Trade-Offs and NoSQL

Shan-Hung Wu, NetDB
CS, NTHU

Outline

- SAE revisited
- Non-relational partitioned DDBMS
  - Trade-off: load balancing vs. short latency
  - DDBMS moves: DBA or even graph cuts
  - NoSQL moves
- Non-relational replicated DDBMS
  - Trade-Off: C vs. short Latency
  - Trade-Off: CA@P
  - DDBMS moves: L-(A@P)
  - NoSQL moves
- Elasticity in non-relational DDBMS
- Remarks

Outline

- SAE revisited
- Non-relational partitioned DDBMS
  - Trade-off: load balancing vs. short latency
  - DDBMS moves: DBA or even graph cuts
  - NoSQL moves
- Non-relational replicated DDBMS
  - Trade-Off: C vs. short Latency
  - Trade-Off: CA@P
  - DDBMS moves: L-(A@P)
  - NoSQL moves
- Elasticity in non-relational DDBMS
- Remarks

Why So Hard to Get SAE?

- Horizontal "S" requires good data partitioning
  - I/O overhead < communication overhead
- 99.999% "A" requires cross-WAN replication
  - Network quality affects system performance
- "E" means workload-aware data (re-)partitioning + live migration

Outline

- SAE revisited
- Non-relational partitioned DDBMS
  - Trade-off: load balancing vs. short latency
  - DDBMS moves: DBA or even graph cuts
  - NoSQL moves
- Non-relational replicated DDBMS
  - Trade-Off: C vs. short Latency
  - Trade-Off: CA@P
  - DDBMS moves: L-(A@P)
  - NoSQL moves
- Elasticity in non-relational DDBMS
- Remarks

Load Balancing vs. Short Latency

- Data are partitioned to avoid hot zones
- When a tx access data across multiple partitions, it becomes slow
  - Distributed locking (may be)
  - 2PC
- How to avoid such slowdown?
- Partition data such that
  - Each machine gets equal load
  - #dist. txs are minimized

Outline

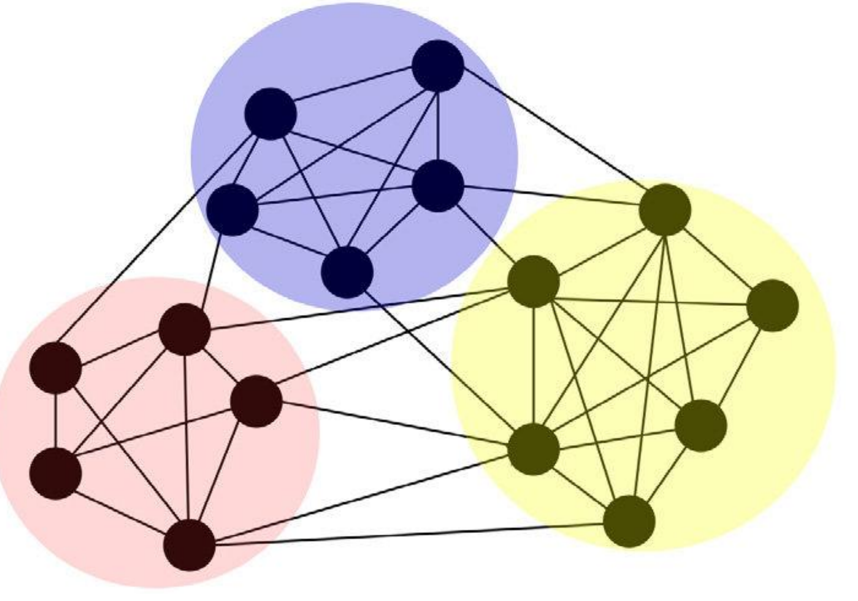
- SAE revisited
- Non-relational partitioned DDBMS
  - Trade-off: load balancing vs. short latency
  - DDBMS moves: DBA or even graph cuts
  - NoSQL moves
- Non-relational replicated DDBMS
  - Trade-Off: C vs. short Latency
  - Trade-Off: CA@P
  - DDBMS moves: L-(A@P)
  - NoSQL moves
- Elasticity in non-relational DDBMS
- Remarks

DDBMS Solution: Hire an Experienced DBA

- Traditionally, data partitioning requires careful examination of the target workload
  - Usually done by experienced DBA
- Cons:
  - Experienced DBAs are expensive
  - Time-consuming
  - Cannot adapt to the fast-changing workloads

