Replication and Synchronization Algorithms for Distributed Databases

Lena Wiese

Research Group Knowledge Engineering
Institut für Informatik
Uni Göttingen

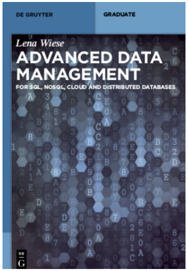
Sep 19th, 2015

Short CV Dr. Lena Wiese

- University of Göttingen (Research Group Leader Knowledge Engineering)
- University of Hildesheim (Visiting Professor for Databases)
- National Institute of Informatics, Tokyo, Japan
- Robert Bosch India Ltd., Bangalore, India
- Master/PhD: TU Dortmund
- Teaching and Research
    - NoSQL databases (lecture, seminars, projects)
    - Database security (encryption for Cassandra and HBase)
- Web: http://wiese.free.fr/

Book / Workshop

- Master's level text book (in English):
  Lena Wiese: Advanced Data Management for SQL, NoSQL, Cloud and Distributed Databases
  © 2015 DeGruyter/Oldenbourg
  (several pictures in this talk taken from the book)



- Workshop NoSQL-Net
  - organized jointly with Irena Holubova (Charles Unversity, Prague) and Valentina Janev (Instituto Mihailo Pupin, Belgrade)
  - 3rd edition to be organized at DEXA conference 2016
  - We welcome industrial application papers

Outline

1 Distributed Database Systems
2 Epidemic Protocols
3 Data Allocation and Replication
4 Multiversion Concurrency Control
5 Version Vectors
6 Quorums and Consensus
7 Implementations
8 Conclusion

Distributed Database Systems

- Scaling up (vertical scaling) vs Scaling out (horizontal scaling)
- Scaling out on commodity hardware in a shared-nothing architecture

Distributed Database Systems

A distributed database is a collection of data records that are physically distributed on several servers in a network while logically they belong together.

Distribution Transparency

- For a user it must be transparent how the DBMS internally handles data storage and query processing in a distributed manner.
- **Access transparency:** access independent of the structure of the network or the storage organization
- **Location transparency:** data distribution hidden from the user
- **Replication transparency:** user may access any replica
- **Fragmentation transparency:** fragmentation/sharding handled internally; user can query the database as if it contained the unfragmented data set
- **Migration transparency:** data reorganization does not affect data access
- **Concurrency transparency:** operations of multiple users must not interfere
- **Failure transparency:** continue processing user requests even in the presence of failures

Outline

1. Distributed Database Systems
2. Epidemic Protocols
3. Data Allocation and Replication
4. Multiversion Concurrency Control
5. Version Vectors
6. Quorums and Consensus
7. Implementations
8. Conclusion

Epidemic Protocols

- Propagation of information (like membership lists or data updates) in the network of database servers difficult:
  - message loss
  - high churn: change due to insertions or removals of servers
- Epidemic protocols spread new information like an infection or like gossip
  - **infected** nodes have received a new message that they want to pass on to other
  - **susceptible** nodes so far have not received the new message
  - **removed** nodes have received the message but are no longer willing to pass it on

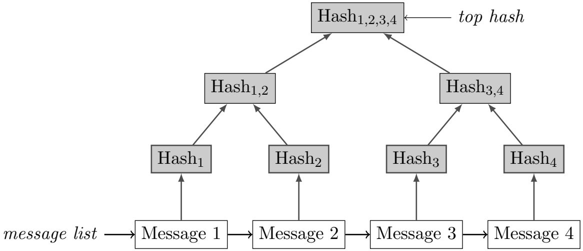
Variants of Epidemic Protocols

- Anti-entropy: periodic task (for example, run once every minute)
  - Anti-entropy is a simple epidemic: any server is either susceptible or infective (there are no removed servers)
  - the infection process does not degrade over time or due to some probabilistic decision
- Rumor spreading: the infection can be triggered by the arrival of a new message (in which case the server becomes infective); or it can be run periodically
  - Rumor spreading is a complex epidemic: the infection proceeds in several rounds
  - The amount of infections decreases with every round as the number of removed servers grows.
  - probabilistic: After each exchange with another server, the server stops being infective with a certain probability
  - counter-based: After a certain number k of exchanges, the server stops being infective

Alan J. Demers, Daniel H. Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard E. Sturgis, Daniel C. Swinehart, and Douglas B. Terry. Epidemic algorithms for replicated database maintenance. Operating Systems Review, 22(1):8-32, 1988.

