Distributed Systems

- -ilities (non-functional requirements)
  - Reliability
    -perform required functions under stated conditions for a specified period
    -> Mean Time Between Failure
  - Fault-tolerant
  - Availability
    Proportion of time in functioning state
  - Recoverable
  - Consistent
  - Scalable (horizontally)
  - Predictable performance
  - Security
  - Heterogeneous
  - Open
- Sychronous DS:
  - Upper bound on transmission delay
  - Ordered transmissions
  - Synchronized clocks over all nodes
- Asynchronous DS:
  - Clocks can be out of sync
  - No upper bound on transmission delay
- Key-value stores
  - simpler semantics
    increased scalability
    Performance: High speed for single-record read and writes
    Availability and geo distribution
    Flexibility: Adapt to changing data definitions
  - No schema
  - raw byte access
  - no relations
  - horizontal partitioning (key ranges)
  - Single-row operations
  - Main operations
    - put(key, value)
    - get(key)
    - delete(key)
- DBMS/relational
  - Data types
  - Foreign Keys
  - Full SQL support (joins, transactions etc.)
- BigTable
  - Architecture:
    - Master: Metadata operations, Load balancing
    - Client library: Client operations
    - Tablet Servers: Data operations
  - Master:

+ Assigning tablets to TS
+ Detects addition and expiration of TS
+ Balance TS load

- Tablet Server:
  + Handles Read and Write Requests
  + Splits Tablets
  + Up to 1000 Tablets
- Chubby: -> understand? Master?
  + Highly-available, persistens, distributed lock, coordination service
  + Ensure one active BT-Master
  + Manage TS lifetime
  + Store schema information ??

-
- Clocks in DS
-
- /// Christian's Algorithm
  - Choose Server with smalles RTT
  - Assume delays split equally
  - Acc = +-(RTT/2 - minTransTime)
  - /// Clock stew:
    instantaneous difference between readings of two clocks
    Clock drift:
    rate at which skew changes
  - external synchronization (from time server)
- Berkley Algorithm
  - Acceptable Deviation
  - Set new time to Avg of all processes' times
  - Internal synchronization
  - send clock skew only
- NTP Network Time Protocol
  - + Provides UTC (Coordinated Universal Time) over network
    + Redundant servers and paths enable Fault Folerance and reconfiguration
    + Scalability by hierarchy (Stratum(s?))
    + Authentication -> security
  - /// Synchronization modes:
    1. Multicast - low Acc: periodically update
    2. RPC - medium: ad-hoc synchronization of one node (similar to Christian's Algorithm)
    3. Symmetric - high: Sync between stratum servers
- Physical clock <-> Logical clock (Lamport clock)
- Logical Clock Algorithm:
  + single timestamp dimension
- Vector Clock Algorithm:
  + d timestamp dimensions
- a -> b => L(a) < L(b)
  L(a) < L(b) =/> a -> b
  (happened before & ??)
- Atomic Total Order Broadcast:

+ Acknowledge incoming Messages to everybody (everybody knows if everybody received any message)
+ Deliver Messages iff it ranks highest in tuple <timestamp, id> (timestamp might be concurrent, so id defines global order)

- Casual Broadcast:
  + Only deliver if all messages of the same source were delivered (broadcast only so you get all messages with every possible value in vector clock dimension)
-
- Coordination & Agreement (Leader Election/ CS coordination)
-
- /// States:
  + Released
  + Wanted
  + Held
- /// Requirements for Mutual Exclusion:
  - Safety: At most one P in CS
  - Liveniness: every P eventually gets to the CS (no deadlocks)
  - Fairness: Order of requests is followed (no starvation)
  - Ordering: happend-before => Fainess?
- /// Chang and Roberts Algorithm for ring based ME
  - Each process knows its successor
  - Highest wins election
  - Don't forward pass j < i if participant
- Bully Algorithm
  - Types of messages:
    + Election (Wait for answer or victory)
    + Answer (Tell, you are higher)
    + Coordination / Victory (tell you won election after nobody answered to you)
  - While waiting after election message:
    if election message:
      cancel election if j is higher
      else ignore
  - Triggered by any process that detects crashed leader
- /// Dolev-Strong Consensus
  - synchronous system with f failures
  - f+1 rounds required
  - each process broadcasts all values it has not sent
  - Choose minimum value
- Leader Election Requirements
  - Safety: for every process their variable elected is either 'none' or Pj where Pj has the highest ID which did not crash until the end of the election
  - Liveness:
    Every process either crashes or elects something unequal to 'none'
- Paxos
  - guarantees Safety but maybe no progress (dueling proposers)
    + ordering using proposal number
    + future proposals take older accepted values

