High Scale Relational Storage at Salesforce built with Apache HBase and Apache Phoenix

Andrew Purtell
Architect, Cloud Storage
apurtell@salesforce.com
@akpurtell
Safe Harbor

Safe harbor statement under the Private Securities Litigation Reform Act of 1995:

This presentation may contain forward-looking statements that involve risks, uncertainties, and assumptions. If any such uncertainties materialize or if any of the assumptions proves incorrect, the results of salesforce.com, inc. could differ materially from the results expressed or implied by the forward-looking statements we make. All statements other than statements of historical fact could be deemed forward-looking, including any projections of product or service availability, subscriber growth, earnings, revenues, or other financial items and any statements regarding strategies or plans of management for future operations, statements of belief, any statements concerning new, planned, or upgraded services or technology developments and customer contracts or use of our services.

The risks and uncertainties referred to above include – but are not limited to – risks associated with developing and delivering new functionality for our service, new products and services, our new business model, our past operating losses, possible fluctuations in our operating results and rate of growth, interruptions or delays in our Web hosting, breach of our security measures, the outcome of any litigation, risks associated with completed and any possible mergers and acquisitions, the immature market in which we operate, our relatively limited operating history, our ability to expand, retain, and motivate our employees and manage our growth, new releases of our service and successful customer deployment, our limited history reselling non-salesforce.com products, and utilization and selling to larger enterprise customers. Further information on potential factors that could affect the financial results of salesforce.com, inc. is included in our annual report on Form 10-K for the most recent fiscal year and in our quarterly report on Form 10-Q for the most recent fiscal quarter. These documents and others containing important disclosures are available on the SEC Filings section of the Investor Information section of our Web site.

Any unreleased services or features referenced in this or other presentations, press releases or public statements are not currently available and may not be delivered on time or at all. Customers who purchase our services should make the purchase decisions based upon features that are currently available. Salesforce.com, inc. assumes no obligation and does not intend to update these forward-looking statements.
whoami

Architect, Cloud Storage at Salesforce.com
• Data Platform Team

Open Source Contributor, since 2007
• Committer, PMC, and Project Chair, Apache HBase
• Committer and PMC, Apache Phoenix
• Committer, PMC, and Project Chair, Apache Bigtop
• Member, Apache Software Foundation

Distributed Systems Nerd, since 1997
Agenda

Motivation
Open Source and the Data Platform Team
What are Apache HBase and Apache Phoenix?
HBase@Salesforce
  • The View from 30Kft
  • Keeping Up Appearances
  • Engineering The Whole Stack Holistically
Q&A
Motivation
The Data Management Challenge

Scale data requires a single platform for analysis ("data gravity")

Salesforce already manages over 100B records of customer data today

More than 3 billion transactions per day on the platform

Exponential growth rate

Batch and stream compute data locality requirements

Service continuity requirements
BigObjects

Systems of record provide reliable, highly available, and secure data storage
- SObjects: Traditional Salesforce platform objects
- Salesforce Files: Blob storage
- BigObjects: Scale out storage for immutable data

Systems of Record
- SObjects: Traditional Salesforce platform objects
  - Transactional data. Rows can be added and updated
  - Example: Accounts, Contacts, Custom Objects
- Salesforce Files: Blob storage
  - Blob storage for *semi- or un-structured* data
  - Example: CSV extracts from external systems, weblogs, monitoring logs
- BigObjects: Scale out storage for immutable data
  - New Object type for immutable data.
  - Optimized for large volumes of data
  - Example: event data, purchase history, product usage data

Platform Connect
- External Object: New proxy object connected to an external oData source
Data Pipelines

- Snapshot data from SObjects / External Objects

- Manipulate and analyze data sets
  - Transformations
  - Joins
  - Calculations and enrichment
Data Pipelines

Snapshot data from SObjects / External Objects

Manipulate and analyze data sets
- Transformations
- Joins
- Calculations and enrichment

Apache Pig + Hadoop for processing framework and scripting language

Apache HBase for BigObjects persistence

Apache Phoenix for indexing and relational access

Visual Diagram:
- Data Snapshot
- External Objects
- Salesforce Files
- Data Pipeline
- Apache Phoenix
- BigObjects
- Apache HBase

