Phoenix

We put the SQL back in NoSQL
https://github.com/forcedotcom/phoenix

James Taylor
@JamesPlusPlus
http://phoenix-hbase.blogspot.com/
Agenda

- What/why HBase?
Agenda

- What/why HBase?
- What/why Phoenix?
Agenda

- What/why HBase?
- What/why Phoenix?
- How does Phoenix work?
Agenda

- What/why HBase?
- What/why Phoenix?
- How does Phoenix work?
- Demo
Agenda

- What/why HBase?
- What/why Phoenix?
- How does Phoenix work?
- Demo
- Roadmap
Agenda

- What/why HBase?
- What/why Phoenix?
- How does Phoenix work?
- Demo
- Roadmap
- Q&A
What is HBase?

• Developed as part of Apache Hadoop
What is HBase?

- Developed as part of Apache Hadoop
- Runs on top of HDFS
What is HBase?

- Developed as part of Apache Hadoop
- Runs on top of HDFS
- Key/value store
What is HBase?

- Developed as part of Apache Hadoop
- Runs on top of HDFS
- Key/value store
  - Map
What is HBase?

- Developed as part of Apache Hadoop
- Runs on top of HDFS
- Key/value store

Map
Distributed
What is HBase?

- Developed as part of Apache Hadoop
- Runs on top of HDFS
- Key/value store
  Map
  Distributed
  Sparse
What is HBase?

- Developed as part of Apache Hadoop
- Runs on top of HDFS
- Key/value store
  - Map
  - Sorted
  - Distributed
  - Sparse
What is HBase?

- Developed as part of Apache Hadoop
- Runs on top of HDFS
- Key/value store
  - Map
  - Sorted
  - Distributed
  - Consistent
  - Sparse
What is HBase?

- Developed as part of Apache Hadoop
- Runs on top of HDFS
- Key/value store

- Map
- Sorted
- Distributed
- Consistent
- Sparse
- Multidimensional
Cluster Architecture

Client

Client finds RegionServer addresses in ZooKeeper

Client reads and writes rows by directly accessing the RegionServers

ZK Quorum
  - ZK Peer
  - ZK Peer
  - ZK Peer

HMaster

HMaster

Master assigns regions and achieves load balancing

RegionServer

RegionServer

RegionServer

HDFS

Software icon
Why Use HBase?

- If you have lots of data
Why Use HBase?

- If you have lots of data
  - Scales linearly
Why Use HBase?

- If you have lots of data
  - Scales linearly
  - Shards automatically
Why Use HBase?

- If you have lots of data
  - Scales linearly
  - Shards automatically
- If you can live without transactions
Why Use HBase?

- If you have lots of data
  - Scales linearly
  - Shards automatically
- If you can live without transactions
- If your data changes
Why Use HBase?

- If you have lots of data
  - Scales linearly
  - Shards automatically
- If you can live without transactions
- If your data changes
- If you need strict consistency
What is Phoenix?
What is Phoenix?

- SQL skin for HBase
What is Phoenix?

- SQL skin for HBase
- Alternate client API
What is Phoenix?

- SQL skin for HBase
- Alternate client API
- Embedded JDBC driver
What is Phoenix?

- SQL skin for HBase
- Alternate client API
- Embedded JDBC driver
- Runs at HBase native speed
What is Phoenix?

- SQL skin for HBase
- Alternate client API
- Embedded JDBC driver
- Runs at HBase native speed
- Compiles SQL into native HBase calls
What is Phoenix?

- SQL skin for HBase
- Alternate client API
- Embedded JDBC driver
- Runs at HBase native speed
- Compiles SQL into native HBase calls
- So you don’t have to!
Cluster Architecture

Client finds RegionServer addresses in ZooKeeper

Client reads and writes rows by directly accessing the RegionServers

ZK Quorum
- ZK Peer
- ZK Peer
- ZK Peer

HMaster
- HMaster
- HMaster

Master assigns regions and achieves load balancing

RegionServer

RegionServer

RegionServer

HDFS
Cluster Architecture

- Client
  - Phoenix
  - Client finds RegionServer addresses in ZooKeeper
  - Client reads and writes rows by directly accessing the RegionServers