Hash Trees

- Hash a list of messages level-wise
  - Quick detection of identical message lists
  - Detection of different subtrees



- BUT: all servers need the same sorting order in their message lists

Ralph C. Merkle. A digital signature based on a conventional encryption function. In International Cryptology Conference (CRYPTO), pages 369-378. Springer, 1987.

Outline

1 Distributed Database Systems

2 Epidemic Protocols

3 Data Allocation and Replication

4 Multiversion Concurrency Control

5 Version Vectors

6 Quorums and Consensus

7 Implementations

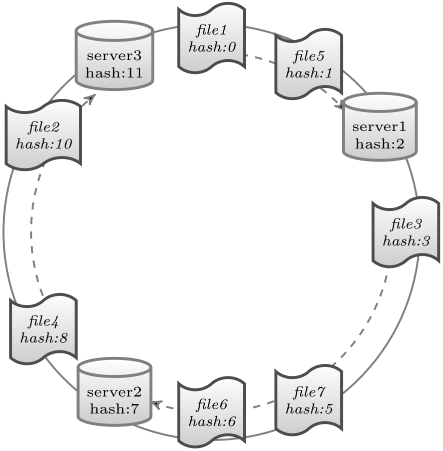
8 Conclusion

Consistent Hashing

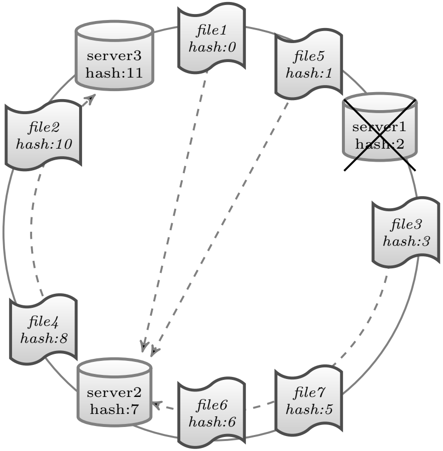
- Allocation of servers and data items on a hash ring
- Based on hash of server name and file name or key/ID
- Allocate file to next (clock-wise) server on hash ring
- Can handle churn

David R. Karger, Eric Lehman, Frank Thomson Leighton, Rina Panigrahy, Matthew S. Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In Frank Thom- son Leighton and Peter W. Shor, editors, Twenty-Ninth Annual ACM Symposium on the Theory of Computing, pages 654-663, 1997.