Automation: Graph Partitioner

- Model the recent workload as a graph
  - Nodes: data objects, with weight denoting their access frequency
  - Edges: common access by txs, also weighted
- Find the minimal balanced partition of this graph
  - Machines are evenly loaded
  - Dist. txs are minimized



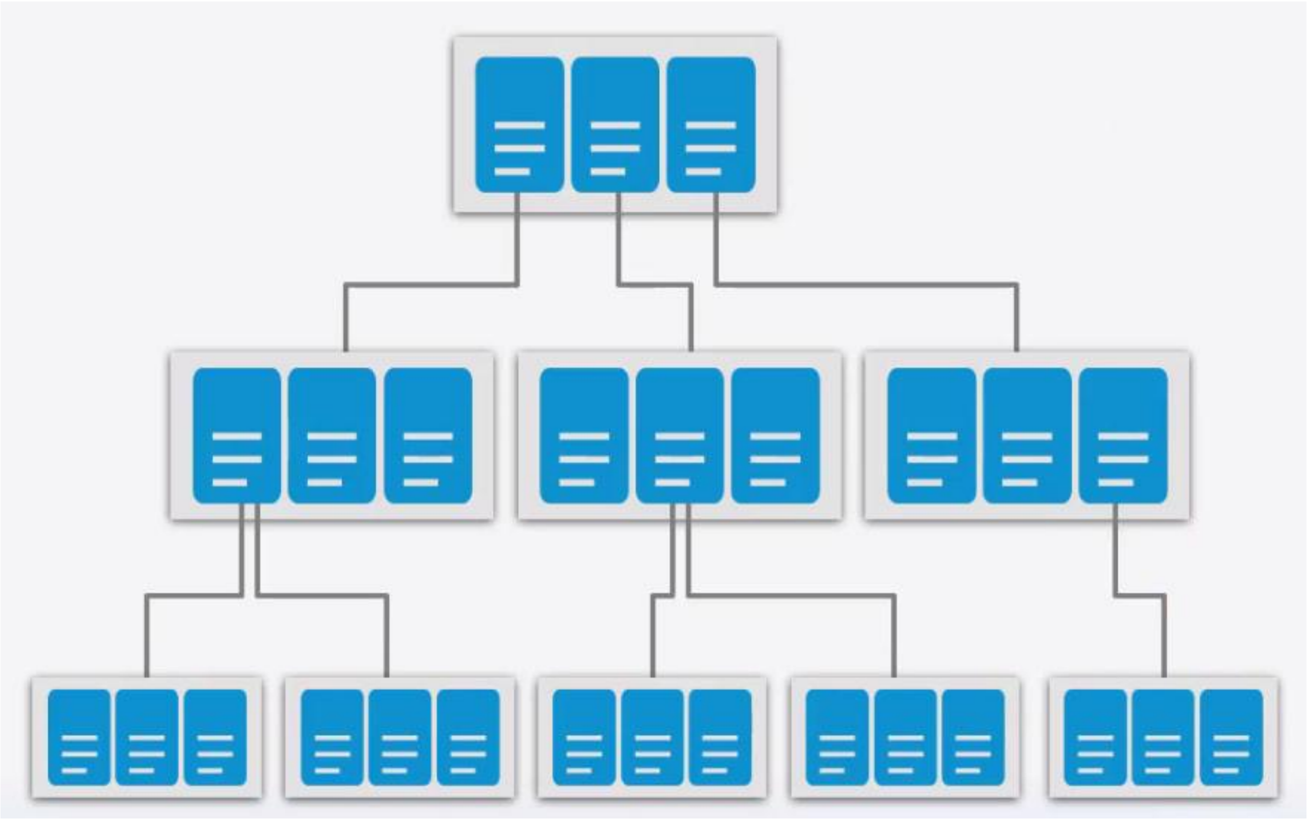
Outline

- SAE revisited
- Non-relational partitioned DDBMS
  - Trade-off: load balancing vs. short latency
  - DDBMS moves: DBA or even graph cuts
  - NoSQL moves
- Non-relational replicated DDBMS
  - Trade-Off: C vs. short Latency
  - Trade-Off: CA@P
  - DDBMS moves: L-(A@P)
  - NoSQL moves
- Elasticity in non-relational DDBMS
- Remarks

Industry: It’s Too Complicated!

