Phoenix

We put the SQL back in NoSQL

https://github.com/forcedotcom/phoenix

James Taylor

@JamesPlusPlus

http://phoenix-hbase.blogspot.com/

Your success.
Our cloud.

salesforce.com

SOFTWARE

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

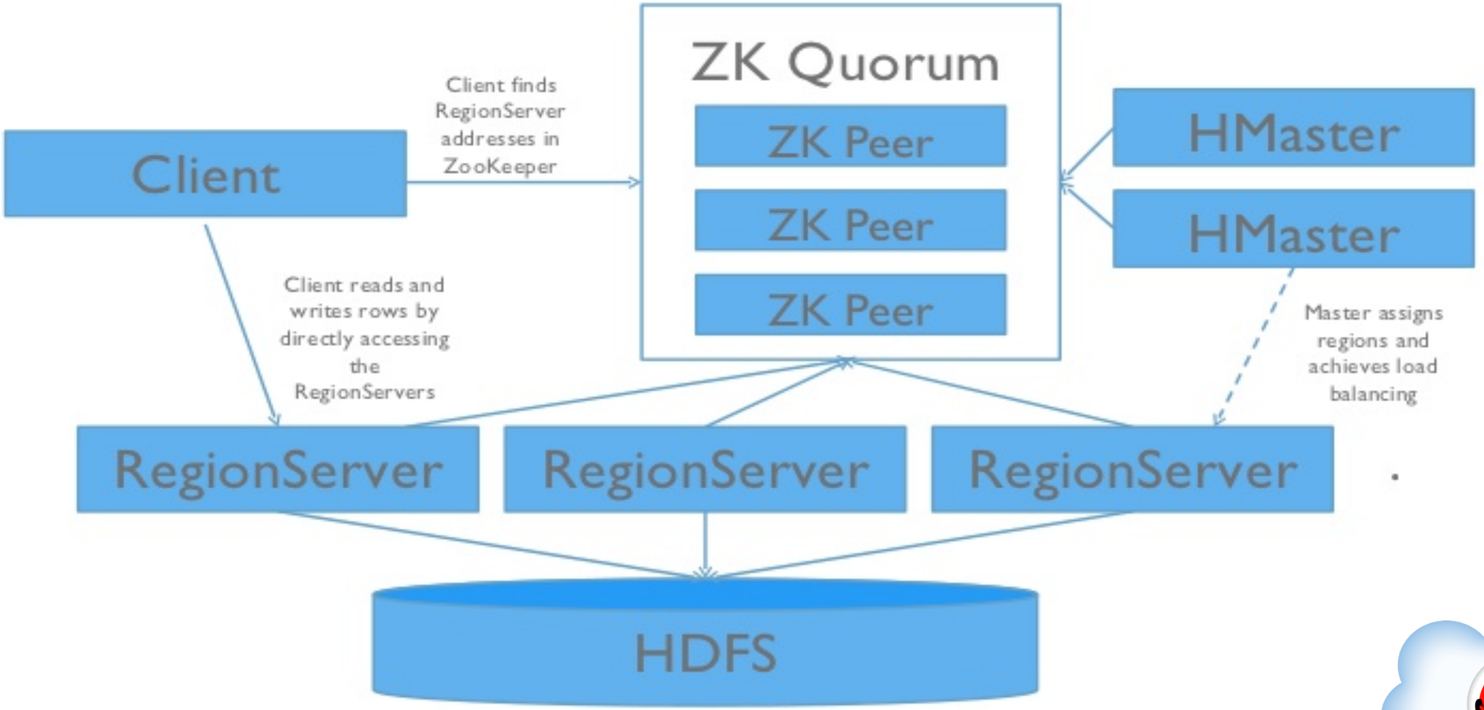
Map	Sorted
Distributed	Consistent
Sparse	

What is HBase?

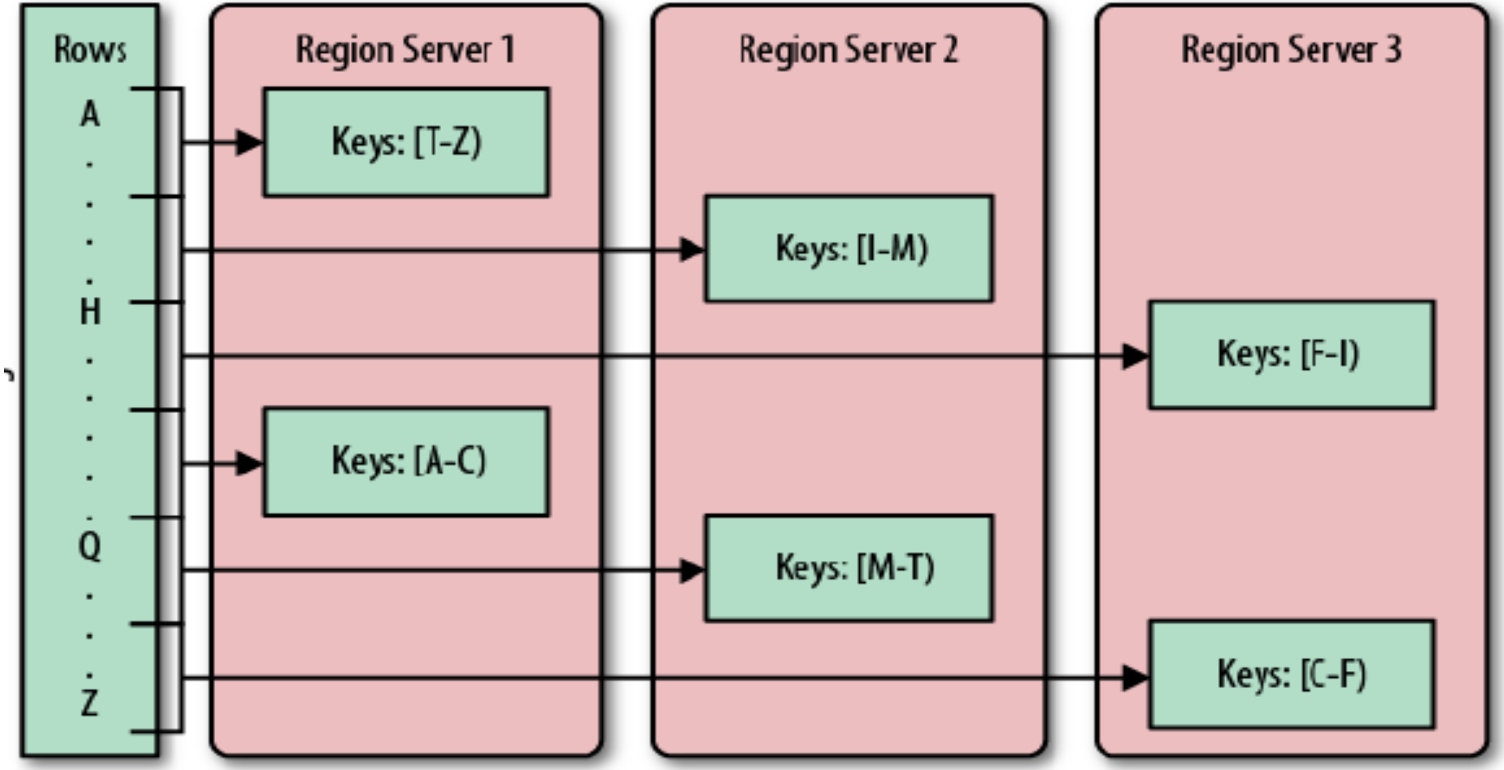
- Developed as part of Apache Hadoop
- Runs on top of HDFS
- Key/value store

