

# Verteilte Systeme

Preq: Data struct / DB (seq), OS, SysProg, SE concepts, OOP

Evolution: 1985 → single standards, power processors, Networks → system containing lots of PCs

Our Task: Mainframe Corp, Workstations & LAN, Client-Server, Internet-scale, sensor/actor networks, mobile, ad-hoc, adaptive, pervasive (ubiquitous) sys, <1% Pingers for PCs

2.4.7 Direct Alarm System: Emergency Hub: Command Control Center → vehicle tracking & Logistics

HVAC (Heat, Ventil, Air Cond)? circulation blo? Water Eco, Air Eco, Monitoring

ICTs for energy saving (Villages, Factories, Schools, Hospitals, etc.)

Remote Service Maintenance: retail, life/safety, security & surveillance, data center, hotels, education, energy, airport, health, transport, buildings, industrial

Def. Disadv: collection independent! Corp that appears as single coherent system

order: collection autonomous corp in network with SW to operate integrated

order: present, if a crash is due to a corp you never heard of

Types: Object/Corp based (CORBA), File based (NFS), Doc based (WWW), Event based/ coordinate based (RCP), Resource oriented (P2P), Service oriented (P2P)

/ dist Computing (Cloud), dist InfoSys (TP), dist Pervasive Sys (P2P) ← omnipresent

Concepts: Communication, Concurrency / OS supp., Naming/Discovery, Sync/Acces, consistency/repl., fault-tolerance, security

Why distribute? users → resources & services, depend & secure (avail, fault tol, intrusion, take over) performance & latency, throughput) ⇒ otherwise do not

goals: resource sharing, transparency, hiding internal & complex, services by standard rules, scalability, ability to expand system, Concurrency (parallel), fault-tolerance & availability

hopes: 1. network reliable, 2. QoS, 3. bandwidth, 4. network is secure,

5. topology doesn't change, 6. 1 Admin, 7. Ongoing cost, 8. not much to optimize

connecting: access & store, business models & policies, collaboration by info exchange, communication, groupware & virtual orgs, electronic trade commerce, sensors/sensor in automation & pervasive computing, security and privacy?

Business Models: Infrastructure (key) policies, activities, resources

value proposition

Financials (cost structure, revenue structure)

Customer (relationship, channel, customer segments)

⇒ Quality of Services clients indicate level of service (SLA)

f.i.: real-time voice → guaranteed delivery below time req.

Financial → encrypted VS factor ⇒ Trade off

⇒ Transparency: Hide diff aspects from client (encrypt) ⇒ Learning the system

Scale Perform. Fault → use servers over substrate, service layer > OoS  
Basic Repli Concurre → diff in representation, how accessed; location → location,  
Access Locati Migration → hide migration changes, Relocation → able to use migration,  
Replication → hide fail, Concurrency → hide the sharing part, Failure → fail & recovery  
Degree 3 don't try blind, perf difficult? (P2P); transp/coord. trade-off (failure recov, repl. cost)  
⇒ consider non-functional requirements → particular demands

⇒ Operability: services via standards, functional protocols, interfaces (replicates integrable; Maintainable)  
Flexible (composition, configuration, replacement, extensibility)

Policy II Mechanism: Granularity (obj vs app), Comp interaction/corp standards, web code/policy??  
⇒ providers diff Service Chars info/people, new can be introduced, plug interfaces for new

⇒ Scalability: ability to grow, Sze/Geo/Admin, reason effective, no chg changes, Scale/fn/secur

Challenges: costs physical resources; performance direct to size of data =  $O(n)$   
performance loss: balancing data for performance not worse than  $O(log n)$   
prevent resources running out: IPs vs Oracle RTB restrict  
external bottleneck: no central services  
complete info

Centralized Services (1 server), Centralized Data (1 database), Access/route on

Decentralized Algorithms - Principles: 1. no remote sys state, 2. decisions based on local info,

3. failure doesn't kill the whole, 4. no global clock

Geo-Scale: LAN (Sync comm, fast broadcast, highly reliable)

WAN (Sync comm, slow, point2point, unreliable)

Admin-Scale: web browser portability, 1. resource usage, 2. Billing & Management, 3. Security

⇒ between admin domains trusted dom, enforced limitations

Scaling Techniques: hide latency: (async → batch, parallel opps), reduce overall communication

distribution: hierarchies, domains, zones

replication: available, load balance, de-correlation, Coding, consistency? ? what

⇒ server/client decks? combine messages?