- Balanced graph partitioning is an NP-hard problem
- Why not just drop the support relational model?
  - NoSQL data model: key-value, document based, entity-group based, etc.
- Data are perfectly partitionable □ no dist. Txs □ ***ultimate scalability***

Example: Document Model



- Used by Google Firestore, MongoDB, etc.

Outline

- SAE revisited
- Non-relational partitioned DDBMS
  - Trade-off: load balancing vs. short latency
  - DDBMS moves: DBA or even graph cuts
  - NoSQL moves
- Non-relational replicated DDBMS
  - Trade-Off: C vs. short Latency
  - Trade-Off: CA@P
  - DDBMS moves: L-(A@P)
  - NoSQL moves
- Elasticity in non-relational DDBMS
- Remarks

Outline

- SAE revisited
- Non-relational partitioned DDBMS
  - Trade-off: load balancing vs. short latency
  - DDBMS moves: DBA or even graph cuts
  - NoSQL moves
- Non-relational replicated DDBMS
  - Trade-Off: C vs. short Latency
  - Trade-Off: CA@P
  - DDBMS moves: L-(A@P)
  - NoSQL moves
- Elasticity in non-relational DDBMS
- Remarks

C vs. Short Latency

	Eager MM	Lazy M/S	Lazy MM
Consistency	Strong	***Eventual***	***Weak***
Latency	***High***	Low	Low
Throughput	***Low***	High	High
Availability upon failure	Read/write	Read-only	Read/write
Data loss upon failure	None	Some	Some
Reconciliation	No need	No need	User or rules
Bottleneck, SPF	None	Master	None

- Short latency wins in traditional DDBMS

Outline

- SAE revisited
- Non-relational partitioned DDBMS
  - Trade-off: load balancing vs. short latency
  - DDBMS moves: DBA or even graph cuts
  - NoSQL moves
- Non-relational replicated DDBMS
  - Trade-Off: C vs. short Latency
  - Trade-Off: CA@P
  - DDBMS moves: L-(A@P)
  - NoSQL moves
- Elasticity in non-relational DDBMS
- Remarks

CAP Theorem

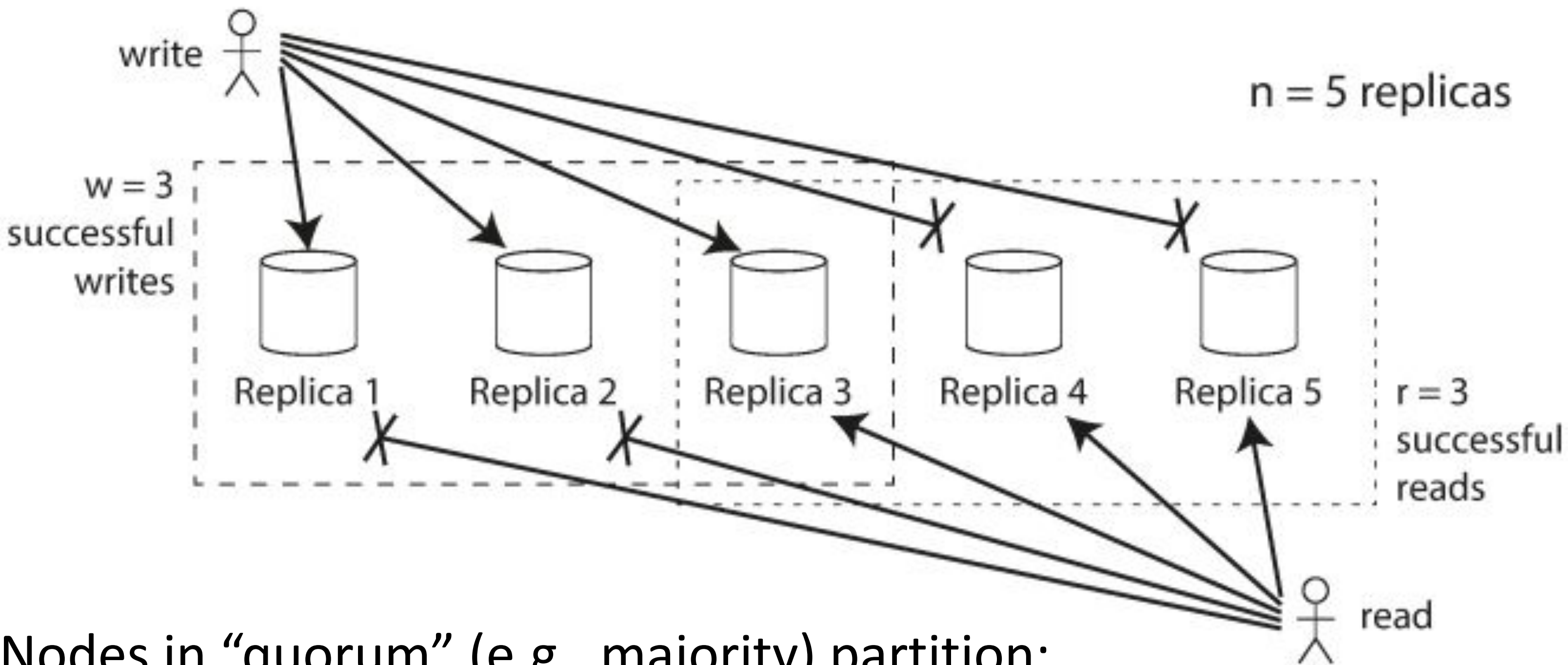
- All-node availability
  - Every node (if not failed) always serve requests
- Consistency
  - Data in every node are consistent