Map	Sorted
Distributed	Consistent
Sparse	Multidimensional

Cluster Architecture



Sharding



Rows
A
H
Q
Z
Region Server 1
Keys: [T-Z)
Keys: [A-C)
Region Server 2
Keys: [I-M)
Keys: [M-T)
Region Server 3
Keys: [F-I)
Keys: [C-F)

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

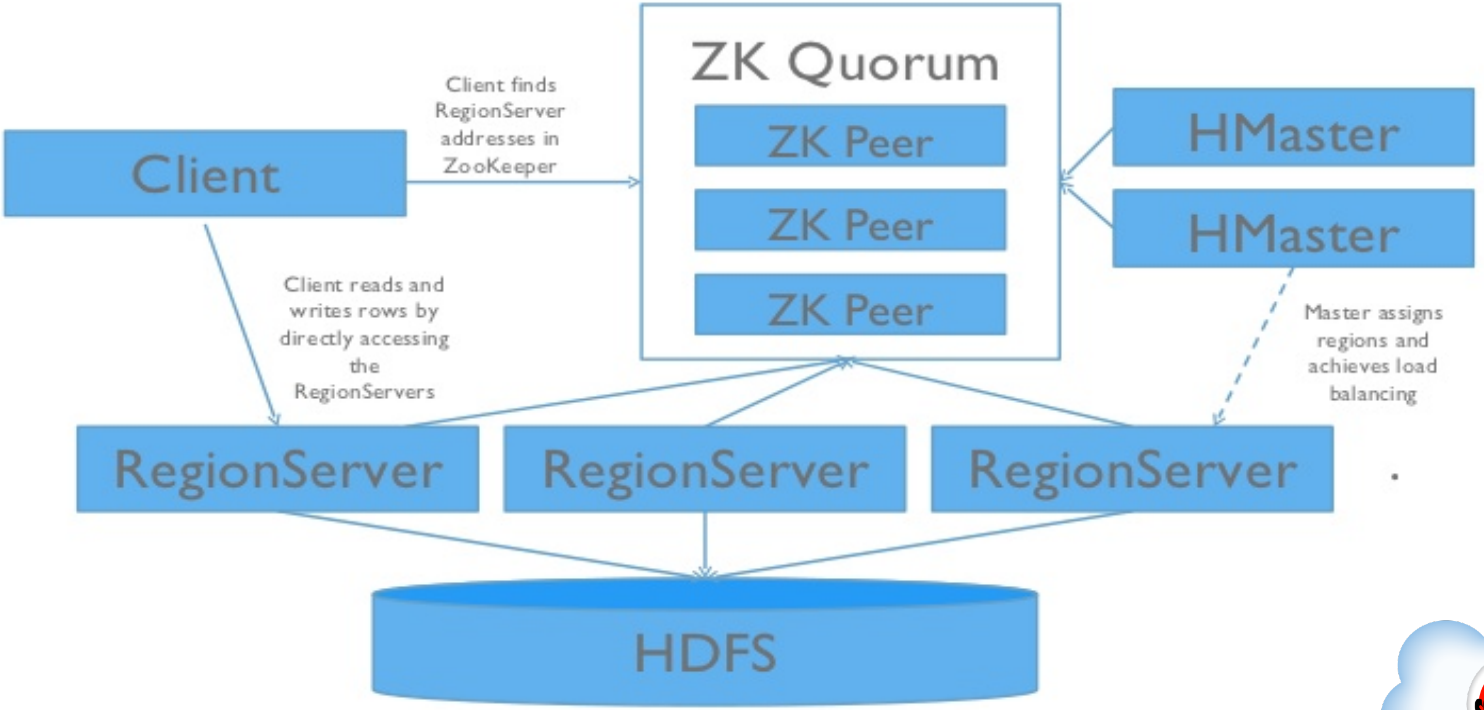
What is Phoenix?

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

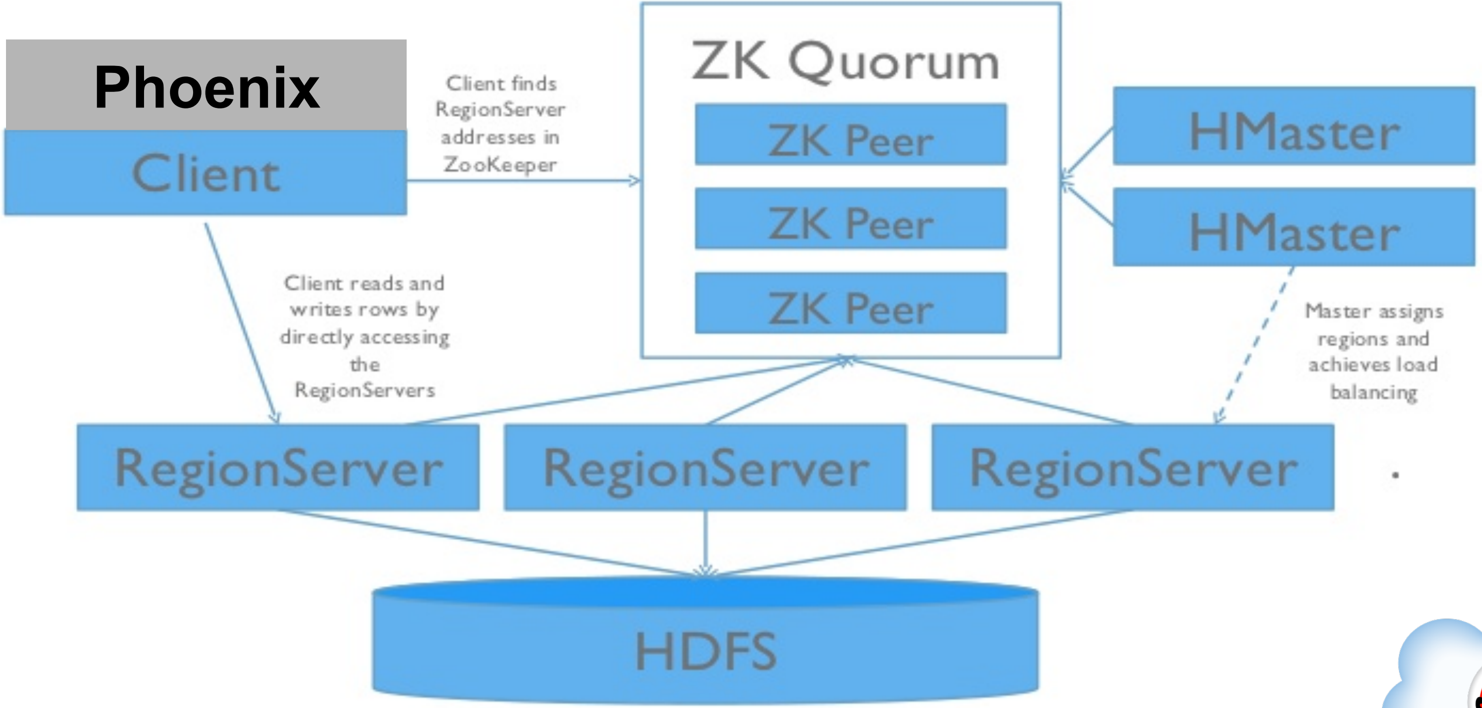
What is Phoenix?

