cloudera
Ask Bigger Questions

Apache HBase 0.96 and What's Next

Jonathan Hsieh | @jmhsieh

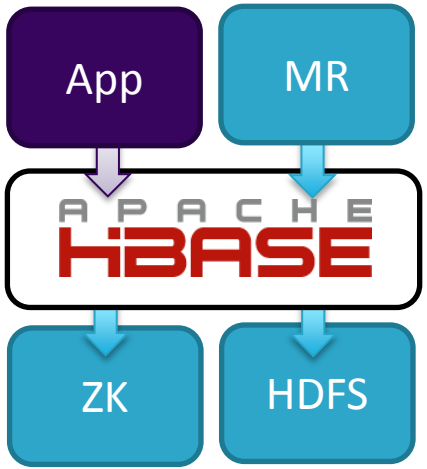
Software Engineer at Cloudera | HBase PMC Member

October 29, 2013

Who Am I?

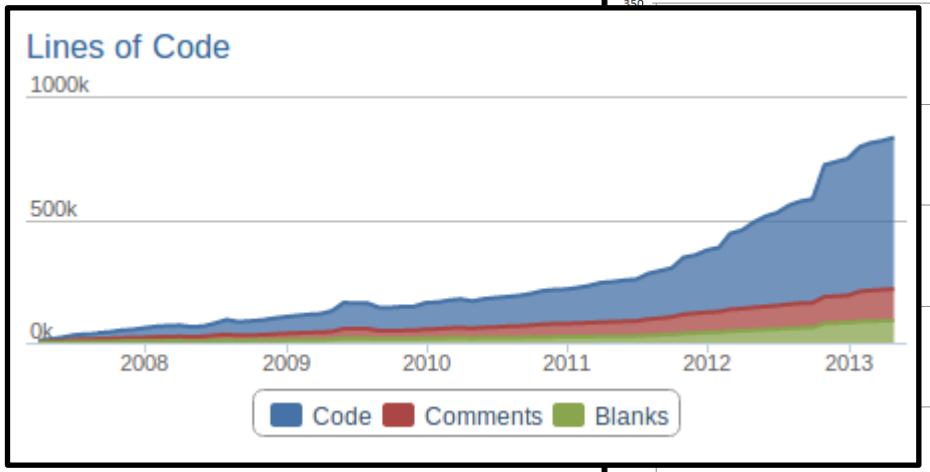
- **Cloudera:**
  - **Software Engineer**
  - **Tech Lead HBase Team**
  - **Apache HBase** committer / PMC
  - **Apache Flume** founder / PMC
- **U of Washington:**
  - Research in Distributed Systems

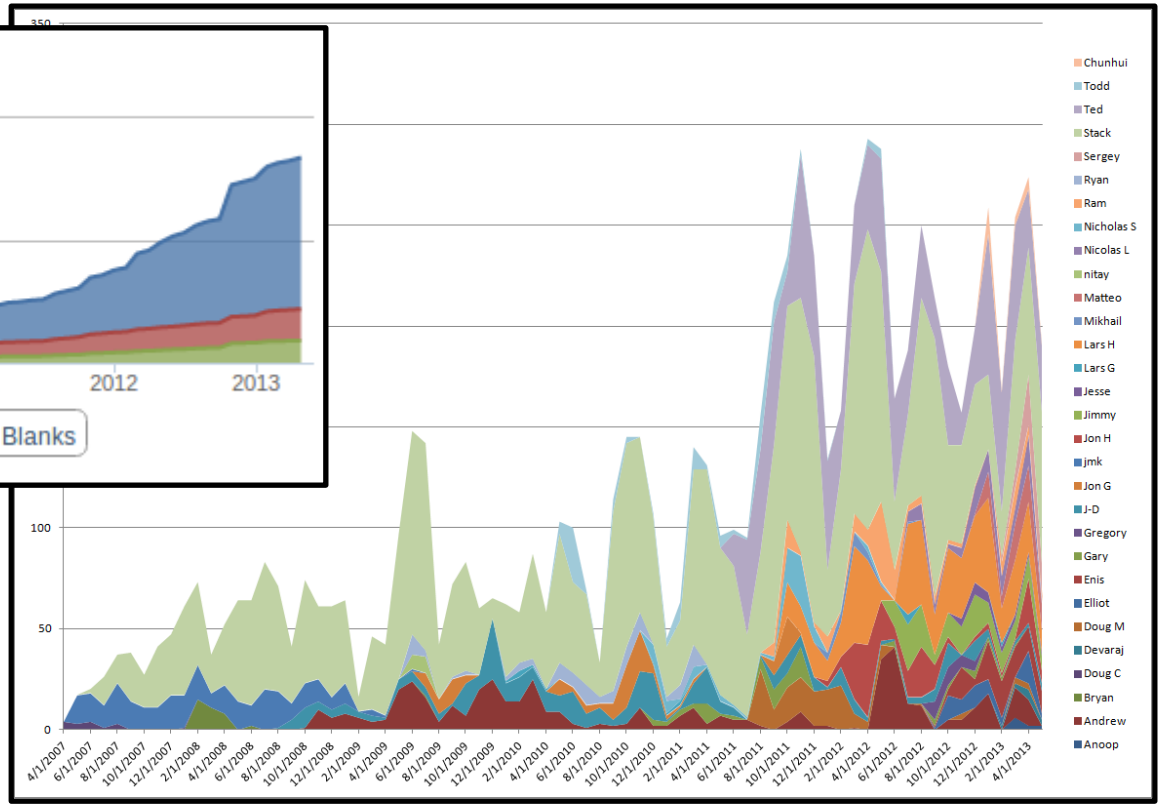
What is Apache HBase?



Apache HBase is an open source, distributed, consistent, non-relational database that provides low-latency, random read/write operations.

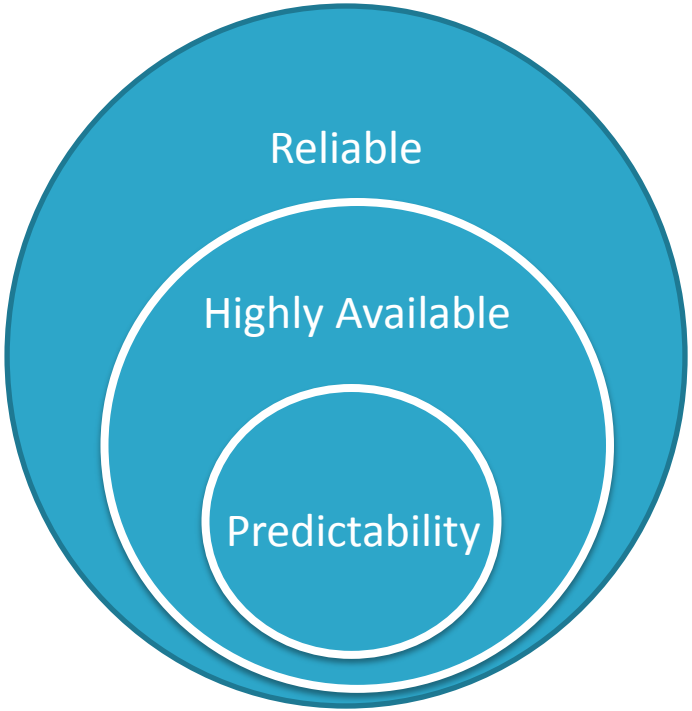
Developer Community





- Vibrant, Highly Active community!
- We're Growing!

