# A Brief Introduction to Key-Value Stores

Jakob Blomer

ALICE Offline Week, CERN
July 1st 2015

1. A Little Bit on NoSQL, ACID, BASE, and the CAP Theorem

2. Three Examples: Riak, ZooKeeper, RAMCloud

3. Blueprint of Twitter's Real-Time Data Analytics Platform

# Thoughts behind Distributed Key-Value Stores

∼2000–2010

**SQL databases do not scale to the needs of large web services**

1. Can we scale out a database ("horizontal scaling") if we give up on the relational data model?
   - Dictionary as a simple, easy to distribute yet useful data structure

Scaling

# Thoughts behind Distributed Key-Value Stores

$\sim$2000–2010

**SQL databases do not scale to the needs of large web services**

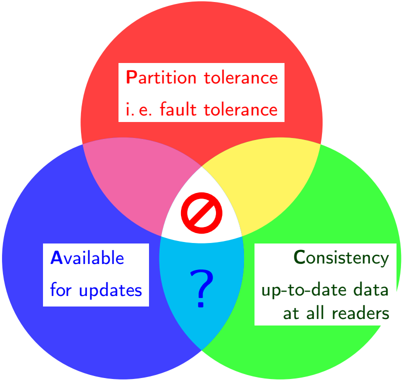
1. Can we scale out a database ("horizontal scaling") if we give up on the relational data model?
   - Dictionary as a simple, easy to distribute yet useful data structure

   Scaling

2. In the presence of inevitable faults, can we still make progress ("availability")?
   - Boosted by Amazon Dynamo paper SOSP'07
   - Idea of "eventual consistency": heal data inconsistency once the system recovers from faults
   - Even though the *interface* is simple, the *implementation* of a distributed key-value store is highly non-trivial

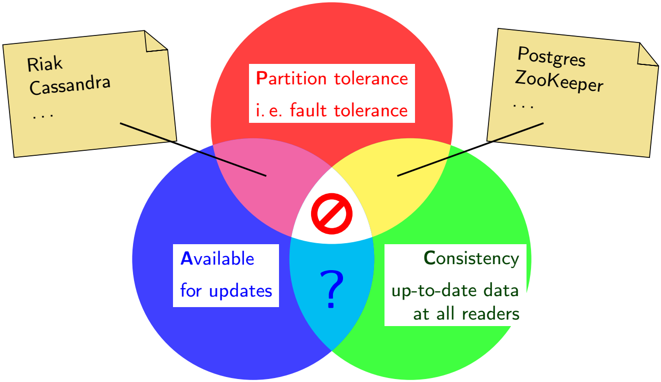
   Fault-Tolerance

*A distributed storage system can have at most two out of three desirable properties* Brewer '97 Brewer '12

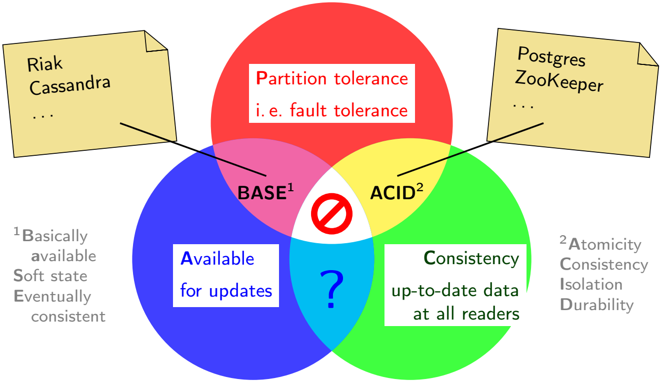


# CAP Theorem

*A distributed storage system can have at most two out of three desirable properties* ▸ Brewer '97 ▸ Brewer '12



