1. Communication

Middleware

- Mellem netværkslaget og applikationslaget
- Definerer protokoller som applikationer kan benytte
- Application-independent services
- Skaber transparency

Kommunikationstyper

- Persistent vs. transient
  - Message queueing systems (persistent)
  - RPC (transient)
- Synkron vs. asynkron
  - RPC (som regel synkron)
  - Email (asynkron)
- Persistent + synkron giver ikke mening

RPC

- Transparency - arbejder med “lokalt” objekt
- Klient stub, server stub - send klient stub til server
- Marshalling - tolk data
- Call by reference
  - Generelt - call-by-copy/restore
  - Java - send klient stub, two-way transparency og RPC
    - Potentielt meget trafik
- Drawbacks
  - Koblet i tid (synkron, transient)
  - Kræver stubbe på serveren

Decentraliseret kommunikation

- Hvorfor? Centralisering ikke mulig/attraktiv
  - Undgå flaskehalse
- Alle enheder er equal - ingen koordinator
- Epidemiske protokoller
- Push baseret vs. pull baseret
- Gossiping
  - Baseret på real life gossiping/hearsay
  - Push-baseret
  - Send data til interesserede enheder
  - Mist interesse efterhånden som alle har data
  - Ved joins benyttes pull

2. Naming

Hvad skal navngives?

- Enheder - kan skifte adresse, have flere access points
- Access points - kan skifte enhed
- Adresser - human-readable, cs.au.dk > 130.225.12.1

Tre typer

- Flat naming
  - Siger intet om name resolution
  - Broadcast i LANs
  - Hashing i DHT-systemer
- Structured naming
  - Namespace representeret som DAG
  - Resolution ved at følge kanter
- Attribute-based naming
  - Attribute-value pairs
  - Brugbart for søgemaskiner, søg på attributter
  - LDAP - standardiserer attributter og deres værdi-domæner

Flat naming

1. Broadcasting and multicasting
   - Network flooding
   - OK til LAN
2. Forwarding pointers
   - Lang kæde af pointers, many points of failure
   - Shortcuts, men “efterladenskaber”, som ikke kan fjernes
3. Home base
   - Kombinere forwarding pointers med en home location
   - Home location er statisk, og opdateres hver gang enheden ændrer location
   - Kan kræve meget unødig trafik, give unødige delays

- Andre approaches på decentraliserede systemer

Structured naming

- DAG over namespace, følg kanter
  - Enkelte elementer skæres af strengen efterhånden - cs.au.dk, cs.au, cs
- Namespace distribution layers: global, administrational, managerial
- Iterative vs. recursive resolution
- Eksempler: DNS, filsystemer

3. Time

Hvorfor synkronisere tid?

- Vi vil gerne kende til rækkefølgen af nogle events
  - Skete A før B? Er B afhængig af A?
  - Trivielt på ikke-distribuerede systemer
- Nødvendigt fx i GPS
- Make-files og compilers, input nyere/ældre end output?

Clock skew

- Lav tegning med perfect, slow og fast clocks
- Krystaller er ikke ens, nogle er hurtigere end andre
- Synkronisering skal ske oftere end $\delta / 2\rho$
- Justering skal ske gradvist, må ikke hoppe tilbage i tid

Network Time Protocol

- Synkroniser efter server med WWV reciever
- Estimer network delay - tegn figur, beregn $((t_2-t_1) + (t_3-t_4)) / 2$
- Opdel i strata

Berkeley algoritmen

- Uden WWV receiver - opgiv real-time, bliv enige om tid
- Koordinator får alles tider og beregner gennemsnit
- Fortæller derefter hvor meget de enkelte skal justeres

Logiske clocks

- Egentlig tid er ikke nødvendigt
- Vi behøver kun “happens before” relationen
  1. To events i samme proces, a → b
  2. Send → receive
- Lamport clocks
  - Tegn 3 processer som kommunikerer
- evt. Vector clocks

4. Synchronization

Rækkefølge af events

- Trivielt på ikke-distribuerede systemer
- Meget svært i distribuerede systemer
- Clock skew
- Med WWV server - Network Time Protocol
- Uden WWV server - Berkeley algorithm

Logiske clocks

- Egentlig tid er ikke nødvendigt
- Vi behøver kun "happens before" relationen
    1. To events i samme proces, a → b
    2. Send → receive
- Lamport clocks
    - Tegn 3 processer som kommunikerer
- evt. Vector clocks
- Totally-ordered multicast (aflevering)
    - Acknowledgements + timestamps garanterer total-ordning