- ZK Quorum
  - ZK Peer
  - ZK Peer
  - ZK Peer

- HMaster
  - Master assigns regions and achieves load balancing

- RegionServer

- HDFS
Cluster Architecture

- Phoenix
- Client
  - Client finds RegionServer addresses in ZooKeeper
  - Client reads and writes rows by directly accessing the RegionServers
- RegionServer
- Phoenix
- ZK Quorum
  - ZK Peer
  - ZK Peer
  - ZK Peer
- HMaster
- HMaster
  - Master assigns regions and achieves load balancing
- HDFS
Phoenix Performance

![Graph showing Phoenix Performance with three lines: Phoenix (key filter), Phoenix (full table), and Hive over HBase. The x-axis represents Table Row Count, ranging from 10M to 100M, and the y-axis represents Duration (sec), ranging from 0 to 140 seconds.]
Why Use Phoenix?
Why Use Phoenix?

- Give folks an API they already know
Why Use Phoenix?

- Give folks an API they already know
- Reduce the amount of code needed
Why Use Phoenix?

- Give folks an API they already know
- Reduce the amount of code needed

```
SELECT TRUNC(date,'DAY'), AVG(cpu) FROM web_stat WHERE domain LIKE 'Salesforce%' GROUP BY TRUNC(date,'DAY')
```
Why Use Phoenix?

- Give folks an API they already know
- Reduce the amount of code needed
- Perform optimizations transparently
Why Use Phoenix?

- Give folks an API they already know
- Reduce the amount of code needed
- Perform optimizations transparently
  - Aggregation
  - Skip Scan
  - Secondary indexing (soon!)
Why Use Phoenix?

- Give folks an API they already know
- Reduce the amount of code needed
- Perform optimizations transparently
- Leverage existing tooling
Why Use Phoenix?

- Give folks an API they already know
- Reduce the amount of code needed
- Perform optimizations transparently
- Leverage existing tooling
  - SQL client/terminal
  - OLAP engine
How Does Phoenix Work?

- Overlays on top of HBase Data Model
- Keeps Versioned Schema Repository
- Query Processor
Phoenix Data Model

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

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

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

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

Phoenix maps HBase data model to the relational world

<table>
<thead>
<tr>
<th>HBase Table</th>
<th>Column Family A</th>
<th>Column Family B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Qualifier 1</td>
<td>Qualifier 2</td>
</tr>
<tr>
<td>Row Key 1</td>
<td>Value</td>
<td>Value</td>
</tr>
<tr>
<td>Row Key 2</td>
<td>Value</td>
<td>Value</td>
</tr>
</tbody>
</table>
Phoenix Data Model

Phoenix maps HBase data model to the relational world

<table>
<thead>
<tr>
<th>HBase Table</th>
<th>Column Family A</th>
<th>Column Family B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Qualifier 1</td>
<td>Qualifier 2</td>
</tr>
<tr>
<td>Row Key 1</td>
<td>Value</td>
<td>Value</td>
</tr>
<tr>
<td>Row Key 2</td>
<td></td>
<td>Value</td>
</tr>
<tr>
<td>Row Key 3</td>
<td>Value</td>
<td></td>
</tr>
</tbody>
</table>
Phoenix Data Model
Phoenix maps HBase data model to the relational world

<table>
<thead>
<tr>
<th>HBase Table</th>
<th>Column Family A</th>
<th>Column Family B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Qualifier 1</td>
<td>Qualifier 2</td>
</tr>
<tr>
<td>Row Key 1</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>Row Key 2</td>
<td></td>
<td>Value</td>
</tr>
<tr>
<td>Row Key 3</td>
<td>Value</td>
<td>Value</td>
</tr>
</tbody>
</table>

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

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

Phoenix maps HBase data model to the relational world

Phoenix Table

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Column Family A</th>
<th>Column Family B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Qualifier 1</td>
<td>Qualifier 2</td>
</tr>
<tr>
<td>Row Key 1</td>
<td>Value</td>
<td>Value</td>
</tr>
<tr>
<td>Row Key 2</td>
<td></td>
<td>Value</td>
</tr>
<tr>
<td>Row Key 3</td>
<td>Value</td>
<td>Value</td>
</tr>
</tbody>
</table>
Phoenix Data Model

