

## 0.1 Verteilte Systeme/Distributed Systems

### 0.1.1 Orga

VL Di 10-12 (nicht am 23.04.)  
Ue Do 10-12

#### Elektisches

- (kvv)
- Website AG
- Sakai

#### Übungen

- ca. 5 Übungsblätter, 14-tägig
- Vorträge in Gruppen über „verteilte Systeme“

#### Material/Inhalt

##### 1. Hälfte Distributed Systems (Tanenbaum, van Steen)

- Architektur
- Prozesse
- Kommunikation
- Namen
- Synchronisation
- Konsistenz
- Replikation
- Fehlertoleranz

##### 2. Hälfte Distributed Algorithms (Nancy Lynch)

- synchronous network algorithms
- network models (leader election, shortest path, distributed consensus, byzantine agreement)
- asynchronous network algorithms (shared memory, mutual exclusion, resource allocation, consensus)
- timing
- network resource allocation
- failure detectors

## 0.2 Distributed Systems

**Def:** A distributed System is a collection of independent computers that appears to it's users as a single coherent system.

Characteristics:

- autonomous components
- appears as single system

- communication is hidden
- organisation is hidden  
(could be high-performance mainframe or sensor net)
- heterogenous system offers homogenous look/interface

Objectives:

- provide resources (printer, storage, computing)
  - share in a controlled, efficient way
  - grant access
    - ⇒ connect users and resources

Transparency:

hide the fact that processes and resources are physically distributed.

Types of transparency:

access hide differences in representation and how a resource is accessed

location

migration move resources

relocation move resources while using

replication

concurrency

failure

transparency is desirable, but not always perfectly possible

tradeoff between transparency and complexity, maintainability and performance

### **Open System**

- service interfaces specified using Interface Definition Language (IDL)
- service specification as text

**Scalability** is an important property

- scalable in size (number of nodes)
- scalable in geographic spread
- scalable in administration

### **Problems**

- centralized services
- centralized data
- centralized algorithms

### **Scaling techniques**

- use only asynchronous communication
- distribution, split components

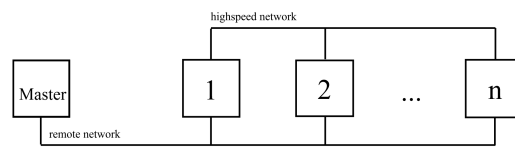


Abbildung 1: cluster computing

- replication of components

#### **pitfalls**

1. reliable network
2. secure network
3. homogenous network
4. constant topology
5. zero latency
6. infinite bandwidth
7. zero transport cost
8. one administrator!

#### **Types of distributed systems**

- computing systems
  - cluster computing
  - grid computing(virtual organisation, geographically distributed and heterogenous))
- distributed inforamtion systems (note the typo)
  - transaction processing systems (database)  
ACID (atomicity, consistency, isolated, durable)
  - enterprise systems
- Distributed pervasive systems
  - small, wireless, adhoc, no administration
  - home automation, health systems, sensor networks

#### **Why do we need distributed systems?**

- performance
- distribution inherent
- reliability
- incremental growth (scalability)
- sharing resources

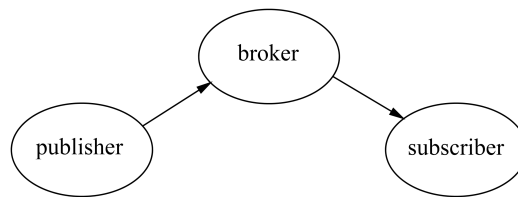


Abbildung 2: publish subscribe system

## 0.3 Architectures of distributed Systems

- how to split software into components  
⇒ Softwarearchitecture
- how to build a system out of the components  
⇒ Systemarchitecture

Middleware can help to create distribution transparency

Architecturestyles:

- Layered architecture  
⇒ network stack, messages or data flow up and down
  - control flow between layers
  - requests down
  - reply up
- Object-based architectures
  - interaction between components
  - e.g. remote procedure calls
  - can be client-server system
- data-centered architectures
  - data is key element
  - communication over data, distributed database
  - web-systems mostly data-centric
- event-based architecture
  - publish-subscribe systems
  - processes communicates through events
  - publisher announces events at broker  
⇒ loose coupling (publisher and subscriber need not to know each other), decoupled in space  
⇒ scalability better than client-server, parallel processing, caching