Architectural Styles:

complexity dealing ⇒ Abstraction (client, server, service), interface vs implementation

Info hiding (encapsulation & interface design)

Separation of concerns (logging, bytes+block+file, client/server, components)

Communication models: multiprocessors & shared memory, multi computer & message passing

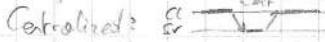
sync in shared memory (semaphores, monitors → diff. invoc synchronization)

Sync in multi comp (blocking & message passing)

req resp

Important Styles: layered, Object-based, Data-centered, Event-based

object based: Obj → Obj, event-based (publish - subscribe), with shared data-space

Centralized: 

Layering (result → user-interface / processing level / data level)

user interface → query gen → database → caching → html → user

user → query gen → database → caching → html → user

Multilevel: simplest organization 2 types client & server 2 Tier client (app) → (OS+OS)

Vertical Dist: User → app → db / tables

Server as client → db / tables

3 Tier client (app) → (OS+OS+OS)

Dist sys: computers in union, pervasive, tech & principles req, abstraction concepts

Communication Concepts

Virtualisation → physical → leads to Processor, logical (VM) Implementation leads to System

Software Stack: Hardware, OS, Middleware/Lib/Runtime Sys, Application

different Programs: Source/Ctr, seq vs app, seq vs parallel, clients vs servers/services

Communication in DS: between processes within single app, among processes of different apps

→ designed for types of env → high perf, M2M, building/home, voice, docs, sensors

→ different network spans → PANs, LANs, CANs, MANs, WANs

→ different layered networks → physical vs network topologies virtual > physical

layered  ← holistic system view

Communication Patterns: one-to-one (client/server), group (1:n, n:1)

Identifiers of entities: → req for communication, local vs global identifier, individual vs group identifier ID multiple layers/entities: Process ID, Machine ID, (IP, name), Access Point (Machine ID+Port), Comm-ID, Port, Group

Pattern 1: user multicasts, agents listen, respond 1-to-1

→ MPI Message Passing Interface

connection-oriented / connection-less: connection-oriented require connection setup

blocking vs non-blocking: [send → transmitting is done, must not have needed target]

blocking → execution suspended until message transmission done, non-blocking doesn't suspend

persistent vs transient: persistent stores message until delivered, trans only kept if both are alive

async vs sync: sync: non-blocking send (callback mechanism), sync: blocking send connection & keepalive

Client Req → send → [ ] → sync → blocking request-reply



stateful vs stateless: less info about prev req, stateless: limited info/lifetime, stateful: maintain per.

out-of-bound-sets: more than one socket → for prioritization, separate data

Communication Protocols:

Conn. patterns: 1:n possible?, Identifier: how?, ConnSetup: without Conn?, MessStruct: language?,

Layered Conn: intermediaries for relay required? = protocol defines rules

protocols may be app-dependent or independent (or middleware layer)

Complex & open → multiple protocols, mostly organised into differ layers

concepts → each layer for certain functionality,

Protocol Suite: set of protocols together in a layered model

OSI Model (Phys > Datalink > Network > Transport > Session > Presentation > App)

Phys: binary transfer, Data: send & detect frames, Net: routing amongst sender & receiver

Trans end-to-end for apps, Sess: conn between apps, Pres: processing format & pass to App

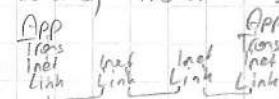
TCP/IP → App / Trans / Internet / Link      HTTP / TCP / IP / HW

IPs def datagram as basic unit, def address space, transmit b/w Net Access & Transport

route to destination, divide & assemble

TCP → App (Data via Streams) Trans (Segments)

UDP → App (Data via Msg) Trans (Packets)



Handshake: → Syn ← syn ack(syn) → ack(syn) req fin ← ack(req+fin) ans fin → ack(fin)

→ not enough to have protocols ⇒ resolving names, elect name coordinator, lock res, sync time

→ middleware: set of general & specific protocols, middleware comm. protocols, spec services

Communication Message / Requests:

Process → OS executes program, threads are within process, thread context switch better than process CS, threads blocking don't block process

Message Passing: processes send & receive, done by set of functions for comm. impl. protocols

Remote calls → remote method invocation local vs remote

Remote object calls → remote object of other process without invocation

process multiple requests → roles (client/server) → many clients, few servers ⇒ concurrent receive  
⇒ impacts on performance, reliability, cost etc