Legend:
- Database
- BigObject for Salesforce
- External Object
Data Management Services

Customer 360
- Enrich customer profile w/ data from external systems

Data Retention
- Keep data for audit purposes
- Maintain scalability of operational systems

Data Quality
- Data enrichment
- Data integration
- Data cleansing

Connected Products
- Event stream archive
- Offline analysis and data mining

Audit & Tracking
- Track what users are doing on platform
- Compliance
Persist the **complete field change history for up to ten years** with Field Audit Trail

Persist user event capture for Event Monitoring and Audit

Support scale out batch and stream compute for Backup, DR, Threat Detection, etc.
Open Source and the Data Platform Team
How Open Source Helps Salesforce

The things we build are better
• Building software with Open Source in mind is inherently beneficial (encourages loose coupling, cohesiveness, quality)
• Positive external pressure to make the right software engineering decisions

The things we don't build are better
• The only thing better than writing a great component is not writing it
• Smart engineers avoid “Not-Invented-Here” syndrome

We are happier
• Open Source extends pride in ownership and visibility beyond the company walls
• We attract better coders who gravitate towards companies that do open source work

We make the world a better place
How We Contribute To And Use Open Source

No forking

• Fork defined as: a departure from the open source repository significant enough to prevent us from contributing patches back or applying patches from the upstream open source repository

Internal repositories as change buffer with local release numbering

• Local repos for HBase, Phoenix, all dependencies (Hadoop, ZooKeeper, etc.)
• Fast forwarded to new upstream releases after consideration
• Updates to local repos are all manual by design
• Hadoop is a special snowflake

Change the upstream repositories first

• Almost all changes begin as patches developed using upstream repositories
• Only critical bug fixes are an exception, or where we require a change locally meantime while working with a slow moving upstream community
Who We Are

Distributed systems engineers
Storage systems architects
Open source contributors
  • Project Chairs: Apache HBase, Phoenix, Bigtop
  • Committers and PMC: Apache HBase, Phoenix, Pig, Bigtop, Incubator
  • Mentors: Apache Phoenix, NiFi, Trafodion
  • Hundreds of commits per year
Apache HBase and Apache Phoenix

A new scale out relational storage option
Apache HBase

A high performance horizontally scalable datastore engine for Big Data suitable as the store of record for mission critical data
Apache HBase

An emerging platform for scale out relational datastores
Apache HBase

A founding member of the Hadoop pantheon

- Introduced as a Hadoop ‘contrib’ module in 2007
HBase Data Model

Tablespaces

Not like a spreadsheet, a “sparse, consistent, distributed, multi-dimensional, sorted map”
HBase Scalability

Table A

Table B

Splits

Assignments

Regions

RegionServers
How is HBase Different from a RDBMS?

<table>
<thead>
<tr>
<th></th>
<th>RDBMS</th>
<th>HBase</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data layout</strong></td>
<td>Row oriented</td>
<td>Column oriented</td>
</tr>
<tr>
<td><strong>Transactions</strong></td>
<td>Multi-row ACID</td>
<td>Single row or adjacent row groups only</td>
</tr>
<tr>
<td><strong>Query language</strong></td>
<td>SQL</td>
<td>None (API access)</td>
</tr>
<tr>
<td><strong>Joins</strong></td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><strong>Indexes</strong></td>
<td>On arbitrary columns</td>
<td>Single row index only</td>
</tr>
<tr>
<td><strong>Max data size</strong></td>
<td>Terabytes</td>
<td>Petabytes*</td>
</tr>
<tr>
<td><strong>R/W throughput limits</strong></td>
<td>1000s of operations per second</td>
<td>Millions of operations per second*</td>
</tr>
</tbody>
</table>

* - No architectural upper bound on data size or aggregate throughput
SQL: In and Out of Fashion

1969: CODASYL (network database)
1979: First commercial SQL RDBMs
1990: Transaction processing on SQL now popular
1993: Multidimensional databases
1996: Enterprise Data Warehouses
2006: Hadoop and other “big data” technologies
2008: NoSQL
2011: SQL on Hadoop
2014: Interactive analytics on Hadoop and NoSQL with SQL

Why?