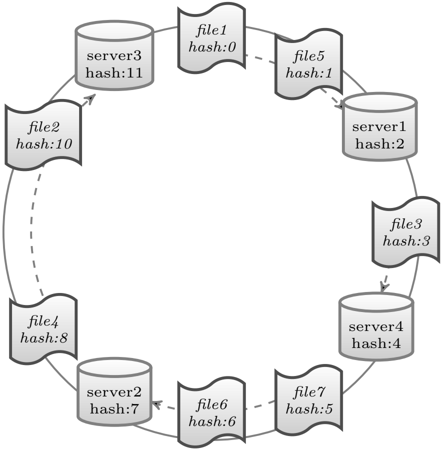
Consistent Hashing



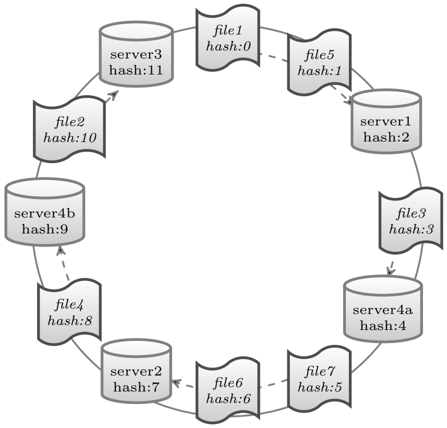
Consistent Hashing



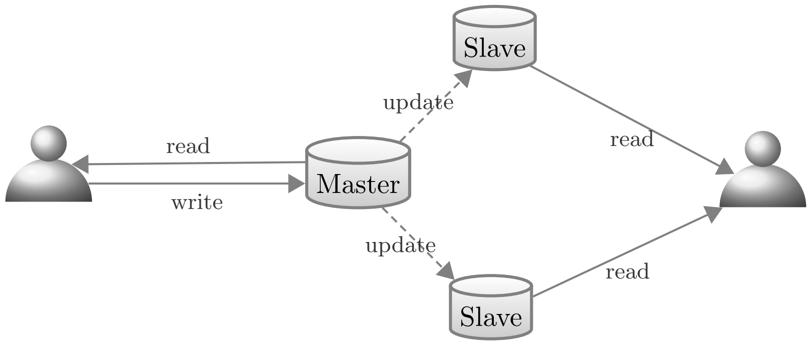
Consistent Hashing



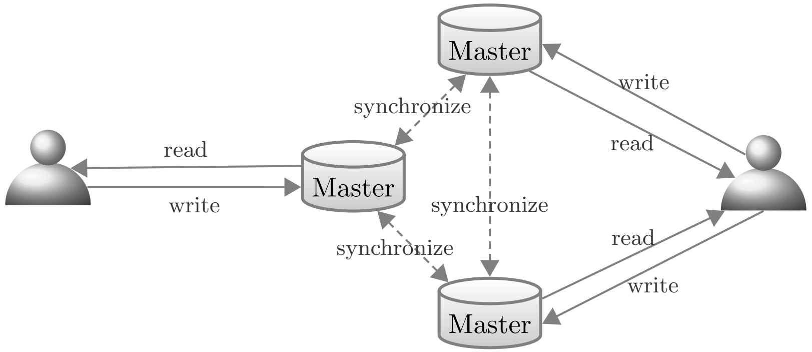
Consistent Hashing with Virtual Servers



Master-Slave Replication



Multi-Master Replication



Outline

1 Distributed Database Systems

2 Epidemic Protocols

3 Data Allocation and Replication

4 Multiversion Concurrency Control

5 Version Vectors

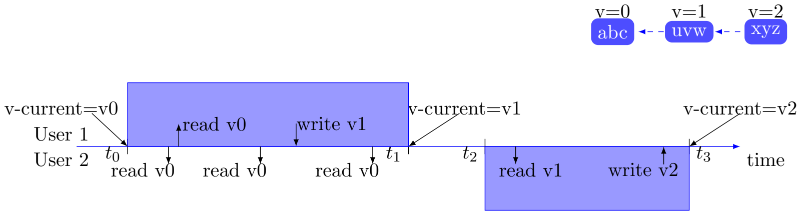
6 Quorums and Consensus

7 Implementations

8 Conclusion

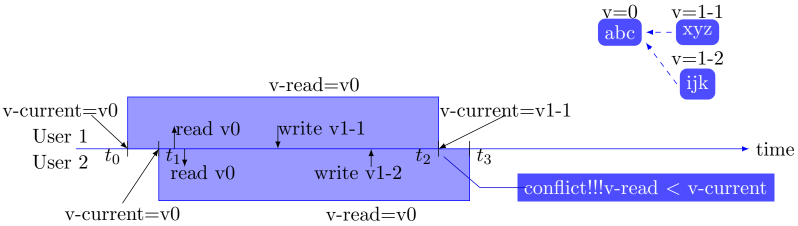
Multiversion Concurrency Control

- Non-blocking reads



Multiversion Concurrency Control

- Abort transaction on concurrent write



Outline

1 Distributed Database Systems

2 Epidemic Protocols

3 Data Allocation and Replication

4 Multiversion Concurrency Control

5 Version Vectors

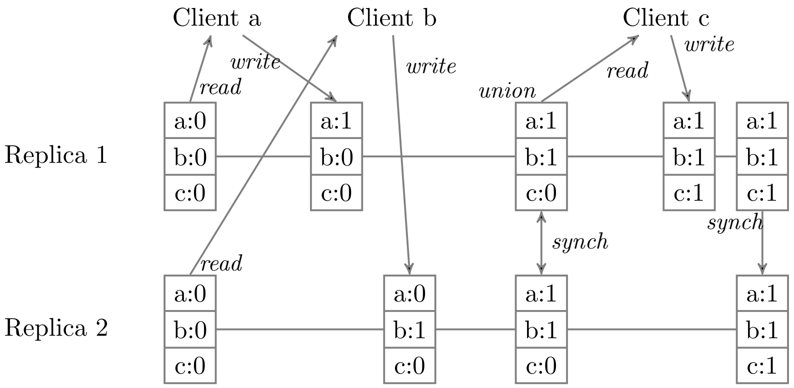
6 Quorums and Consensus

7 Implementations

8 Conclusion

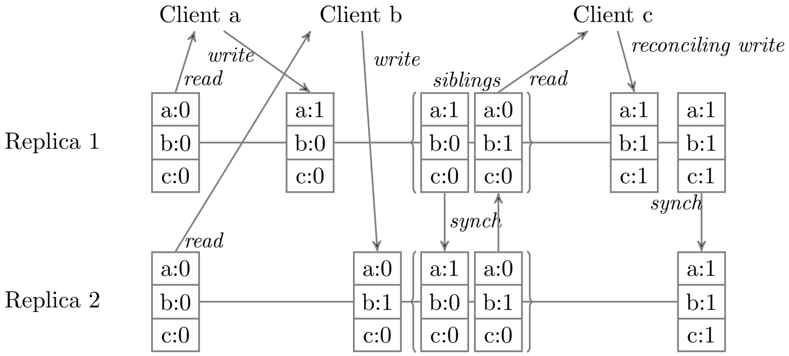
Version Vectors

- Logical counter for each client
- Option 1: Automatic synch with union semantics (e.g. shopping cart)