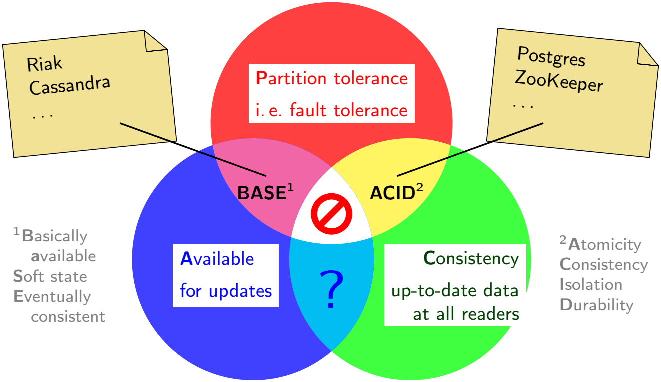
*A distributed storage system can have at most two out of three desirable properties* ▸ Brewer '97 ▸ Brewer '12




[1] **B**asically **a**vailable **S**oft state **E**ventually consistent

[2] **A**tomicity **C**onsistency **I**solation **D**urability

*A distributed storage system can have at most two out of three desirable properties* ▸ Brewer '97 ▸ Brewer '12

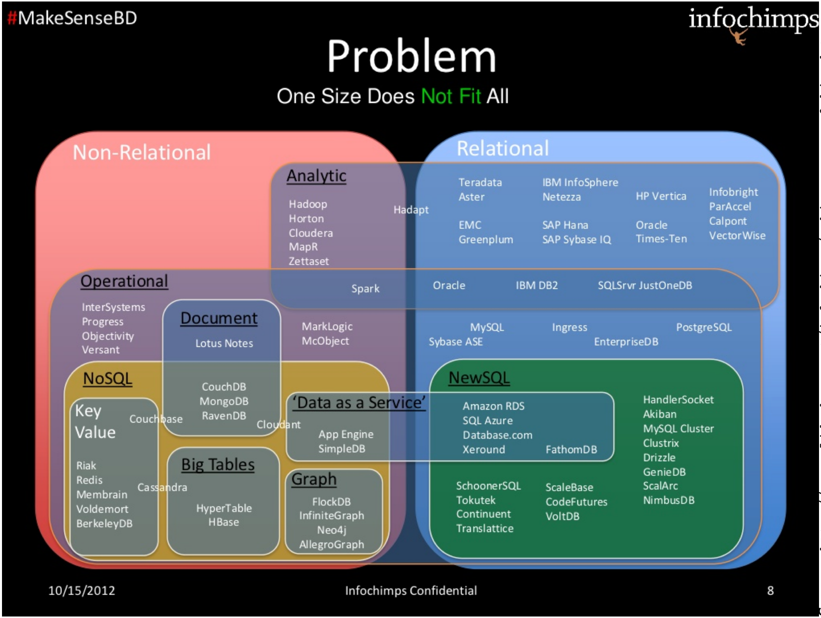



[1] **B**asically **a**vailable **S**oft state **E**ventually consistent

[2] **A**tomicity **C**onsistency **I**solation **D**urability

The tradeoffs between availability and consistency can be granular and subtle
For instance: a disconnected ATM might still allow small withdrawals

# The NoSQL Movement Explores the CAP Space

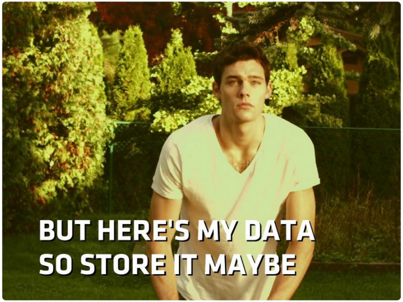



Source: http://www.slideshare.net/infochimps/making-sense-of-big-data

## The Jepsen Experiments

Series of experiments by Kyle Kingsbury (aka aphyr) to
*"demonstrate how some real distributed systems behave when the network fails"*

→ `https://aphyr.com/tags/Jepsen`


Kyle Kingsbury

**Experiment Setup**

Five node server cluster, single client with a series of requests on mutable data (e. g. increment a counter).
At some point two out of five servers fail. They recover some time later.