Today: Apache HBase 0.96.0



- HBase is **fault tolerant**
  - HDFS for **durability** via replication
- HBase has good **availability**
  - Uses ZK for **coordination** via quorums
  - New features for fast RS recovery in 0.96
- HBase strives for **predictable** performance
  - Some experimental features in 0.96, and more improvements in development.

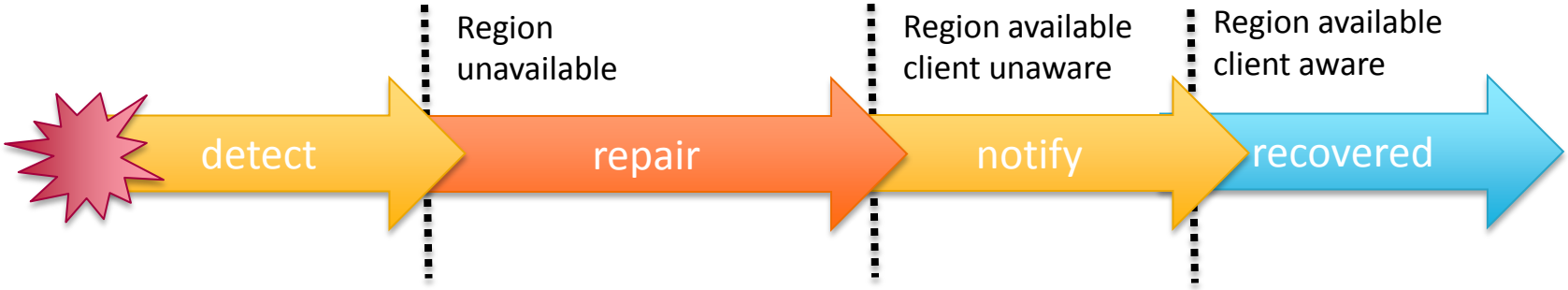
Summary by Version

	0.90 (CDH3)	0.92 /0.94 (CDH4)	0.96 (CDH5)	Next
New Features	Stability	Reliability	Continuity	Multitenancy
MTTR	Recovery in Hours	Recovery in Minutes	Recovery of writes in seconds, reads in 10's of seconds	Recovery in Seconds (reads+writes)
Perf	Baseline	Better Throughput	Optimizing Performance	Predictable Performance
Usability	HBase Developer Expertise	HBase Operational Experience	Distributed Systems Admin Experience	Application Developers Experience

MTTR Recap

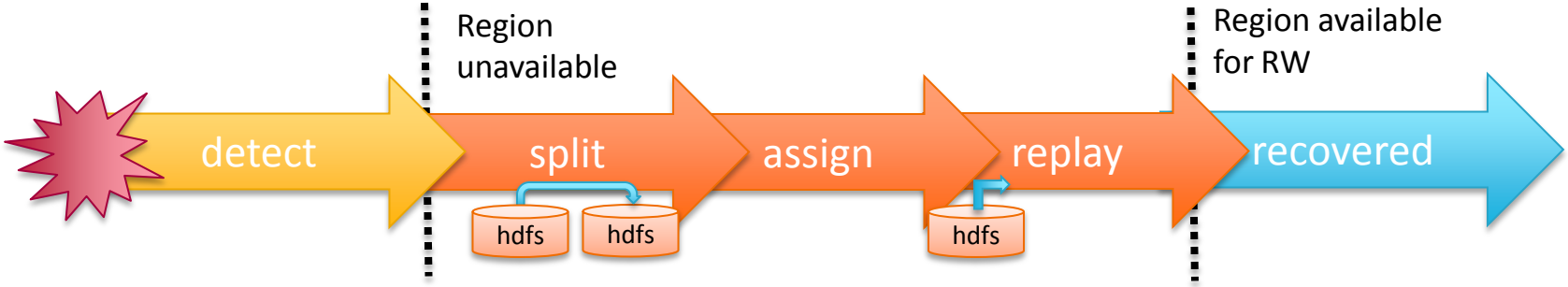
Mean time to recovery

Mean Time to Recovery (MTTR)



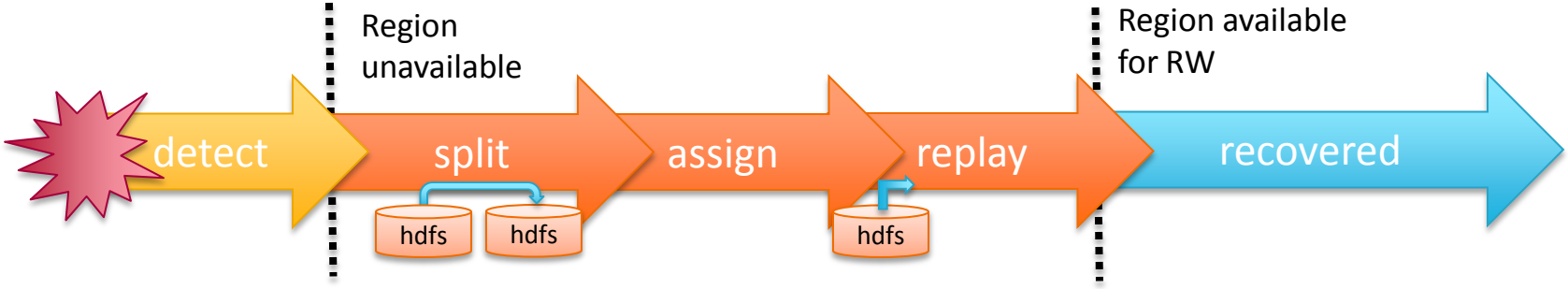
- Machine failures happen in distributed systems
- Average unavailability when automatically recovering from a failure.
- Recovery time for a unclean data center power cycle

Distributed log splitting (0.92)