+ 2f + 1
    - Quorum defined by #acceptors (any two Quorums share at least one member)
    - Phase 1
    - Proposer.prepare(P# N):
      to quorum (all) acceptors
    - Acceptor.promise(N, N' opt, V' opt) -> Proposer:
      Only check if N is greater any previous PROMISED N' and reply with N and (if any ACCEPTED V' with N'') N'' and V'
      Optional: otherwise NACK
    - Phase 2
    - if quorum of acceptors promised:
      Proposer.acceptReq(N, max(V')) -> Acceptors:
      V'' is the maximum V' from the promises
      If none -> set own V (e.g. from client)
    - iff acceptor promised N' <= N
      Acceptor.accepted(N, V) -> Proposers, Learners:
      Optional otherwise NACK -> Proposer
    - Phase 3
    - if 1 single learner receives Quorum of accepted(N, V) -> learner is decided(N, V)
      Learner.clientRes(V)
      -> execute command / communicate leader / etc.
    - End
    - Basic Paxos (decides one value), Multi Paxos, Cheap Paxos..
      Trade-offs between:
      - Number of processes
      - Number of message delays before learning
      - Types of failures tolerated
      - Number of messages sent
-
- Replication
-
- Primary-backup Replication
    - eager -> replication before reply is sent
    -
- Active Replication
    - Client sends request to every replica (no primary)
    - requires TO broadcast
    - ok when ack from all replicas
- Gossiping
    - lazy
    - update pending if not happened before
-
-
- Consistency
    - RSM: all operations in same order on every replica (costly and not scalable)
    - Data-centric consistency model
      contract to specify what the results of read/write Ops are in presence of concurrency

- Strict consistency:
  Every read returns most recent write
  -> all writes must be visible to all processes and absolute global time order is maintained
- Linearizability:
  global but not absolute global time
- Sequential Consistency:
  Order is only valid within processes but all processes see same order
- -- from here Fallunterscheidung für jeden Process --
- Casually Consistent:
  Order can be different for different processes
  and only writes of other processes are important
- FIFO Consistency:
  It suffices for a process to share writes in FIFO order. Different processes may receive writes in with differing delays

- 2-Phase Commit
  - safe but not live
    + Coordinator
    + Transaction (Sequence of Operations)
    + Phase 1 "Can Commit": Voting Phase -> lock
      - if not all answer "yes" -> abort
    + Phase 2 "Do Commit": Commit Phase
- 3-Phase Commit
  - not safe but live
  - 1. Can Commit
  - 2. Pre Commit
    - if at least one participant gets preCommit -> after timeout: commit -> inform recovered processes about commit -> no doCommit
  - 3. After all Acks are at Coordinator -> Do Commit
- MapReduce
  - + Master node Failures are not handled for batch processing
    + Optimization for load-balancing: near end of phase spawn copies of tasks
    - not designed for incremental/streaming tasks
    - low level
    - low per node performance
  - Map(<k, v>) -> <k, v>*:
    + There is a call for ever <k, v> pair
    + k for this step not important
    + Optionally Combiner at the end (e.g. <k, v*> -> <k, sum(v*)>)
  - Sort / Group by k'
    + random Partition Function might be specified for use case
  - Reduce(k', <v'>*) -> <k",v">*
    + keys are sent to Reducer-Nodes by Hash-value
  - Execution Framework:
    + Scheduling
    + Data distribution (Map)
    + Synchronization (Reduce)
    + Errors and faults