- 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!**

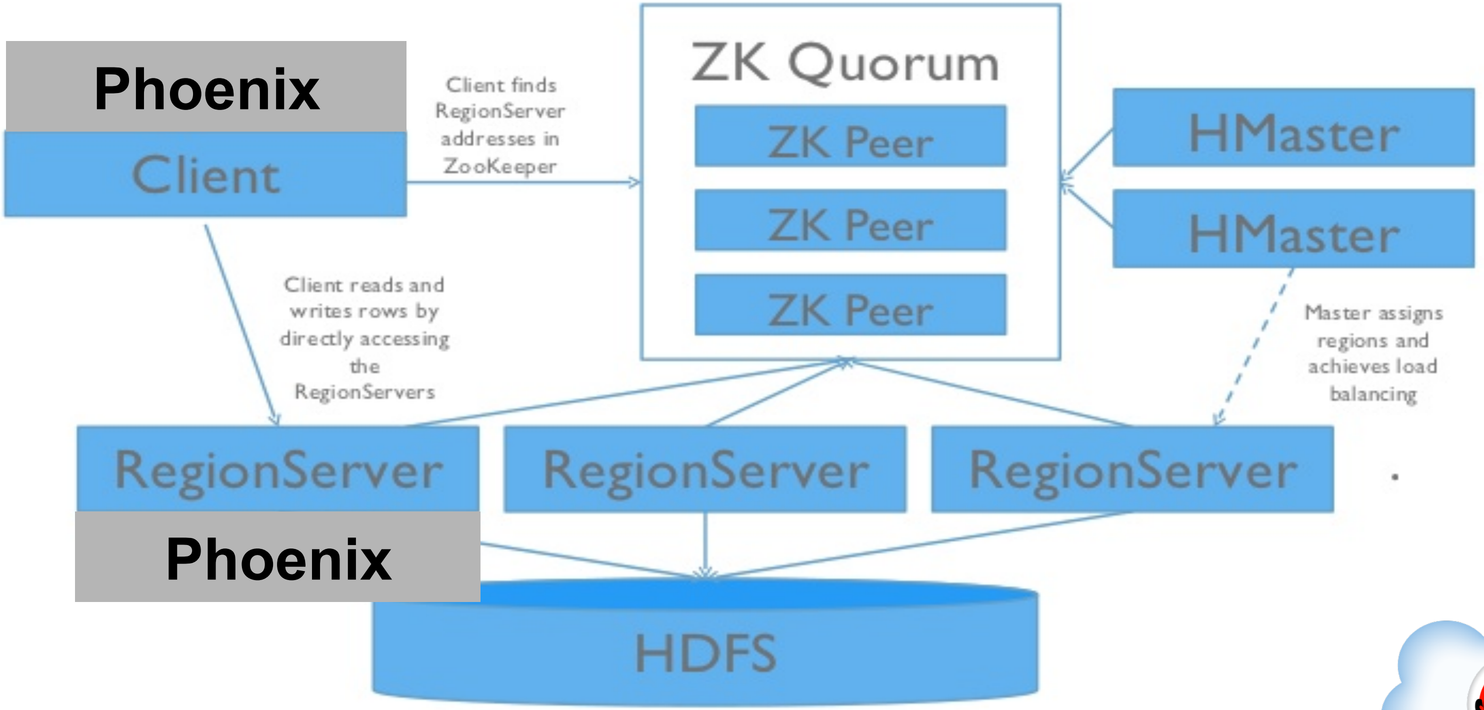
Cluster Architecture



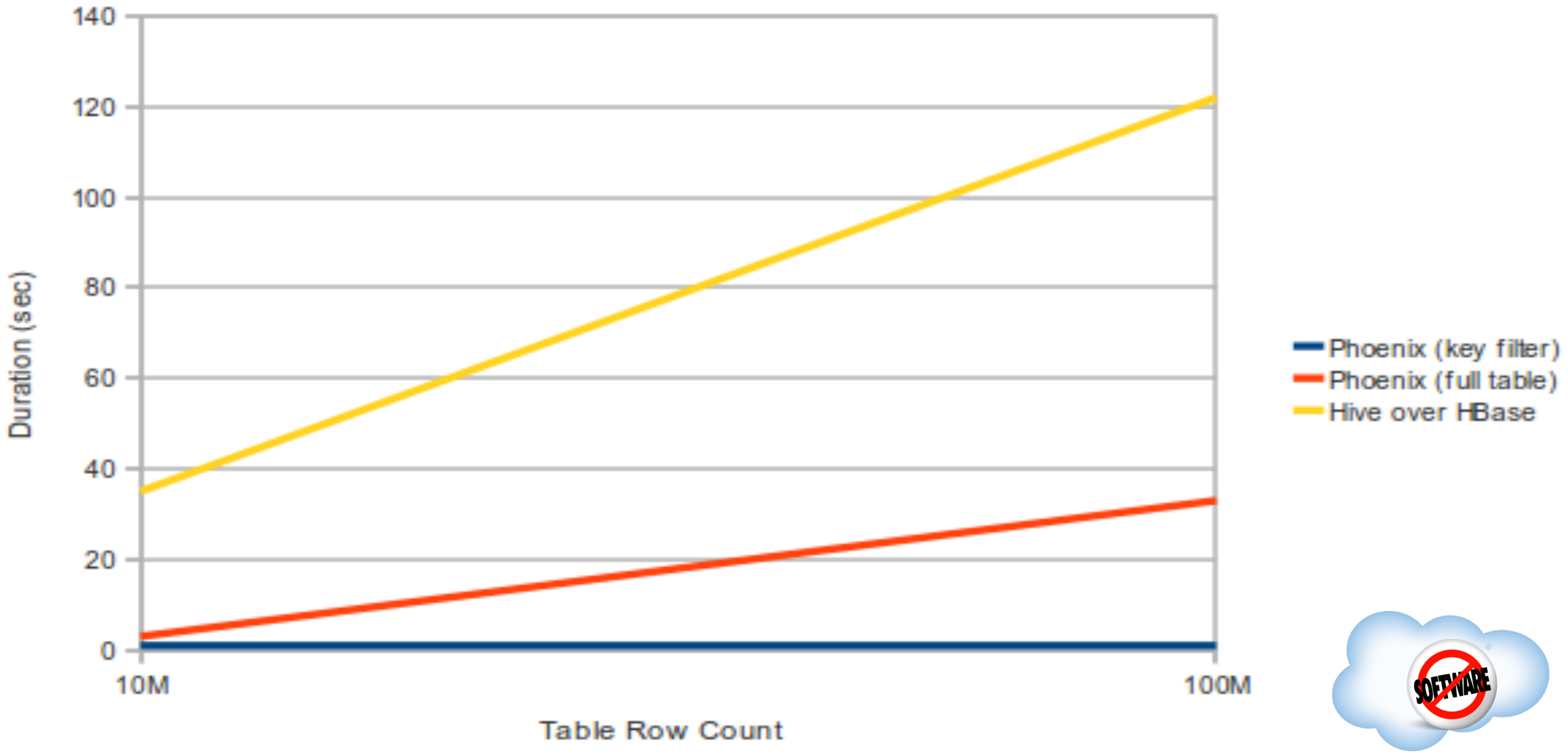
Cluster Architecture



Cluster Architecture



Phoenix Performance



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 Respository
- Query Processor