Mutual exclusion

1. Centraliseret
    - Enkelt server styrer adgang til resourcen
    - Nemt, men single point of failure, bottleneck
2. Decentraliseret
    - Replikeret resource, voting om tilgang til resourcen, kræver m > n/2 votes
    - Mere stabil, men mere kompleks
3. Distribueret
    - Multicast om tilladelse, skal have OK fra alle, timestamp- og queue-baseret
    - Rigtig dårlig - multiple points of failure, mere kompleks, langsommere
4. Token-ring
    - Token passeres rundt mellem alle enheder, tilgang kræver besiddelse af token
    - Simpel og ingen single point of failure, men lost tokens, unødig trafik

Leader election

- Bully algoritmen - "jeg er større end dig", fx højeste proces-ID vinder
- Ring algoritmen - send ELECTION rundt i ring, og derefter COORDINATOR
- Small wireless networks (*ad hoc*) - lav et træ, find den bedste enhed
- Distribuér tokens (*store netværk*) - magnet-princip

5. Consistency

Data-centric consistency

- Continous consistency - tre akser
- **Numerical** deviation
  - fx aktie som varierer med mere end 2 cents eller 0.5%
  - Kan også være antal tentative/pending update operations!
- **Staleness** deviation
  - fx en vejr-rapport ældre end 4 timer
- **Ordering of operations**
  - **Sequential ordering** - samme som total ordering
  - **Causal ordering** - potentiel causal afhængighed

Eventual consistency (*problem løst med client centric consistency*)

- Lazy updating, alle replicas bliver konsistente "før eller senere"
- Fint med få updates og ingen write-write konflikter
- Fint hvis klienter aldrig forbinder til forskellige replicas
  - Ellers opleves inkonsistens
  - Løses ved at garantere client-centric consistency

Client-centric consistency models (*antag data item = 0, og writes inkrementerer*)

- Antag at data item x = 0, og writes inkrementerer
- Antag første operation på R_1, anden operation på R_2
- 1. Monotonic reads
  - På hinanden følgende reads returnerer samme eller nyere værdi
  - På R_1 er x = 5, på R_2 er x >= 5
- 2. Monotonic writes
  - På hinanden følgende writes opererer på samme eller nyere værdi
  - På R_1 giver W(x) x = 1, på R_2 giver W(x) x >= 2
- 3. Read your writes
  - En read efter en write returnerer den skrevne værdi eller nyere
  - På R_1 giver W(x) x = 1, på R_2 er R(x) >= 1
- 4. Writes follow reads
  - En write efter en read opererer på samme eller nyere værdi
  - På R_1 giver R(x) x = 5, på R_2 giver W(x) x >= 6

Implementation af client-centric consistency

- Readset og writeset
- Vector clocks

6. Fault tolerance

Failure models

1. **Crash** - Server crasher
2. **Omission** - En request/response besked bliver tabt
3. **Timing** - Langsom response, timeouts
4. **Response** - Forkert svar
5. **Arbitrary** (*Byzantine*) - vilkårlig fejl på vilkårligt tidspunkt

Løsninger - redundans

- Information - error correction codes
- Time - send igen efter noget tid
- Physical - flere replicas

Reliable Communication

Client-server communication

RPC eksempel, lav tegning

Mange tidspunkter og steder RPC kan crashe/fejle

1. Kan ikke finde server
   - Exception, Lamport's definition af distribuerede systemer ;)
2. Request besked tabt - omission model
   - Send igen, svært at identificere
3. Server crash efter request modtagelse - crash model
   - At least once, at most once, exactly once
   - Idempotens tillader at sende igen, indtil success (exactly once effekt)
4. Response besked tabt - omission model
   - Ligesom request besked tabt
5. Klient crasher efter request afsendelse - crash model
   - Serveren kan ikke sende svar tilbage, vent på klient

Process Resilience

Replikér processer og saml i process groups

Hvis én fejler kan en anden tage over

Flat group - alle er lige, undgår single point of failure, men kompleks, feedback implosion

Hierarchical group - simpel, ingen multicasting, men single point of failure (koordinator)

Recovery

Checkpointing (*individuel, kordineret*)

- Find en recovery line

Logging