# Call Me Maybe (contd.)

**Experiment Setup**

Five node server cluster, single client with a series of requests on mutable data (e. g. increment a counter).
At some point two out of five servers fail. They recover some time later.

> *(W)e demonstrate Redis losing 56% of writes during a partition.*

> *MongoDB is neither AP nor CP. The defaults can cause significant loss of acknowledged writes. The strongest consistency offered has bugs which cause false acknowledgements, and even if they're fixed, doesn't prevent false failures.*

> *Today, we'll see how last-write-wins in Riak can lead to unbounded data loss.*

> *Cassandra lightweight transactions are not even close to correct. Depending on throughput, they may drop anywhere from 1-5% of acknowledged writes—and this doesn't even require a network partition to demonstrate.*

**Experiment Setup**

Five node server cluster, single client with a series of requests on mutable data (e. g. increment a counter).
At some point two out of five servers fail. They recover some time later.

*Kafka's replication claimed to be CA, but in the presence of a partition, threw away an arbitrarily large volume of committed writes.*

*RabbitMQ lost ∼35% of acknowledged writes.*

*Use Zookeeper. It's mature, well-designed, and battle-tested.*

*I look forward to watching both etcd and Consul evolve.*

**Note**

All of the tested systems are incredibly useful and developed by very capable engineers! But we should be aware of the real limitations beyond marketing claims.

A distributed key-value store typically

- is provided by a cluster of commodity servers
- scales horizontally: hundreds or thousands of nodes
- is fault-tolerant in some sense
- has a simple dictionary interface:
  `PUT(<key>, <value>)` and `GET(<key>);`
  for small keys and values (<1 MB)
  sometimes with extensions such as
  atomic operations, transactions, secondary indexes, ...
  or higher order data structures such as queue, stack, ...
  **but no** relational algebra, fuzzy queries, triggers/stored procedures

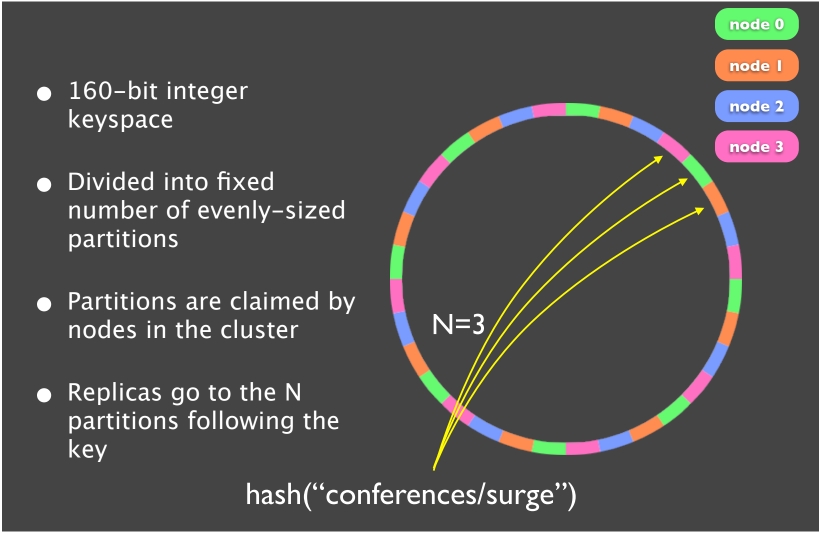
**Examples**

Amazon Dynamo ▸ SOSP'07, Riak, Cassandra, HBase, Hyperdex, etcd, RAMCloud ▸ TOCS'15, ZooKeeper ▸ USENIX'10, redis, ...