- CAP theorem: in presence of partition (P), choose one

Partition Explained

- Machine failure
- ***Network partition***
  - May be ***temporal***, but ***frequent*** (due to, e.g., packet delay/loss)
  - Including single-node partition
  - Cross-WAN
- What’s the difference?
  - Nodes are alive!
  - Some may commit users something that others don’t agree
- “P” covers machine failure

All-node Av ≠ Service Av



- Nodes in "quorum" (e.g., majority) partition:
  - Keeps the service availability
- [Paxos protocol]() allows C ***and*** service Av @P
- Nodes not in quorum partition:
  - Need not make any progress (i.e., act as if they were dead)
  - When partition resolved, need to enter the "recovery" mode first

Outline

- SAE revisited
- Non-relational partitioned DDBMS
  - Trade-off: load balancing vs. short latency
  - DDBMS moves: DBA or even graph cuts
  - NoSQL moves
- Non-relational replicated DDBMS
  - Trade-Off: C vs. short Latency
  - Trade-Off: CA@P
  - DDBMS moves: L-(A@P)
  - NoSQL moves
- Elasticity in non-relational DDBMS
- Remarks

DDBMS with Network Partition

- Lazy M/S replication
  - One master for each data object (globally)
- If a client can reach the masters:
  - No consistency (short latency wins)
- If a client is partitioned from the masters:
  - No availability
  - Data in a master is always correct and durable
- We say DDBMS choose C in present of P
- Overall: L over C, with sacrifised A@P

Outline

- SAE revisited
- Non-relational partitioned DDBMS
  - Trade-off: load balancing vs. short latency
  - DDBMS moves: DBA or even graph cuts
  - NoSQL moves
- Non-relational replicated DDBMS
  - Trade-Off: C vs. short Latency
  - Trade-Off: CA@P
  - DDBMS moves: L-(A@P)
  - NoSQL moves
- Elasticity in non-relational DDBMS
- Remarks

Dynamo/Cassandra/Riak: L-(C@P)

- In normal case: lazy M/S replication
  - L over C
- In the presence of failure: lazy M/M replication
  - ***Degraded consistency level***
  - But R/W availability

Google BigTable/MegaStore: C-(A@P)

