



Introduction to HBase

NYC Hadoop Meetup

Jonathan Gray

February 11, 2010

About Me

- Jonathan Gray
 - HBase Committer
 - HBase User since early 2008
 - Migrated large PostgreSQL instance to HBase
 - In production @ streamy.com since June 2008
 - Core contributor to performance improvements in HBase 0.20
 - Currently consulting around HBase
 - As well as Hadoop/MR and Lucene/Katta

Overview

- Why HBase?
- What is HBase?
- How does HBase work?
- HBase Today and Tomorrow
- HBase vs. RDBMS Example
- HBase and “NoSQL”

Why HBase?

- Same reasons we need Hadoop
 - Datasets growing into Terabytes and Petabytes
 - Scaling out is cheaper than scaling up
 - Continue to grow just by adding commodity nodes
- But sometimes Hadoop is not enough
 - Need to support random reads and random writes

Traditional databases are expensive to scale
and difficult to distribute

What is HBase?

- Distributed
- Column-Oriented
- Multi-Dimensional
- High-Availability
- High-Performance
- Storage System

Project Goal

Billions of Rows * Millions of Columns * Thousands of Versions

Petabytes across thousands of commodity servers

HBase is **not**...

- A Traditional SQL Database
 - No joins, no query engine, no types, no SQL
 - Transactions and secondary indexing possible but these are add-ons, not part of core HBase
- A drop-in replacement for your RDBMS
- You must be OK with RDBMS *anti-schema*
 - Denormalized data
 - Wide and sparsely populated tables

Just say “no” to your inner DBA

How does HBase work?

- Two types of HBase nodes:
 - Master and RegionServer**
- Master (one at a time)
 - Manages cluster operations
 - Assignment, load balancing, splitting
 - Not part of the read/write path
 - Highly available with ZooKeeper and standbys
- RegionServer (one or more)
 - Hosts tables; performs reads, buffers writes
 - Clients talk directly to them for reads/writes

HBase Tables

- An HBase cluster is made up of any number of user-defined tables
- Table schema only defines its **column families**
 - Each family consists of any number of columns
 - Each column consists of any number of versions
 - Columns only exist when inserted, NULLs are free
 - Everything except table/family names are byte[]
 - Rows in a table are sorted and stored sequentially
 - Columns in a family are sorted and stored sequentially

(Table, Row, Family, Column, Timestamp) → Value

HBase Table as Data Structures

- A table maps rows to its families
 - `SortedMap(Row → List(ColumnFamilies))`
- A family maps column names to versioned values
 - `SortedMap(Column → SortedMap(VersionedValues))`
- A column maps timestamps to values
 - `SortedMap(Timestamp → Value)`

An HBase table is a three-dimensional sorted map
(row, column, and timestamp)

