The NoSQL Ecosystem, Relaxed Consistency, and Snoop Dogg

Adam Marcus
MIT CSAIL
marcua@csail.mit.edu / @marcua
About Me

• Social Computing + Database Systems
• Easily Distracted: Wrote *The NoSQL Ecosystem* in *The Architecture of Open Source Applications*¹

Preliminaries

• Almost always, use PostgreSQL or MySQL
• Most problems can be solved by a single (large) machine
• Consider paying a DB vendor to solve problem
But sometimes, need >1 machines

- Analytics on MapReduce or Column Stores
- Facebook famously stores 1B+ users on ~10K machines
Scaling transactional workloads

- Google spends thousands of person-hours on Megastore/Spanner
- Everyone else jumps through hoops to stick to “relational model” and “SQL”
  - Partition data to avoid multinode transactions (2PC)
  - Avoid multirow transactions to prevent locking
History, Compressed
History, Compressed

Late 1990s

Buy RAM

Today
History, Compressed

Buy RAM  Wrap RDBMSs  NoSQL

Late 1990s  Today
History, Compressed

Buy RAM  Wrap RDBMSs  NoSQL

Late 1990s  Today

Not Only SQL
History, Compressed

Buy RAM  Wrap RDBMSs  NoSQL  Commercialize

Late 1990s  Today
History, Compressed
The List, So You Don't Yell at Me

- HBase
- Cassandra
- Voldemort
- Riak
- HyperTable
- Neo4j
- InfoGrid
- MongoDB
- CouchDB
- AllegroGraph
- HyperGraphDB
- Voldemort
- BerkeleyDB
- Redis
- Sones
- Parrot
- FlockDB
- DEX
- MemcacheDB
- Tokyo Cabinet
- VertexDB
- Oracle NoSQL
The List, So You Don't Yell at Me

Marcus' Law of Databases:

The number of persistence options doubles every 1.5 years
history
the parents of nosql
tradeoffs + demos
scaling with snoop dogg
real-world usage
discussion
BigTable

- Google, 2006
- Column store with sloppy schemas
- Strong consistency
- Open source: HBase
Dynamo

- Amazon, 2007
- Key-value store
- Allows eventual consistency
- Open source: Voldemort, Riak
history
the parents of nosql
tradeoffs + demos
scaling with snoop dogg
real-world usage
discussion
understanding NoSQL
= understanding a long series of tradeoffs
Standards-compliant SQL Systems

- Relational model
- Powerful query language
- Transactional semantics
- Predefined schemas
- Strong consistency between replicas*
Standards-compliant SQL Systems

- Relational model
- Powerful query language
- Transactional semantics
- Predefined schemas
- Strong consistency between replicas*

NoSQL: Maybe you don't need all of these?
NoSQL Systems are a Buffet
NoSQL Systems are a Buffet
(of progressively larger grenades)
NoSQL Systems are a Buffet (of progressively larger grenades)

- Data model
- Query model
- Durability
- Transactional consistency
- Partitioning
- Replica consistency
Prof. Madden will not quiz you on specific features of NoSQL systems (i.e., don't memorize that MongoDB is a document store)
Data Model

• Usually key-based...
  • Binary blob: Voldemort
  • Documents: MongoDB, CouchDB, Riak
  • Data structures: Redis
  • Column-families: HBase, Cassandra
Data Model

- Usually key-based...
  - Binary blob: Voldemort
  - Documents: MongoDB, CouchDB, Riak
  - Data structures: Redis
  - Column-families: HBase, Cassandra
- ...but not always
  - Graph stores
Query Model

- Redis: data structure-specific operations
- CouchDB, Riak: MapReduce
- Cassandra, MongoDB: SQL-like languages, no joins or transactions
Query Model

- Redis: data structure-specific operations
- CouchDB, Riak: MapReduce
- Cassandra, MongoDB: SQL-like languages, no joins or transactions
- Third-party
  - High-level: PigLatin, HiveQL
  - Library: Cascading, Crunch
  - Streaming: Flume, Kafka, S4, Scribe
Transactions

- Full ACID for single key
- Redis: multi-key single-node transactions
demo!
history
the parents of nosql
tradeoffs + demos
scaling with snoop dogg
real-world usage
discussion
Single-server durability

- Memory only: memcached
- Single-server durability: the rest
  - fsync every N seconds: most
  - Write-ahead logging: Cassandra, HBase, Redis, Riak
  - Group commit: Cassandra, HBase/HDFS
When one server is not enough

