# HBase

Ryan Rawson

Sr Developer @ SU, HBase committer

June 11th, NOSQL

StumbleUpon

## Quick Backstory

- Needed large data store @ SU
- Started looking back in Jan '09
- Looked at the field of stores, tried:
  - Cassandra
  - Hypertable (fast)
  - HBase
- Ended picking HBase

## Now

- Personally rewritten large portions of HBase for 0.20
  - Code easy to work with, understand, modify
- Recently voted to committer status (thanks!)
- Now giving presentations (hi!)

## Four Point Agenda

- What is HBase?
- Why HBase?
- HBase 0.20
- HBase At Stumbleupon

## What is HBase?

- Clone of Bigtable - http://labs.google.com/papers/bigtable.html
- Created originally at Powerset in 2007
- Hadoop-subproject
  - The usual ASF things apply (license, JIRA, etc)

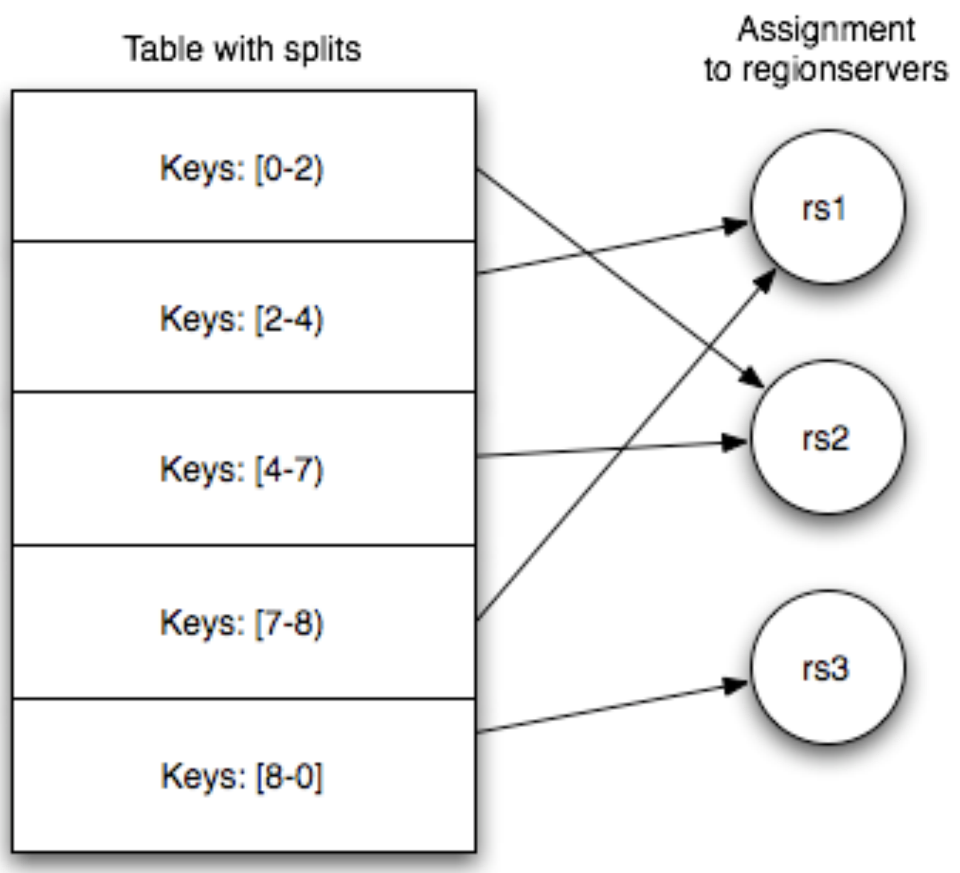
## What is HBase?

- Column-oriented semi-structured data store
- Distributed over many machines
  - Bigtable known to scale to >1000 nodes
- Tolerant of machine failure
- Layered over HDFS (&  KFS)
- Strong consistency (important)