*From “SQL On Everything, In Memory” by Julian Hyde, Strata NYC 2014*
Implementing structured queries well is hard
- Systems cannot just “run the query” as written
- Relational systems require the algebraic operators, a query planner, an optimizer, metadata, statistics, etc.

However, the result is very useful to non-technical users
- Dumb queries (e.g. tool generated) can still get high performance
- Adding new algorithms or reorganizations of physical data layouts or migrations from one data store to another are transparent

What about blending the scale and performance of non-relational scale out stores with the ease of use of SQL?
Apache Phoenix

“Putting the SQL back into NoSQL”

A relational system built on HBase

• Reintroduces the familiar declarative SQL interface to data
• Reintroduces typed data and query optimizations possible with it
• Secondary indexes, joins, query optimization, statistics, ...
• Integrates with just about everything as a JDBC data source

With some new advantages

• Dynamic columns extend schema at runtime
• Schema is versioned – for free by HBase – allowing flashback queries using prior versions of metadata

And new challenges

• Distributed transactions are hard; a work in progress building on Tephra*
## Apache Phoenix

<table>
<thead>
<tr>
<th>Feature</th>
<th>Supported?</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE / DROP / ALTER TABLE</td>
<td>Yes</td>
</tr>
<tr>
<td>UPSERT / DELETE</td>
<td>Yes</td>
</tr>
<tr>
<td>SELECT</td>
<td>Yes</td>
</tr>
<tr>
<td>WHERE / HAVING</td>
<td>Yes</td>
</tr>
<tr>
<td>GROUP BY / ORDER BY</td>
<td>Yes</td>
</tr>
<tr>
<td>LIMIT</td>
<td>Yes</td>
</tr>
<tr>
<td>JOIN</td>
<td>Yes, with limitations</td>
</tr>
<tr>
<td>Views</td>
<td>Yes</td>
</tr>
<tr>
<td>Secondary indexes</td>
<td>Yes</td>
</tr>
<tr>
<td>Statistics and query optimization</td>
<td>Yes</td>
</tr>
<tr>
<td>Transactions</td>
<td>No, work in progress</td>
</tr>
</tbody>
</table>
Phoenix Integration With HBase

SELECT * FROM users WHERE id = 'xxxabc'

Coprocesors (HBase server side extensions)

RegionServers

Regions

JDBC driver
Phoenix Data Model

Phoenix maps the HBase data model to the relational world
Phoenix Data Model

Phoenix maps the HBase data model to the relational world.

![Diagram showing the Phoenix data model, mapping HBase data to relational tables.]

Key concepts:
- **Table**: Represents a HBase table.
- **Row**: Represents a row key in HBase.
- **Column**: Represents a column family in HBase.
- **Key**: Represents a key used to access data in HBase.
- **Value**: Represents a value associated with a key in HBase.
- **Column Family**: A set of columns (key-value pairs) organized under a common name.
Phoenix maps the HBase data model to the relational world

**Phoenix Data Model**

Phoenix table

Primary key constraint
Phoenix Data Model

Phoenix maps the HBase data model to the relational world
Phoenix maps the HBase data model to the relational world.
Dynamic columns

• Specify a subset of columns at CREATE time; the remainder can be optionally specified at query time
• Surfaces HBase's schema flexibility

```
CREATE TABLE "t" (
    K VARCHAR          PRIMARY KEY,
    "f1"."col1" VARCHAR,
    "f1"."col2" VARCHAR)

SELECT * FROM "t" ("f1"."col2" VARCHAR);
```
Native multitenancy

- Multitenant isolation via a combination of multitenant tables and tenant-specific connections
- Tenant-specific connections only access data that belongs to the tenant
- Tenants can create their own schema addendums (views, columns, indexes)

```sql
CREATE TABLE event (
    tenant_id VARCHAR,  
    type CHAR(1),
    event_id BIGINT,
    ...
CONSTRAINT pk PRIMARY KEY (tenant_id, type, event_id))
MULTI_TENANT=true;
```

First PK column identifies tenant ID

Tenant-specific connection

```sql
DriverManager.connect("jdbc:phoenix:localhost;tenantId=me");
```
Secondary Indexes

Three index types

• Mutable indexes
  • Global mutable indexes
    Server side intercepts primary table updates, builds and writes entries to secondary index tables
    For read heavy, low write use cases
  • Local mutable indexes
    Index data and primary data are placed together (index in shadow column family)
    For write heavy, space constrained use cases

• Immutable indexes
  Managed entirely by the client, writes scale best
  For use cases where rows are immutable after write
Secondary Indexes

...  