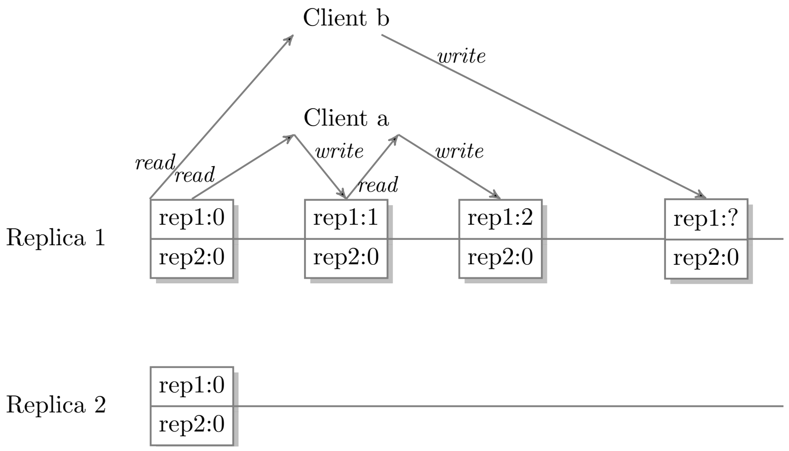
Version Vectors

- Logical counter for each client
- Option 2: Sibling versions and reconciliation by client context



Version Vectors

- Caution: Logical counter for each replica leads to lost updates!



Outline

1 Distributed Database Systems

2 Epidemic Protocols

3 Data Allocation and Replication

4 Multiversion Concurrency Control

5 Version Vectors

6 Quorums and Consensus

7 Implementations

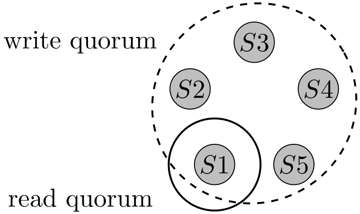
8 Conclusion

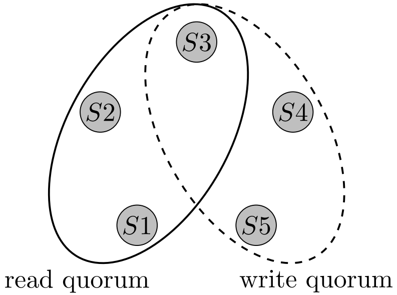
Write and Read quorums

- Use quorums to ensure consistency
- A read quorum is defined as a subset of replicas that have to be contacted when reading data; for a successful read, all replicas in a read quorum have to return the same answer value.
- A write quorum is defined as a subset of replicas that have to be contacted when writing data; for a successful write, all replicas in the write quorum have to acknowledge the write request.
- Quorum rules for N replicas, read quorum size R, write quorum size W
    - $R + W > N$ (consistent reads)
    - $2W > N$ (strongly consistent writes)
- Partial quorums only ensure weak consistency (for example, during failures)