Iterative vs concurrent processing: listen after processing vs dispatch processor while listening

replication e.g. requires load balancing

multiple threads → dispatcher thread handles worker threads

message brokers / topic repositories ⇒ blocking queue

Summ: complex patterns & models, choices on requirements, examples

Message-oriented transient communication:

how does app use transport layer? → socket interface (APIs) popular: Java Sockets

Windows/BSD

Sockets: communication end-point for send/receive in application

Client → send/receive, Server → bind, listen/accept, receive incoming, process send back

Primitives: Socket = end-point, Bind = attach local address, Listen = announce willing to accept  
Accept = block until request, Connect = advise connection attempt,  
Send = , Receive = Close = release connection

Server: SOCKET → BIND → LISTEN → ACCEPT → READ → CLOSE  
Client: SOCKET → CONNECT → WRITE → READ → CLOSE

MPI: for parallel processing, clusters / host perf sys, one-to-one/group sync/async

Concepts: communicators/groups → identifying set of processes to be communicated  
Rank: VID, set of functions to manage exec env, point-to-point funcs  
collective comm funcs, funcs handling data types

MPI Init → init exec env, Comm size → size of group, Comm rank → rank

Send() → send blocking, Recv() → receive blocking, Broadcast() → to others

Reduce() → all values of all processes to single value, Finalize() → termination

Message-oriented persistent communication:

Msg-Queuing / MOM (Msg-Oriented Middleware)

well supp. for persistent but asynd, scalable, several implementations  
stack → PUT, GET, POLL, NOTIFY running / passive

Sender ⇒ lookup Transport-level Address ⇒ sent to QueueReceiver ⇒ Queue-level address

Message brokers ⇒ publish/subscribe: msg match to app, transform: from one format  $\xrightarrow{\text{to app}}$

Remote Procedure Call: how to call remotely similar to local?

→ hide complexity in procedures

Passing Parameters & Results → agreed msg format ⇒ marshaling/unmarshaling  
→ different representations? interfaces?

Interface desc + Msg Format + Transport Info = Stubs: code for encode/decode

client call → stub encodes → transfer → receive → stub decodes → local call

One-Way RPC → call & continue

async RPC → call → request → accept request & tell client

→ Bestätigung sender, dass er bearbeitet wird & später result zurücksenden

RPC implementations: rpcgen - SUN RPC (IDL interface, XDR msg, TCP transp)  
XML-RPC (XML msg, HTTP transport), JSON-RPC (JSON msg, HTTP/TCP transp)  
Server: RMI-Client locates in Server RMI-Registry, that holds published objects  
→ invoke object methods in RMI-Server after location

Streaming data programming:

Data Stream = sequence of data units (read file → send bytes)  
for continuous media, discrete

timings: delay problem & sync: no limit, sync: upper limit, isoc: upper/lower limit  
⇒ multiple streams (complex / multiple streams data processing)

Group Communications:

multicast msg → IP level or app-level

atomic multicast: either each one or no one

reliable multicast: in best effort, not guaranteed

App-level Comm Multi: organized into overlay networks → tree or mesh

→ join to nearest member trees

Gossip-based Data Dissemination → fast spread

N nodes → need to send data ⇒ keep in b, send t times to f random members

Basic Concepts & Design Principles:

Entity: any object (process, file, host, printer...)

Dir Types & Complex Dependencies: among diff. levels, e.g. printer → network, command point → data  
⇒ create & manage entity names & identify

1o Address BINDS to resource, 2o Process ACCESSES, 3o Identifier REFERS TO

Name: set of bits/chars → for entities, collections, specific unique context

Identifier: name that uniquely identifies an entity → to only one entity

Address: the name of an access point

Naming Design Principles: based on specific system organizations & characteristics

Ex: broadcast net: Network/Ethernet, Identifier: IP&MAC, Name res: network address to data link address  
independent nodes: P2P, Identifier: m-bit key, Name res: distributed hash tables

as based on different purposes

Data Structure: simple? no structure at all → avoid set of bits

complex? several data types to reflect aspects

can include location info, GLN (Global Location Number) in logistics

for name-to-address binding: name → address/entity, can change over time → dyn/static

Flat Naming: constructed / flat → just bits ⇒ simple, no location info, Internet Address or Network log  
or n-bit numbers in Distributed Hash Tables

Broadcast based resolution: work to find accesspoint of 'en', broadcast identifier broadcast (ID(en))

→ only en will return accesspoint when msg arrives ⇒ f.i. ARP IP to MAC (broadcast)

Dynamic sys: nodes, no control, join/leave anytime, large amount, each one knows subset

