Overview
Overview: History

- Began as project by Powerset to **process massive amounts of data** for natural language search

- Open-source implementation of Google’s **BigTable**
  - Lots of **semi-structured data**
  - Commodity Hardware
  - Horizontal Scalability
  - Tight integration with **MapReduce**

- Developed as part of Apache’s **Hadoop** project and runs on top of **HDFS (Hadoop Distributed Filesystem)**
  - Provides **fault-tolerant** way of storing **large quantities of sparse data**.
Overview: What is HBase?

- Non-relational, distributed database
- Column-Oriented
- Multi-Dimensional
- High Availability
- High Performance
Data Model & Operators
Data Model

- **A sparse, multi-dimensional, sorted** map
  - \{row, column, timestamp\} -> cell
- **Column = Column Family**: Column Qualifier

- Rows are **sorted lexicographically** based on row key
- **Region**: contiguous set of sorted rows
- **HBase**: a large number of columns, a low number of column families (2-3)
Operators

- Operations are based on **row keys**

- **Single-row operations:**
  - Put
  - Get
  - Scan

- **Multi-row operations:**
  - Scan
  - MultiPut

- No built-in joins (use MapReduce)
Physical Structures
Physical Structures: Data Organization

- **Region**: unit of distribution and availability
- Regions are split when grown too large
- Max region size is a tuning parameter
  - Too low: prevents parallel scalability
  - Too high: makes things slow

**Memstore**

- **HLog**
  - (Append only WAL on HDFS)
  - (Sequence File)
  - (one per RS)

- **HFile** (on HDFS)
- **HFile** (on HDFS)
- **HFile** (on HDFS)
Physical Structures: Need for Indexes

- HBase has **no built-in support for secondary indexes**
- API only exposes operations by **row key**

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Name</th>
<th>Position</th>
<th>Nationality</th>
</tr>
</thead>
<tbody>
<tr>
<td>“1”</td>
<td>Nowitzki, Dirk</td>
<td>PF</td>
<td>Germany</td>
</tr>
<tr>
<td>“2”</td>
<td>Kaman, Chris</td>
<td>C</td>
<td>Germany</td>
</tr>
<tr>
<td>“3”</td>
<td>Gasol, Paul</td>
<td>PF</td>
<td>Spain</td>
</tr>
<tr>
<td>“4”</td>
<td>Fernandez, Rudy</td>
<td>SG</td>
<td>Spain</td>
</tr>
</tbody>
</table>

**Find all players from Spain?**
- With built-in API, scan the entire table
- Manually build a secondary index table
- Exploit the fact that rows are sorted lexicographically by row key based on byte order
Physical Structures: Secondary Index

- **Data Table:**

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Name</th>
<th>Position</th>
<th>Nationality</th>
</tr>
</thead>
<tbody>
<tr>
<td>“1”</td>
<td>Nowitzki, Dirk</td>
<td>PF</td>
<td>Germany</td>
</tr>
<tr>
<td>“2”</td>
<td>Kaman, Chris</td>
<td>C</td>
<td>Germany</td>
</tr>
<tr>
<td>“3”</td>
<td>Gasol, Paul</td>
<td>PF</td>
<td>Spain</td>
</tr>
<tr>
<td>“4”</td>
<td>Fernandez, Rudy</td>
<td>SG</td>
<td>Spain</td>
</tr>
</tbody>
</table>

- **Index table on nationality column**
  - a scan operation
  - start row = "Spain"
  - stop scanning: set a RowFilter with a BinaryPrefixComparator on the end value("Spain")
  - range queries are also supported

- **Index table on Row Key**

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Dummy</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Germany 1”</td>
<td>Germany 1</td>
</tr>
<tr>
<td>“Germany 2”</td>
<td>Germany 2</td>
</tr>
<tr>
<td>“Spain 3”</td>
<td>Spain 3</td>
</tr>
<tr>
<td>“Spain 4”</td>
<td>Spain 4</td>
</tr>
</tbody>
</table>
Physical Structures: Secondary Index (cont.)

- Find all power forwards from Spain?
  - A composite index

- Row keys are **plain byte arrays**
  - Byte order = your desired order?
  - Convert strings, integers, floats, decimals carefully to bytes
  - Default sorting is ascending; if descending indexes are needed, reverse bit order
**Physical Structures: More Indexing**

- **Lily’s HBase Indexing Library**
  - Aids in building and querying indexes in HBase
  - Hides the details of playing with byte[] row keys

- **HBase + full text indexing and searching systems**
  - Apache Lucene (Apache Solr, elasticsearch)
  - Lily, HAvroBase (HBase + Solr), HBasene (HBase + Lucene)
System Architecture
System Architecture: Overview
System Architecture: Write-Ahead-Log Flow
System Architecture: HFile and KeyValue
APIs
APIs: Overview

- **Java**
  - Get, Put, Delete, Scan
  - IncrementColumnValue
  - TableInputFormat - MapReduce Source
  - TableOutputFormat - MapReduce Sink

- Rest
- Thrift
- Scala
- Jython
- Groovy DSL
- Ruby shell
- Java MR, Cascading, Pig, Hive
ACID Properties
ACID Properties

- HBase **not ACID-compliant**, but does guarantee certain specific properties

**Atomicity**
- All mutations are atomic within a row. Any put will either wholly succeed or wholly fail.
- APIs that mutate several rows will *not* be atomic across the multiple rows.
- The order of mutations is seen to happen in a well-defined order for each row, with no interleaving.