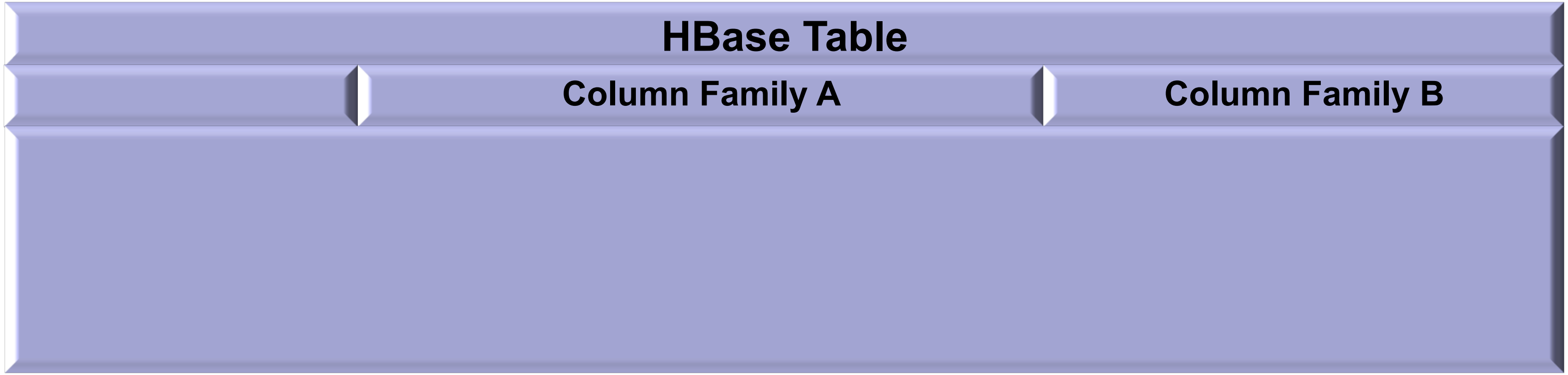
Phoenix Data Model

Phoenix maps HBase data model to the relational world

HBase Table

Phoenix Data Model

Phoenix maps HBase data model to the relational world



Phoenix Data Model

Phoenix maps HBase data model to the relational world

HBase Table			
	Column Family A		Column Family B
	Qualifier 1	Qualifier 2	Qualifier 3

Phoenix Data Model

Phoenix maps HBase data model to the relational world

HBase Table			
	Column Family A		Column Family B
	Qualifier 1	Qualifier 2	Qualifier 3
Row Key 1	*Value*		

Phoenix Data Model

Phoenix maps HBase data model to the relational world

HBase Table			
	Column Family A		**Column Family B**
	Qualifier 1	Qualifier 2	Qualifier 3
Row Key 1	*Value*		
Row Key 2		*Value*	*Value*

Phoenix Data Model

Phoenix maps HBase data model to the relational world

HBase Table			
	Column Family A		**Column Family B**
	Qualifier 1	Qualifier 2	Qualifier 3
Row Key 1	*Value*		
Row Key 2		*Value*	*Value*
Row Key 3	*Value*		

Phoenix Data Model

Phoenix maps HBase data model to the relational world

HBase Table			
	Column Family A		**Column Family B**
	Qualifier 1	Qualifier 2	Qualifier 3
Row Key 1	*Value*		
Row Key 2		*Value*	*Value*
Row Key 3	*Value*		

Phoenix Data Model

Phoenix maps HBase data model to the relational world

HBase Table			
	Column Family A		**Column Family B**
	Qualifier 1	Qualifier 2	Qualifier 3
Row Key 1	*Value*		
Row Key 2		*Value*	*Value*
Row Key 3	*Value*		

Phoenix Data Model

Phoenix maps HBase data model to the relational world

HBase Table			
	Column Family A		**Column Family B**
	Qualifier 1	Qualifier 2	Qualifier 3
Row Key 1	*Value*		
Row Key 2		*Value*	*Value*
Row Key 3	*Value*		

Multiple Versions

Phoenix Data Model

Phoenix maps HBase data model to the relational world

Phoenix Table

HBase Table			
	Column Family A		**Column Family B**
	Qualifier 1	Qualifier 2	Qualifier 3
Row Key 1	*Value*		
Row Key 2		*Value*	*Value*
Row Key 3	*Value*		

Phoenix Data Model

Phoenix maps HBase data model to the relational world

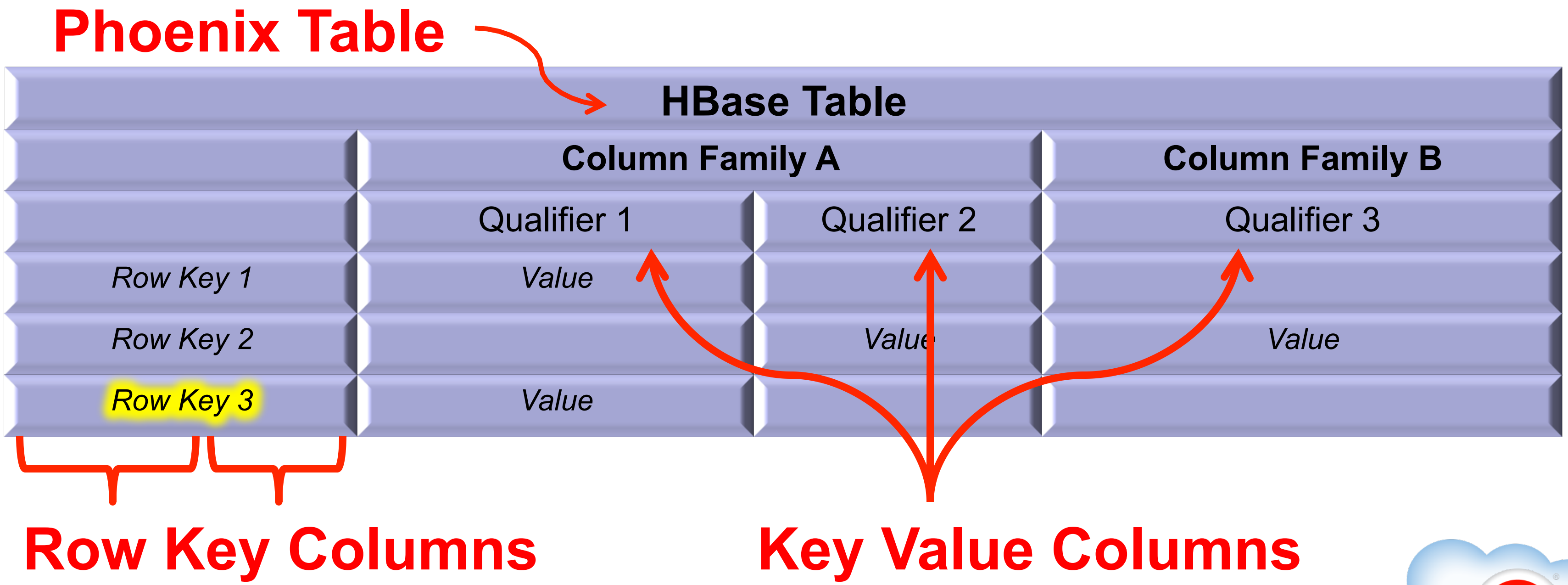
Phoenix Table

HBase Table			
	Column Family A		Column Family B
	Qualifier 1	Qualifier 2	Qualifier 3
Row Key 1	*Value*		
Row Key 2		*Value*	*Value*
Row Key 3	*Value*		

Key Value Columns

Phoenix Data Model

Phoenix maps HBase data model to the relational world



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:

SERVER METRICS	
HOST	VARCHAR
DATE	DATE
RESPONSE_TIME	INTEGER
GC_TIME	INTEGER
CPU_TIME	INTEGER
IO_TIME	INTEGER
…	

Row Key

Example

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

SERVER METRICS	
HOST	VARCHAR
DATE	DATE
RESPONSE_TIME	INTEGER
GC_TIME	INTEGER
CPU_TIME	INTEGER
IO_TIME	INTEGER
…	

Key Values

Example

With 90 days of data that looks like this:

SERVER METRICS			
HOST	DATE	RESPONSE_TIME	GC_TIME
sf1.s1	Jun 5 10:10:10.234	1234	
sf1.s1	Jun 5 11:18:28.456		8012
…			
sf3.s1	Jun 5 10:10:10.234	2345	
sf3.s1	Jun 6 12:46:19.123		2340
sf7.s9	Jun 4 08:23:23.456	5002	1234
…			

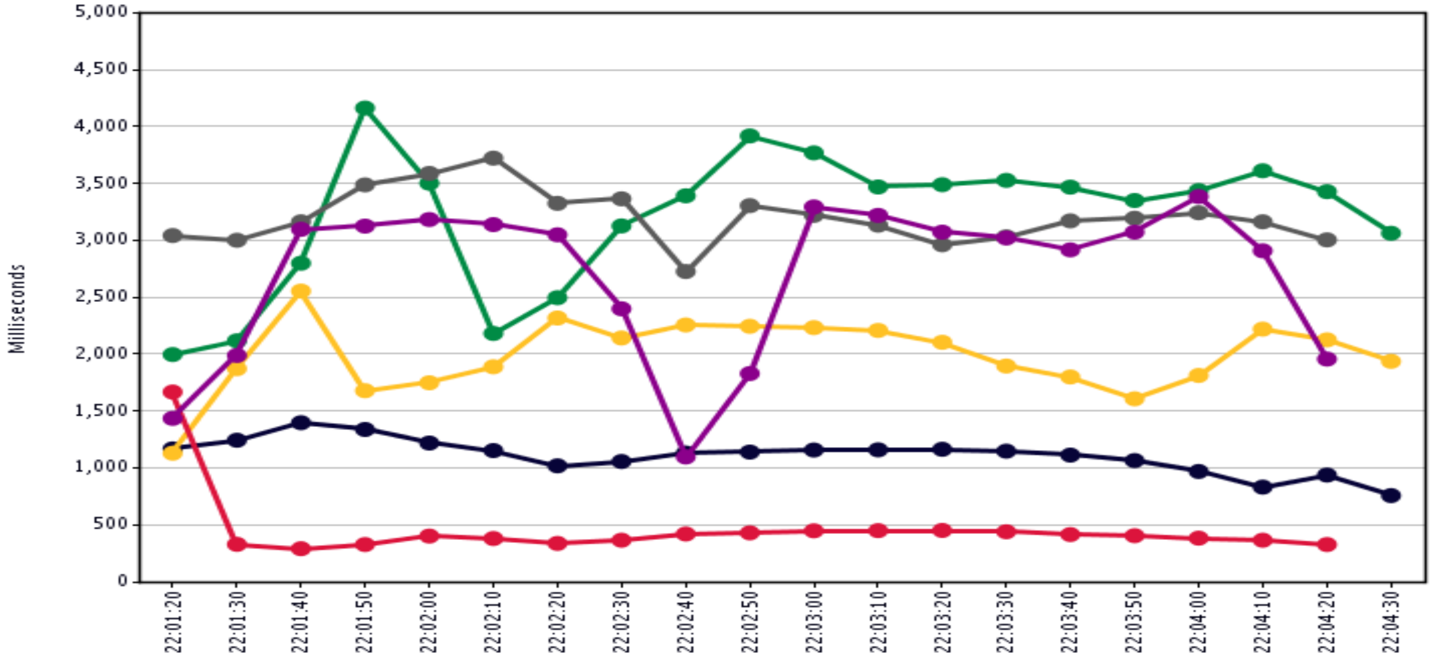
Example

Walk through query processing for three scenarios

Example

Walk through query processing for three scenarios

1. Chart Response Time Per Cluster

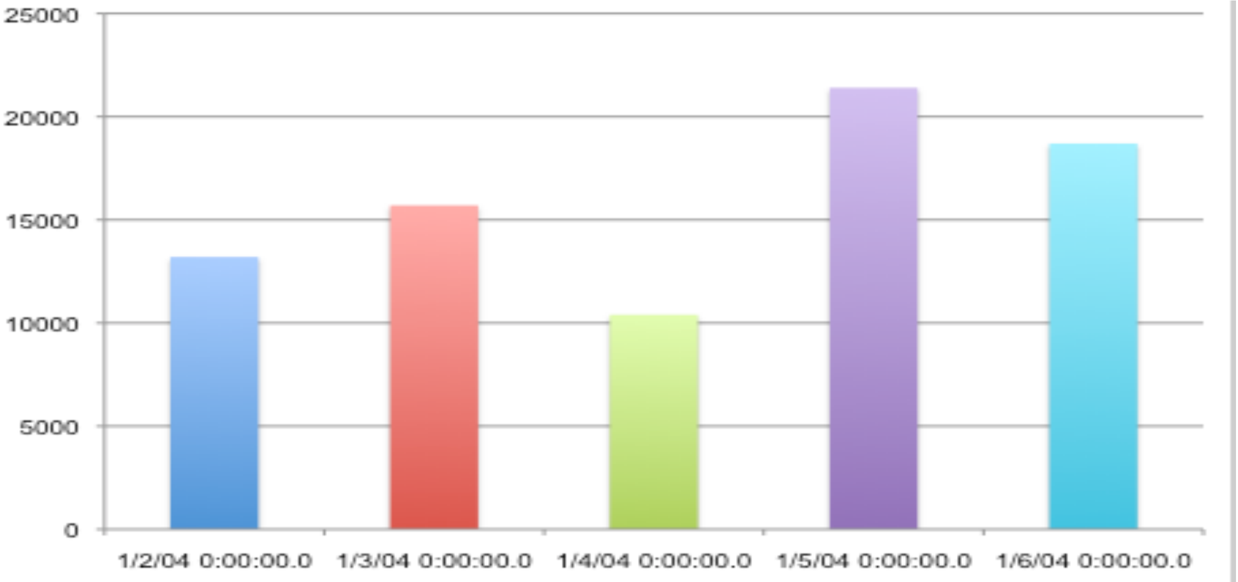


Example

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')

Row Key Ranges	
HOST	DATE

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	
HOST	DATE

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	
HOST	DATE

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	
HOST	DATE
sf1	

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	
HOST	DATE
sf1	
sf3	

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')