- Repair == split, assign, replay
  - When many servers fail, many logs to recover
  - Instead of just at the master, RS's split logs in parallel
  - Huge win in recovery time

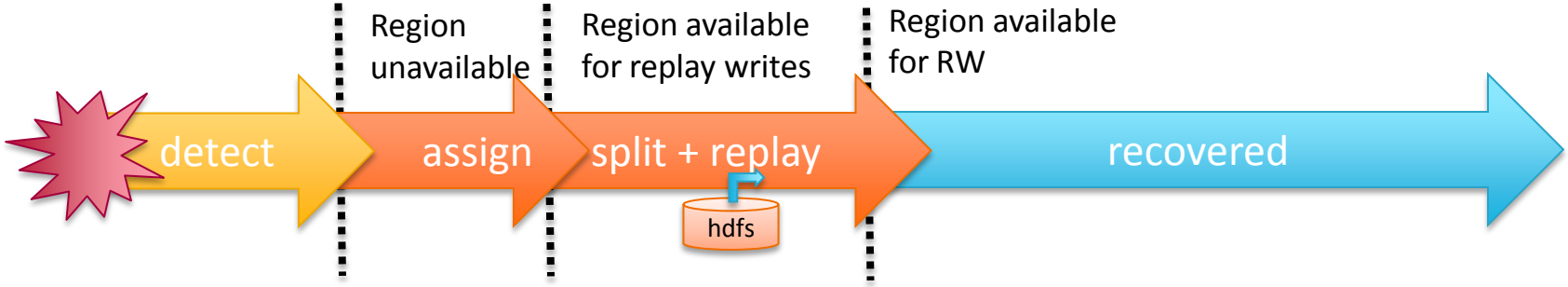
Fast notification and detection (0.96)



- Proactive notification of HMaster failure (0.96)
- Proactive notification of RS failure (0.96)
- Notify client on recovery (0.96)

- Fast server failover (Hardware)

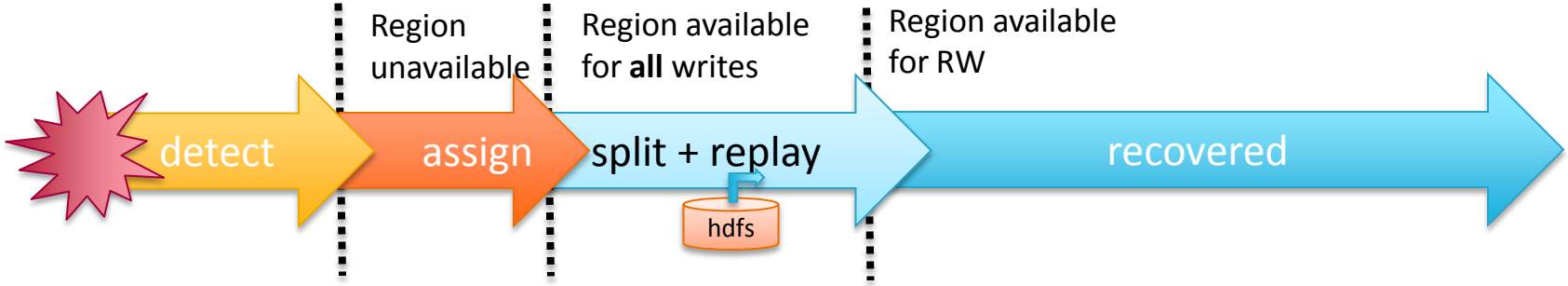
Distributed log replay (0.96)



- Previously had two IO intensive passes:
  - Log splitting to intermediate files
  - Assign and log replay
- Now just one IO heavy pass: Assign first, then split+replay.
  - Improves read and write recovery times.
  - Off by default currently*.

*Caveat: If you override time stamps you could have READ REPEATED isolation violations (use tags to fix this)

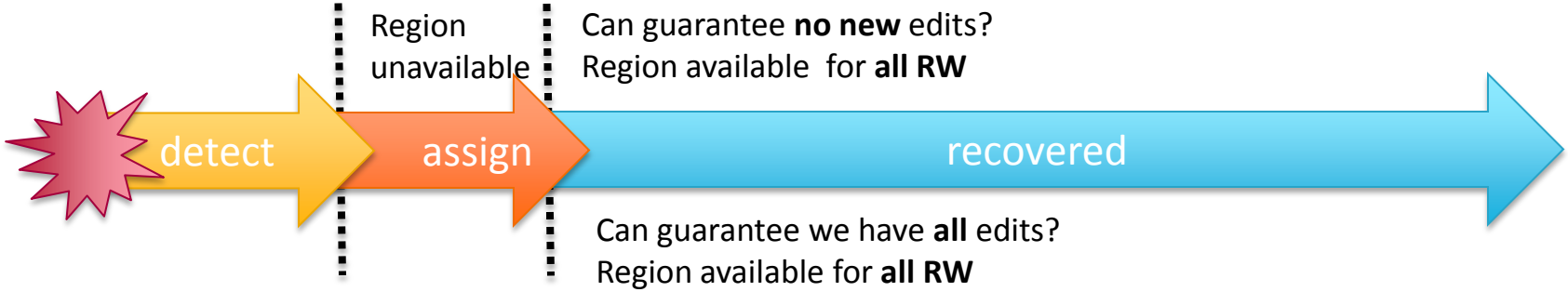
Distributed log replay with fast write recovery



- Writes in HBase do not incur reads.
- With distributed log replay, we've already have regions open for write.
- Allow fresh writes while replaying old logs*.

*Caveat: If you override time stamps you could have READ REPEATED isolation violations (use tags to fix this)

Fast Read Recovery



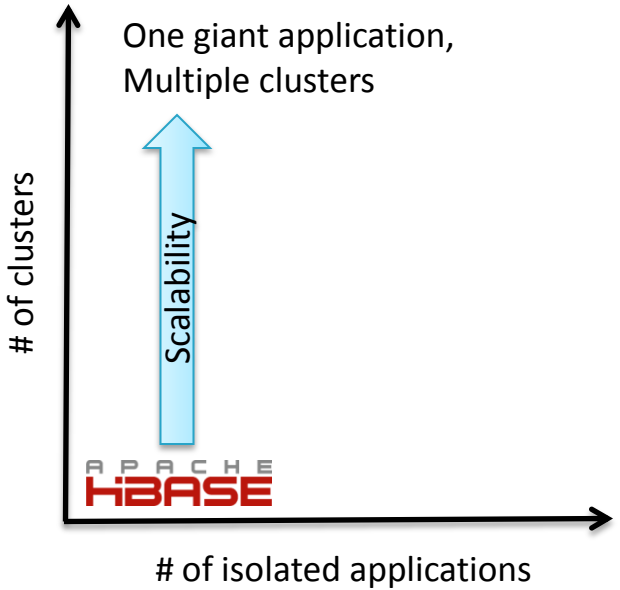
- Idea: Pristine Region fast read recovery
  - If region not edited it is consistent and can recover RW immediately
- Idea: Shadow Regions for fast read recovery
  - Shadow region tails the WAL of the primary region
  - Shadow memstore is one HDFS block behind, catch up recover RW
- Currently some progress for trunk