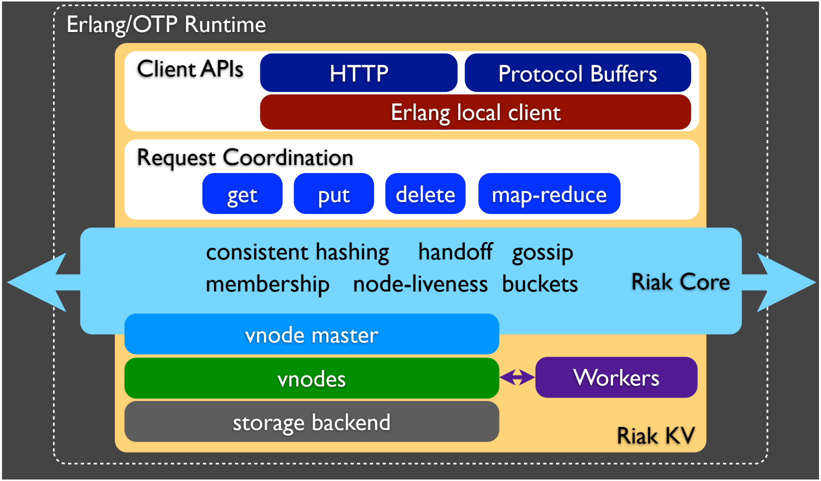
**There is no standard set of data structures and operations (like with SQL). Key value stores all differ from each other.**

**Riak Brief**

- Developed by Basho Technologies since ∼2008
- Apache licensed, with additional commercial options
- ≈80 k lines of code, mostly Erlang
- Available for many distributions, including OS X
- Open implementation of Amazon Dynamo
- AP system with some nobs to trade off consistency/availability fully decentralized
- Used by many large companies




Source: Ian Plosker / Basho




Source: Ian Plosker / Basho

## Note on Eventual Consistency

- Objects can end up on both sides of a network partition
- On recovery, conflict resolution needs to be handled by the user/application
  1. Addition only of immutable key-value pairs
  2. Use of a custom merge function
  3. Use of *Conflict-Free Replicated Data Types* CRDT'11

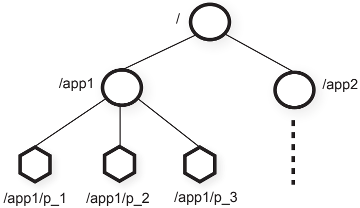
     → `http://basho.com/posts/technical/distributed-data-types-riak-2-0`

## Performance

- Small scale CMS benchmarks testing Riak for conditions data → `http://cern.ch/go/cw8T`
- Throughput (not latency!) should scale nicely with more machines; remains to be tested

## ZooKeeper Brief

- Developed by Yahoo (publication in 2010), now Apache project
- Apache licensed
- ≈100 k – 150 k lines of code, mostly Java
- Available for many distributions
- Distributed consensus system,
  all writes need to be acknowledged by a majority of nodes
  typical cluster size: 5 nodes
- Decent throughput of tens of thousands to hundreds of thousands of requests per second
- Often used in addition to other key-value stores to keep high-level information
  e. g. cluster membership, table placement