→ f.i. p2p sys, Chord, CAN, Pastry

Distributed Hash Tables: m-bit for keyspace (all identifiers, (Processing)) Node identifier is one key,  
entity identified by  $K = \text{hash}(en)$ ; each node manages range of keys → stores after node  
, nodes relay until message reaches

Chord's circular passing of messages, has successors, Schrittweite w: da nächsten

Successoren wird über Reparatur erreicht 1. Lehr, 2. Lehr, 3. Lehr, 4. Lehr, 18. Lehr

e.g. 1, 4, 9, 11, 14, 18 → FingerTable von 1: 4, 4, 9, 9, 18

Hierarchical approach: top-level domain, directly root of S, subdomain S of T, leaf in S

directory node → several location records, location records: keep info about entity in

& directory, directory nodes have location records & pointers

lookup mechanisms: do i know? no? search @ bis es weiter geht

Structured naming: name spaces are organized into name space

→ as graph (leaf & directories) node == entity absolute from root, relative in btree

Name resolution: N := (1, 12, (n)) dot N → lookup ((1, ident1) in N) directory

→ lookup ((2, ident2) in ident1's directory)

Closure Mechanism → determine where & how name resolution would be started

Enable Aliases Using Links → Hard (link multiple absolute paths), symbolic links  
(left node stores absolute path)  
mounting → remote can be mounted locally using nfs

### None Space Implementation

~ Distributed none management → several servers manage

~ Many Dist. Layers → Global (root & close nodes), Admin (directory within organ), Managerial (large regions)

Characteristics of distribution layers: Global - Admin - Managerial

Geo scale: www, organ, department, #nodes: few, many, West numbers,

responsive: seconds, ms, immediate / update prop: long, immediate, immediate

#replicas: many, none/few, none client config: yes, yes, sometimes

None resolution: iterativ (1 by 1), iterativ server-side (1 server gefragt, erfragt weitere server für Antwort), recursive (server fragt in die Tiefe weiter)

in re Internet: human-readable none hierarchy, machines use IP → DNS (domain → IP)

SOA → Zone, A → Host (IP), MX → Domain (mail server), SRV → Domain (spec server),

NS → Zone (none server implementing zone), CNAME → None (symbol link) for primary none

PTR → Host (canonized none), HINFO → Host (info to host), TXT → org (entity info)

Authoritative name server → owner for zone

Primary & Secondary → main & replicas

Caching server

DNS-Queries: simple hostname res, Email server res, reverse (IP to name), hostinfo, other  
Attributed-based naming: attribute-value tuples for properties → describe entity

→ directory services: naming res (query-mechanism, whole space dealt) → LDAP, RDF

LDAP → Object (info by type → hierarchical)

Directory Entity → for particular Obj, slices or subtrees

Directory Info Base → Collection (each one has DN distinguished name)

Directory Info Tree → tree structure for DiB

### WEB Naming Services:

Web Service → entity providing function via network interface

address points

Web Service identifier → WSDL → serv addresser identifies where & how call service

Service publisher to Registry, Client searches & uses result to call address points

OPEN ID → individual identifiers, no single provider → standardization

access entity → access relying party & authenticate with provider → access

Physical Clock Sync? → accountability of processes, consistency of msg-processing, validity of important msg, fairness for requests

challenge: impossible to sync different computers → clock drift

→ establish/decide accurate timing system (clock) → UTC (Coordinated Universal Time)

UTC bases on high accurate physical clocks (atomic)

→ generate/utilize by providing UTC time

→ sync using time sync algorithms

everyone has clock, radio clock → via waves (connects to UTC source)

GPS → system of satellites that broadcast → position, time stamp

Cristia Algo: send A → rec B → send B → rec A      RTT = rec A - send A - (send B - rec B)  
→ t rec A = max (t rec A, t rec B +  $\frac{RTT}{2}$ )

Berkeley Algo: sonnelt alle Differenzen, dividiert Differenz durch # Servers  
sucht korrekturent zu allen Servern (1 Time beacon)

Network: UTC ↔ Primary ⇔ Secondary: UDP Multicast, Gossip, Symmetric

Logical Clocks? no need for exact phys timing, only maintain physical causality

→ correct order of execution

happens-before relation ( $\rightarrow$ )  $a \rightarrow b$  → a may affect b      I proofs

compts logical clock + sync before exec → push clock to other process, no adjust

→ realized using a middleware clock

Limitation? Weiterdenken → Kausale Zusammenhang von 2 Sources sollte + eingespielt?