Multi-tenancy

Many apps and users in a single cluster

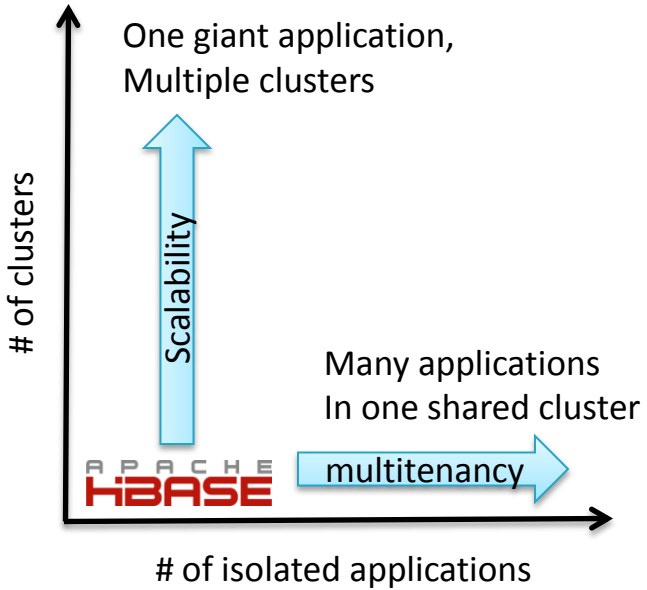
Growing HBase

- Pre 0.96.0: scaling up HBase for single HBase applications
  - Essentially a single user for single app.
  - Ex: Facebook messages, one application, many hbase clusters
    - Shard users to different pods
- Focused on continuity and disaster recovery features
  - Cross-cluster Replication
  - Table Snapshots
  - Rolling Upgrades



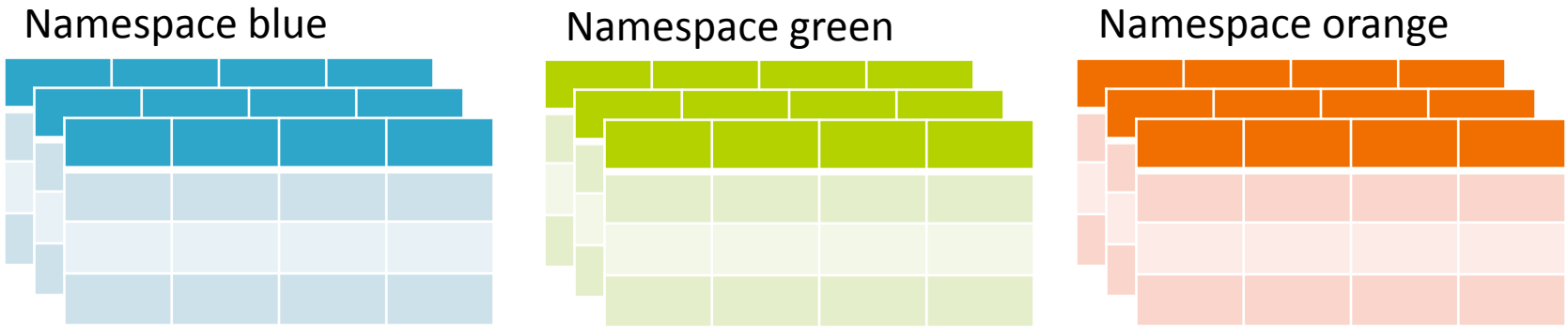
Growing HBase

- In 0.96 we introduce primitives for supporting **Multitenancy**
  - Many users, many applications, one HBase cluster
  - Need to have some control of the interactions different users cause.
  - Ex: Manage for MR analytics and low-latency serving in one cluster.



Namespaces (0.96)

- Namespaces provide an abstraction for multiple tenants to create and manage their own tables within a large HBase instance.

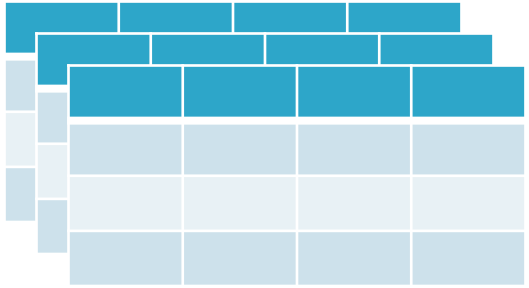


Multitenancy goals

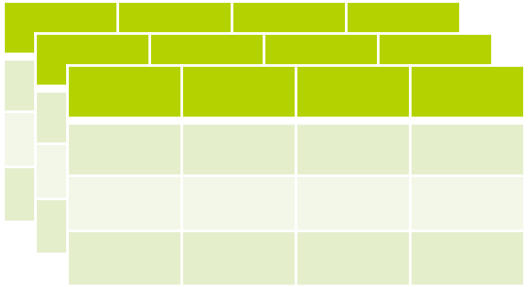
- Security (0.96)
  - A separate admin ACLs for different sets of tables
- Quotas (in progress)
  - Max tables, max regions.
- Performance Isolation (in progress)
  - Limit performance impact load on one table has on others.
- Priority (future)
  - Handle some tables before others