- /// Spark
  - In-memory processing instead of IO hard drive accesses (as in MapReduce)
- /// CAP theorem
  - Consistency:
    All reads get the latest value (or error)
  - Availability:
    All requests should return successfully
  - Partition-tolerance:
    The system can tolerate arbitrary number of (communication) failures
  - theorem: impossible for replicated read-write store in asynchronous network to maintain the three guarantees in face of partitions
- Web Caching
  - Goals:
    1. Each node about same share of objects
    2. Client know what node to query for specific obj
    3. Failure tolerance and addition/deletion of nodes
  - Consistent Hashing: Add nodes on Hash Table to partition but do not change Hash Function
  - Consistens Hashing similar to CHORD -> distribute Websites on ring
- Reed Solomon Error Correction
  - Enc * X = Dec -> cut failure rows -> Enc^-1 * Dec = X
  - -> hier vermutlich begründe-Fragen -> Slides lesen
-
- -- Distributed Hash Tables --
-
- CHORD
  - search compares succ-side
- CAN (Content Addressable Network)
  - Search in O(d*n^(1/d))
  - vertical split = vertical DB
  - When inserting, pick random point
- Pastry
  - + Routing Tables RT & Leaf Sets LS
    + B = Address length
    + L = Leaf set size
    + populate cells with neighbors if possible
  - Search:
    1. Check if in LS-range and forward to closest ID (maybe node itself)
    2. get p = common prefix + 1
       if possible forward to node with longer common prefix
       else forward to node with prefix p
       else forward to any node with equal common prefix p-1 but which is numerically closer
  - When joining the network, copy Rows of Routing Table of nodes on way to closest node
  - Nodes ordered -> Node is responsible for all numbers between predecessor and self
- Bit Torrent

- seed: peer with entire file
  leech: peer downloading file -> becomes seed
- + Each peer tries to maximize its download rate
  + selectively deciding to whom to upload or choke
  + no central resource allocation
- 1. .torrent has address of tracker
  2. tracker keeps track of peers downloading file
  3. connect to available peers
  4. Beginning: first piece
  5. Middle: piece 2 - n-x
  6. End: piece n-x-1 - n
- + Random first policy -> after first complete -> Middle phase
  + Strict priority policy -> complete every piece before starting next
  + Rarest first: Scan all pieces at all neighbors and request piece owned by fewest peers
  + Engame mode: Send request for final blocks to all peers

? • Gnutella
- + TTL bis 0
  + Rounds
  + QUERY
  + QUERYHIT
  + connect to 3 peers

•
• Publish - Subscribe Systems which model have we learnt in ML
•
- + Pattern for Many-to-Many communication
  + Matching Problem:
  match publication e to matching subscriptions s
  + Broker, Publisher and Subscriber
- Advertisement-based routing model:
  + Publishers advertise their type of publication to every broker
  + Subscriptions are transferred to all Brokers on path
  Subscription Routing Table SRT: Advertisements
  Publication Routing Table PRT: Subscriptions
- Subscription-based routing model:
  + The same only the other way round
- Rendezvous-based routing model:
  + One Broker is rendezvous point
  + Each Broker knows how to reach the RP
  + Matching at RP with Bloom filters
  + RP knows topology of the network with link IDs
- Bloom filters:
  + False Positives
- Subscription covering:
  Merge overlapping subscription IF same 'next jump' from that node

? • Blockchain need deeper understanding
- Applications:
  + Smart Contracts

+ Cryptocurrency
- Distributed Ledger:
  + append-only log of transactions
  + fully replicated
- Byzantine Generals Problem
  + Generals are either Commander or Lieutenant
  + Lamports solution requires N >= 3f + 1
- PoW Proof of Work (Mining)
  + 1000x more energy expensive than for credit cards
  <-> Proof of Stake:
  + Mine with virtual currency (like random choice from all coins)
- "Blockchain Puzzle"
  -> nonce = solution to computational problem to solve?