Phoenix maps HBase data model to the relational world

<table>
<thead>
<tr>
<th>Phoenix Table</th>
<th>HBase Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column Family A</td>
<td>Column Family A</td>
</tr>
<tr>
<td>Column Family B</td>
<td>Column Family B</td>
</tr>
</tbody>
</table>

| Row Key 1 | Value |
| Row Key 2 | Value |
| Row Key 3 | Value |

Key Value Columns
Phoenix Data Model

Phoenix maps HBase data model to the relational world

Phoenix Table

HBase Table

Column Family A
- Qualifier 1
- Qualifier 2
- Qualifier 3

Column Family B

Row Key Columns

Key Value Columns

Row Key 1
Row Key 2
Row Key 3

Value
Value
Value
Value
Phoenix Metadata

- Stored in a Phoenix HBase table
Phoenix Metadata

- Stored in a Phoenix HBase table
  - SYSTEM.TABLE
Phoenix Metadata

- Stored in a Phoenix HBase table
- Updated through DDL commands
Phoenix Metadata

- Stored in a Phoenix HBase table
- Updated through DDL commands
  - CREATE TABLE
  - ALTER TABLE
  - DROP TABLE
  - CREATE INDEX
  - DROP INDEX
Phoenix Metadata

- Stored in a Phoenix HBase table
- Updated through DDL commands
- Keeps older versions as schema evolves
Phoenix Metadata

- Stored in a Phoenix HBase table
- Updated through DDL commands
- Keeps older versions as schema evolves
- Correlates timestamps between schema and data
Phoenix Metadata

- Stored in a Phoenix HBase table
- Updated through DDL commands
- Keeps older versions as schema evolves
- Correlates timestamps between schema and data
  - Flashback queries use schema that was in-place then
Phoenix Metadata

- Stored in a Phoenix HBase table
- Updated through DDL commands
- Keeps older versions as schema evolves
- Correlates timestamps between schema and data
- Accessible via JDBC metadata APIs
Phoenix Metadata

- Stored in a Phoenix HBase table
- Updated through DDL commands
- Keeps older versions as schema evolves
- Correlates timestamps between schema and data
- Accessible via JDBC metadata APIs
  - java.sql.DatabaseMetaData
  - Through Phoenix queries!
Example

Over metrics data for clusters of servers with a schema like this:

<table>
<thead>
<tr>
<th>SERVER METRICS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>HOST</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>DATE</td>
<td>DATE</td>
</tr>
<tr>
<td>RESPONSE_TIME</td>
<td>INTEGER</td>
</tr>
<tr>
<td>GC_TIME</td>
<td>INTEGER</td>
</tr>
<tr>
<td>CPU_TIME</td>
<td>INTEGER</td>
</tr>
<tr>
<td>IO_TIME</td>
<td>INTEGER</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Example

Over metrics data for clusters of servers with a schema like this:

<table>
<thead>
<tr>
<th>SERVER METRICS</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOST</td>
</tr>
<tr>
<td>DATE</td>
</tr>
<tr>
<td>RESPONSE_TIME</td>
</tr>
<tr>
<td>GC_TIME</td>
</tr>
<tr>
<td>CPU_TIME</td>
</tr>
<tr>
<td>IO_TIME</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Example
With 90 days of data that looks like this:

<table>
<thead>
<tr>
<th>HOST</th>
<th>DATE</th>
<th>RESPONSE_TIME</th>
<th>GC_TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>sf1.s1</td>
<td>Jun 5 10:10:10.234</td>
<td>1234</td>
<td></td>
</tr>
<tr>
<td>sf1.s1</td>
<td>Jun 5 11:18:28.456</td>
<td></td>
<td>8012</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sf3.s1</td>
<td>Jun 5 10:10:10.234</td>
<td>2345</td>
<td></td>
</tr>
<tr>
<td>sf3.s1</td>
<td>Jun 6 12:46:19.123</td>
<td></td>
<td>2340</td>
</tr>
<tr>
<td>sf7.s9</td>
<td>Jun 4 08:23:23.456</td>
<td>5002</td>
<td>1234</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

Walk through query processing for three scenarios
Example

Walk through query processing for three scenarios

1. Chart Response Time Per Cluster
Example

Walk through query processing for three scenarios

1. Chart Response Time Per Cluster

2. Identify 5 Longest GC Times
Example

Walk through query processing for three scenarios

1. Chart Response Time Per Cluster
2. Identify 5 Longest GC Times
3. Identify 5 Longest GC Times again and again
Scenario 1