Event-based and data-based can be combined

⇒ shared Data space

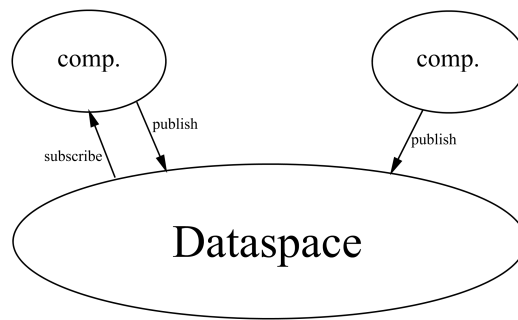


Abbildung 3: shared data space

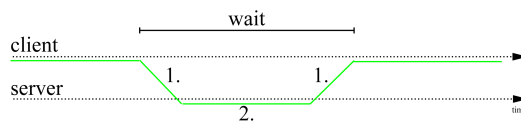


Abbildung 4: client server simple waiting situation

### 0.3.1 System architectures

#### 1. centralized architectures client - server

- (i) single point of failure
- (ii) performance (server is bottleneck)

(a) communication problems

(b) server problems

can request be repeated without harm?  
⇒ request is idempotent

#### (iii) application layering Layers:

- 1.) User interface
- 2.) processing
- 3.) data level

⇒ a lot of waiting  
⇒ does not scale

#### 2. Decentralized architectures

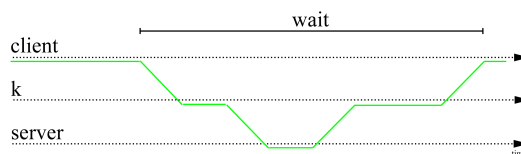


Abbildung 5: application layer

- vertical distribution (layering)  
different logic on different machines
- horizontal distribution  
replicated client/server operating on different data  
⇒ overlay-underlay hides physical structure by adding logical structure

#### Structured P2P architectures

- most popular technique is distributed hashtables (DHT)
- randomly 128 bit or 160 bit ke for data and nodes. Two or more keys are very unlikely
- Chord system arranges items in a ring
- data item  $k$  is assigned to node with smallest identifier  $id \geq k$

ie item 1 belongs to node 1

item 2 belongs to node 2

for each item  $k_i$   $\text{succ}(k)=id$

returns the name of the node  $k$  is assigned to

to find data item  $k$  the function  $\text{LOOKUP}(k)$  returns the adress of  $\text{succ}(k)$  in  $O(\log(N))$ (later!)

membership management

join:

create SHA1 identifier

$\text{LOOKUP}(id) = \text{succ}(id)$

contact  $\text{succ}(id)$  and  $\text{pred}(id)$  to join ring

leave:

node  $id$  informs  $\text{succ}(id)$  and  $\text{pred}(id)$  and assigns it's data to  $\text{succ}(id)$

#### Content adressable network (CAN)

- d-dimensional cartesian space
- every node draws random number
- space is divided among nodes
- every data draws identifier (coordinates) which assigns a node
- join
  - select random point
  - half the square in which  $id$  falls
  - assign item to centers
- leave
  - one node takes the rectangle
  - ⇒ reassign rectangles periodically

#### Unstructured P2P Network

- random graph
- each node maintains a list of  $c$  neighbours
- partial view or neighbourhood list with age

- nodes exchange neighbour information  
active thread  
select peer

#### PUSH

select  $c/2$  youngest entries+myself  
send to peer

#### PULL

receive peer buffer  
construct new partial view  
increment age

passive thread  
recieve buffer from peer

#### PULL:

select  $c/2$   
send to peer  
construct new partial view increment age

## 0.4 PeerSim

## 0.5 Processes

### processes

- execution of program
- processor creates virtual processor
- for each program everyting is stored in process table
- transparent sharing of resources,(processor, memory) separation
- each virtual processor has it's own independent adress space
- process switch is expensive, (save cpu context, pointers, translation lookaside buffer (TLB), memory management unit (MMU))
- perhaps even swaps to disk, if memory exhausted

2 possible solutions:

1. scheduler activation, upcall to achieve process switch
2. light-weight processes (LWP)  
user level thread package  
execute scheduler and run thread of parent  
may block on systemcall, then other LWP may run  
triggered from userspace