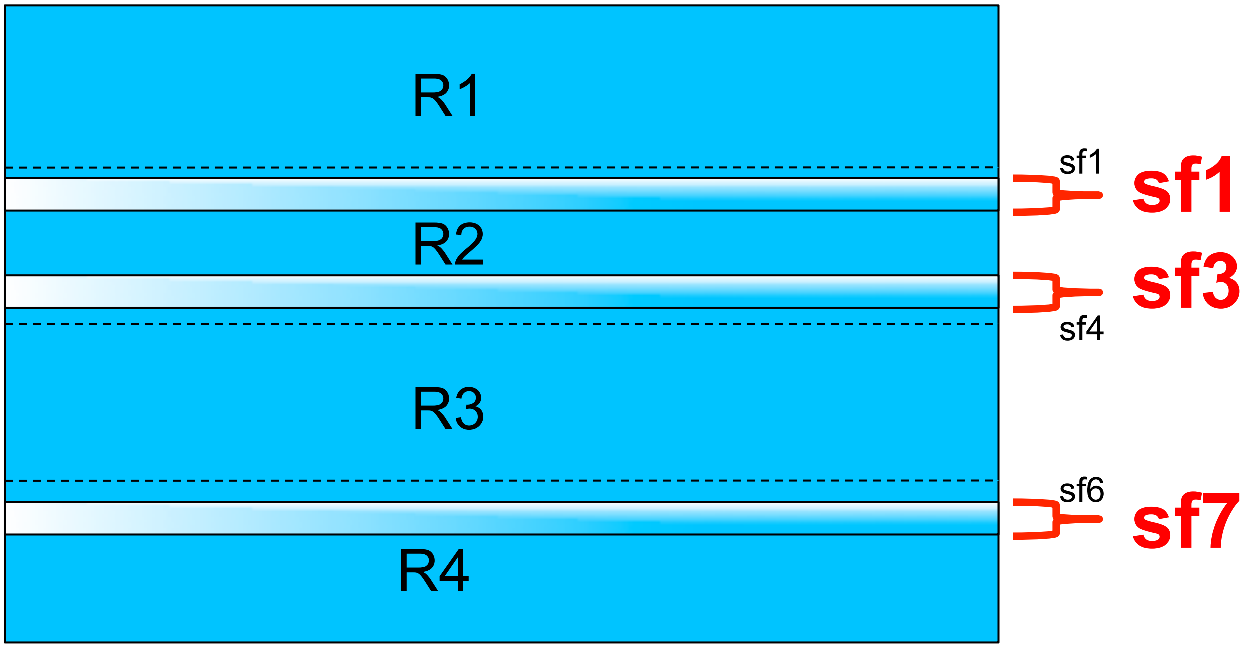
Row Key Ranges	
HOST	DATE
sf1	
sf3	
sf7	

Step 1: Client
Identify Row Key Ranges from Query

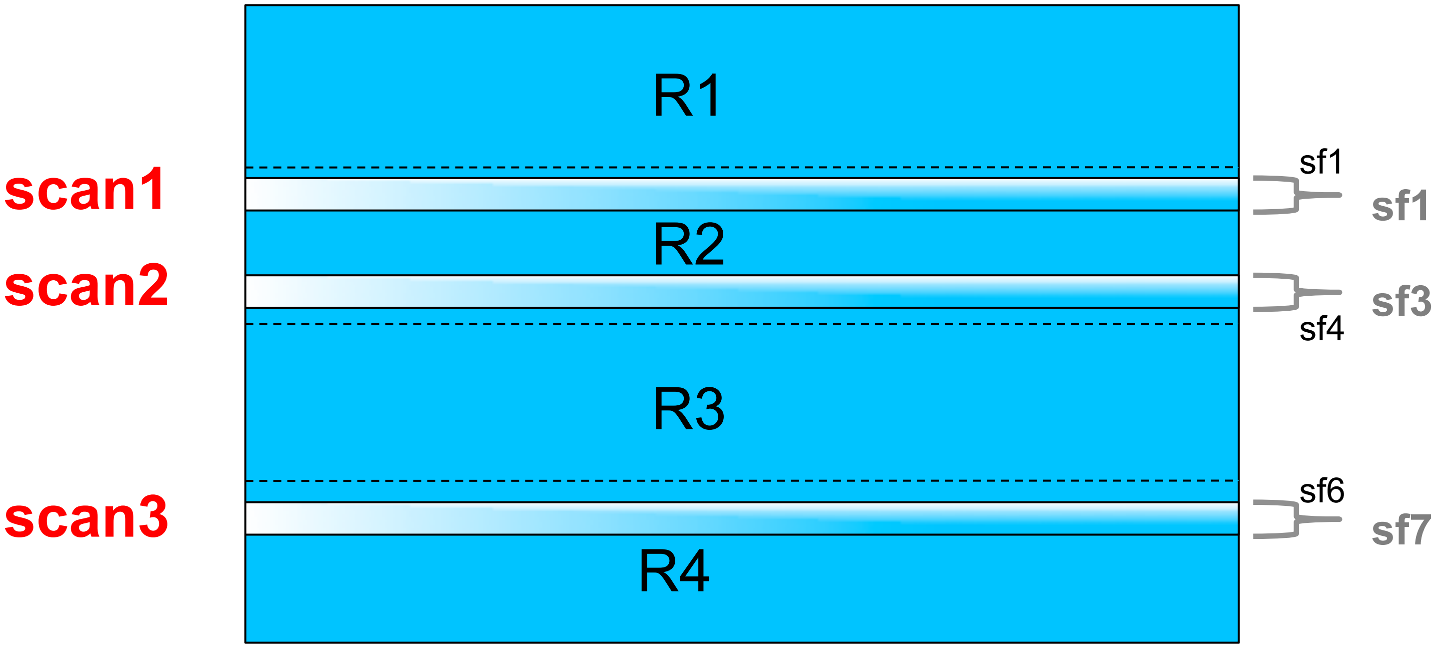
SELECT substr(host,1,3), trunc(date,'DAY'), avg(response_time)
FROM server_metrics
WHERE date > CURRENT_DATE() – 7
AND substr(host, 1, 3) IN ('sf1', 'sf3', 'sf7')
GROUP BY substr(host, 1, 3), trunc(date,'DAY')