Quorums

Example 1: ROWA read one write all (left)
Example 2: Majority quorums (right)





Hinted Handoff and Read Repair

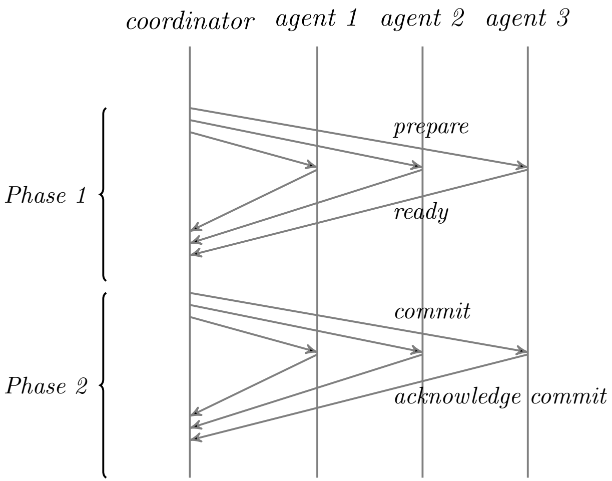
- Hinted handoff: handle temporary failures
  - if a replica is unavailable, write requests are delegated to another available server
  - hint that these requests should be relayed to the replica server as soon as possible
- Read repair: a coordinator node sends out the read request to a couple of replicas
  - coordinator contacts a set of replicas larger than the needed quorum
  - returns the majority response to the client
  - sends repair (update) instructions to those replicas that are not yet synchronized

2-Phase Commit

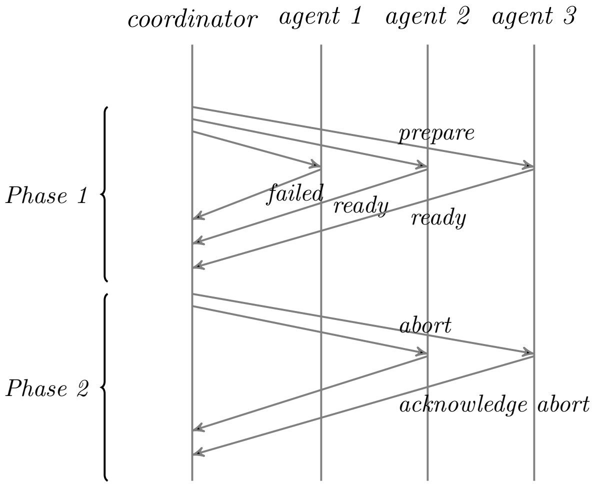
- Execution of a distributed transaction where all agents have to acknowledge a successful finalization of the transaction
- 2PC has a voting phase and a decision phase
- Failure of a single agent will lead to a global abort of the transaction
- The state between the two phases – before the coordinator sends his decision to all agents – is called the in-doubt state
- In case the coordinator irrecoverably fails before sending his decision, the agents cannot proceed to either commit or abort the transaction.

Jim Gray. Notes on Data Base Operating Systems. In: M. J. Flynn, J. N. Gray, A. K. Jones, K. Lagally H. Opderbeck, G. J. Popek, B. Randell J. H. Saltzer, H. R. Wehle: Operating Systems: An Advanced Course. Lecture Notes in Computer Science, volume 60, pp. 393-481. Springer, 1978.