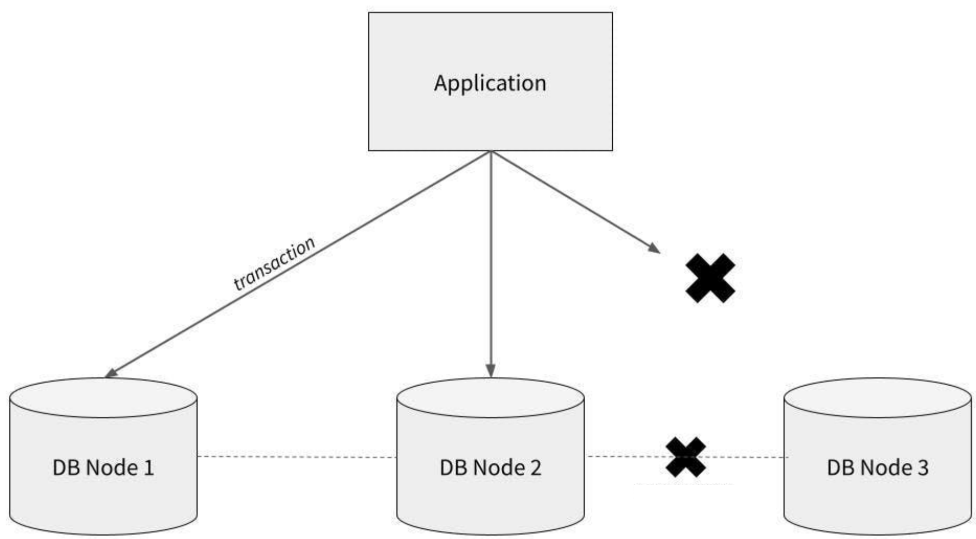
- In normal case: eager replication
  - ***C***
  - Long latency but scalable, as each partition (entity group) runs an instance of eager replication protocol
- @P: adopt eager replication protocol that is robust to P
  - ***Paxos***
  - If a client can reach any machine in the quorum, the system acts normally

Outline

- SAE revisited
- Non-relational partitioned DDBMS
  - Trade-off: load balancing vs. short latency
  - DDBMS moves: DBA or even graph cuts
  - NoSQL moves
- Non-relational replicated DDBMS
  - Trade-Off: C vs. short Latency
  - Trade-Off: CA@P
  - DDBMS moves: L-(A@P)
  - NoSQL moves
- Elasticity in non-relational DDBMS
- Remarks

Elasticity

- Targets: workload-aware (re-)partitioning + live migration
- Challenges: distributed txs, 2PC/log shipping, etc.
  - NoSQL: limited support for distributed txs



When to Migrate?

- NoSQL makes it easy: achieved using a “master” in each datacenter
  - Monitors the loads of “working” servers
  - Split/merge loads if they are too hot/cold
- Example?
  - GFS, MapReduce, BigTable
- Elasticity is usually per-datacenter basis
  - Enough to cope with “hot datacenters” (due to concentrated active users)
  - No data partitioning across datacenters (i.e., fully replicated)

Problems

- Master is a single point of failure
  - Solution: hot-stand-by's
- Master may be the performance bottleneck
  - Solution: preventing master from dealing with data traffic (but control messages)
- Hard to develop apps:
  - Non-relational data model
  - No/reduced dist. tx support
- Data migration vs. A and C

Data Migration vs. Tx execution

- If data being migrated are locked, conflict txs stall
  - No A
- Otherwise, no C
- Current approach: “live migration” (A -> B)
  - A continues executing txs, and additionally pushes the results sets to B
  - B takes over tx execution only it catches up A (after receiving data and applying result sets)
- Keeps A and C
- But B cannot help system performance timely
  - Many txs are still executed by A after the migration decision
- Still a research problem!

Outline

- SAE revisited
- Non-relational partitioned DDBMS
  - Trade-off: load balancing vs. short latency
  - DDBMS moves: DBA or even graph cuts
  - NoSQL moves
- Non-relational replicated DDBMS
  - Trade-Off: C vs. short Latency
  - Trade-Off: CA@P
  - DDBMS moves: L-(A@P)
  - NoSQL modes
- Elasticity in non-relational DDBMS
- Remarks

Why So Hard to Get SAE?

- Horizontal S requires good partitioning (and replication)
  - Experienced DBA is needed
  - Dist. Txs are too costly (due to 2PC)
- Extreme A requires cross-WAN replication
  - Trade-off in normal operations: consistency vs. latency
  - Trade-off when failure occurs: consistency vs. availability (CAP theorem)
- E means workload-aware (re-)partitioning + live migration
  - Trade-off in deciding partitions: equal volume vs. few dist. txs
  - Join optimization needed for multi-tenant DBMS
  - Migration blocks ongoing txs