Isolation with Region Server Groups

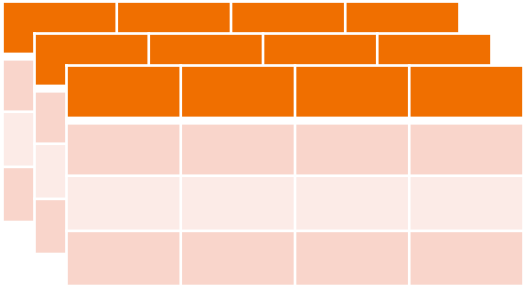
Namespace blue



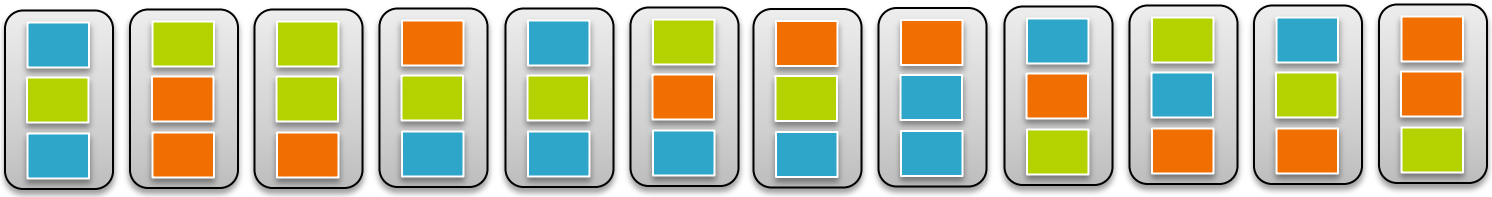
Namespace green



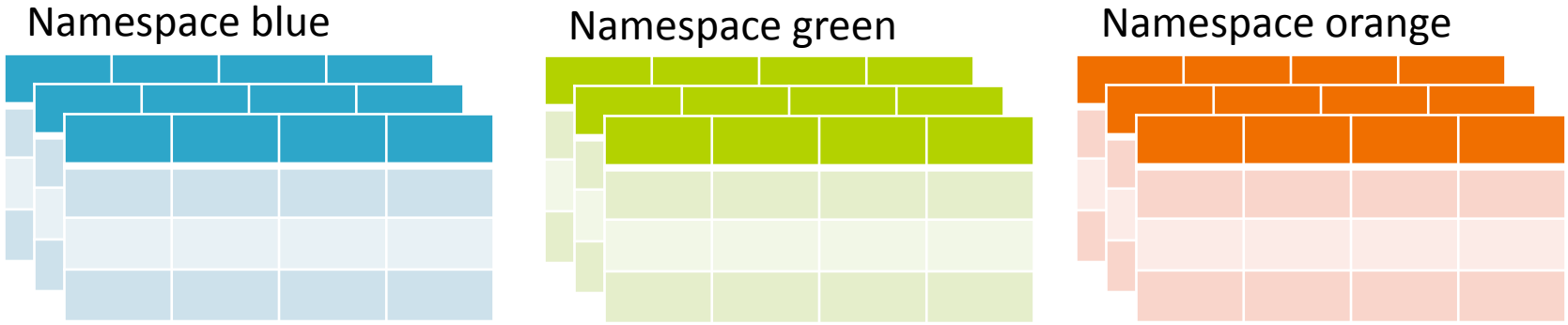
Namespace orange



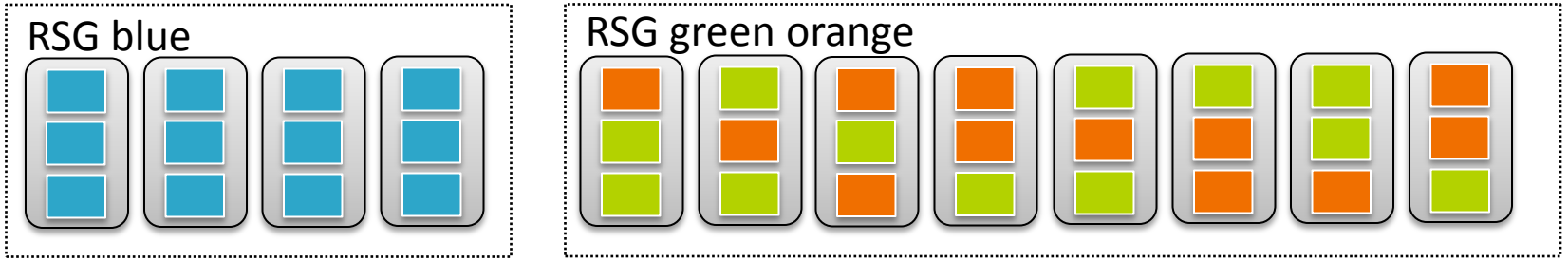
Region assignment distribution (no region server groups)



Isolation with Region Server Groups



Region assignment distribution with Region Server Groups (RSG)



Cell Tags

- Mechanism for attaching arbitrary metadata to Cells.
- Motivation: Finer-grained isolation
  - Use for Accumulo-style cell-level visibility
- Main feature for 0.98 (in development).
- Other uses:
  - Add sequence numbers to enable correct fast read/write recovery
  - Potential for schema tags

Improving Predictability

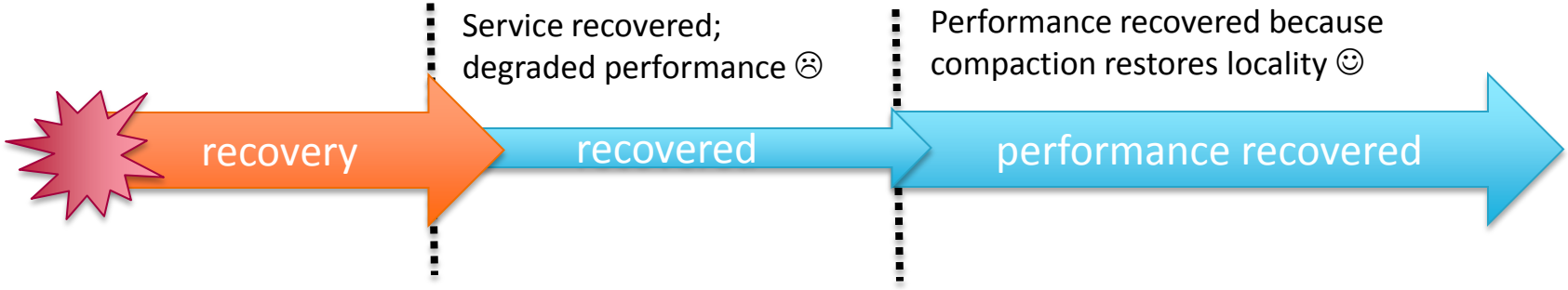
Improving the 99%tile

Common causes of performance variability

- **Locality Loss**
  - **Favored Nodes, HDFS block affinity**
- Compaction*
  - Exploring compactor
- GC
  - Off-heap Cache
- **Hardware hiccups**
  - **Multi WAL, HDFS speculative read**