Row Key Ranges	
HOST	DATE
sf1	t_1 - *
sf3	
sf7	

Step 2: Client
Overlay Row Key Ranges with Regions



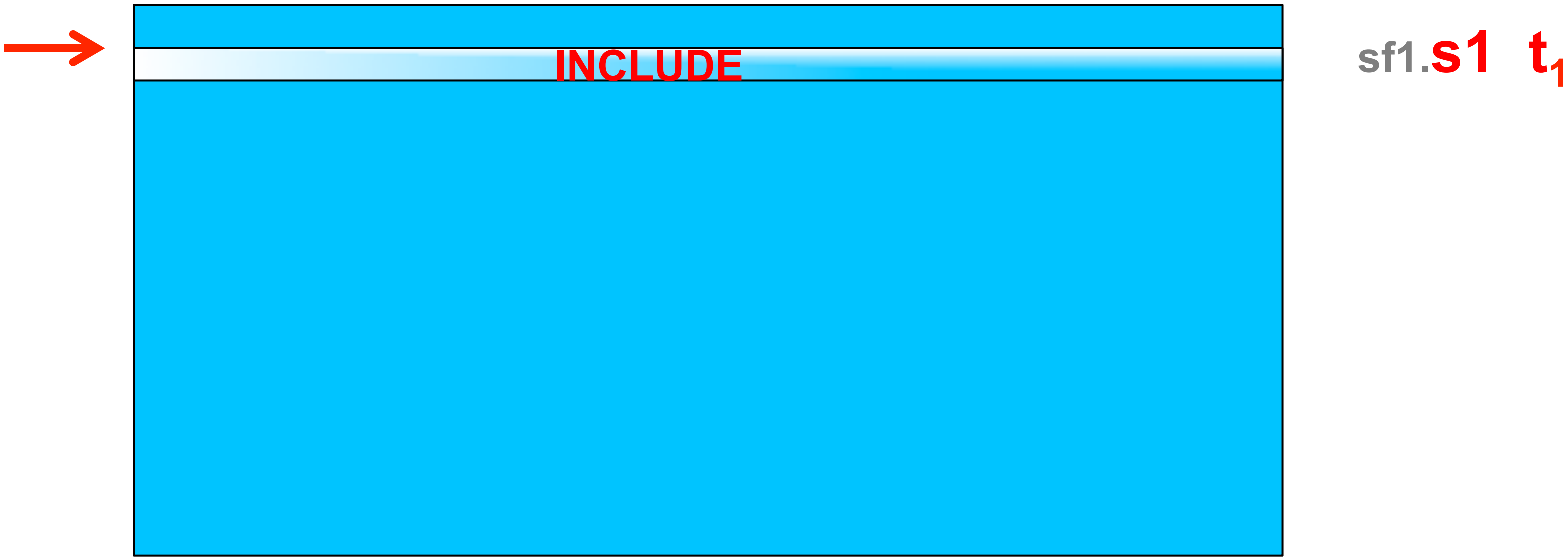
Step 3: Client
Execute Parallel Scans



Step 4: Server
Filter using Skip Scan

sf1.s1 t_0

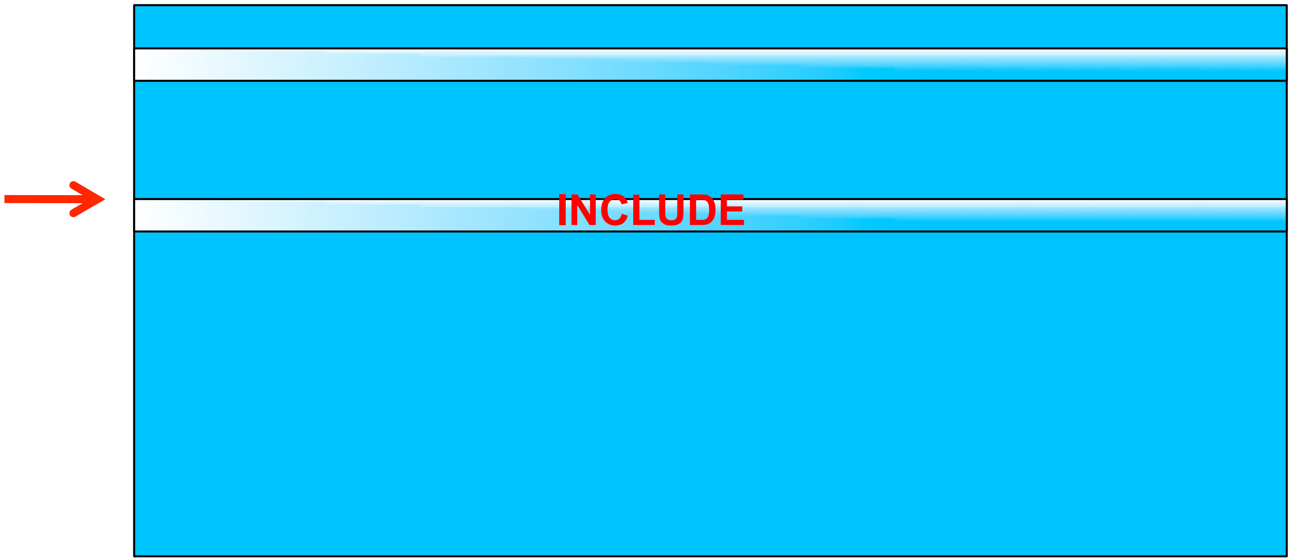
Step 4: Server Filter using Skip Scan



Step 4: Server
Filter using Skip Scan

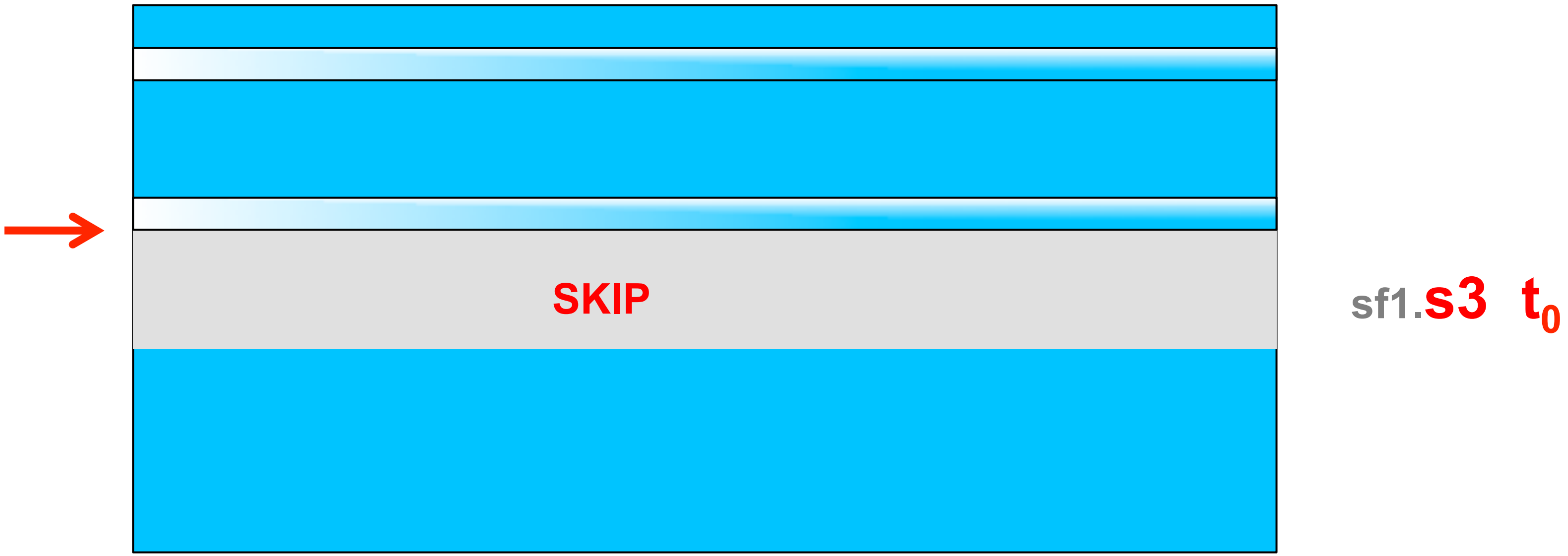
sf1.s2 t_0

Step 4: Server Filter using Skip Scan

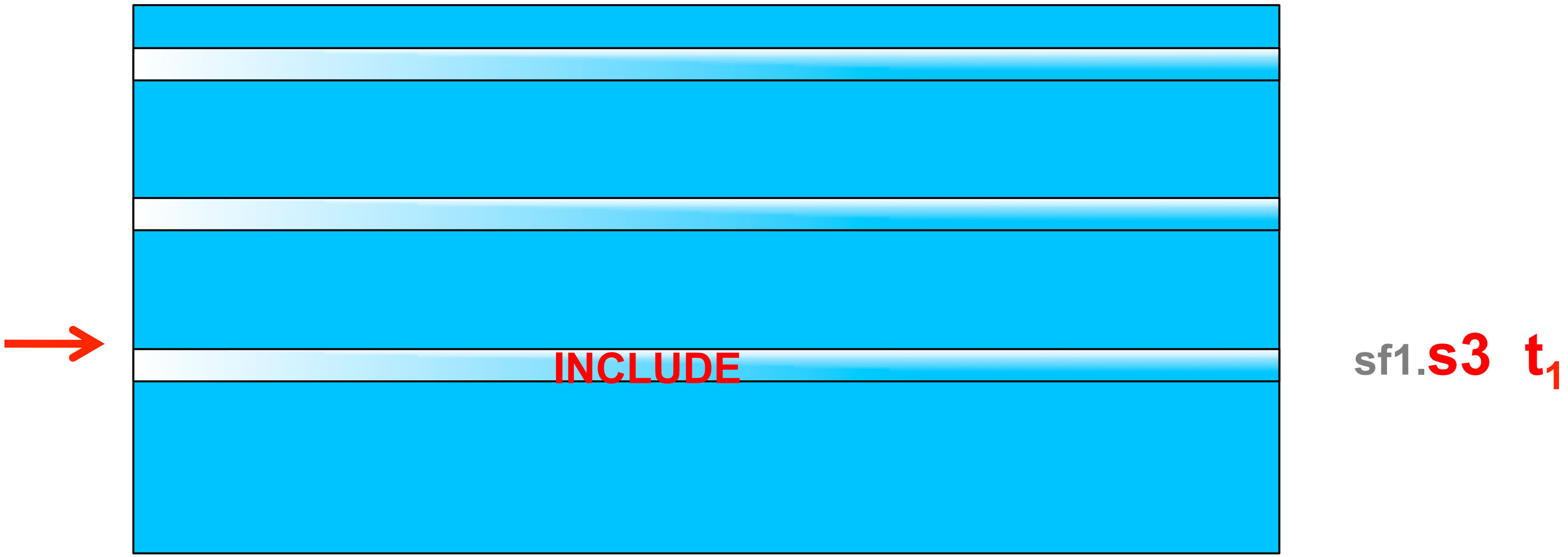


sf1.**s2** t_1

Step 4: Server Filter using Skip Scan



Step 4: Server
Filter using Skip Scan

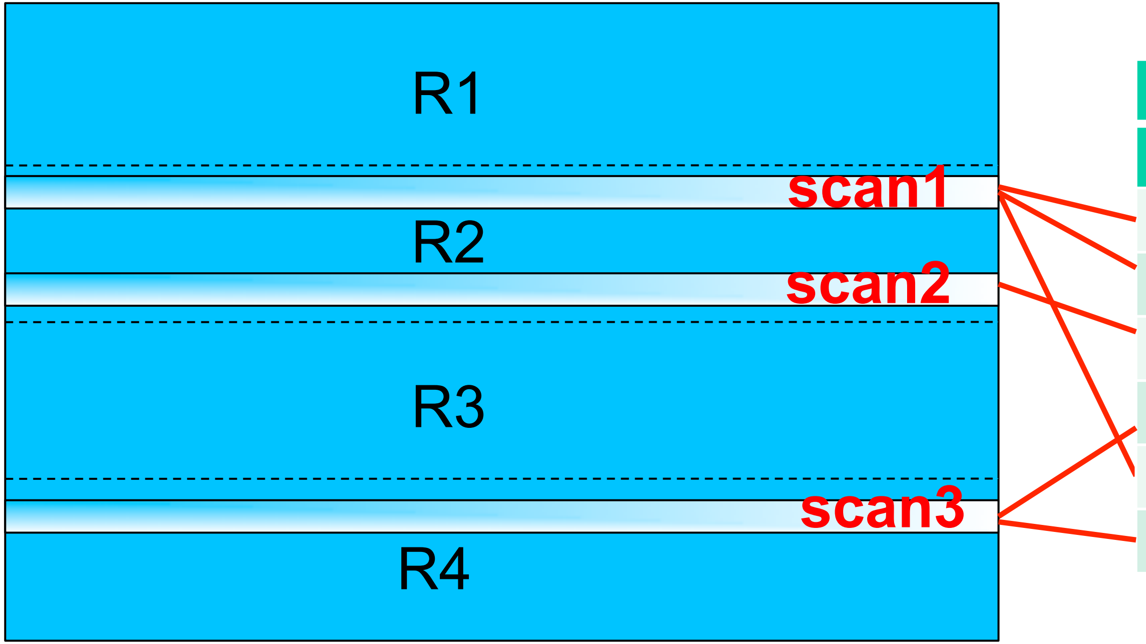


Step 5: Server Intercept Scan in Coprocessor

SERVER METRICS	
HOST	DATE
sf1.s1	Jun 2 10:10:10.234
sf1.s2	Jun 3 23:05:44.975
sf1.s2	Jun 9 08:10:32.147
sf1.s3	Jun 1 11:18:28.456
sf1.s3	Jun 3 22:03:22.142
sf1.s4	Jun 1 10:29:58.950
sf1.s4	Jun 2 14:55:34.104
sf1.s4	Jun 3 12:46:19.123
sf1.s5	Jun 8 08:23:23.456
sf1.s6	Jun 1 10:31:10.234

SERVER METRICS		
HOST	DATE	AGG
sf1	Jun 1	…
sf1	Jun 2	…
sf1	Jun 3	…
sf1	Jun 8	…
sf1	Jun 9	…

Step 6: Client
Perform Final Merge Sort



SERVER METRICS		
HOST	DATE	AGG