2-Phase Commit



2-Phase Commit

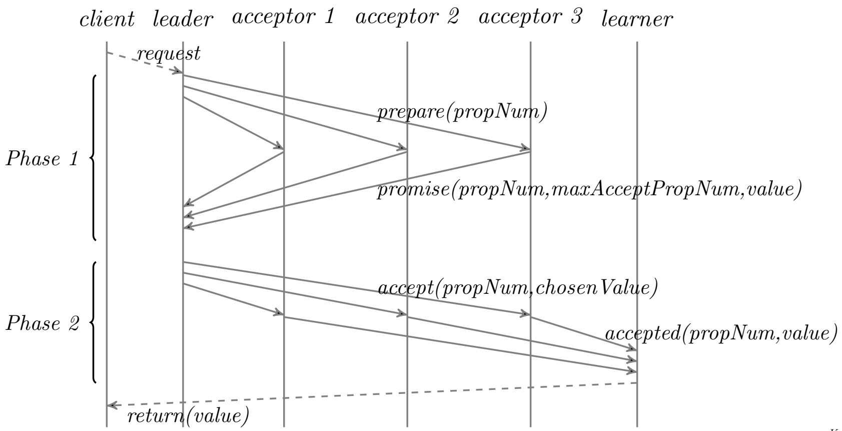


Paxos

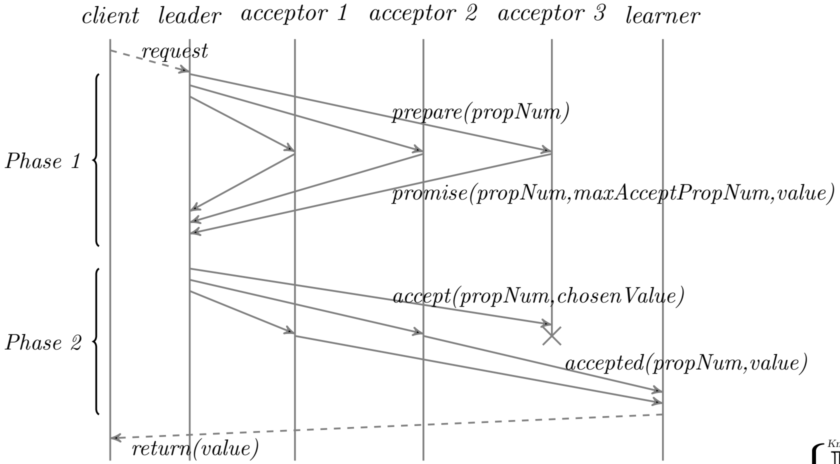
- Paxos algorithm can be applied to keep a distributed DBMS in a consistent state under certain failures
    - A client can issue a read request
    - Database servers have to come to a consensus on what the current state (and hence the most recent value) of the record is
- Database servers as agents
    - Proposer
    - Leader
    - Acceptor
    - Learner

Leslie Lamport. The part-time parliament. ACM Transactions on Computer Systems (TOCS), 16(2):133–169, 1998
Leslie Lamport. Generalized consensus and Paxos. Technical report, Technical Report MSR-TR-2005-33, Microsoft Research, 2005

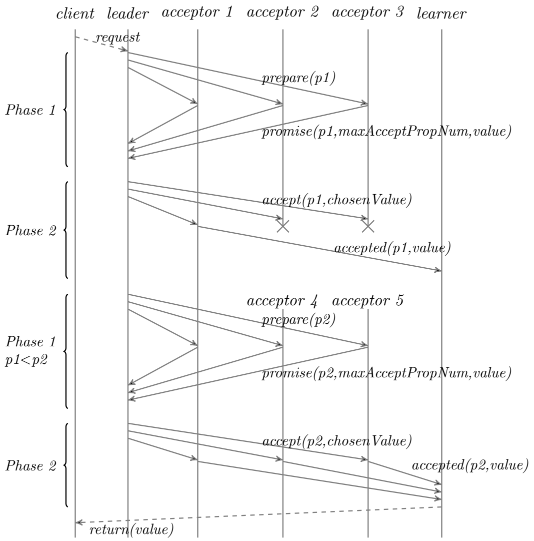
Paxos


client
leader
acceptor 1
acceptor 2
acceptor 3
learner
request
Phase 1
prepare(propNum)
promise(propNum,maxAcceptPropNum,value)
Phase 2
accept(propNum,chosenValue)
accepted(propNum,value)
return(value)