Vector Clock: if  $VC(a) < VC(b)$  → a precedes b

$VC_i(j) = k$  → k events in  $P_j$  causal related to  $P_i$

Apps: reply totally order multicast, atomic multicast & accept msgs in some order

MM (real-time teleconference) causal multicast → broadcast order: recv order

→ causal counter  $(1, 1, 0)$  not be accepted by  $(0, 0, 0)$ , only by  $3$

$(1, 0, 0)$  or  $(0, 1, 0)$

Mutual Exclusion: only 1 user can access a resource at given time

→ prevent inconsistency / corruption

→ approaches: token / permission

Centralized: lock & unlock SEMAPHORE

Distributed Algo (Ricart, Agmonalo, Lamport)

Sort ist? wenn jeder jemals sagt  $\rightarrow$  OK, wenn jemand sagt, in quelle Stellung  $\rightarrow$  wenn früherer später vorkommt, der spätere hat nicht zu geprüft hat  $\rightarrow$  OK (sonst queue)

Ring Algo if token  $\rightarrow$  access & release  $\rightarrow$  pass token (or just pass token)

ELECTION ALGORITHMS:

Leader Election: coordinator, selected from set of processes

why challenge? distributed, multi processes involved

algos: to elect, uniquely defined, during int or coordinator fail

Bully algo: 1st fails election, asking the highest priority processes

$\rightarrow$  they respond "OK" and tells below them  $\rightarrow$  highest rated gets reward

Ring algo: (LeLan, Chong, Roberts) open in ring, int non-participant

election & coordination messages, forwarded, precedes & sent clockwise

forward in cycle until greatest process is found and propagated through

Simple flood: directed graph

$\rightarrow$  Primitives have VID, at round pass this VID, after n rounds

Process sees its VID and gets leader

Consistency & replication:

Replication: process of maintaining several copies at different locations

Consistency: process of keeping copies same or changes

$\rightarrow$  Ø<sub>1</sub>: ~~no replicas~~ serve role servers, retire lottery & bandwidth (check load)

$\rightarrow$  Ø<sub>2</sub>: replica up-to-date consumes bandwidth, updates are not immediately

pick two: Performance, Scalable, Consistency

updates require agreement to all replicas  $\rightarrow$  costly

Mitigation = avoid instantaneous global sync

Maintaining Performance & Scalability: keep consistent  $\rightarrow$  ensure conflicting operations are done in order  
read-write (concurrently) & write-write

Issue: global ordering may be costly  $\rightarrow$  scalability  $\Rightarrow$  weaker consistency

Data-centric consistency: consistency model  $\rightarrow$  contract what read/write should result in

Observation: degree of consistency: differ in numerical value, relative staleness

differ in performed update operations

consistency unit: how consistency is measured (webpage, table entry, entire table)

Price  $\rightarrow$  to max 10ms diff, to detect for staleness each 10sec,

$\rightarrow$  to no more than 1's unperformed updates

Contract: prot. Unterschied schreibt wird repliziert hinzufügt Schrift

Ordering Operations: we wait read from most recent write  $\rightarrow$  which one?

Relax timing: intervals R/W, how it conflicts?, agree on consistent global ordering

Seq consistent: We W<sub>b</sub> R<sub>b</sub> R<sub>a</sub>  $\rightarrow$  other process reads a before b  $\rightarrow$  inconsistent

$\Rightarrow$  every reader reads in the same order

Consistent: writes must be seen in some order

$\rightarrow$  no read before consistent related writes are done

FIFO consistent: writes of one process are ok, but from different processes?

$\rightarrow$  writes are always assumed to be concurrent

Grouped operations: access to sync vars are sequential consistent

no access to sync var previous writes are pending

no access to table until all sync variable accesses were performed

$\rightarrow$  save view on lock, no unlock until view is synced, no just if view is sync'd changes  $\rightarrow$  result of effect must be known

Avoid Polarizing consistency: concurrent processes: sudden updates, consistency & isolation required.

Lock of Hen: lock of updates, easy readable inconsistencies, focus on single client

## Client-Centric consistency models:

- ~ System model: a readic reads a monotonic writes  $w_1, w_2, \dots, w_n$
- ~ write-follows-reads

Google focus on client writes instead of server maintenance  
for mobile users?

work on location A, then on location B  $\Rightarrow$  inconsistencies?

A wasn't propagated to B, newer things than the ones in A, conflicts?