Chart Response Time Per Cluster

SELECT substr(host,1,3), trunc(date,'DAY'), avg(response_time)
FROM server_metrics
WHERE date > CURRENT_DATE() – 7
AND substr(host, 1, 3) IN ('sf1', 'sf3', 'sf7')
GROUP BY substr(host, 1, 3), trunc(date,'DAY')
Scenario 1
Chart Response Time Per Cluster

SELECT substr(host, 1, 3), trunc(date,’DAY’), avg(response_time)
FROM server_metrics
WHERE date > CURRENT_DATE() – 7
AND substr(host, 1, 3) IN (‘sf1’, ‘sf3’, ‘sf7’) 
GROUP BY substr(host, 1, 3), trunc(date,’DAY’)
Scenario 1
Chart Response Time Per Cluster

SELECT substr(host,1,3), trunc(date,'DAY'), avg(response_time)
FROM server_metrics
WHERE date > CURRENT_DATE() - 7
AND substr(host, 1, 3) IN ('sf1', 'sf3', 'sf7')
GROUP BY substr(host, 1, 3), trunc(date,'DAY')
Scenario 1
Chart Response Time Per Cluster

SELECT substr(host, 1, 3), trunc(date,'DAY'), avg(response_time) 
FROM server_metrics 
WHERE date > CURRENT_DATE() – 7 
AND substr(host, 1, 3) IN ('sf1', 'sf3', 'sf7') 
GROUP BY substr(host, 1, 3), trunc(date,'DAY')
Scenario 1
Chart Response Time Per Cluster

```
SELECT substr(host,1,3), trunc(date,'DAY'), avg(response_time)
FROM server_metrics
WHERE date > CURRENT_DATE() – 7
AND substr(host, 1, 3) IN ('sf1', 'sf3', 'sf7')
GROUP BY substr(host, 1, 3), trunc(date,'DAY')
```
Step 1: Client
Identify Row Key Ranges from Query

```
SELECT substr(host,1,3), trunc(date,’DAY’), avg(response_time)
FROM server_metrics
WHERE date > CURRENT_DATE() – 7 
AND substr(host, 1, 3) IN (‘sf1’, ‘sf3’, ‘sf7’)
GROUP BY substr(host, 1, 3), trunc(date,’DAY’)
```

<table>
<thead>
<tr>
<th>Row Key Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOST</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Step 1: Client
Identify Row Key Ranges from Query

SELECT substr(host,1,3), trunc(date,'DAY'), avg(response_time)
FROM server_metrics
WHERE date > CURRENT_DATE() – 7
AND substr(host, 1, 3) IN ('sf1', 'sf3', 'sf7')
GROUP BY substr(host, 1, 3), trunc(date,'DAY')
Step 1: Client
Identify Row Key Ranges from Query

SELECT substr(host,1,3), trunc(date,'DAY'), avg(response_time)
FROM server_metrics
WHERE date > CURRENT_DATE() – 7
AND substr(host, 1, 3) IN ('sf1', 'sf3', 'sf7')
GROUP BY substr(host, 1, 3), trunc(date,'DAY')

<table>
<thead>
<tr>
<th>Row Key Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOST</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Step 1: Client
Identify Row Key Ranges from Query

SELECT substr(host,1,3), trunc(date,'DAY'), avg(response_time)
FROM server_metrics
WHERE date > CURRENT_DATE() – 7
AND substr(host, 1, 3) IN ('sf1', 'sf3', 'sf7')
GROUP BY substr(host, 1, 3), trunc(date,'DAY')

<table>
<thead>
<tr>
<th>HOST</th>
<th>DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>sf1</td>
<td></td>
</tr>
</tbody>
</table>
Step 1: Client

Identify Row Key Ranges from Query

SELECT substr(host,1,3), trunc(date,'DAY'), avg(response_time) FROM server_metrics
WHERE date > CURRENT_DATE() – 7
AND substr(host, 1, 3) IN ('sf1', 'sf3', 'sf7')
GROUP BY substr(host, 1, 3), trunc(date,'DAY')

<table>
<thead>
<tr>
<th>HOST</th>
<th>DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>sf1</td>
<td></td>
</tr>
<tr>
<td>sf3</td>
<td></td>
</tr>
</tbody>
</table>
Step 1: Client

Identify Row Key Ranges from Query

SELECT substr(host,1,3), trunc(date,'DAY'), avg(response_time)
FROM server_metrics
WHERE date > CURRENT_DATE() – 7
AND substr(host, 1, 3) IN ('sf1', 'sf3', 'sf7')
GROUP BY substr(host, 1, 3), trunc(date,'DAY')