### threads

- several threads share CPU
- thread context has little memory information, perhaps mutex lock
- threads avoid blocking application (e.g. spreadsheet,computation of dependent cells, intermediate backup)
- thread switch is fast
- user level threads allow parallel computation of program sections
- I/O or other blocking system calls block all threads, but thread creation/deletion is kernel task = expensive
- advantages of threads over processes vanishes

Advantages of LWP and user-level thread package:

1. creation, deletion etc is easy, no kernel intervention
2. blocking syscall does not suspend process if enough LWPs are available
3. applications do not see LWP. They only see user-level threads
4. LWP can run on different processors in multiprocessor systems

Disadvantages:

1. LWP creation as expensive as creation of kernel-level thread

Advantages:

- a blocking systemcall blocks only thread, not process  $\Rightarrow$  system call for communication in distributed systems

Multiple threads in clients and servers

#### **Clients:**

- multiple thread may hide communication delay (distribution transparency)
- web browser opens several connections to load parts of a document/page
- web server may be replicated in same or different location  
 $\Rightarrow$  truly parallel access to items and parallel download

#### **Servers:**

- single threaded, e.g. file server  
thread serves incoming request, waits for disk, returns file  
serves next
- multithreaded  
dispatcher thread receives request  
hands over to worker thread  
waits for disk etc.  
dispatcher takes next request
- finite state machine  
only one thread  
examines request, either read from ... or from disk  
during wait stores requests in table  
serves next request  
manage control either new request or reply from disk  
act accordingly  
process acts as finite state machine that receives messages and acts/changes state

#### **summary:**

model	characteristics
single thread	no parallelism, blocking syscalls
multi thread	parallelism, blocking syscalls
finite state machine	parallelism, non-blocking syscalls

### **0.5.1 Virtualisation**

V pretends there are more resources than available.

Reasons for the need for V.

-hardware changes much faster than SW



⇒ improves portability  
-networks consist of different hardware  
⇒ enables portability of programs for all  
usage (distributed applications, network protocols)

## 2 Types of Architectures for Virtualisation:

### 1. Runtime system providing instruction set

- interpreted as Java
- emulated as for Windows applications on UNIX-platform processes VM

### 2. Virtualisation shields hardware and offers instruction set of the same or other hardware

- can host different OS that run simultaneously  
⇒ VMM such as VMware, Xen

## 0.5.2 Client-/Serverprocesses

### Clients:

- b) allows to store data at the server
- **thin client** e.g. X-windows
- thin client should separate application logic from user interaction
- often not implemented ⇒ poor performance
- compression of interaction commands as solution
- compound documents where user interaction triggers several processing steps on the server. must be implemented (e.g. rotation of picture changes placement in texts)

### Servers:

- serves requests on behalf of the client
- Types of servers
  - **iterative Server** handles requests itself
  - **concurrent server** passes requests to worker, e.g. multithreaded server
- server listens to port, endpoint to the client; some ports are reserved for special services
- superserver listens to several ports, replacing several (mostly idle) servers
- stateless servers, keeps no information on state of client → change state without informing the client, e.g. web server
- soft state server, maintains client state for limited time, e.g. servers informing about updates
- stateful server keeps information about client (file server keeps (client, file) table), often better performance, fault-tolerance poorer
- cookies allow to share information for server upon next visit client sends its cookies, allows state information for stateless server

## Distributed Servers

- servers in different locations that have different ip-addresses in DNS under the same name
- MIPv6: mobility support for IPv6
- mobile node has home network with stable home adress (HoA)
- special router is home agent and takes care of traffic to the mobile node
- mobile node receives care-of-adress (CoA), never seen by client
- route optimisation avoids routing through home agent

### 0.5.3 Code Migration

- Code migration on (running) process - Why?
- service placement in distributed system  $\Rightarrow$  minimize communication cost
- load balancing in multiprocessor machine or cluster  $\Rightarrow$  performace
- (security)

#### Models of Migration

- or process model
  1. code segment, instructions
  2. resource segement, references to external resources, i.e.e. file, printer, devices
  3. execution segement, execution state process, stack, private data, programm counter
- Migration types
  - weak mobility, transfer code, (1), mabe 3)), which executes from beginning (i.e. java applets)
  - strong mobility, transfer 1)3), stop executions, transfer, resume