$A = B \rightarrow$  consistent

$\Rightarrow$  update at one replica, lousy propagation, ev. all are updated, consistent after

Mutual Reads: if one read  $x$ , every other node reads no earlier value of  $x$   
replicas:  $WS(x; \Sigma)$  Write, version; time  $t \rightarrow$ ; leads to pos

$x; \Sigma_t; v_j[t] \rightarrow$  first is part of read

M writes  $\Rightarrow$  write is done before any other may write on it

Read your writes: always read after write  $WS(x; \Sigma)$

Write follows read: writes only on last read or no read value

## Know your Consistency models!

Replica Consistency Concerns: consistent to read, no updates  $\Rightarrow$  problem

Access-to-update  $\uparrow \Rightarrow$  replica helps

Updates-to-access  $\uparrow \Rightarrow$  updates will not be consumed

ideal  $\rightarrow$  only update those that will be accessed, help replicas in priority to clients

Replica Management: placement = management/commercial issue

Content replication & placement  $\Rightarrow$  distribution (state vs operator, push vs pull vs lease, blocking vs non-blocking, unicost vs multicast)

Placement: figure k-best out of N possible locations

$\Rightarrow$  average distance to clients is minimal  $\Rightarrow$  computational expensive

$\Rightarrow$  select with largest bandwidth system  $\Rightarrow$  put server to best connected host

$\Rightarrow$  distribute latency equivalent  $\rightarrow$   $k$  regions of highest density = cheap

Content replication: distinguish diff processes

$\Rightarrow$  persistent: always hold on, initial set, LAN cluster, geo + max

$\Rightarrow$  server init: can dynamically hot replace on request / performance (push mode, web basing server) reduce servo load  $\rightarrow$  proximity of req. clients

$\Rightarrow$  client-init: process to dyn. hold replicate (client cache)

Server init  $\rightarrow$  count requests for a file  $\uparrow$  threshold prep  $\downarrow$  threshold  $\rightarrow$  drop

between  $\rightarrow$  migrate file

content distribution  $\Rightarrow$  client servers only propagate notify & invalidation,

transfer solo or passive replication / propagate update protocol  $\rightarrow$  active replication

no single approach is best  $\Rightarrow$  depends on usage & write-read ratio's

push-based protocols: updates are pushed, used by general servers instead

mainly  $\uparrow$  degree of consistency, if server keeps track of clients

$\Rightarrow$  they have coded sole  $\Rightarrow$  stateful server (limited scalability, b. fault tolerance)

multicast is more efficient

pull/drift based protocols: push for all updates, for client updates, client

pulls for updates, high responsive if code size, unicasting

(client/server content distribution)

pushing = server-init; pulling = client-init

state of server  $\rightarrow$  list of client codes / none

resp. to be excl  $\rightarrow$  update (batch update) / poll & update

resp. time of client  $\rightarrow$  immediate (or batch update time) / fetch-update file

switch b/w push & poll  $\rightarrow$  push in the stamps

Issue: loose expiration times on system's behaviour separation

$\Rightarrow$  Age-based: file hasn't changed for long  $\rightarrow$  long-lasting-load

$\Rightarrow$  request freq based: the more requests, the longer the expiration time

$\Rightarrow$  state-based: the more loaded, the shorter the expiration time

$\Rightarrow$  to reduce servers load while still strong consistent

blocking vs non-blocking: when push propagate?

sync: blocking/earlier, immediate replicate, then reply

async: non-blocking/lazy, apply to one copy, reply, propagate afterwards

Consistency protocols:

Continuous consistency, primary-based protocols, replicated write protocols

Conf. consistency: numerical errors

Principal question: server has log, numerical changes by whom weight older write

Weight forwarded to one of the N replicas,  $TW[\Sigma_{ij}]$  are writes  
executed by server  $S_i$  that originated from  $S_j$

## NUMERIC ERRORS WITH?

Primary-based protocols:

writes zu Primary weitergeschickt werden, updated alle backups,

Wichtig: Bestätigung erwartet und bestätigt dann

Sync-replica:  $\oplus$  no inconsistency, reading local copies is up-to-date, static changes

$\ominus$  write updates everything, slow, not resilient against network/node failure

async-replica:  $\oplus$  fast, only primary updates immediately, resilient against ?