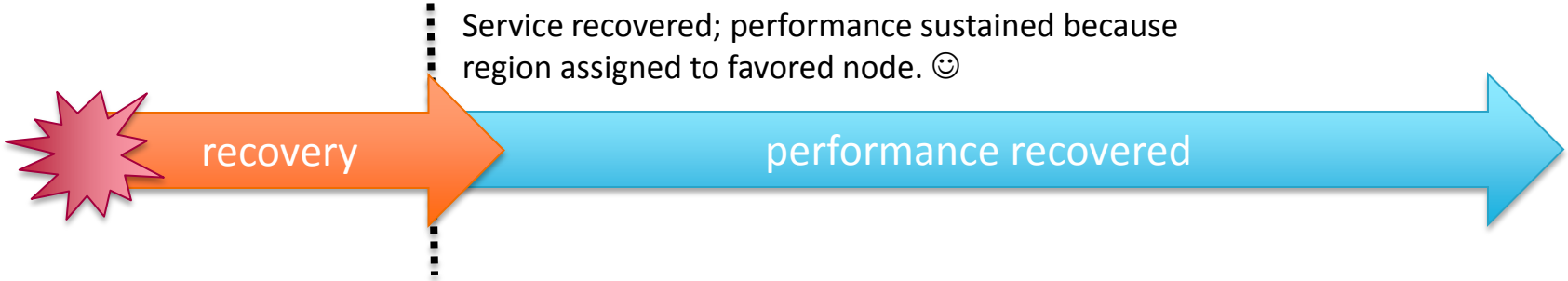
*See my Hadoop Summit 2013 talk

Performance degraded after recovery



- After recovery, reads suffer a performance hit.
  - Regions have lost locality
  - To maintain performance after failover, we need to regain locality.
  - Compact Region to regain locality
- We can do better by using HDFS features

Read Throughput: Favored Nodes (0.96)

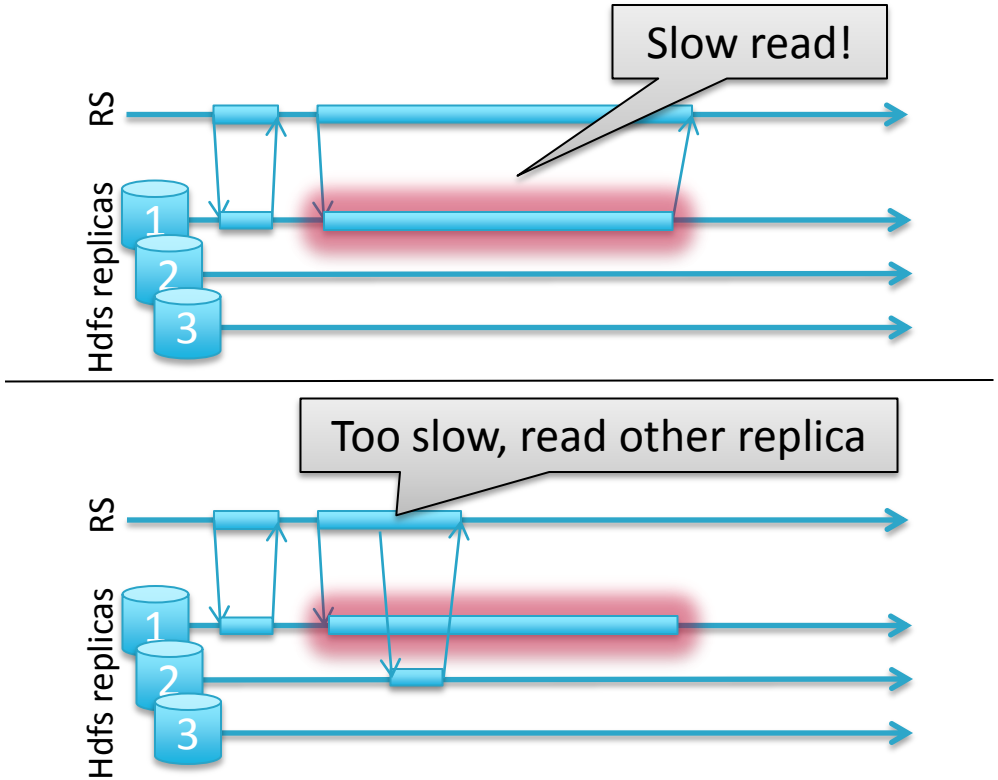


- Control and track where block replicas are
  - All files for a region created such that blocks go to the same set of **favored nodes**
  - When failing over, assign the region to one of those favored nodes.
- Currently a preview feature in 0.96
  - Disabled by default because it doesn't work well with the latest balancer or splits.
  - Will likely use upcoming HDFS block affinity for better operability
- Originally on Facebook's 0.89, ported to 0.96

Read latency: HDFS speculative read

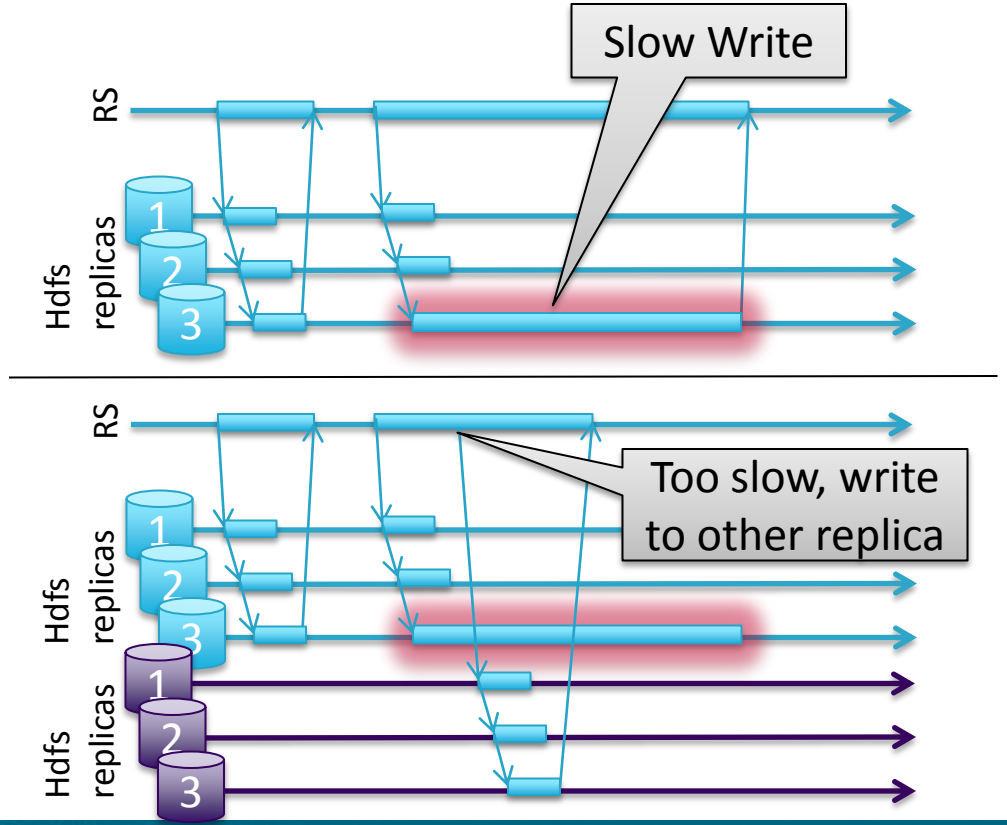
- HBase's HDFS client reads 1 of 3 replicas
- If you chose the slow node, your reads are slow.

