&
sqrrl
Secure. Scale. Adapt

Adam Fuchs, CTO
sqrrl data, INC.
February 21, 2013
Today’s Talk

1. An overview of what matters for big data analysis.
3. Approaches to sorted key-value table design.
4. An overview of Sqrrl Enterprise.
The Value of Sqrrl and Accumulo

- Security
- Scalability
- Adaptivity

Sqrrl Enterprise and Apache Accumulo
Google publishes Bigtable paper

NSA open sources Accumulo

Accumulo becomes a top-level Apache project

First sqrrl release planned

2005  2006  2007  2008  2009  2010  2011  2012  2013

Google Publishes Papers:
  GFS (2003)
  Map Reduce (2004)

NSA begins development of Accumulo

sqrrl is founded
Apache Accumulo

- Sorted, Distributed Key/Value Store
- Based on Google’s Big Table Design
- Built on Top of Apache Hadoop and Apache Zookeeper
- Augments and Integrates With the Hadoop ecosystem
- Originally developed at the National Security Agency, now an Apache Software Foundation project
Accumulo’s Strengths

Apache Accumulo excels at:

• **Security**
  – Cell-level security reduces the cost of application development in the presence of complex legal or policy restrictions on data use
  – Identity and access management and encryption tie into enterprise security standards

• **Scalability**
  – Proven reliability and performance at the multi-petabyte scale
  – High-performance parallel I/O library

• **Adaptivity**
  – Flexible schema support to quickly ingest new data sources
  – Sorted key/value paradigm supports a multitude of search and analysis applications
  – Server-side programming framework “iterator trees” support best-in-class aggregation, filtering, and complex query semantics
Big Data Lessons Learned

• **Start small, but design for scalability**
  – One application first, then grow to hundreds
  – One gigabyte first, then grow to petabytes

• **Iterative schema refinement**
  – Initially, let the data define the schema
  – Refine the schema in bulk as you better understand the data
  – Middle ground between flat files and complete ontologies

• **Discovery analytics as application building blocks**
  – Universal search: structured and unstructured data, low latency
  – Basic statistics: aggregations of query results, parallelized, low latency, to support big picture analysis
  – Graphs: scalable graph analytics for analyzing how everything is connected

• **Data-centric security**
  – Separate modeling of security and analysis
  – Simplifies multi-tenancy and application accreditation
Definition: A form of security in which data carries with it the elements of provenance that are required to make policy decisions on its releasability.

- Separate data modeling for Security and Analysis
- Reusability of applications across security domains
- Distributed development of ingest and query applications
- Supported by Accumulo’s cell-level security
Security Policy Evaluation

Traditional Security

- Original Records
  - Insert and Model Security
- Transformed Records
  - Transform Data and Model Security
- Query Mechanism
  - Filters 1000s Per Second
  - Access Checks
- Enterprise Security Server
  - Unfiltered
  - Filtered
Data-Centric Security

- Insert and Model Security
- Transform and Preserve Labels
- Filter Billions Per Second

Original Records
Transformed Records
Query Mechanism
Access Checks
Enterprise Security Server

Secure. Scale. Adapt.
Visibility Syntax & Semantics

Document Labels

\[ \begin{align*}
\text{Doc}_1 & : \text{(Federation)} \\
\text{Doc}_2 & : \text{(Klingon|Vulcan)} \\
\text{Doc}_3 & : \text{(Federation\&Human\&Vulcan)} \\
\text{Doc}_4 & : \text{(Federation\&(Human|Vulcan))}
\end{align*} \]

User Authorization Sets

\[ \begin{align*}
\text{CptKirk} & : \{\text{Federation, Human}\} \\
\text{MrSpock} & : \{\text{Federation, Human, Vulcan}\}
\end{align*} \]

Semantics

\[ \begin{align*}
\frac{(T \Rightarrow \tau) \land (\tau \in A)}{(T, A) \models \text{true}} & \quad \text{term} \\
\frac{(T \Rightarrow T_1 \& T_2) \land ((T_1, A) \models \text{true}) \land ((T_2, A) \models \text{true})}{(T, A) \models \text{true}} & \quad \text{and} \\
\frac{(T \Rightarrow T_1 \mid T_2) \land (((T_1, A) \models \text{true}) \lor ((T_2, A) \models \text{true}))}{(T, A) \models \text{true}} & \quad \text{or} \\
\frac{(T \Rightarrow (T_1)) \land (T_1 \models \text{true})}{(T, A) \models \text{true}} & \quad \text{paren}
\end{align*} \]

Syntax

\[ \begin{align*}
\text{WORD} & \Rightarrow [a-zA-Z0-9_] + \\
\text{CLAUSE} & \Rightarrow \text{AND} \\
& \quad \Rightarrow \text{OR} \\
\text{AND} & \Rightarrow \text{AND \& AND} \\
& \quad \Rightarrow (\text{CLAUSE}) \\
\text{OR} & \Rightarrow \text{OR | OR} \\
& \quad \Rightarrow (\text{CLAUSE}) \\
& \quad \Rightarrow \text{WORD}
\end{align*} \]
Accumulo stores sorted key/value pairs (entries).

An Accumulo key is a 5-tuple, consisting of:

- **Row**: Controls Atomicity
- **Column Family**: Controls Locality
- **Column Qualifier**: Controls Uniqueness
- **Visibility Label**: Controls Access
- **Timestamp**: Controls Versioning

Keys are sorted:

- **Hierarchically**: Row first, then column family, and so on.
- **Lexicographically**: Compare first byte, then second, and so on.