$\ominus$  inconsistency possible, local read not always up-to-date

Primary based possible:

$\oplus$  at least one is up-to-date, ordering  $\rightarrow$  easy to achieve (no intern sync req)

$\ominus$  Primary is bottleneck & point of failure, high recovery cost if fail

Quorum-based shot

Distributed File Systems: have fs transparently available

$\rightarrow$  Remote-Access model: requests & so, file stays on server

$\rightarrow$  Upload / Download model: 1. DL 2. Access 3. Upload

NFS  $\rightarrow$  Virtual File System abstraction

Local FS  $\leftarrow$  VFS  $\rightarrow$  NFS client  $\rightarrow$  RPC stub cl  $\rightarrow$  RPC stub svr  $\rightarrow$  NFS server  $\rightarrow$  VFS  $\rightarrow$  Local FS

VFS provides standard FS interface

Cluster-based FS: for  $\uparrow$  data collections  $\rightarrow$  speed up accesses  $\Rightarrow$  striping (parallel fetch)  
 $\rightarrow$  mehrere Platten utilizieren, Files zerlegen und auf dix. Platten laden

Google FS  $\Rightarrow$  64 MB chunks, verteilt & repliziert, moster main memory: filename/chunk lookuptable  
replication via primary-backup master kept out of the loop

$\Rightarrow$  concurrent modification & consistency models

UNIX storage semantics: read = lost write  $\rightarrow$  only implementable if one instance of a file exists

Transactions semantics: FS supports transactions  $\rightarrow$  how allow concurrent to plus distributed file

Session semantics: read & write local, what happens on close?

Code: transaction wise  $\rightarrow$  readopen Session becomes invalidated if written during viewing

Cors & repl: disk-side coding for performance & server-side impl. for fault tolerance

$\rightarrow$  Cors  $\rightarrow$  schreiben, wenn. most recent lokaler id, muss nicht wieder herabladen

repl code  $\Rightarrow$  detection of cors. Accessible Vol Star Group  $\rightarrow$  Version Vector

Security: low lv = convenient, higher lv  $\rightarrow$  allows secure comm. over insecure channel = SSL over TCP  
 $\rightarrow$  weakest link principle  $\Rightarrow$  typically the human

security must base on tech & math facts, not security by obscurity

Cryptography: symmetric & asymmetric keypairs

hosting: no decryption  $\Rightarrow$  one-way, weak/strong collision resistances

sym  $\rightarrow$  interceptor, asym  $\rightarrow$  fabrication, hosting  $\rightarrow$  modification

Shared Secret Authentication  $\xrightarrow{A \rightarrow} R_B \rightarrow K_{A,B}(R_B) \rightarrow R_A \leftarrow K_{A,B}(R_A)$

Opt?  $\rightarrow A, R_A \& R_B K_{A,B}(R_A) \rightarrow K_{A,B}(R_B)$

$\Rightarrow$  Reflection reuses sent & received secrets  $\rightarrow$  entrieder als middle man

$\rightarrow$  anti man-in-the-middle: key distribution center  $\rightarrow$  oder nur provider

asym key pairs  $\Rightarrow$  comp expensive  $\Rightarrow$  negotiable sym key using asym

Diffie-Hellman:  $\rightarrow n, g, g^x \bmod n$   $(g^y \bmod n)^x = g^{xy} \bmod n$

$$g^y \bmod n \quad (g^x \bmod n)^y = g^{xy} \bmod n$$

How to know identity?  $\Rightarrow$  Certification Authorities  $\Rightarrow$  pub key mapping  
 $\Rightarrow$  wear-and-tear  $\Rightarrow$  expiration, revocation

Integrity? sign with pair, prove with pub OR Hash Dig Signature

Access Control? request needs to be authorized

no Access Control Matrices as Access Control Lists no Protection Domains

Matrices: Access Lvs RWT, RW, R, -

Lists? request  $\rightarrow$  if dc in ACL  $\rightarrow$  grant access

Common Attacks:

$\rightarrow$  compromiseable on every layer

+ Buffer Overflows? input  $\geq$  space, overwrites bytes (possibly executable code)

+ SQL Injection? input validation missing

+ XSS: inject arbitrary scripts into trusted web site

+ DDos: overload resources, how protect? how identify?