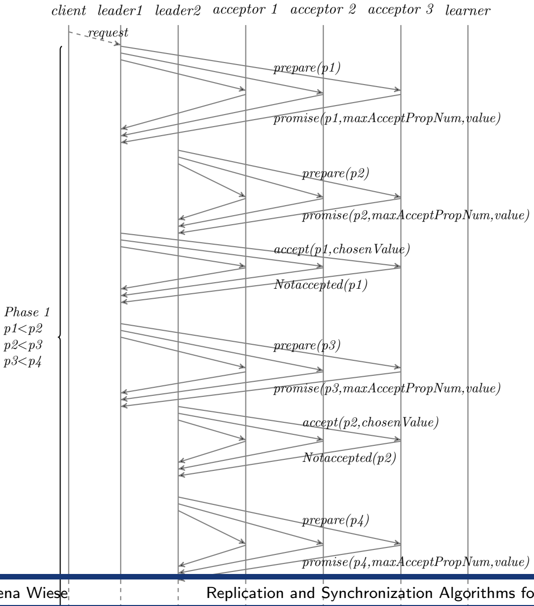
Paxos with minority of failing acceptors



Paxos with majority of failing acceptors



Paxos with dueling proposers

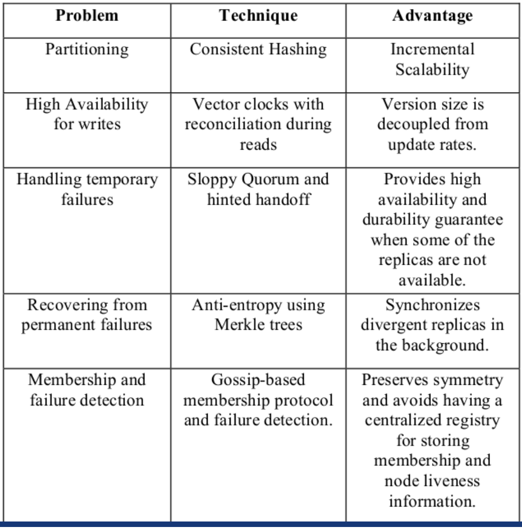


Outline

1. Distributed Database Systems
2. Epidemic Protocols
3. Data Allocation and Replication
4. Multiversion Concurrency Control
5. Version Vectors
6. Quorums and Consensus
7. Implementations
8. Conclusion

Amazon Dynamo

Table 1: Summary of techniques used in *Dynamo* and their advantages.



Problem	Technique	Advantage
Partitioning	Consistent Hashing	Incremental Scalability
High Availability for writes	Vector clocks with reconciliation during reads	Version size is decoupled from update rates.
Handling temporary failures	Sloppy Quorum and hinted handoff	Provides high availability and durability guarantee when some of the replicas are not available.
Recovering from permanent failures	Anti-entropy using Merkle trees	Synchronizes divergent replicas in the background.
Membership and failure detection	Gossip-based membership protocol and failure detection.	Preserves symmetry and avoids having a centralized registry for storing membership and node liveness information.

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. In Sympo- sium on Operating Systems Principles (SOSP), pages 205–220. ACM, 2007.

Riak

- Uses SHA1 for consistent hashing with virtual servers to distribute the load
- Uses “dotted” version vectors (no locks) with sibling semantics
- Exposes them to the users as a context for client-side conflict resolution
- Uses hinted handoff (temporary and permanent “ownership transfer”) and read repair
- Uses Anti-Antropy with hash trees as a background task to detect inconsistent data

http://basho.com/posts/technical/why-riak-just-works/

MongoDB

- Uses master-slave replication
- Replica set with one primary and several secondaries
- If primary fail a new one is selected from the replica set
- Heartbeat: by default 10 seconds
- Write concern is used to configure the write quorum (0,1,majority,...)

Cassandra

- "A true masterless architecture"
- Supports consistent hashing with RandomPartitioner
- Uses read repair and hinted handoff
- Write quorums can be configured ("tunable consistency": ONE, QUORUM, or ALL)

Conclusion

Modern distributed databases provide a revival / an implementation of established ideas on novel hardware.

Thank you!