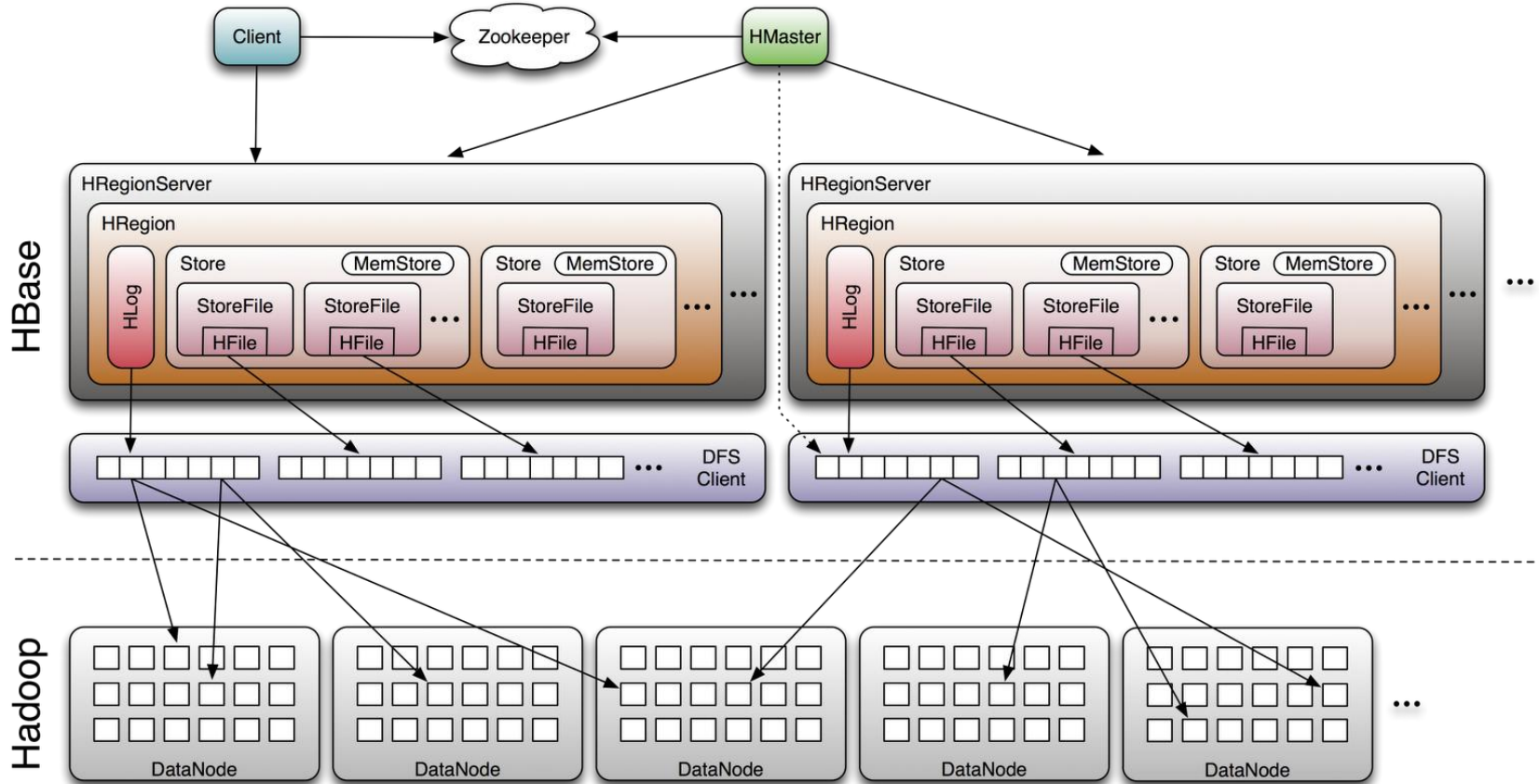
HBase Regions

- Table is made up of any number of **regions**
- Region is specified by its **startKey** and **endKey**
 - Empty table:
(Table, NULL, NULL)
 - Two-region table:
(Table, NULL, “MidKey”) and (Table, “MidKey”, NULL)
- A region only lives on one RegionServer at a time
- Each region may live on a different node and is made up of several HDFS files and blocks, each of which is replicated by Hadoop