+ Sidechannels: find out secret, timing, steal priv keys, reverse-engineering

+ Social Engineering: target human, easily if good follower, Phishing

Dependability: components with services  $\rightarrow$  each one must operate correctly  $\Rightarrow$  dependable and Available, Reliable (ephemeral), Safe (no catastrophic conseq), Integrity, Maintainable

Threats: Failure (functionality may), Error (deviation of expected state), Fault (error cause)

$\Rightarrow$  Fault (bug)  $\rightarrow$  Error (Wrong return value)  $\rightarrow$  Failure (incorrect service)

$\Rightarrow$  Fault (select JS)  $\rightarrow$  Error (bit I/O Errors)  $\rightarrow$  Failure (not correct copied)

Fault classes: developed, operational, HW, SW, Malicious Accident, Incompleteness

Failures: crash (comp halts), omission (no response), timing (performance), response (good input, bad output), arbitrary (byzantine - fault & the arbitrary)

Fault Tolerance: it will happen, partial failure  $\rightarrow$  only one component

$\rightarrow$  Fault prevention,  $\rightarrow$  Fault forecast,  $\rightarrow$  Fault Tolerance (crash & continue)

$\rightarrow$  Fault removal ( $\downarrow$  # & severity)

$\Rightarrow$  Approaches: redundancy  $\rightarrow$  hide occurrence

info  $\rightarrow$  extra info good, time  $\rightarrow$  repeat requests, phys  $\rightarrow$  add components

info (parity bit, error check codes, RAID 4+5), time (retransmission, recall)

phys (backup server, HW Raid 1, different implementations same functionality)

$\rightarrow$  Basics: organize data processes in groups (flat or hierarchical)

hierarchical  $\Rightarrow$  coordinator & worker, not (really) fault/scalable  $\rightarrow$  but easy

flat  $\Rightarrow$  info always exchanged, overhead? control is distributed  $\rightarrow$  hard to implement

k-fault  $\Rightarrow$  most k concurrent member failures

how large? crash/part k+1, arbitrary/binary 2k+1  $\Rightarrow$  members are identified via order

Byzantine Agreement: 1) Message 2) Vector results 3) Spread vector, 4) prove majorities

$\Rightarrow$  agreement!  $\rightarrow$  need majority required for k-fault

remote? 1) no locate 2) request (as) 3) zero crash by response lost 5) crash crash

1) exception handling 2) resend 3) now/after exec/before exec

$\Rightarrow$  how recover? at least once carried out, at most once

client-side? always receive, never receive, only if no ACK, only if ACK

$\Rightarrow$  can't cover every case

4) did it crash? operation carried out? repeat? no "real" solution

5) option coordination  $\rightarrow$  kill option / reincarnation, tell server / expiration

Recovery: identify consistent state  $\rightarrow$  error-free state, forward & backward

backward  $\rightarrow$  check pointing, is applicable, relative costs, re-exec?, irreversible?

$\Rightarrow$  recovery be fitable?

$\Rightarrow$  global coordinator 1) issue checkpoint request, 2) on receive pause & report

3) all reported? good 4) continue

Msg Logging: less costly, not entirely free of checkpoints  $\Rightarrow$  try replay cam behaviour

basic assumption: seq of state intervals  $\rightarrow$  state starts nondeterministic execution is

$\Rightarrow$  record non-deterministic events to allow replay

no options if good logged

Internet of Things: phys obj seemlessly integrated, active participants in business processes

phys obj  $\rightarrow$  smart obj technologies: RFID, sensor networks, IPv6

future: use power of diff companies, tools for creation, optimization & exchange

Internet of Services: services, REST, UML, SOAP, cloud computing

IoS = SOA? SOA = concept & structures in a company

P2P: direct interaction, no client/server asymmetry, similar services & consumers can peer

$\Rightarrow$  self-organizing, equal peers, shared usage of no dedicated

$\Rightarrow$  avoids central services  $\Rightarrow$  peers act as server & client

+ scalability, autonomy, decentralization, self-organization, shared resources

Overlay network: Virtual network over internet

Why? costs are outsourced, high extensibility, scalable, fault tolerant, resilient to loss  $\leftarrow$  3

Motivation: globalization, cross-organizational workflows, Business Process Outsourcing, flexibility?

Service-oriented computing  $\rightarrow$  workflows, invoke other services

Roles: service provider, consumer, intermediary (broker)

Cloud Computing: demand varies (peak loads), demand is unknown in advance, bold analytics Shd=165  
on demand provision of resources

NIST Def: o-access of web interfaces, broad net access, virtualization techniques,  
rapid elasticity (virtually unlimited capacity & scalability) pay-as-you-go

Cloud: Infrastructure as a Service  $\rightarrow$  deliver infra as service (VM)

Platform as a Service  $\rightarrow$  computing platform (execution environment)

Software as a Service  $\rightarrow$  ERP Software

Deployment Methods: Private Cloud (only for one), Community Cloud (shared)

Public Cloud (open/sell by services), Hybrid Cloud (composition of methods)