Migrating resource segment 2) is difficult  
Consider process to resource binding

1. binding by identifier, URL, ftp-server-name
2. binding by value, libraries for programming
3. binding by type, local device, monitor

#### Resource-machine-binding

1. unattached
2. fastend
3. fixed

pass tp resource binding	unattached	fastened	fixed
by identifier	MV	GR(or MV)	GR
by value	CP	GR(or CP)	GR
by type	RB	RB(or GR,CP)	RB(or GR)

ference, CP: copy value, RB: rebind to locally available resource

MV:move, GR, global re-

## 0.6 Communication

we skip networking → Telematik

Consider:

- Remote procedure call
- Message-oriented communication
- Stream-oriented communication
- Multicast communication

### 0.6.1 RPC

Remote procedure call uses stubs to pack parameters in message value parameters

⇒ value packed in messages, transfer byte-by-byte ⇒ problem can be little endian vs big endian systems  
reference parameters are: extremely difficult; create array and pass by value; how to handle graphs, linked lists...

Remote procedure calls

### 0.6.2 Asynchronous RPC

- hide communication, communication transparency

### 0.6.3 Message oriented communication

- avoids synchronous communication which blocks sender Berkeley sockets UNIX TCP/IP  
server |socket|->|bind|->|listen|->|accept|->|read|<->|write|->|close|  
client |socket|----->|connect|->|read|<->|write|->|close|

### 0.6.4 Message-passing-interface (MPI)

- standard for communication
- communication within group of processes
- each group/member has identifier

### 0.6.5 Message-queuing-system, Message-oriented-middleware (MoM)

- asynchronous persistent communication
- store messages
- transfer may take minutes, not milliseconds
- applications communicate by inserting messages into queues
- messages are inserted into local queue

- message carries destination address
- queue manager may act as relays, router
- message broker transform type A into type B, using a set of rules
- applications are email, workflow, batch processing, queries accross several databases

#### 0.6.6 stream oriented communication

- temporal relationship between items important
- multimedia data is compressed
- QoS is important
  - bit rate
  - max delay for session setup
  - max end-to-end delay
  - max delay variance (jitter)
  - max round trip delay
- networking solution such as differentiated services
- synchronisation of streams

#### 0.6.7 Multicast communication

- application level multicast uses overlay
- tree, unique path between each pair of nodes
- mesh, more robust, fault-tolerant

**Example:** Construct overlay tree for chord

- node that wants to start multicast generates key 128bit/160bit (nid) randomly
- lookup of succ(nid) finds node responsible for key mid  
⇒ succ(nid) becomes root of tree
- join: lookup (nid) creates lookup message with join request routed from P to succ(nid)
- request is forwarded Q (first time forward), Q becomes forwarder  
⇒ P child of Q
- request is first time forwarded by R, R becomes forwarder  
⇒ Q becomes child of R
- multicast: lookup(nid) sends message to the root  
multicast from root

#### Efficiency?

Quality of application level tree

1. Link stress, number of traversals of same link per packet
2. stretch, relative delay penalty (RDP)  
$$\frac{\text{transmission time in overlay}}{\text{transmission time in delay/network}} \Rightarrow \text{minimize aggregated stretch, average RDP over all note pairs}$$
3. tree cost, minimize aggregated link cost, link cost = cost between end points  
⇒ find minimal spanning tree

### 0.6.8 Gossip-based-communication

- epidemic behaviour
- a node does not have new data (susceptible), it has the data (infected) or is unwilling to spread (removed)  
Anti-entropy-model  
P chooses randomly Q
  1. P pushes its data to Q
  2. P pulls Q's data
  3. P and Q exchange data
- if many nodes are infected probability for selecting susceptible node is low  
⇒ low probability of data dissemination
- pull works when many nodes are infected. Susceptible node determines spread. They have a high probability to contact infected nodes
- if only one node is infected push/pull is best
- Round is period in which each node at least once selects a neighbor  
number of rounds needed to spread  $\approx \mathcal{O}(\log(N))$ ,  $N$  is number of nodes

#### Rumor spreading, gossiping:

function of nodes that never obtain data:  $s = e^{-(k+1)(1-s)}$

e.g.  $k = 4, \ln(s) = 4,97$

⇒  $s = 0,007$

less than 0,7 remain without data

removing data is difficult : delete message is send via gossiping