sf1	Jun 5	…
sf1	Jun 9	…
sf3	Jun 1	…
sf3	Jun 2	…
sf7	Jun 1	…
sf7	Jun 8	…

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

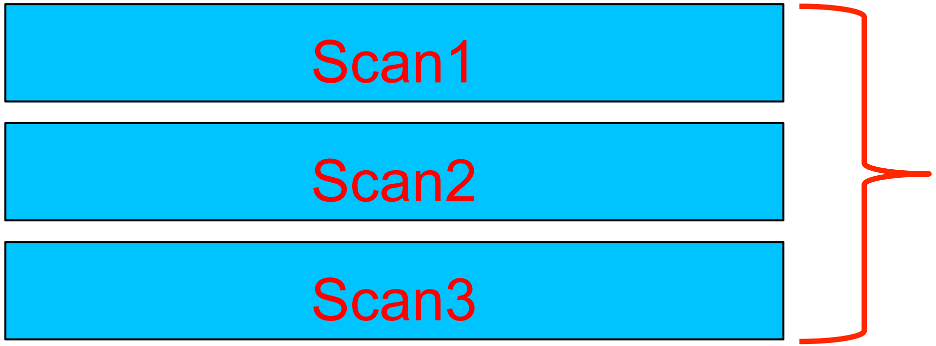
Scenario 2
Find 5 Longest GC Times

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

SERVER METRICS		
HOST	DATE	GC_TIME
sf1.s1	Jun 2 10:10:10.234	22123
sf1.s1	Jun 3 23:05:44.975	19876
sf1.s1	Jun 9 08:10:32.147	11345
sf1.s2	Jun 1 11:18:28.456	10234
sf1.s2	Jun 3 22:03:22.142	10111

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		
HOST	DATE	GC_TIME
sf1.s1	Jun 2 10:10:10.234	22123
sf1.s1	Jun 3 23:05:44.975	19876
sf1.s1	Jun 9 08:10:32.147	11345
sf1.s2	Jun 1 11:18:28.456	10234
sf1.s2	Jun 3 22:03:22.142	10111

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)
```

GC_TIME_INDEX	
GC_TIME	INTEGER
DATE	DATE
HOST	VARCHAR
RESPONSE_TIME	INTEGER

Row Key

Scenario 3
Find 5 Longest GC Times

CREATE INDEX gc_time_index
ON server_metrics (gc_time DESC, date DESC)
INCLUDE (host, response_time)

GC_TIME_INDEX	
GC_TIME	INTEGER
DATE	DATE
HOST	VARCHAR
RESPONSE_TIME	INTEGER

Key Value

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?