• Global mutable indexes – covered indexes
  
  CREATE TABLE t (k VARCHAR PRIMARY KEY, v1 VARCHAR, v2 INTEGER);
  
  CREATE INDEX i ON t (v1) INCLUDE (v2);

• Data in covered columns will be copied into the index

• This allows a global index to be used more frequently, as a global index will only be used if all columns referenced in the query are contained by it

Note: Rebuilds must currently be done using offline tooling (MapReduce)
Joins

Standard join syntax is supported, with some limitations

- Only equality (=) is supported in joining conditions
  - No restriction on other predicates in the ON clause
- Some queries may exceed server side resources and require rewrites by hand
- Work in progress
Query Optimization

Client side rewriting
- Parallel scanning with final client side merge sort
- RPC batching
- Use secondary indexes if available
- Rewrites for multitenant tables

Statistics
- Use guideposts to increase intra-region parallelism

Current Work-in-Progress
- Integration with Apache Calcite
  - ~120 rewrite rules

Server side push down
- Filters
- Skip scans
- Partial aggregation
- TopN
- Hash joins
Query Optimization

Example query plan for a 32 region table
Query Optimization
With a secondary index on RESPONSE_TIME

```
0: jdbc:phoenix:localhost:2181:/hbase> CREATE INDEX response_time on server_metrics (RESPONSE_TIME) INCLUDE (GC_TIME);
No rows affected (0.336 seconds)
0: jdbc:phoenix:localhost:2181:/hbase> EXPLAIN SELECT COUNT(*) AS COUNT, GC_TIME FROM SERVER_METRICS WHERE RESPONSE_TIME > 1000 GROUP BY GC_TIME ORDER BY COUNT DESC LIMIT 100;
```

```
PLAN

CLIENT PARALLEL 32-WAY RANGE SCAN OVER RESPONSE_TIME [1,000] - [*
  SERVER AGGREGATE INTO DISTINCT ROWS BY [GC_TIME]
  CLIENT MERGE SORT
  CLIENT TOP 100 ROWS SORTED BY [COUNT(1) DESC]

4 rows selected (0.058 seconds)
0: jdbc:phoenix:localhost:2181:/hbase>
```
The View from 30Kft

So Why HBase?

Durability
Consistency
Modeling constructs
Atomicity
Concurrency
Queryability
Schema mutability
Data size
Data portability and backup
Scalability
Reliability
Latency
Recoverability
Data Integrity
Required Downtime
Manageability
Cost
Support
Comprehensibility
License
Community
The View from 30Kft

So Why HBase?

Durability
Consistency
Modeling constructs
Atomicity
Concurrency
Queryability
Schema mutability
Data size
Data portability and backup
Scalability
Reliability
Latency
Recoverability
Data Integrity
Required Downtime
Manageability
Cost
Support
Comprehensibility
License
Community

Scalability
Consistency
Operations
License
Community
The View from 30Kft

So Why HBase?

Durability
Consistency
Modeling constructs
Atomicity
Concurrency
Queryability
Schema mutability
Data size
Data portability and backup
Scalability
Reliability
Latency
Recoverability
Data Integrity
Required Downtime
Manageability
Cost
Support
Comprehensibility
License
Community

Scalability
Consistency
Operations
License
Community

HBase
The View from 30Kft

Instances and instance groups

Instances are identical collections of hardware and software that support a discrete subset of our customers.

An instance group is a collection of instances, in one location, configured as a single resource pool and failure domain.

For every instance or instance group, there is an identical mirror available in another datacenter.