- Replicate

- Partition
When one server is not enough

- Replicate
  - Performance
  - K-safety

- Partition
When one server is not enough

- Replicate
  - Performance
  - K-safety

- Partition
  - Vertical
  - Horizontal
When one server is not enough

- Replicate
  - Performance
  - K-safety

\[
\begin{array}{c}
\text{Consistency/Availabilty} \\
\end{array}
\]

- Partition
  - Vertical
  - Horizontal
When one server is not enough

- Replicate
  - Performance
  - K-safety

- Partition
  - Vertical
  - Horizontal

\{ \text{Consistency/Availability} \}
\{ \text{Partitioning Scheme} \}
Eventual vs. Strong Consistency: FIGHT!
Eventual vs. Strong Consistency: FIGHT!

\[ N = \# \text{ replicas} \]
\[ W = \# \text{ write acknowledgements} \]
\[ R = \# \text{ read acknowledgements} \]
\[ N = \# \text{ replicas} \]
\[ W = \# \text{ write acknowledgements} \]
\[ R = \# \text{ read acknowledgements} \]

N = 3

Sally: 100K

N = 3
$N = \# \text{ replicas}$

$W = \# \text{ write acknowledgements}$

$R = \# \text{ read acknowledgements}$

Give Sally a raise!

Sally: 100K

Sally: 100K

Sally: 100K

$N = 3$
N = # replicas
W = # write acknowledgements
R = # read acknowledgements

Give Sally a raise!

120K

Sally: 100K
Sally: 100K
Sally: 100K
N = # replicas
W = # write acknowledgements
R = # read acknowledgements

N = 3, W = 1, R = 1

Give Sally a raise!

Sally: 100K
Sally: 100K
Sally: 100K

120K
$N = \# \text{ replicas}$
$W = \# \text{ write acknowledgements}$
$R = \# \text{ read acknowledgements}$

$N = 3, W = 1, R = 1$

Sally: 120K
Sally: 120K
Sally: 100K
N = # replicas
W = # write acknowledgements
R = # read acknowledgements

Sally: 120K
Sally: 120K
Sally: 100K

N = 3, W = 1, R = 1
N = # replicas
W = # write acknowledgements
R = # read acknowledgements

N = 3, W = 1, R = 1

Sally's Salary?
N = # replicas
W = # write acknowledgements
R = # read acknowledgements

N = 3, W = 1, R = 1

Sally's Salary?

100K

120K 120K 100K

Sally: Sally: Sally:
N = # replicas
W = # write acknowledgements
R = # read acknowledgements

Sally:
100K

120K

Sally: 100K
Sally: 100K

Give Sally a raise!

N = 3
N = # replicas
W = # write acknowledgements
R = # read acknowledgements

N = 3,
W = 3,
R = 1

Give Sally a raise!

Sally: 100K
Sally: 100K
Sally: 100K

N = 3, W = 3, R = 1
N = # replicas
W = # write acknowledgements
R = # read acknowledgements

N = 3, W = 3, R = 1

Waste of time, Waste of time, Waste of time
N = # replicas
W = # write acknowledgements
R = # read acknowledgements

N = 3, W = 3, R = 1
N = # replicas
W = # write acknowledgements
R = # read acknowledgements

N = 3, W = 3, R = 1
N = # replicas
W = # write acknowledgements
R = # read acknowledgements

N = 3, W = 3, R = 1

Sally's Salary?

Sally: 120K

N = 3, W = 3, R = 1
\[ N = \text{# replicas} \]
\[ W = \text{# write acknowledgements} \]
\[ R = \text{# read acknowledgements} \]

N = 3, W = 3, R = 1

Sally's Salary?

Sally: 120K

N = 3, W = 3, R = 1
N = # replicas
W = # write acknowledgements
R = # read acknowledgements

Give Sally a raise!

Sally: 100K
Sally: 100K
Sally: 100K

120K

N = 3
N = # replicas
W = # write acknowledgements
R = # read acknowledgements

N = 3, W = 2, R = 2

Give Sally a raise!

Sally: 100K

Sally: 100K

Sally: 100K
N = \# replicas
W = \# write acknowledgements
R = \# read acknowledgements

N = 3, W = 2, R = 2
\( N = \# \text{ replicas} \)
\( W = \# \text{ write acknowledgements} \)
\( R = \# \text{ read acknowledgements} \)

\( N = 3, \; W = 2, \; R = 2 \)
$N =$ number of replicas
$W =$ number of write acknowledgements
$R =$ number of read acknowledgements

$N = 3, W = 2, R = 2$

Sally's Salary?
\[ N = \# \text{ replicas} \]
\[ W = \# \text{ write acknowledgements} \]
\[ R = \# \text{ read acknowledgements} \]

\[ N = 3, \ W = 2, \ R = 2 \]