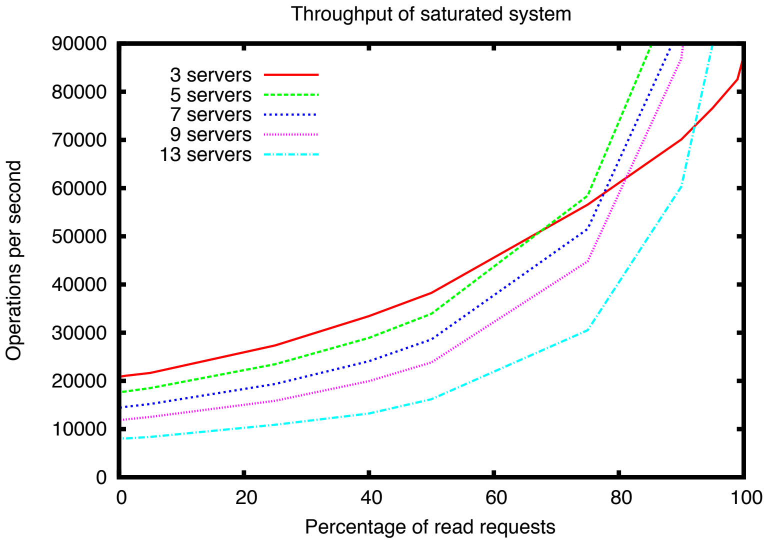



**RPCs**

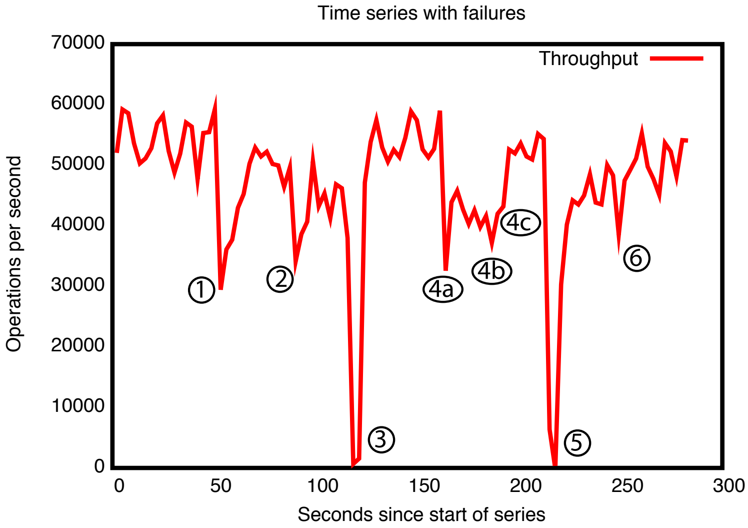
```
create(path, data, flags)
delete(path, version)
exists(path, watch)
getData(path, watch)
setData(path, data, version)
getChildren(path, watch)
```

**Hierarchical key-value store (resembles a file system)**

# ZooKeeper Benchmarks




From 2010, GbE, USENIX'10




From 2010, GbE, USENIX'10

## RAMCloud Brief

- Developed since 2011 at Stanford University
- MIT license
- Aims at production grade software (e. g. fully unit-tested)
- $\approx$100 k lines of C++ code
- Easy to deploy: compiles on SL6
- Highly performance-tuned: low latency at large scale
  an order of magnitude smaller latency than other key-value stores
  **Also:** very well understood performance (tail latency, individual components, . . . )
- CP system, linearizable ("exactly once") semantics is almost fully implemented
- Used, for instance, by ONOS to store the routing information for software-defined networks

# RAMCloud Data Model

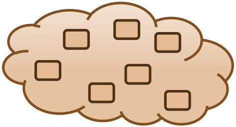
## Entities

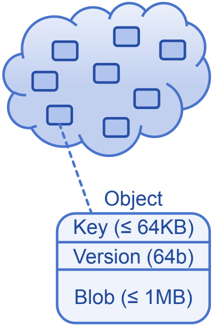
- Table
- Object (row): Key + Value + Version
- Tablet: partition of a table (block of rows)

## Operations

- `read(tableId, version)` $\rightarrow$ blob, version
- `write(tableId, key, value)` $\rightarrow$ version
- `delete(tableId, key)`
- `cwrite(tableId, key, value, version)` $\rightarrow$ version
  conditional write, simplifies concurrency control
- Atomic increment
- Secondary indices (range queries)
- Enumerate objects in a table