Values are byte arrays.
### Key/Value Examples

<table>
<thead>
<tr>
<th>Row</th>
<th>Col. Fam.</th>
<th>Col. Qual.</th>
<th>Visibility</th>
<th>Timestamp</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane Doe</td>
<td>Friends</td>
<td>John Doe</td>
<td>JD</td>
<td>20121130</td>
<td></td>
</tr>
<tr>
<td>Jane Doe</td>
<td>Phone Number</td>
<td>555-1212</td>
<td></td>
<td>20090115</td>
<td></td>
</tr>
<tr>
<td>John Doe</td>
<td>Friends</td>
<td>Jane Doe</td>
<td>JD</td>
<td>20121201</td>
<td></td>
</tr>
<tr>
<td>John Doe</td>
<td>Notes</td>
<td>PCP</td>
<td>PCP_JD</td>
<td>20120912</td>
<td>Patient suffers from an acute ...</td>
</tr>
<tr>
<td>John Doe</td>
<td>Test Results</td>
<td>Cholesterol</td>
<td>JD</td>
<td>PCP_JD</td>
<td>20120912</td>
</tr>
<tr>
<td>John Doe</td>
<td>Test Results</td>
<td>Mental Health</td>
<td>JD</td>
<td>PSYCH_JD</td>
<td>20120801</td>
</tr>
<tr>
<td>John Doe</td>
<td>Test Results</td>
<td>Mental Health</td>
<td>PSYCH_JD</td>
<td>20120801</td>
<td>Crazy!</td>
</tr>
<tr>
<td>John Doe</td>
<td>Test Results</td>
<td>X-Ray</td>
<td>JD</td>
<td>PHYS_JD</td>
<td>20120513</td>
</tr>
</tbody>
</table>
Well-Known Location (zookeeper)

Root Tablet
-∞ to ∞

Metadata Tablet 1
-∞ to “Encyclopedia:Ocelot”

Metadata Tablet 2
“Encyclopedia:Ocelot” to ∞

Collections of entries form Tables

Tables are partitioned into Tablets

Metadata tablets hold info about other tablets, forming a 3-level hierarchy

A Tablet is a unit of work for a Tablet Server

Table: Adam’s Table

Data Tablet
-∞ : thing
Data Tablet
thing : ∞

Table: Encyclopedia

Data Tablet
-∞ : Ocelot
Data Tablet
Ocelot : Yak
Data Tablet
Yak : ∞

Table: Foo

Data Tablet
-∞ to ∞
Iterator Framework

Iterator Operations:
- File Reads
- Block Caching
- Merging
- Deletion
- Isolation
- Locality Groups
- Range Selection
- Column Selection
- Cell-level Security
- Versioning
- Filtering
- Aggregation
- Partitioned Joins
Table Design

- No built-in secondary indices
- Sort Order ↔ Index
- Balance between ingest and query
- Avoid introducing bottlenecks
- Preserve cell-level security and scalability
# Forward and Inverted Indexes

<table>
<thead>
<tr>
<th>Table:</th>
<th>Forward Index</th>
<th>Inverted Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row:</td>
<td>&lt;UUID&gt;</td>
<td>&lt;Term&gt;</td>
</tr>
<tr>
<td>Column Family:</td>
<td>&lt;Type&gt;</td>
<td>&lt;Type&gt;</td>
</tr>
<tr>
<td>Column Qualifier:</td>
<td>&lt;Field&gt;</td>
<td>&lt;UUID&gt; + &lt;Field&gt;</td>
</tr>
<tr>
<td>Value:</td>
<td>&lt;Term&gt;</td>
<td>&lt;Digest of Event&gt;</td>
</tr>
</tbody>
</table>
Forward and Inverted Indexes

Ingest Process

- Map Terms to UUID
- Map UUID to Event

Query Process

- (opt.) Fetch Events by UUID
- Find UUIDs and Digests for Term

Event

Inverted Index

Term
sqrrl extends Accumulo with:

• Discovery analytics
  – Real-time search across structured and unstructured data
  – Graph analysis primitives to link across datasets
  – Distributed computation of ad-hoc statistics
  – Online statistics indexing for real-time report generation
  – Custom indexing to support your applications

• Enterprise integration
  – Support for standard query languages, including Lucene and a subset of SQL
  – I/O support for hierarchical documents, including JSON
  – Identity and Access Management with existing Kerberos, AD, LDAP, and other installations
  – Polyglot service layer, including Python, Ruby, C, and other languages
sqrrl analytics Architecture

- Applications
  - Analytics APIs
  - Security & Access Controls
  - Data Integration

- Accumulo
  - Search, Statistics, Graph
  - Geospatial, SQL, Machine Learning, Custom Extensions

- IAM, Encryption, DAM, Secure Code

- ETL, Hadoop
Adaptability

• Flexible Schema:
  – Generic hierarchical structured/unstructured document store (JSON, XML, etc.)
  – Graph store (entities, relationships)
  – Schema stats for iterative refinement

• Core Discovery Analytics:
  – Structured/unstructured text search (Lucene)
  – Big-Picture Analysis, Aggregation (SQL)
  – Graph search (frontier expansion)
  – Additional scalable indexes (Geo, Image, Video ...
Lightweight Apps

The future of Big Data innovation is Apps, built on:

- Universal Search
- Schema-less Statistics
- Graphs
- Intuitive Languages
- Secure, Scalable, and Adaptable platforms
Analytics Demo

SELECT count(uuid) WHERE 'love' GROUP BY doc('user/followers_count'),doc('user/friends_count') LIMIT 500

copyright sqrrl data INC 2013
Contact

sqrrl data, Inc.
275 Third St.
Cambridge, MA 02142

(617) 902-0784
www.sqrrl.com
@sqrrl_inc
info@sqrrl.com