BlueRunner: Building an Email Service in the Cloud

Jun Rao
IBM Almaden Research Center

Apache Cassandra Committer
The Team

User Systems and Experience Research

Jerald Schoudt
Stefan Nusser
Jan Pieper

Information Management Research

Sandeep Tata
Bo Shekita
Jun Rao

Hernan Badenes
Julian Cerruti
Outline

- BlueRunner overview
- Scalable Row Stores
- BlueRunner design in Cassandra
- Preliminary performance results
- Summary
What’s BlueRunner

- Research prototype for hosted emails at IBM ARC
- Browser-based email client + Cassandra backend
  - 3+ years on the client
  - ~1 year on the Cassandra backend
- Many advanced features in client
  - Scrolling
  - Foldering/Tagging
  - Sorting/Pivoting
  - Threading
  - Orienteering and Usability Improvements
- Backend
  - Thin client; most operations pushed to the backend
  - Designed for large-scale hosted environment
    - 100K mailboxes, each with 100K messages
How does the client look?
Why a new backend?

<table>
<thead>
<tr>
<th>Limitations in Traditional DBs</th>
<th>limiting scalability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fault-tolerance relies on expensive reliable storage</td>
<td></td>
</tr>
<tr>
<td>Weak elasticity—hard to grow a cluster incrementally</td>
<td></td>
</tr>
<tr>
<td>No automatic load-balancing</td>
<td></td>
</tr>
<tr>
<td>Rigid relational schema</td>
<td></td>
</tr>
<tr>
<td>No versioning support</td>
<td>mismatch for many apps</td>
</tr>
</tbody>
</table>

limiting scalability

every app
The Cloud Landscape for Scalable Backend

- Voldemort
- BigTable
- Cassandra
- Facebook
- HBase
- MongoDB
- Drizzle
- Amazon SimpleDB™

**Flurry of activity in this space motivated by**

- RDBMS too rigid/heavy for some apps
- Existing RDBMS engines missing many key cloud requirements
What is a Scalable Row Store?

- Middle ground btw a DBMS and a file system
  - Much simpler API then SQL
  - Designed to scale

<table>
<thead>
<tr>
<th>Limitations in Traditional DBs</th>
<th>Scalable Row Stores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fault-tolerance relies on expensive reliable storage</td>
<td>Fault-tolerance done in software; replication on commodity disks</td>
</tr>
<tr>
<td>Weak elasticity--- hard to grow a cluster incrementally</td>
<td>Can grow a cluster incrementally and online</td>
</tr>
<tr>
<td>No automatic load-balancing</td>
<td>Built-in automatic load-balancing</td>
</tr>
<tr>
<td>Rigid relational schema</td>
<td>No strong schema required</td>
</tr>
<tr>
<td>No versioning support</td>
<td>Built-in versioning</td>
</tr>
</tbody>
</table>
Cassandra

- Google’s Bigtable data model + Amazon’s Dynamo scalable architecture
- Developed by Facebook in 2007
  - Used in production for a few apps (e.g., inbox search for 200M users)
- Became an Apache Incubator project early 2009
  - active community
  - additional committers from Rackspace and IBM
  - contributors from Digg, Twitter, etc
Cassandra Data Model

- Familiar relational tables, rows, and columns, but more flexible
  - No upfront schema required
  - New columns can be added any time and columns can vary from row to row

<table>
<thead>
<tr>
<th>row key</th>
<th>col name</th>
<th>col value</th>
</tr>
</thead>
<tbody>
<tr>
<td>row 1</td>
<td>k127</td>
<td>type: capacitor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>farads: 12mf</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cost: $1.05</td>
</tr>
<tr>
<td>row 2</td>
<td>k187</td>
<td>type: resistor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ohms: 8k</td>
</tr>
<tr>
<td></td>
<td></td>
<td>label: banded</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cost: $.25</td>
</tr>
<tr>
<td>row 3</td>
<td>k217</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

- Columns grouped into Column Families
  - Column families are stored separately (like vertical partitioning)
Cassandra Distributed Architecture

- Based on DHT ring
- Replication cross-rack and cross-datacenter (sync or async)
- No single point of failure
- Gossip protocols for membership, failure, DHT map, etc

Insert (row1 ...)

Scale, Fault-tolerance, Elasticity, Low-cost
Eventual Consistency

- **CAP Theorem** [Brewer00]
  - Can only get 2 of **C**onsistency, **A**vailability, or **P**artition tolerance
- Cassandra relaxes **C** to eventual consistency
  - Emphasis is on performance and availability
  - Allow concurrent read/write to any replica – latest write wins on conflict
- **Knobs to tradeoff consistency and performance**
  - Writes are sent to all N replicas in parallel
  - Can choose to read from **R** replicas and wait for **W** acks for writes
  - Tune R and W to 1,2,3...,N for latency requirements
Email Schema in Cassandra

- **Row key - User id**
  - All data for one user is on a single node
  - Currently no sharing of messages across users

- **Column Families**
  - MailList - Message id : full message
  - HeaderList - Message id : message headers + metadata
  - CollectionIndex - Collection id + sort key : message id
    
    e.g., Inbox/Date/2009-07-10-14:20:56 : message1000
    Inbox/Sender/Mike Brown : message1000
    
  - Others
    - CollectionMetadata, ThreadList, ThreadIndex

- **Full message stored separately from index and metadata**

- **Data format - JSON**
Typical Operations

- **Cassandra APIs used**
  - `get_column(row, CF, column)`
  - `get_columns(row, CF, columns[])`
  - `get_slice(row, CF, startColumn, asc/desc, count)`
  - Efficient with row/column index support in Cassandra

- **ListMessages**
  - `get_slice(Jun, CollectionIndex, Inbox/Date/current_date/, desc, 50)` to obtain the first 50 messageIDs in Inbox
  - `get_columns(Jun, HeaderList, messageID[])`

- **GetMessage**
  - `get_column(Jun, MailList, messageID)`

- **SortMessages by Sender**
  - `get_slice(Jun, CollectionIndex, Inbox/Sender//, asc, 50)`
  - `get_columns(Jun, HeaderList, messageID[])`
BlueRunner Deployment

Mail Client

WebServer (Tomcat)

HTTP/REST

Thrift

MailStore

Cassandra
Experimental Setup

- 6-node cluster and each node with
  - 2 quad-core CPUs
  - 16 GB memory
  - 5 SATA disks (1 for Cassandra commit log and 4 for data)

- Data
  - generated 1800 mailboxes per node
    - 250-16K messages per mailbox
    - ~50GB w/o replication
  - Cassandra replication set to 2

- Workload
  - Varying # of concurrent clients from 1600 to 4100 per node
  - Each client repeatedly
    - opens up an inbox
    - looks at a few message
    - go to sleep for a while
Preliminary Result (median response time in ms)

- Able to sustain 2500 concurrent clients per node with reasonable response time
  - average: ~100 requests per sec per node
Summary

- Cassandra-based backend very promising
  - Enabling scalability, availability, and elasticity
  - Flexible data model a good fit

- Future work
  - Many places can be improved
    - Alternative schema design
    - Secondary index/full-text index support
    - Enabling MapReduce-based analytics on the backend
  - Other potential collaborative apps on Cassandra
  - Research on better reasoning btw consistency and availability