Sally:
120K
Sally:
120K

100K

Sally: 120K
Sally: 120K
Sally: 100K

QUORUM!
N = # replicas
W = # write acknowledgements
R = # read acknowledgements

N = 3, W = 2, R = 2
Consistency wild west

Available, inconsistent on failure

\[ N = 3, W = 1, R = 1 \]
Consistency wild west

Available, inconsistent on failure

\[ N = 3, W = 1, R = 1 \]

Consistent, unavailable writes on failure

\[ N = 3, W = 3, R = 1 \]
Consistency wild west

Available, inconsistent on failure

\[ N = 3, W = 1, R = 1 \]

Consistent, unavailable writes on failure

\[ N = 3, W = 3, R = 1 \]

Available, consistent with 1 failure

\[ N = 3, W = 2, R = 2 \]
Consistency wild west

Available, inconsistent on failure
N = 3, W = 1, R = 1

Consistent, unavailable writes on failure
N = 3, W = 3, R = 1

Available, consistent with failure
N = 3, W = 2, R = 2
Consistency
Consistency → (not acId)
Consistency  \rightarrow  \text{(not aCid)}
Consistency

- Strong: Appears that all replicas see all writes
- Eventual: If left alone, replicas eventually converge
- Weak: Replicas have different, divergent versions
Consistency

- Strong: Appears that all replicas see all writes
- Eventual: If left alone, replicas eventually converge
- Weak: Replicas have different, divergent versions

The magical CAP theorem line
Consistency math
Consistency math

Strong: $N < R + W$
Consistency math

Strong: $N < R + W$
Easy: $W = N$
Tricky: $W < N$
Consistency math

Strong: $N < R + W$
Easy: $W = N$
Tricky: $W < N$
Eventual/Weak: $N \geq R + W$
Consistency Options

- BigTable (HBase): Strong
- Dynamo (Voldemort/Riak/Cassandra): tunable strong or eventual
Consistency Options

• BigTable (HBase): Strong
• Dynamo (Voldemort/Riak/Cassandra): tunable strong or eventual
• ...and many others (Yahoo! PNUTs has timeline consistency)
How do we get consistent (eventually)?

Sally: 120K

Sally: 120K

Sally: 100K

Sally: 120K

Sally: 120K

Sally: 120K

Sally: 120K
How does Dynamo go from weak to eventual consistency?

- Version w/ vector clocks, read repair
- Hinted handoff on failure
- Anti-entropy
- Gossip-based membership
Version w/ vector clocks

Sally: 120K  

?  

Sally: 100K
How replicas get out of sync
How replicas get out of sync

N = 3, W = 1, R = 1

Sally: 100K

120K

Sally: 100K

Sally: 100K

Give Sally a raise!
How replicas get out of sync

$N = 3, W = 1, R = 1$
How replicas get out of sync

N = 3, W = 1, R = 1

Sally: 120K
Sally: 120K
Sally: 100K
How replicas get out of sync

N = 3, W = 1, R = 1

Sally: 120K
Sally: 120K
Sally: 100K

MOAR RAISE!
How replicas get out of sync

N = 3, W = 1, R = 1

Sally: 120K

Sally: 120K

Sally: 130K
How replicas get out of sync

Sally Salary?

N = 3, W = 1, R = 1

N = 3, W = 1, R = 1
Simple versioning doesn't work!

Sally: 100K
Version: 20

Sally: 100K
Version: 20

Sally: 100K
Version: 20
Simple versioning doesn't work!

Sally: 120K  
Version: 21

Sally: 120K  
Version: 21

Sally: 100K  
Version: 20
Simple versioning doesn't work!

- Sally: 120K  Version: 21
- Sally: 120K  Version: 21
- Sally: 130K  Version: 21
Vector clocks to the rescue

- Maintain vector with version per replica
- Each client request sent to one replica, which forwards to others
- Routing replica updates its vector entry by 1
Vector clocks to the rescue

- Maintain vector with version per replica
- Each client request sent to one replica, which forwards to others
- Routing replica updates its vector entry by 1
Vector clocks to the rescue

- Maintain vector with version per replica
- Each client request sent to one replica, which forwards to others
- Routing replica updates its vector entry by 1
Vector clocks to the rescue

- Maintain vector with version per replica
- Each client request sent to one replica, which forwards to others
- Routing replica updates its vector entry by 1

\[ 120\text{K} \]

- Sally:
  - Version: \(<2,1,1>\)
  - 120K

- Sally:
  - Version: \(<1,1,1>\)
  - 100K

- Sally:
  - Version: \(<1,1,1>\)
  - 100K

\[ \text{Version:<2,1,1>} \]
Vector clocks to the rescue

- Maintain vector with version per replica
- Each client request sent to one replica, which forwards to others
- Routing replica updates its vector entry by 1
Vector clocks to the rescue

- Maintain vector with version per replica
- Each client request sent to one replica, which forwards to others
- Routing replica updates its vector entry by 1
Read repair

- Client reads a value from multiple replicas
- Compares vector clocks
  - \(<2,1,1> = <2,1,1>\) (in sync)
  - \(<2,1,1> > <1,1,1>\) (update \(<1,1,1>\) before conflict)
  - \(<2,1,1> !! <1,1,2>\) (conflict)
How does Dynamo go from weak to eventual consistency?