## Table & Regions

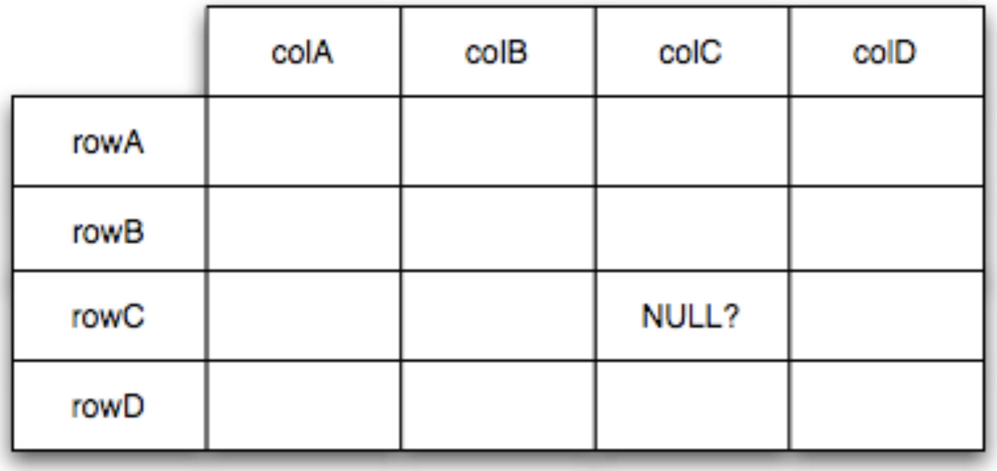
- Rows stored in byte-lexographic sorted order
- Table dynamically split into “regions”
- Each region contains values [startKey, endKey)
- Regions hosted on a regionserver

## Table & Regions



## Column Storage

- In HBase, don't think of a spreadsheet:

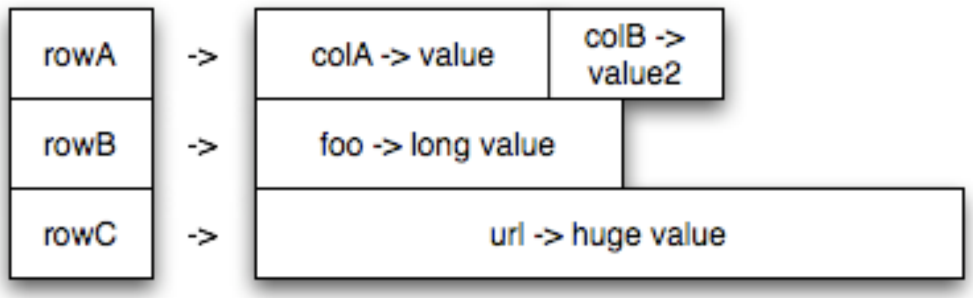


|  | colA | colB | colC | colD |
|---|---|---|---|---|
| rowA |  |  |  |  |
| rowB |  |  |  |  |
| rowC |  |  | NULL? |  |
| rowD |  |  |  |  |

All columns same 'size' and present (as NULL)

## Column Storage

- Instead think of tags. Values any length, no predefined names or widths:



Column names carry info (just like tags)

## Column Families

- Table consists of 1+ “column families”
- Column family is unit of performance tuning
- Stored in separate set of files
- Column names scoped like so:
  - “Family:qualifier”

## Sorting

- Rows stored in byte-lexographical order (row keys are raw bytes, not just strings)
- Furthermore within a row, columns stored in sorted order
- Fast, cheap easy to scan adjacent rows & columns

## Sorting (but there's more!)

- Not just scanning, but can do partial-key lookups
- When combined with compound keys, has the same properties as leading-left edge indexes in standard RDBMS
  - (Except your index is distributed of course)
- Can use a second table to index a primary table.

## Values

- Row id, column name, value all byte []
- Can store ascii, any binary or use serialization (eg: thrift, protobuf)

- Atomic increments available
- Serialization good for structs that are always read in one unit (eg: Address book entry)

## Values & Versions

- Each row id + column – stored with timestamp
- HBase stores multiple versions

- Can be useful to recover data due to bugs!
- Use to detect write conflicts/collisions

## API Example

```
Scan scan = new Scan(startRow,
   endRow).addFamily(“family”);
ResultScanner scanner = table.getScanner(scan);
Result result;
while ( (result=scanner.next()) != null) {
   Entity e = new Entity();
   dser.deserialize(e, result.getValue("default”, “0”);
}
scanner.close();
```

## Why HBase?

- Community is highly active, diverse, helpful
- User list Email activity for May: 78 threads
- IRC Channel #hbase highly active
- Helpful people in multiple timezones, email answered all hours of the day/night/weekend.

## Why HBase?

- Committer & contributor base broad:
  - PSet, Streamy, SU, Trend Micro, Openplaces, and more!
- No monopoly on experts – deep knowledge at these companies and more!

- (We're really friendly… honest!)