<table>
<thead>
<tr>
<th>HOST</th>
<th>DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>sf1</td>
<td></td>
</tr>
<tr>
<td>sf3</td>
<td></td>
</tr>
<tr>
<td>sf7</td>
<td></td>
</tr>
</tbody>
</table>
Step 1: Client
Identify Row Key Ranges from Query

```
SELECT substr(host,1,3), trunc(date,'DAY'), avg(response_time)
FROM server_metrics
WHERE date > CURRENT_DATE() - 7
AND substr(host, 1, 3) IN ('sf1', 'sf3', 'sf7')
GROUP BY substr(host, 1, 3), trunc(date,'DAY')
```

Row Key Ranges

<table>
<thead>
<tr>
<th>HOST</th>
<th>DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>sf1</td>
<td>$t_1$ - *</td>
</tr>
<tr>
<td>sf3</td>
<td></td>
</tr>
<tr>
<td>sf7</td>
<td></td>
</tr>
</tbody>
</table>
Step 2: Client
Overlay Row Key Ranges with Regions

R1

R2

R3

R4

sf1

sf3

sf4

sf6

sf7
Step 3: Client
Execute Parallel Scans

- scan1
  - R1
  - sf1

- scan2
  - R2
  - sf3
  - sf4

- scan3
  - R3
  - sf6
  - sf7

- R4
Step 4: Server
Filter using Skip Scan

SKIP

sf1.s1 t₀
Step 4: Server
Filter using Skip Scan

INCLUDE

sf1.s1  t₁
Step 4: Server
Filter using Skip Scan
Step 4: Server
Filter using Skip Scan

INCLUDE

sf1.s2 t₁
Step 4: Server
Filter using Skip Scan

sf1.s3 t₀
Step 4: Server
Filter using Skip Scan
Step 5: Server Intercept Scan in Coprocessor

### SERVER METRICS

<table>
<thead>
<tr>
<th>HOST</th>
<th>DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>sf1.s1</td>
<td>Jun 2 10:10:10.234</td>
</tr>
<tr>
<td>sf1.s2</td>
<td>Jun 3 23:05:44.975</td>
</tr>
<tr>
<td>sf1.s2</td>
<td>Jun 9 08:10:32.147</td>
</tr>
<tr>
<td>sf1.s3</td>
<td>Jun 1 11:18:28.456</td>
</tr>
<tr>
<td>sf1.s3</td>
<td>Jun 3 22:03:22.142</td>
</tr>
<tr>
<td>sf1.s4</td>
<td>Jun 1 10:29:58.950</td>
</tr>
<tr>
<td>sf1.s4</td>
<td>Jun 2 14:55:34.104</td>
</tr>
<tr>
<td>sf1.s4</td>
<td>Jun 3 12:46:19.123</td>
</tr>
<tr>
<td>sf1.s5</td>
<td>Jun 8 08:23:23.456</td>
</tr>
<tr>
<td>sf1.s6</td>
<td>Jun 1 10:31:10.234</td>
</tr>
</tbody>
</table>

### SERVER METRICS

<table>
<thead>
<tr>
<th>HOST</th>
<th>DATE</th>
<th>AGG</th>
</tr>
</thead>
<tbody>
<tr>
<td>sf1</td>
<td>Jun 1</td>
<td>...</td>
</tr>
<tr>
<td>sf1</td>
<td>Jun 2</td>
<td>...</td>
</tr>
<tr>
<td>sf1</td>
<td>Jun 3</td>
<td>...</td>
</tr>
<tr>
<td>sf1</td>
<td>Jun 8</td>
<td>...</td>
</tr>
<tr>
<td>sf1</td>
<td>Jun 9</td>
<td>...</td>
</tr>
</tbody>
</table>
Step 6: Client
Perform Final Merge Sort