HBase is currently operating both in instance and instance group configurations:

- We started with smaller configurations (~10s of nodes).
- Recently we have begun migrating to larger configurations (~100s of nodes).
- Total fleet size is a “few thousand” servers.
Keeping Up Appearances

Apache Phoenix was motivated by the impedance mismatch between the HBase API and the expectations of platform developers

SELECT * FROM foo WHERE bar > 30 VS.

```
SELECT * FROM foo WHERE bar > 30
```

```
HTable t = new HTable("foo");
RegionScanner s = t.getScanner(new Scan(...,
    new ValueFilter(CompareOp.GT,
        new CustomTypedComparator(30)), ...));
while ((Result r = s.next()) != null) {
    // blah blah blah Java Java Java
}
s.close();
t.close();
```
Engineering The Whole Stack Holistically
Engineering The Whole Stack Holistically

Kernel tunables, filesystem, krb, nscd

Salesforce.com Application Servers
- Phoenix (JDBC client)
- Pig (Data Manipulation)
- Hive (Structured Query)
- GraphX (Graph analysis framework)
- MLLib (Data mining)

Sshoop (Real-time Data Collector)
- Phoenix (Query execution engine)

Flume (Log Data Collector)
- HBase (Distributed Database)

HDFS 2.0
- Hadoop Distributed File System

Zookeeper Coordination
- YARN (MRv2) (Cluster Resource Manager / MapReduce)
- Spark (Iterative In-Memory Computation)

The Java Virtual Machine
- Hadoop Common JNI

Linux
Engineering The Whole Stack Holistically

- GC tuning, local patches
- Kernel tunables, filesystem, krb, nscd
Engineering The Whole Stack Holistically

- **GC tuning, local patches**
- **Kernel tunables, filesystem, krb, nscd**
- **Multi-standby (HDFS-6440), fsync, tuning, custom UGI**

---

**Salesforce.com Application Servers**

- Phoenix (JDBC client)
- Pig (Data Manipulation)
- Hive (Structured Query)
- GraphX (Graph analysis framework)
- MLLib (Data mining)

**Cluster Components**

- **Snoop** (RDD Data Collector)
- **Flume** (Log Data Collector)
- **HBase** (Distributed Database)
- **Hadoop Distributed File System (HDFS 2.0)**
- **YARN (MRv2)** (Cluster Resource Manager / MapReduce)
- **Zookeeper** (Coordination)
- **Spark** (Iterative In-Memory Computation)
- **Hadoop Common JNI**

**The Java Virtual Machine**

**Linux**
Engineering The Whole Stack Holistically

Many bug fixes and enhancements, tuning

Multi-standby (HDFS-6440), fsync, tuning, custom UGI

GC tuning, local patches

Kernel tunables, filesystem, krb, nscd
Engineering The Whole Stack Holistically

Relational engine built on HBase

Many bug fixes and enhancements, tuning

Multi-standby (HDFS-6440), fsync, tuning, custom UGI

GC tuning, local patches

Kernel tunables, filesystem, krb, nscd
Engineering The Whole Stack Holistically

- Data access layer for internal users
- Relational engine built on HBase
- Many bug fixes and enhancements, tuning
- Multi-standby (HDFS-6440), fsync, tuning, custom UGI
- GC tuning, local patches
- Kernel tunables, filesystem, krb, nscd

Salesforce.com Application Servers

- Phoenix: JDBC client
- Pig: Data Manipulation
- Hive: Structured Query
- GraphX: Graph analysis framework
- MLLib: Data mining

- Phoenix: Query execution engine
- HBase: Distributed Database
- YARN (MRv2): Cluster Resource Manager / MapReduce
- Spark: Iterative In-Memory Computation

Zookeeper Coordination

HDFS 2.0: Hadoop Distributed File System

Hadoop Common JNI

The Java Virtual Machine

Linux
Engineering The Whole Stack Holistically

Circuit breaker, relogin thread

Data access layer for internal users

Relational engine built on HBase

Many bug fixes and enhancements, tuning

Multi-standby (HDFS-6440), fsync, tuning, custom UGI

GC tuning, local patches

Kernel tunables, filesystem, krb, nscd