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 transparancy:

access	hide differences in representation and how a resource is accessed
location	
migration	
relocation	
replikation	
concurrency	
failure	

transparancy is desireable, but not always perfectly possible
tradeoff between transparancy and complexity, maintainablility 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 techiques)

- use only asynchronous communication
- distribution, split components

- replication of components

pitfalls

reliable network

secure network

homogenous network

constant topologgy

zero latency

infinite bandwith

zero transport cost

one administrator!

Types of distributed systems

- computing systems
  - cluster computing
  - grid computing(virtual organisation, geographically distributed and heterogenous))
- distributed inforamtion systems
  - 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

0.3 Architectures of distributed Systems

- how to split software into components
  ⇒ Softwarearchiticture
- 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 threough 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

0.3.1 System architectures

centralized architectures
client - server

(i) single point of failure

(ii) performance (server is bottleneck)
can request be repeated without harm?
⇒ request is idempotent

(iii) aplication layering
Layers:

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

⇒ a lot of waiting
⇒ does not scale

Decentralized architectures

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 assigened 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 succ(k)=id
returns the name of the node k is assigened to
to find data item k the function LOOKUP(k) returns the adress of succ(k) in O(log(N)(later!)

membership management
join:
create SHA1 identifier
LOOKUP(id) = succ(id)
contact succ(id) and pred(id) to join ring

leave:
node id informs succ(id) and pred(id) and assigns it's data to succ(id)

Content adressable network (CAN)

- d-dimensional cartesian space
- every node draws random number
- space is divided among nodes
- every data draws identifier (coodinates) 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