R1

R2

R3

R4

SERVER METRICS

<table>
<thead>
<tr>
<th>HOST</th>
<th>DATE</th>
<th>AGG</th>
</tr>
</thead>
<tbody>
<tr>
<td>sf1</td>
<td>Jun 5</td>
<td></td>
</tr>
<tr>
<td>sf1</td>
<td>Jun 9</td>
<td></td>
</tr>
<tr>
<td>sf3</td>
<td>Jun 1</td>
<td></td>
</tr>
<tr>
<td>sf3</td>
<td>Jun 2</td>
<td></td>
</tr>
<tr>
<td>sf7</td>
<td>Jun 1</td>
<td></td>
</tr>
<tr>
<td>sf7</td>
<td>Jun 8</td>
<td></td>
</tr>
</tbody>
</table>
Scenario 2
Find 5 Longest GC Times

SELECT host, date, gc_time
FROM server_metrics
WHERE date > CURRENT_DATE() - 7
AND substr(host, 1, 3) IN ('sf1', 'sf3', 'sf7')
ORDER BY gc_time DESC
LIMIT 5
Scenario 2
Find 5 Longest GC Times

- Same client parallelization and server skip scan filtering
Scenario 2
Find 5 Longest GC Times

- Same client parallelization and server skip scan filtering
- Server holds 5 longest GC_TIME value for each scan

### SERVER METRICS

<table>
<thead>
<tr>
<th>HOST</th>
<th>DATE</th>
<th>GC_TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>sf1.s1</td>
<td>Jun 2 10:10:10.234</td>
<td>22123</td>
</tr>
<tr>
<td>sf1.s1</td>
<td>Jun 3 23:05:44.975</td>
<td>19876</td>
</tr>
<tr>
<td>sf1.s1</td>
<td>Jun 9 08:10:32.147</td>
<td>11345</td>
</tr>
<tr>
<td>sf1.s2</td>
<td>Jun 1 11:18:28.456</td>
<td>10234</td>
</tr>
<tr>
<td>sf1.s2</td>
<td>Jun 3 22:03:22.142</td>
<td>10111</td>
</tr>
</tbody>
</table>
Scenario 2
Find 5 Longest GC Times

- Same client parallelization and server skip scan filtering
- Server holds 5 longest GC_TIME value for each scan
- Client performs final merge sort among parallel scans

### SERVER METRICS

<table>
<thead>
<tr>
<th>HOST</th>
<th>DATE</th>
<th>GC_TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>sf1.s1</td>
<td>Jun 2 10:10:10.234</td>
<td>22123</td>
</tr>
<tr>
<td>sf1.s1</td>
<td>Jun 3 23:05:44.975</td>
<td>19876</td>
</tr>
<tr>
<td>sf1.s1</td>
<td>Jun 9 08:10:32.147</td>
<td>11345</td>
</tr>
<tr>
<td>sf1.s2</td>
<td>Jun 1 11:18:28.456</td>
<td>10234</td>
</tr>
<tr>
<td>sf1.s2</td>
<td>Jun 3 22:03:22.142</td>
<td>10111</td>
</tr>
</tbody>
</table>
Scenario 3
Find 5 Longest GC Times

CREATE INDEX gc_time_index
ON server_metrics (gc_time DESC, date DESC)
INCLUDE (host, response_time)
Scenario 3
Find 5 Longest GC Times

CREATE INDEX gc_time_index
ON server_metrics (gc_time DESC, date DESC)
INCLUDE (host, response_time)
Scenario 3
Find 5 Longest GC Times

CREATE INDEX gc_time_index
ON server_metrics (gc_time DESC, date DESC)
INCLUDE (host, response_time)
Scenario 3
Find 5 Longest GC Times

CREATE INDEX gc_time_index
ON server_metrics (gc_time DESC, date DESC)
INCLUDE (host, response_time)
Scenario 3
Find 5 Longest GC Times

CREATE INDEX gc_time_index
ON server_metrics (gc_time DESC, date DESC)
INCLUDE (host, response_time)
Scenario 3
Find 5 Longest GC Times

SELECT host, date, gc_time
FROM server_metrics
WHERE date > CURRENT_DATE() – 7
AND substr(host, 1, 3) IN ('sf1', 'sf3', 'sf7')
ORDER BY gc_time DESC
LIMIT 5
Demo

- Phoenix Stock Analyzer
- Fortune 500 companies
- 10 years of historical stock prices
- Demonstrates Skip Scan in action
- Running locally on my single node laptop cluster
Phoenix Roadmap

- Secondary Indexing
- Count distinct and percentile
- Derived tables
- Hash Joins
- Apache Drill integration
- Cost-based query optimizer
- OLAP extensions
- Transactions
Thank you!
Questions/comments?