## Why HBase?

- Used in production at many companies
- 12 companies listed on http://wiki.apache.org/hadoop/Hbase/PoweredBy
- Openplaces, Streamy, SU serve websites out of HBase

- Lots of experience to draw upon!

## Why HBase? (Features)

- Full web management/monitoring UI (master & regionservers)
- Push metrics to log files & Ganglia
- Rolling upgrades possible! (Including master!)
- Non-SQL shell – re-enforces the non-SQL-ness of HBase

## HBase Features

- Easy integration with Hadoop MR – table input and output formats ship
- Cascading connectors for input and output
- Other ancillary open source activities around the edges (ORM, schema management, etc)

## Why HBase?

- But… HBase is slow!
- That metabrew/last.fm blog post said so!
  - (Also other people too…)

- “It’s much more than a KV store, but latency is too great to serve data to the website.”

- Answer: 0.20

## HBase 0.20

- Two major and exciting themes:

- #1:  Performance
- #2:  ZooKeeper integration, multiple masters

## HBase 0.20 vs 0.19

| | 0.19 | 0.20 |
|---|---|---|
| Master | Single master – if it fails, so does the cluster | Master election and membership via ZK |
| Compression | Not really | GZ, LZO |
| Memory usage | Small values cause big indexes and OOM | New file-format limits index size (800kB for 10m entries) |
| Scan Speed | 300-600ms per 500 rows | 20-30ms per 500 rows |

## Zookeeper?

- A highly available configuration storage system
- Set up in a 2N+1 quorum
- Hadoop subproject

## Master & Zookeeper

- Store membership info in ZK
- Detect dead servers (via ephemeral nodes)
- Master election and recovery

- Can kill master and cluster continues
- New master determines state and continues

## Performance

- Significant performance gains in 0.20
- New file format with 0-copy infrastructure
- Scan and get improvements
- LZO compression
- Block caching
- Speed increases as much as 30x!

## Performance

- 0.20 is not the final word on performance:
- Other RPC-related performance improvements
- Other Java-related improvements (G1?, 1.7?)

## Performance Numbers

- 1m rows, 1 column per row (~16 bytes)
  - Sequential insert: 24s, .024ms/row
  - Random read: 1.42ms/row (avg)
  - Full Scan: 11s, (117ms/10k rows)
- Performance under cache is very high:
  - 1ms to get single row
  - 20 ms to read 550 rows
  - 75ms to get 5500 rows

## HBase at Stumbleupon

- Strong commitment to HBase @ SU
- Supports a HBase committer
- Looking to hire more HBase hackers

## Big accomplishments @ SU

- Over 9b small rows in single table
  - Sustained import performance – 3-4 days to import 9b rows (mysql limiting speed)
- 1.2m row reads/sec on 19 nodes (!!)
  - That is 60-100k reads/sec/node sustained, 2hrs
  - Scalable with more nodes
  - HBase has been improved since then

## Fast accomplishments @ SU

- Extremely high speed increments and writes
- Supports su.pr analytics
- Su.pr reads from HBase with no intervening caches
- Integrated with PHP

## HBase & PHP @ SU

- PHP access via Thrift gateway
- Easy (PHP) deployment with Thrift
- App developers like soft-schema, easy querying and writing
- Want to use HBase for more features and applications!

## HBase deployment trivia

- Nodes are 8x16 w/2TB (best price point)
  - Don’t use RAID1. Use RAID0 or JBOD support
- Ganglia allows overall cluster performance monitoring
- Clusters won’t span datacenters
  - We want fully duplicate data for DR anyways
- Update master with code & config
  - Rsync to other nodes (1 dir, very easy)
  - Controlled restart for rolling upgrade

## HBase deployment trivia

- HDFS – set xciever limit to 2048, Xmx2000m
  - Never get HDFS problems even under heavy load
- For 9b row import, randomized key insert order gives substantial speedup
- Give HBase enough ram, you wouldn't starve mysql!
- Import speeds of 200k ops/sec on 19 machines possible!
  - Hard to provide a SQL-based source fast enough
  - 100k ops/sec typical for sustained

## HBase deployment trivia

- Consider dual writes or logs to get HBase up to date but without actually moving your data
- Duplicate data in indexes (already done in mysql)
- Have to think about read patterns when designing table key order!

## HBase future @ SU

- Latency sensitive cluster
- Batch/analytics cluster
- Use replication to keep latter up to date
- Allows batch jobs to go full throttle against reasonably up to date data without risking the website

## Q&A

- Questions?
- Stumbleupon is hiring awesome HBase hackers!