**Consistency and Isolation**
- All rows returned via any access API will consist of a complete row that existed at some point in the table's history.
**Consistency of Scans**
- A scan is not a consistent view of a table. Scans do not exhibit snapshot isolation.
- Those familiar with relational databases will recognize this isolation level as "read committed".

**Durability**
- All visible data is also durable data. That is to say, a read will never return data that has not been made durable on disk.
- Any operation that returns a "success" code (e.g. does not throw an exception) will be made durable.
- Any operation that returns a "failure" code will not be made durable (subject to the Atomicity guarantees above).
- All reasonable failure scenarios will not affect any of the listed ACID guarantees.
Users
Users: Just to name a few…

facebook

twitter

mozilla

Adobe

Meetup

Trend Micro

NING

Su.pr

Yahoo!
The New Messages

Texts, chat and email together in one simple conversation.

All your messages together

Get Facebook messages, chats and texts all in the same place.

- Include email by activating your optional Facebook email address
- Control who can send you messages through your privacy settings

Full conversation history

See everything you’ve ever discussed with each friend as a single conversation.

- No need for subject lines or other formalities
- Easily leave large conversations that no longer interest you

Kate
awesome

Drew
k cya
May 23

Drew
Around?
June 18

Kate
yeah

The messages you want

Focus on messages from your friends.

- Messages from unknown senders and bulk email go into the Other folder
- Spam is hidden from view automatically

For more information, read the top questions about the new Messages.

Request an Invitation
- **Previous Solution:** Cassandra
- **Current Solution:** HBase
- **Why?** Cassandra's replication behavior
Results for: nosql

**nosqlupdate**  NoSQL Update
So you want to keep yourself updated on the #NoSQL movement? Just start following!

**al3xandru**  Alex Popescu
NOSQL Dreamer
http://nosql.mypopescu.com, Software architect, Founder/CTO InfoQ.com, Web aficionado, Speaker,

**spyced**  Jonathan Ellis
Riptano co-founder and project chair for Apache Cassandra. At Mozy, I built a multi-petabyte, scalable storage system based on Reed-Solomon encoding.

**cassandra**  Cassandra Database
The Cassandra distributed database combines the replication model of Amazon’s Dynamo with the data model of Google’s BigTable

**CouchDB**  CouchDB
HTTP + JSON Document Database with Map Reduce views and peer-based Replication

**DataStax**  DataStax
DataStax is the commercial leader in Apache Cassandra™, and helps customers build and operate massively scalable, cloud-optimized applications and data services
- Customer Indexing

- **Previous Solution:** offline process at a single node

- **Current Solution:**
  - Import user data into HBase
  - Periodically MapReduce job reading from HBase
  - Hits FlockDB and other internal services in mapper
  - Write data to sharded, replicated, horizontally scalable, in-memory, low-latency Scala service

- **Vs. Others:**
  - HDFS: Data is mutable
  - Cassandra: OLTP vs. OLAP?
Users: Mozilla - Socorro

Firefox Crash Data
3 days 7 days 14 days 28 days

Crashes per Active Daily User

Firefox Top Crashers

Firefox 4.0/15/16
no signature

mozglk: :acm: lck: const*: const::NEQ_DebugBreak: P: 1/Port/MDMainThrea
debug (no signature)

Firefox 4.0
mozglk: :acm: lck: const*: const::NEQ_DebugBreak: P: 1/Port/MDMainThrea
debug (no signature)

Firefox 3.6.15
getUserMediaProc churn: window

Mozilla - Socorro

Mozilla Crash Reports - Powered by Socorro
Server Status | Project Info | Source Code | Breakpad Wiki | Privacy Policy

Log In
Users: Mozilla – Socorro (cont.)

- **Socorro**, Mozilla’s crash reporting system (https://crash-stats.mozilla.com/products)
  - Catches, processes, and presents crash data for Firefox, Thunderbird, Fennec, Camino, and Seamonkey.

- 2.5 million crash reports per week, 320GB per day

- **Previous Solution**: NFS (raw data), PostgreSQL (analyze results)
  - 15% of crash reports are processed

- **Current Solution**: Hadoop (processing) + HBase (storage)
HBase vs. RDBMS
<table>
<thead>
<tr>
<th>HBase</th>
<th>RDBMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column-oriented</td>
<td>Row oriented (mostly)</td>
</tr>
<tr>
<td>Flexible schema, add columns on the fly</td>
<td>Fixed schema</td>
</tr>
<tr>
<td>Good with sparse tables</td>
<td>Not optimized for sparse tables</td>
</tr>
<tr>
<td>No query language</td>
<td>SQL</td>
</tr>
<tr>
<td>Wide tables</td>
<td>Narrow tables</td>
</tr>
<tr>
<td>Joins using MR – not optimized</td>
<td>Optimized for joins (small, fast ones too!)</td>
</tr>
<tr>
<td>Tight integration with MR</td>
<td>Not really…</td>
</tr>
</tbody>
</table>
HBase vs. RDBMS (cont.)

<table>
<thead>
<tr>
<th>HBase</th>
<th>RDBMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>De-normalize your data</td>
<td>Normalize as you can</td>
</tr>
<tr>
<td>Horizontal scalability – just add hardware</td>
<td>Hard to shard and scale</td>
</tr>
<tr>
<td>Consistent</td>
<td>Consistent</td>
</tr>
<tr>
<td>No transactions</td>
<td>Transactional</td>
</tr>
<tr>
<td>Good for semi-structured data as well as structured data</td>
<td>Good for structured data</td>
</tr>
</tbody>
</table>
Questions?
Thanks!