- Idea: If a read is taking too long, speculatively go to another that may be faster.



Write latency: Multiple WALs

- HBase's HDFS client writes 3 replicas
- Min write latency is bounded by the slowest of the 3 replicas
- Idea: If a write is taking too long let's duplicate it on another set that may be faster.



HBase Extensions

An Ecosystem of projects built on HBase

Making HBase easier to use and tune.

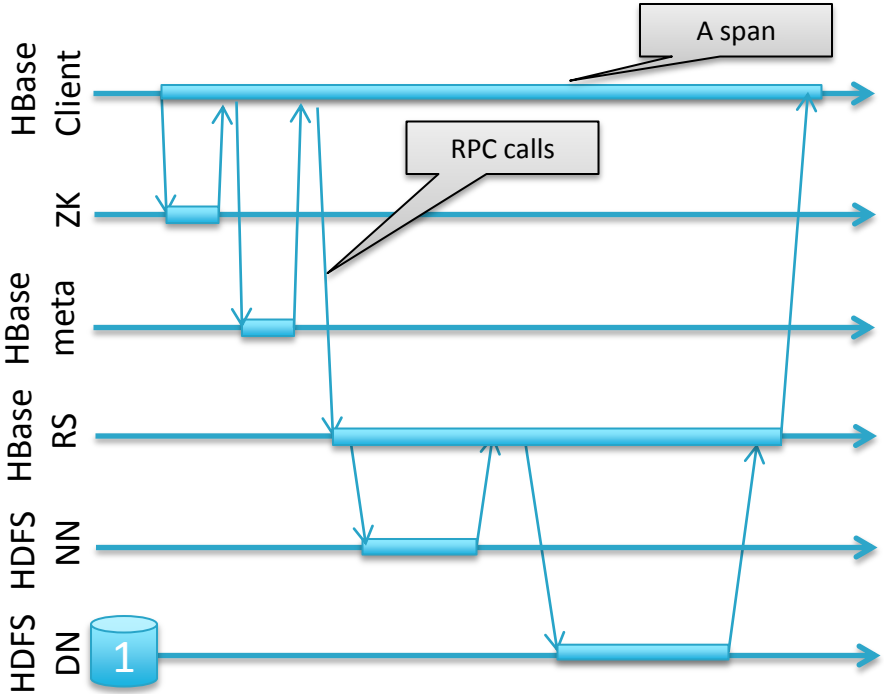
- *With great power comes great responsibility.*

- Difficult to see what is happening in HBase
- Easy to make poor design decisions early without realizing

- New Developments
  - HTrace + Zipkin
  - Frameworks for Schema design

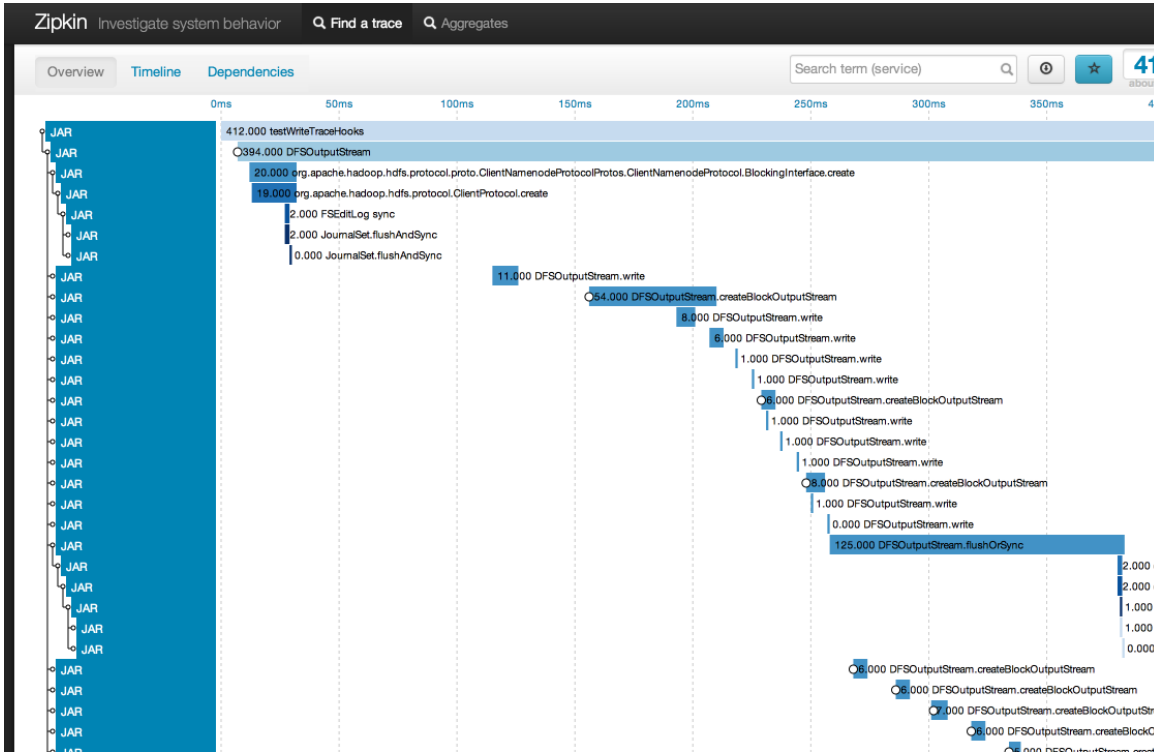
HTrace: Distributed Tracing in HBase and HDFS

- Framework Inspired by Google Dapper
- Tracks time spent in calls in RPCs across different machines.
- Threaded through HBase (0.96) and future HDFS.



Zipkin – Visualizing Spans

- UI + Visualization System
  - Written by Twitter
- Zipkin HBase Storage
- Zipkin HTrace integration
- View where time from a specific call is spent in HBase, HDFS, and ZK.



HBase Schemas

- HBase Application developers must iterate to find a suitable **HBase schema**
  - **Schema critical for Performance at Scale**
  - How can we make this easier?
  - How can we reduce the expertise required to do this?
- Old option: Learn the architecture. Use the API. Experiment and learn.

How should I arrange my data?

- Isomorphic data representations!