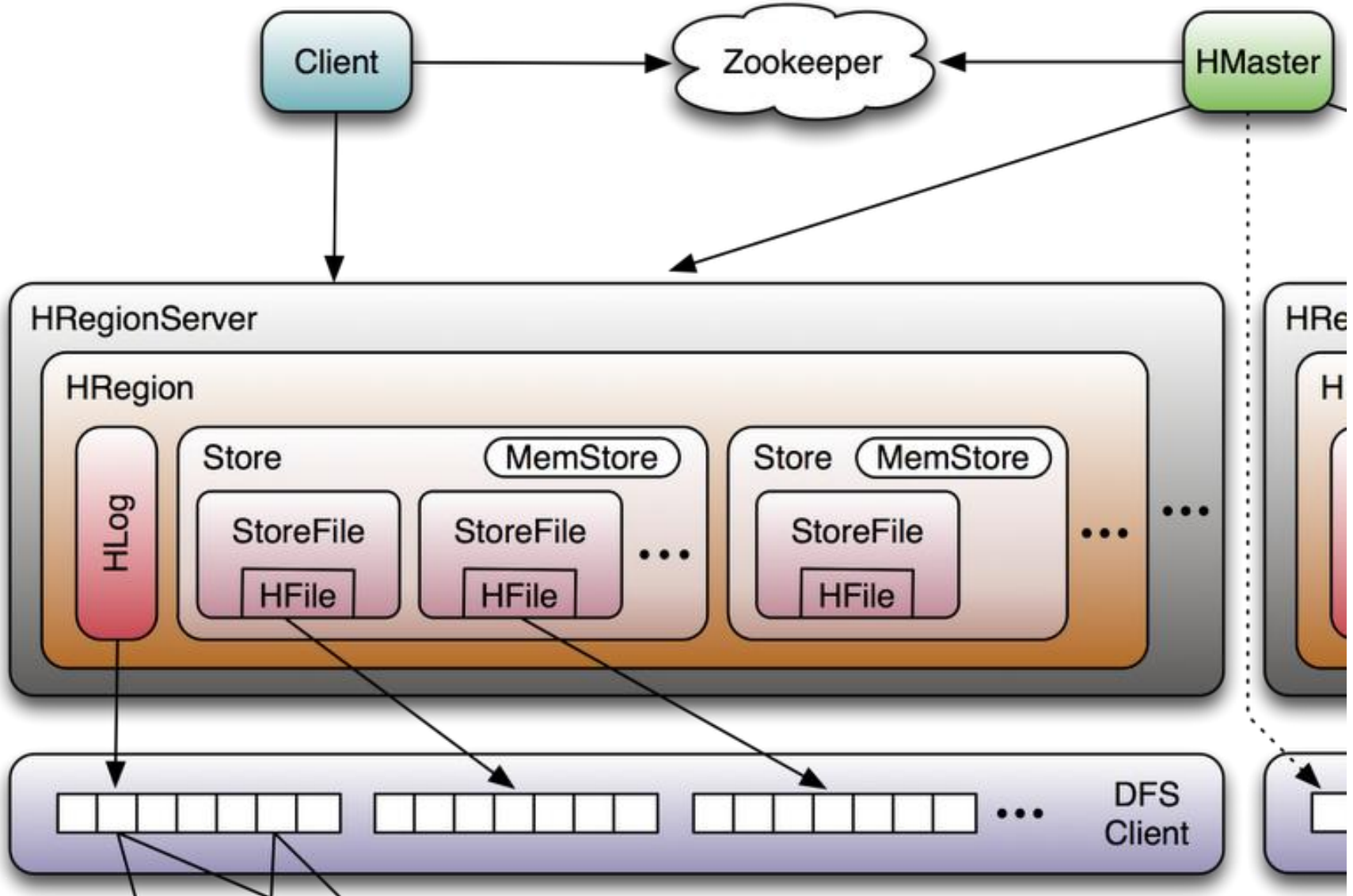
More HBase Architecture

- Region information and locations stored in special tables called catalog tables
 - ROOT**- table contains location of meta table
 - .META.** table contains schema/locations of user regions
- Location of -ROOT- is stored in ZooKeeper
 - This is the “bootstrap” location
- ZooKeeper is used for coordination / monitoring
 - Leader election to decide who is master
 - Ephemeral nodes to detect RegionServer node failures

System Architecture



System Architecture



HBase Key Features

- Automatic partitioning of data
 - As data grows, it is automatically split up
- Transparent distribution of data
 - Load is automatically balanced across nodes
- Tables are ordered by row, rows by column
 - Designed for efficient scanning (not just gets)
 - Composite keys allow ORDER BY / GROUP BY
- Server-side filters
- No SPOF because of ZooKeeper integration

HBase Key Features (cont)

- Fast adding/removing of nodes while online
 - Moving locations of data doesn't move data
- Supports creating/modifying tables online
 - Both table-level and family-level configuration parameters
- Close ties with Hadoop MapReduce
 - TableInputFormat/TableOutputFormat
 - HFileOutputFormat

Connecting to HBase

- Native Java Client/API
 - Get, Scan, Put, Delete classes
 - HTable for read/write, HBaseAdmin for admin stuff
- Non-Java Clients
 - Thrift server (Ruby, C++, PHP, etc)
 - REST server (stargate contrib)
- HBase Shell
 - Jruby shell supports put, delete, get, scan
 - Also supports administrative tasks
- TableInputFormat/TableOutputFormat

HBase Add-ons

- MapReduce / Cascading / Hive / Pig
 - Support for HBase as a data source or sink
- Transactional HBase
 - Distributed transactions using OCC
- Indexed HBase
 - Utilizes Transactional HBase for secondary indexing
- IHbase
 - New contrib for in-memory secondary indexes
- HBql
 - SQL syntax on top of HBase

HBase Today

- Latest stable release is **HBase 0.20.3**
 - Major improvement over HBase 0.19
 - Focus on performance improvement
 - Add ZooKeeper, remove SPOF
 - Expansion of in-memory and caching capabilities
 - Compatible with Hadoop 0.20.x
 - Recommend upgrading from earlier 0.20.x HBase releases as 0.20.3 includes some important fixes
 - Improves logging, shell, faster cluster ops, stability

HBase in Production

- Streamy
- StumbleUpon
- Adobe
- Meetup
- Ning
- Openplaces
- Powerset
- SocialMedia.com
- TrendMicro

The Future of HBase

- Next release is HBase 0.21.0
 - Release date will be ~1 month after Hadoop 0.21
- Data durability is fixed in this release
 - HDFS append/sync finally works in Hadoop 0.21
 - This is implemented and working on TRUNK
 - Have added group commit and knobs to adjust
- Other cool features
 - Inter-cluster replication
 - Master Rewrite
 - Parallel Puts
 - Co-processors

HBase Web Crawl Example

- Store web crawl data
 - Table **crawl** with family **content**
 - Row is URL with Columns
 - *content:data* stores raw crawled data
 - *content:language* stores http language header
 - *content:type* stores http content-type header
 - If processing raw data for hyperlinks and images, add families **links** and **images**
 - *links:<url>* column for each hyperlink
 - *images:<url>* column for each image

Web Crawl Example in RDBMS

- How would this look in a traditional DB?
 - Table **crawl** with columns **url**, **data**, **language**, and **type**
 - Table **links** with columns **url** and **link**
 - Table **images** with columns **url** and **image**
- How will this scale?
 - 10M documents w/ avg10 links and 10 images
 - 210M total rows versus 10M total rows
 - Index bloat with links/images tables

What is “NoSQL”?

- Has little to do with *not being SQL*
 - SQL is just a query language standard
 - HBql is an attempt to add SQL syntax to HBase
 - Millions are trained in SQL; resistance is futile!
 - Popularity of Hive and Pig over raw MapReduce
- Has more to do with anti-RDBMS architecture
 - Dropping the **relational** aspects
 - Loosening **ACID** and **transactional** elements

NoSQL Types and Projects

- Column-oriented
 - HBase, Cassandra, Hypertable
- Key/Value
 - BerkeleyDB, Tokyo, Memcache, Redis, SimpleDB
- Document
 - CouchDB, MongoDB
- Other differentiators as well...
 - Strong vs. Eventual consistency
 - Database replication vs. Filesystem replication

That's it! Questions?