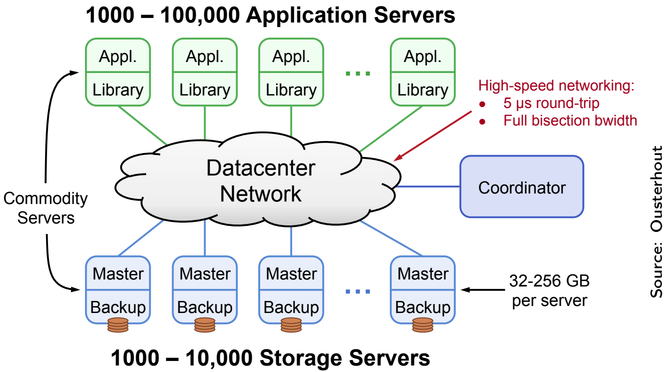
**Tables**






**Source: Ousterhout**

# RAMCloud – System Overview




**Key Parameters**

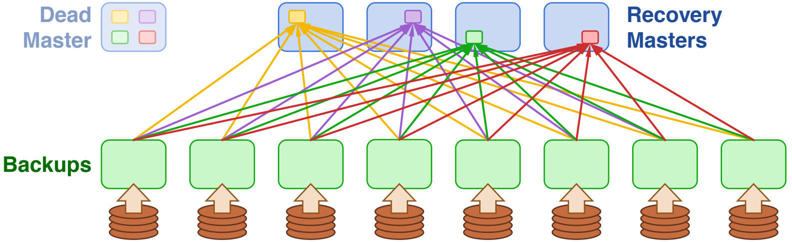
- All data guaranteed to be in **memory**, thus up to 1M ops/sec/server
- Extra **low latency** (InfiniBand): 5 µs to read, 15 µs to write
- Reliable, $k$ **replicas on disk** (buffered log, no disk write during store)

Some publications: TOCS'15 SOSP'11 Raft'14 HotOS'13 FAST'14

**Starting point:** RAMcloud keeps a single copy in RAM and $k$ copies on disk

Recovery of 32 GB of memory in 1 s to 2 s leveraging scale

1. Data backups are scattered over entire cluster in 8 MB segments
   ⇒ recovery can read with 100 MB/s from hundreds of nodes
2. Data on a server is *partitioned*
   ⇒ recovery can write with 1 GB/s to tens of new masters




Source: Ongaro

- Twitter Heron replacing Storm as real-time stream data processing platform for a shared cluster ▸ SIGMOD'15
- Used for real-time active user counts, ads evaluation, ...
- Every job: a topology of data sources and sinks and transformational tasks

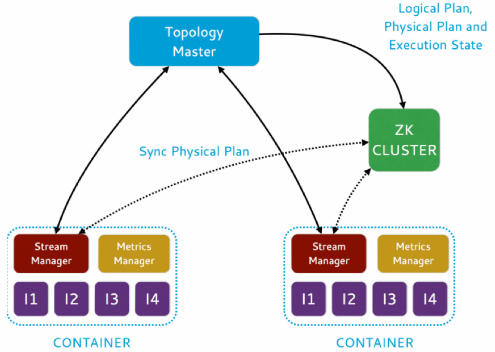



*Figure 2. Topology Architecture*

Off-the-shelf components: ZooKeeper registry, Mesos scheduler, Linux containers

Source: `https://blog.twitter.com/2015/flying-faster-with-twitter-heron`

# Heron Benchmarks

## Word count

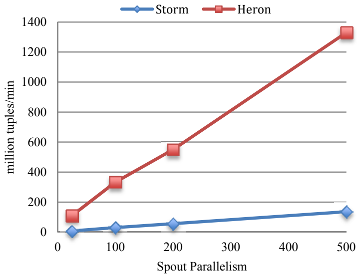



**Figure 9: Throughput with acknowledgements**

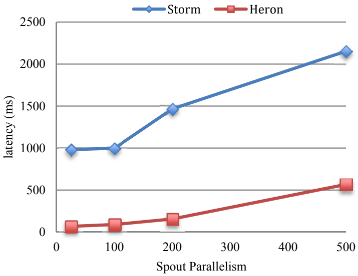



**Figure 10: End-to-end latency with acknowledgements**

SIGMOD'15

Thank you for your time!