Short Fat Table using column qualifiers

Rowkey	d:col1	d:col2	d:col3	d:col4
bob	aaaa	bbbb	cccc	dddd
jon	eeee	ffff	gggg	hhhhh

Short Fat Table using column families

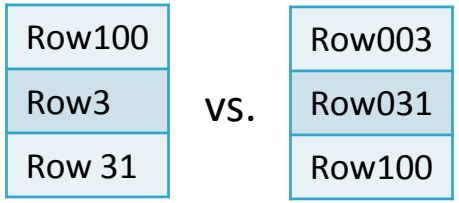
Rowkey	col1:	col2:	col3:	col4:
bob	aaaa	bbbb	cccc	dddd
jon	eeee	ffff	gggg	hhhhh

Tall skinny with compound rowkey

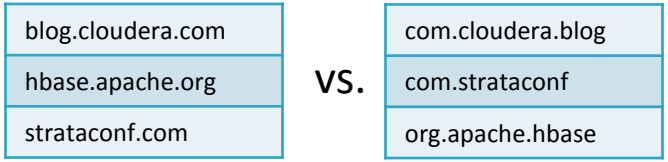
rowkey	d:
bob-col1	aaaa
bob-col2	bbbb
bob-col3	cccc
bob-col4	dddd
jon-col1	eeee
jon-col2	ffff
jon-col3	gggg
jon-col4	hhhh

Row key design techniques

- Numeric Keys and lexicographic sort
  - Store numbers big-endian.
  - Pad ASCII numbers with 0's.



- Use **reversal** to have most significant traits first.
  - Reverse URL.
  - Reverse timestamp to get most recent first.
    - (MAX_LONG - ts) so "time" gets monotonically smaller.



- Use **composite keys** to make key distribute nicely and work well with sub-scans
  - Ex: User-ReverseTimeStamp
  - Do not use current timestamp as first part of row key!

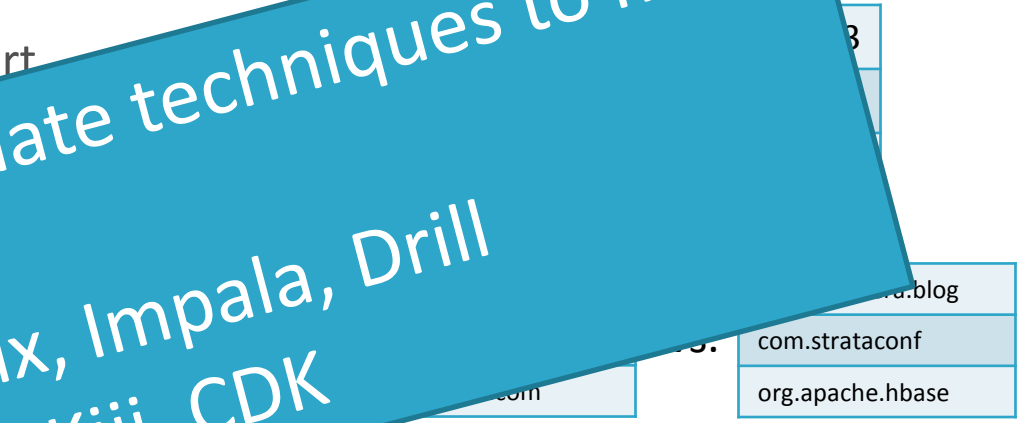
Row key design techniques

- Numeric Keys and lexicographic sort
  - Store numbers big-endian
  - Pad ASCII numbers
- Use Hashed keys
  - Hashed key distribute nicely and work well with MapReduce
  - User-ReverseTimeStamp
  - Do not use current timestamp as first part of row key!

Frameworks encapsulate techniques to make it easier for users.

SQL skins: Phoenix, Impala, Drill

App frameworks: Kiji, CDK



Phoenix

- A SQL skin over HBase targeting low-latency queries.
- JDBC SQL interface
- Highlights
  - Adds Types
  - Handles Compound Row key encoding
  - Secondary indices in development
  - Provides some pushdown aggregations (coprocessor).

- Open sourced by Salesforce.com
  - Work from James Taylor, Jesse Yates, et al
  - https://github.com/forcedotcom/phoenix

Impala

cloudera IMPALA

- Scalable Low-latency SQL querying for HDFS (and HBase!)
- ODBC/JDBC driver interface
- Highlights
  - Use's Hive metastore and its hbase-hbase connector configuration conventions.
  - Native code implementation, uses JIT for query execution optimization.
  - Authorization via Kerberos support
- Open sourced by Cloudera
  - https://github.com/cloudera/impala

Kiji

- APIs for building big data applications on HBase
  - based on Google's Bigtable usage experience
- Highlights
  - Provides types via Avro serialization and Schema encoding
  - Provides locality group that logically maps to HBase's physical column families.
  - Manages schema evolution
  - Provides framework for applying machine learning to data
- Open sourced by WibiData
  - http://www.kiji.org/

Cloudera Development Kit (CDK)

- APIs that provides a **Dataset abstraction**
  - Provides get/put/delete API in avro objects
  - HBase Support in progress
- Highlights
  - Supports multiple components of the hadoop distro (flume, morphlines, hive, crunch, hcat)
  - Provides types using Avro and parquet formats for encoding entities
  - Manages schema evolution
- Open source by Cloudera
  - http://cloudera.github.io/cdk/docs/current

Conclusions

Summary by Version

	0.90 (CDH3)	0.92 /0.94 (CDH4)	0.96 (CDH5)	Next
Major Features	**Stability** •True Durability • Replication • Import/Export/Copy	**Reliability** •True Consistency •Master-Master Replication •Coprocs + Security	**Continuity** • Protobufs • Snapshots • Namespace Security	**Multitenancy** • Cell-level Tags • Namespaces Isolation • Namespace Quotas
MTTR	**Recovery in Hours** •Distributed log splitting*	**Recovery in Minutes** •Distributed log splitting	**Recovery of writes in seconds, reads in 10's of Seconds** • Distributed log replay† • Fast Failure Notification	**Recovery in Seconds (reads+writes)** • Pristine Region read recover •Shadow Regions
Perf	**Baseline** •Metrics	**Better Throughput** • CF+Region Metrics • HFile Checksums • Short Circuit HDFS Read • Blooms and Big Hfiles	**Optimizing Performance** • HTrace • Prefix Tree Encoding† • Exploring compaction • Stochastic load balancer	**Predictable Performance** • Multi WAL • Speculative Reads • Favored nodes
Usability	**HBase Developer Expertise**	**HBase Operational Experience**	**Distributed Systems Admin Experience**	**Application Developers Experience**

† experimental in progress *backported

Questions?
@jmhsieh

More questions?
Come to the Cloudera
booth for an ask the expert
session with Jon and Lars George!