- Version w/ vector clocks, read repair
- Hinted handoff on failure
- Anti-entropy
- Gossip-based membership
How does Dynamo go from weak to eventual consistency?

- Version w/ vector clocks, read repair
- Hinted handoff on failure
- Anti-entropy
- Gossip-based membership

Proactively stay in sync
How does Dynamo go from weak to eventual consistency?

- Version w/ vector clocks, read repair
- Hinted handoff on failure
- Anti-entropy
- Gossip-based membership

Proactively stay in sync
Track replicas symmetrically
Replication

• Consistency
• Availability
• Playing catchup
Partitioning

- Consistent hashing: Voldemort, Riak
- Range partitioning: HBase, MongoDB
- Both: Cassandra
Your partitioning scheme matters

- Partition so all requests go to one machine
  - Ideally, all joins happen on one machine
  - Example: map all GMail for a user to one machine
Your partitioning scheme matters

- Partition so all requests go to one machine
  - Ideally, all joins happen on one machine
  - Example: map all GMail for a user to one machine
- Highly networked data (Facebook News) hard to partition: roughly two options
  - Each request floods many partitions
  - Denormalize data (replicate), partition by user
history
the parents of nosql
tradeoffs + demos
scaling with snoop dogg
real-world usage
discussion
NoSQL Use-cases

- Cassandra at Netflix
- HBase at Facebook
- MongoDB at Craigslist
Cassandra

- BigTable data model: key→column family
- Dynamo sharding model: consistent hashing
- Eventual or strong consistency
Cassandra at Netflix

- Transitioned from Oracle
- Store customer profiles, customer:movie watch log, and detailed usage logging

“Replicating Datacenter Oracle with Global Apache Cassandra on AWS” by Adrian Cockcroft
http://www.slideshare.net/adrianco/migrating-netflix-from-oracle-to-global-cassandra
Cassandra at Netflix

- Transitioned from Oracle
- Store customer profiles, customer:movie watch log, and detailed usage logging
- In-datacenter: 3 replicas, per-app consistency

“Replicating Datacenter Oracle with Global Apache Cassandra on AWS” by Adrian Cockcroft
http://www.slideshare.net/adrianco/migrating-netflix-from-oracle-to-global-cassandra
Cassandra at Netflix (cont'd)

- Benefit: async inter-datacenter replication
- Benefit: no downtime for schema changes
- Benefit: hooks for live backups
HBase

- Data model: key→column family
- Sharding model: range partitioning
- Strong consistency
HBase

- Data model: key → column family
- Sharding model: range partitioning
- Strong consistency

- Applications
  - Logging events/crawls, storing analytics
  - Twitter: replicate data from MySQL, Hadoop analytics
  - Facebook Messages
HBase for Facebook Messages

- Cassandra/Dynamo eventual consistency was difficult to program against
HBase for Facebook Messages

- Cassandra/Dynamo eventual consistency was difficult to program against

- Benefit: simple consistency model
- Benefit: flexible data model
- Benefit: simple partitioning, load balancing, replication
MongoDB

- Document-based data model
- Range-based partitioning
- Consistency depends on how you use it
MongoDB: Two use-cases

- Archiving at Craigslist
  - 2.2B historical posts, semi-structured
  - Relatively large blobs: avg 2KB, max > 4 MB
MongoDB: Two use-cases

- Archiving at Craigslist
  - 2.2B historical posts, semi-structured
  - Relatively large blobs: avg 2KB, max > 4 MB
- Checkins at Foursquare
  - Geospatial indexing
  - Small location-based updates, partitioned on user
history
the parents of nosql
tradeoffs + demos
scaling with snoop dogg
real-world usage
discussion
Polyglot Persistence

- Account data in MySQL
- Cache in memcached
- Counters + queues in redis
- Statistics in MongoDB
- Event logging in HDFS/HBase/Cassandra
When you see “NoSQL,” think “tradeoffs”

- Data model
- Query Model
- Transactions
- Consistency vs. Availability
- Partitioning schemes

Adam Marcus
marcua@csail.mit.edu
@marcua