persistent spaces: stored until removed / lease time run out
- Associative spaces: objects accessed via associative lookup
- Transaction oriented spaces: atomic opreations
- Spaces support exchange of executable code

8.4.2 Data structures

- Entry interface (Serializable) for objects in space (ne.jini.core.entry), extended by classes storing variable values. Public constructor setting variables
- SpaceAccessor: JavaSpace s = SpaceAccessor.getSpace(); spaces Jini service / RMI lookup

8.4.3 Basic operations

- read, take (read, remove), write, notify (notify process matching entry has arived, can be requested)
- write: Lease write (Entry e, Transaction txn, long lease) throws RemoteExecption, TransactionException
- read and take: read remote object and copy to local process + remove from space, process needs template. SharedVar template = new SharedVar(); SharedVar result = (SharedVar) space.take(template, null, Long.MAX_VALUE); If several matching objects, any can be selected, waiting till entry available
- Matching rules: template class matches or is super class, if template field is null, matches any value, if field is specified, objects field must match
- Atomicity: Basic operations are atomic ⇒ No race-conditions (if take is used for editing objects)

9 Object-based Distributed Systems

9.1 Object Management Architecture (OMA)

- Also: Common Object Request Broker Architecture (CORBA)
- possible middleware for object-oriented distributed applications
- ORB communication through request/reply protocol. Only mediates between application objects (localization, messages delivery, ...)

9.2 Object Request Brokers (ORB)

- Connects distributed objects at runtime
- Support invocation of distributed objects

9.2.1 Features

- Static (interface determined on compiling) & dynamic (interface determined at runtime) invocations
- Interfaces for higher programming language
- self-descriptive
- location transparcency
- security checks
- polymorphic method invocation (execution depends on objects instance) – ORB calls objects method (vs. RPC calls server function)
- hierachical object naming

9.2.2 ORB structure

- *Picture page 209*
- components
  - ORB core (kernel): mediates request between client/server, network communication
  - Static invocation interface: determine operations/parameters on compilation
  - Dynamic invocation interface: identical for all ORB implementation (only one dynamic interface)
  - ORB interface: ORB service calls (conversion object reference to strings)
  - Interface repository (signature of methods for dynamic invocation)
  - Object adapter: brige between CORBA/IDL interfaces and programming language interfaces

  - Runtime repository: information about server (supported) object (classes)
  - Skeletons: language of server created by IDL compiler (several static, one dynamic skeleton)
- Embedding in distributed applications
  - ORB as library
  - e.g. ORBIX & TAO

9.3 Common object services

- System level services extend ORB functionality
- Life-cycle Service: create, copy, migrate, delete objects
- Persistence: object storage e.g. databases
- Name: locate objects by name e.g. LDAP
- Event: register events
- Concurrency Control: lock manager
- Transaction: 2-phase commit coordination
- Relationship: create relations between objects, navigation, referntial integrity
- Query: SQL operations

9.4 Inter-ORB protocol

communication between ORBs based on General Inter-ORB protocol (GIOP)

9.4.1 GIOP Features

- Message formats (request, reply) + common data representation (CDR)
- Remote object references
- Internet Inter-ORB Protocol (IIOP) is GIOP via TCP/IP

9.4.2 External data representation

- Primitive data types: char, octet, short, ...
- Complex data types (typeCodes: struct, union, sequence (Format described in interface repository)

9.4.3 Object reference

- Identifies object accessed via IOP
- Object reference (IOR profile): IP host address, TCP port, object key

9.4.4 GIOP message

- components: head, header, content
- head: same format for all message types, identifies message type
- message types: Request, Reply, CancelRequest, LocateRequest (destination of object reference), LocateReply, CloseConnection, MessageError, Fragment

9.4.5 RMI over IIOP

- Java Remote Method Protocol (JRMP) for RMI (Java specific) ⇒ i.e. no interoperability with CORBA (any language)
- RMI-IIOP uses JNDI to register objects by names
- Java IDL for CORBA (no JRMP, no RMI)

9.5 Distributed Component Object Model (DCOM)

- COM: Process library, Support development of dynamic components (dll, .exe)
- DCOM: COM + process communication with remote processes, access transparency

9.5.1 Object Model

- DCOM object: implementation of interface with unique, 128-bit Interface Identifier (IID)
- Only binary interfaces (table with pointers to implementation)
- Class instances, transient

9.5.2 Architecture

- Library specifies method signature
- Registry records maping remote call + local file
- Service control manager (SCM) activates objects
- Proxy marshaller transforms code to network stream
- Client proxy unmarshals objects

9.5.3 Object Invocation Model

- Remote-invocation model (synchronous/blocking) ⇒ Canel object to cancel
- Client reference to remote object via interface pointer (proxy implementation), How forward reference? *Image S 219*
- Cobmination with Microsotf Transaction Server (MTS) & Microsoft Message Queue Server (MSMQ) to COM+ (Transaction, integration into Windows)

9.6 .NET-Framework

- Windows framework for distributed applications
- Mainly: Common Language Runtime (CLR) + Framework Class Libary

9.6.1 CLR

- Runtime environment for different languages: memory + thread management
- Encapsulate access to OS functions
- Common intermediate language (MSIL)
- Common Type System (CTS): Possible datatypes / programming constructs uniformly interpreded for interoperability

9.6.2 Frame Class Library

- Common functions for all languages in .NET framework (file access, database interaction)
- Hieracrchy of namespaces (System.Object)

9.6.3 .NET-Remoting

- Remote method invocation (System.Runtime.Remoting)
- Different tarnsport protocols (TCP, HTTP)
- Activation of remote objects