2. • Merkle Tree usage ?
  - SPV Simple Payment Verification:
    if only pruned chain available -> SPV checks if specific transaction exists from
    1. Block headers
    2. related branches
  - "Spent transactions can be pruned"
- Limitations of BITCOIN:
  - slow block time (10min) & and confirmation time (>1 hour for 6 confirmations)
  - slow transaction rate (7tx per second)
- Feather-forking Attack on BITCOIN
  -
  -
  -
  - Know Google and Hadoop names of stuff:
    BigTable - HBase
    GFS - HFS
    Chubby - Zookeeper
    MapReduce -
- Questions:
  - Scalability of Berkeley-Algorithm? Sheet 1.2 d)
  - 2.3: j is undefined? v_i is undefined? -> just hard to read from solution
  - Review Slides: Dolev-Strong: in second example run:
    In Round 1: where does the b in node 2 come from?
    In Round 2: Node 3 should forward {a,b} as it didn't forward b before
    -> Round = all nodes broadcast a value once
  - Paxos: Failure of Proposer: How does the asynchronous DS know that a proposer has failed?
  - What is Paxos good for if it always agrees on the same value?
    -> N instances of Paxos for N possible values
  - 4.1 a) Why is Paxos used for RSM?
    -> makes sure that each command is agreed upon and order is preserved globally
    Do Paxos instances share SeqNumbers?
    "If n is the number of commands issued by the client" in other words does it mean 1. n is the number of possible command types or 2. the number of commands the client will give (e.g. for commands 'write(x,5)' 'read(x)' 'write(x,2)'

1. would be 2
2. would be 3)

- 5.1 iii)
  The primary updated its own state though so when it comes back, it has a different state than the other processes.
- 6.2 a) Why would a crashed process abort after it sent the canCommit? I can't see difference why other processes would commit in 3-PC but not in 2-PC
  b) In solution it says: "This is because P1 has crashed without committing or aborting (since we were only in phase 2)" So it is not possible to abort in Phase 2 of 3-PC?
  In 3-PC after Phase 2 timeout and commit at certain nodes -> do these nodes remove their lock?
- 7.2 b) Combiner:
  The Combiner does not compute any sum (otherwise it could do that in the word-count example as well)
  -> Combiner is locally after Map part
- 9.1 d) it is not mentioned that the Keys and values need to be copied to new node
- 9.2 a) Greedy algorithm does not make use of distance over ring limit edges
  -> simplicity of assignment
- BitTorrent download phase should be :
  5. Middle: piece 2 - n-x
  6. End: piece n-x-1 - n
  because 1 and n-x are in two phases otherwise
- What exactly does the client know about the Merkle Tree before verifying a transaction?
  -> only gets the needed parts of Merkle Tree ?
  Would transaction 95 also be verified even it is not contained?
  -> yes but normally way harder to find a matching doublicate for the hash
- Wenn ich ein CP System hab, dann soll es ja auch wenn sich Partitions bilden, konsistent bleiben. Wie sollen da aber zwei Knoten von unterschiedlichen Partitions von den Updates des jeweils anderen erfahren?
- Bei Dolev Strong, wenn alle außer ein Prozess failen, dann wird einfach der einzige Wert der da ist akzeptiert und das gilt dann als Konsens?

- ToDo:
  - Wichtig: 5.3
  - Slide Summarys durchlesen
  - Zu slides zu Cloud und Misc gab es keine Aufgaben
  - Geodreieck
  - MapReduce aus FDE Blatt und Klausur
  - Moodle Fragen durchlesen
  - Vor Klausur:
    ruhig
    Code anschauen (5.2 machen)
  - Harder Questions:
    1.3 c)
  - Nicht ganz kapiert:
    3.2
    4.2 (Paxos mit RSM)

    - Mail mit Fragen
-
- Facts:
    - to quiesce (no more operation so eventually system state will converge)

- Questions:
    - ~~5.3 4.~~
      ~~Why is it u<3,8,...> instead of u<2,8,...>? Client 2 asked for the update with ID 2 -> Update ID is incremented globally -> third update~~
- ~~synchronous DS?~~