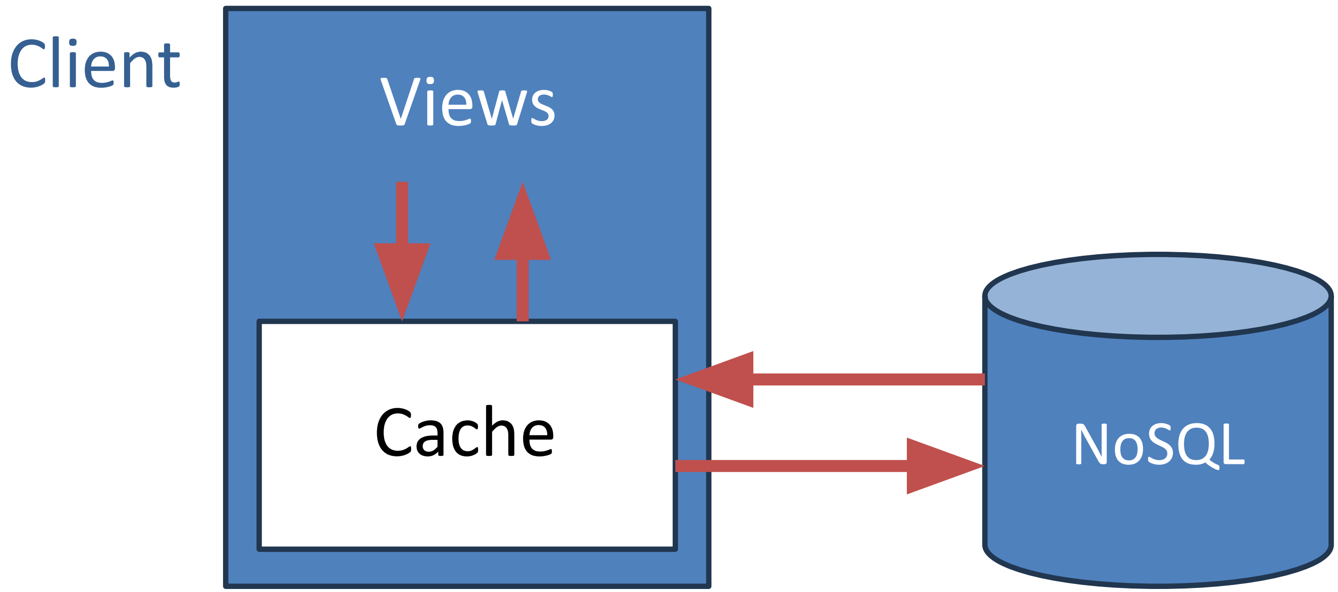
The NoSQL Move

- Drop relational model
  - Assume data can be partitioned
  - No dist. tx S and easier E (although migration still blocks txs)
- Further compromise C (except Google) in both normal and failure cases
  - For low latency in normal case
  - For A in failure case

The Score Sheet

	DDBMS	Dynamo/Cassandra/Riak
Data model	Relational	Key-value
Tx boundary	Unlimited	Key-value
Consistency	W strong, R eventual	W strong, R eventual
Latency	Low (local partition only)	Low
Throughput	High (scale-up)	High (scale-out)
Bottleneck/SPF	Masters	Masters
Consistency (F)	W strong, R eventual	WR eventual
Availability (F)	Read-only	Read-write
Data loss (F)	Some	Some
Reconciliation	Not needed	Needed
Elasticity	No	Manual, blocking migration

Other Benefits of NoSQL



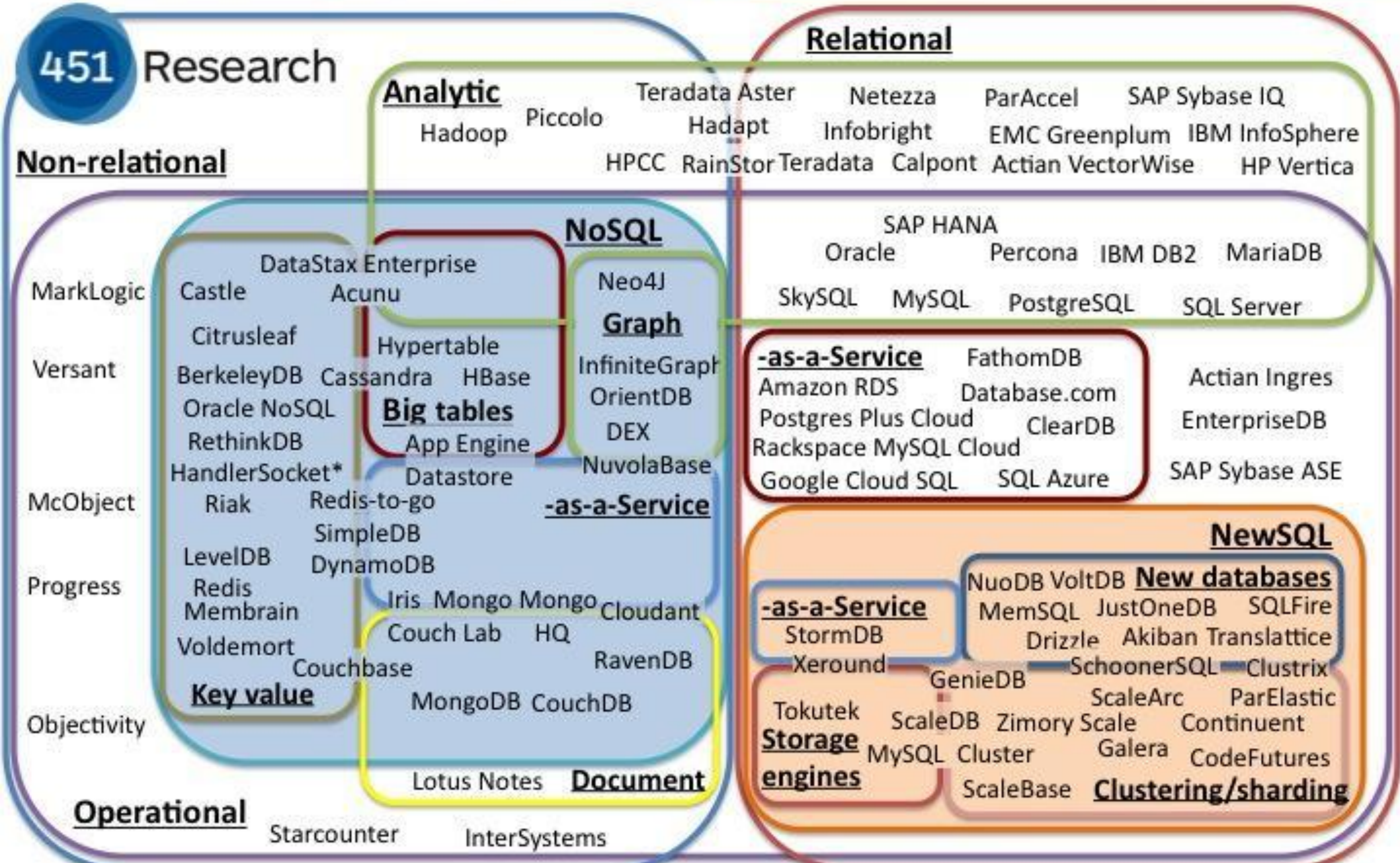
- ***Offline*** support (e.g., Firestore)
  - Writes to local cache first, then on server when client goes back online
  - Conflict resolution (on same doc): the last write wins
- Change ***pushing***
  - Server pushes changes

Denormalization

- How to model data in NoSQL?

- Based on ER and relation model first
- Outline your queries
- Then denormalize fields for every query

The evolving database landscape



References

- Handling Large Datasets at Google: Current Systems and Future Directions, Jeff Dean.
- MapReduce: Simplified Data Processing on Large C lusters, Jeffrey Dean and S Ghemawat, 2004.
- The Google File System, S Ghemawat et al. 2003.
- A Thomson et al. The Case for Determinism in Database Systems. 2010.
- A Thomson et al. Calvin: Fast Distributed Transactions for Partitioned Database Systems. 2012.

References

- Fay Chang et al. Bigtable: A Distributed Storage System for Structured Data. 2006.
- Shute et al, F1 — the Fault-Tolerant Distributed RDBMS Supporting Google's Ad Business. 2012
- Corbett et al, Spanner: Google's Globally-Distributed Database. 2012.
- Matthew Aslett. What we talk about when we talk about NewSQL. http://blogs.the451group.com/information_management/2011/04/06/what-we-talk-about-when-we-talk-about-newsql/
- Matthew Aslett. NoSQL, NewSQL and Beyond. http://blogs.the451group.com/information_management/2011/04/15/nosql-newsql-and-beyond/