Apache Spark – Apache HBase Connector

Feature Rich and Efficient Access to HBase through Spark SQL

Weiqing Yang

Mingjie Tang

June, 2017

HORTONWORKS

POWERING THE FUTURE OF DATA™

About Authors

- Weiqing Yang
  - Contribute to Apache Spark, Apache Hadoop, Apache HBase, Apache Ambari
  - Software Engineer at Hortonworks
- Mingjie Tang
  - SparkSQL, Spark Mllib, Spark Streaming, Data Mining, Machine Learning
  - Software Engineer at Hortonworks
- … All Other SHC Contributors

Agenda

Motivation

Overview

Architecture & Implementation

Usage & Demo

HORTONWORKS®

Motivation

- Limited Spark Support in HBase Upstream
  - RDD level
  - But Spark Is Moving to DataFrame/Dataset

- Existing Connectors in DataFrame Level
  - Complicated Design
    - Embedding Optimization Plan inside Catalyst Engine
    - Stability Impact with Coprocessor
    - Serialized RDD Lineage to HBase
  - Heavy Maintenance Overhead

Overview

Apache Spark– Apache HBase Connector (SHC)

- Combine Spark and HBase
  - Spark Catalyst Engine for Query Plan and Optimization
  - HBase as Fast Access KV Store
  - Implement Standard External Data Source with Build-in Filter, Maintain Easily

- Full Fledged DataFrame Support
  - Spark SQL
  - Integrated Language Query

- High Performance
  - Partition Pruning, Data Locality, Column Pruning, Predicate Pushdown
  - Use Spark UnhandledFilters API
  - Cache Spark HBase Connections

Data Coder & Data Schema

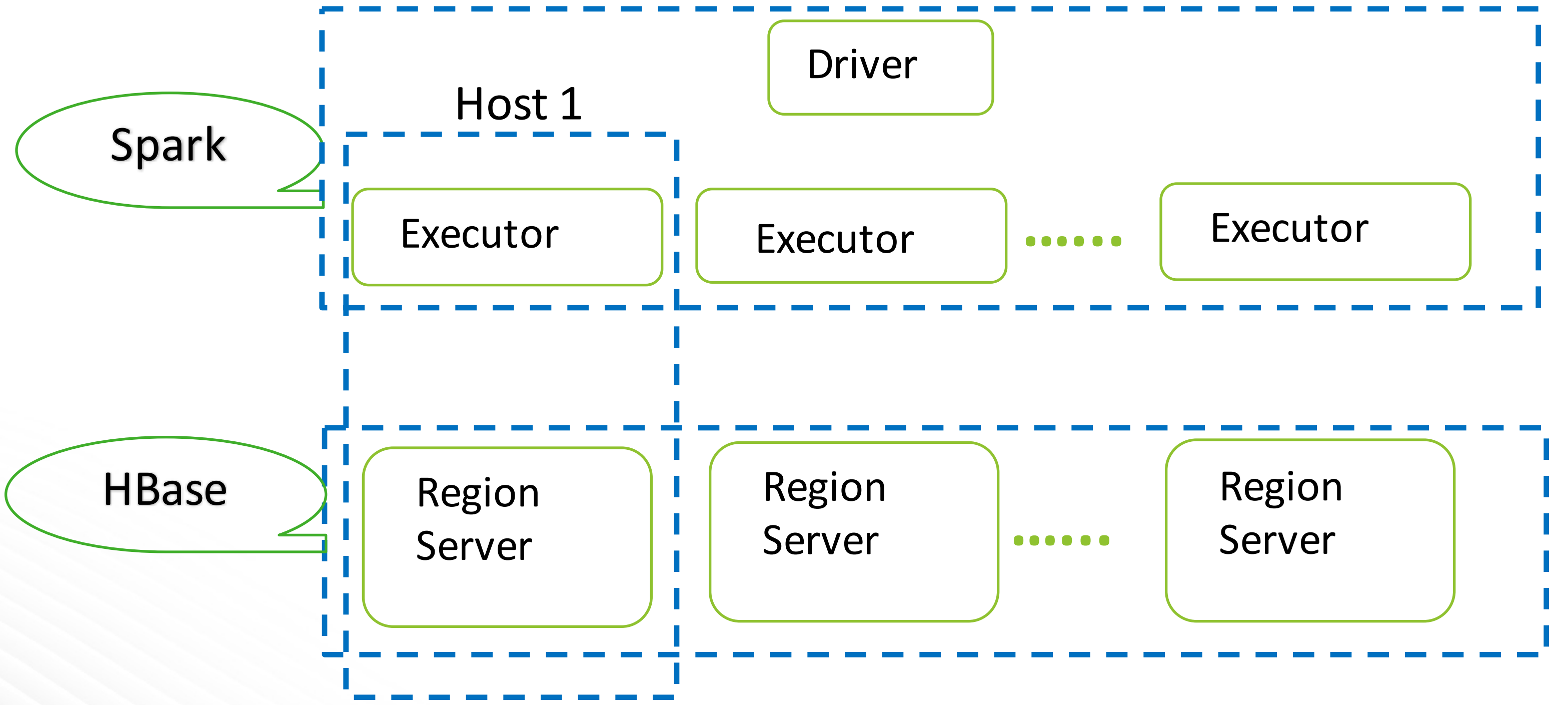
- Support Different Data Coders
  - PrimitiveType: Native Support Java Primitive Types
  - Avro: Native Support Avro Encoding/Decoding
  - Phoenix: Phoenix Encoding/Decoding
  - Plug-In Data Coder
  - Can Run on the Top of Existing HBase Tables

- Support Composite Key
  - def cat = s"""{

```
|"table":{"namespace":"default", "name":"shcExampleTable", "tableCoder":"Phoenix"},
|"rowkey":"key1:key2",
|"columns":{
  |"col00":{"cf":"rowkey", "col":"key1", "type":"string"},
  |"col01":{"cf":"rowkey", "col":"key2", "type":"int"},
  …
  …
```

Architecture & Implementation

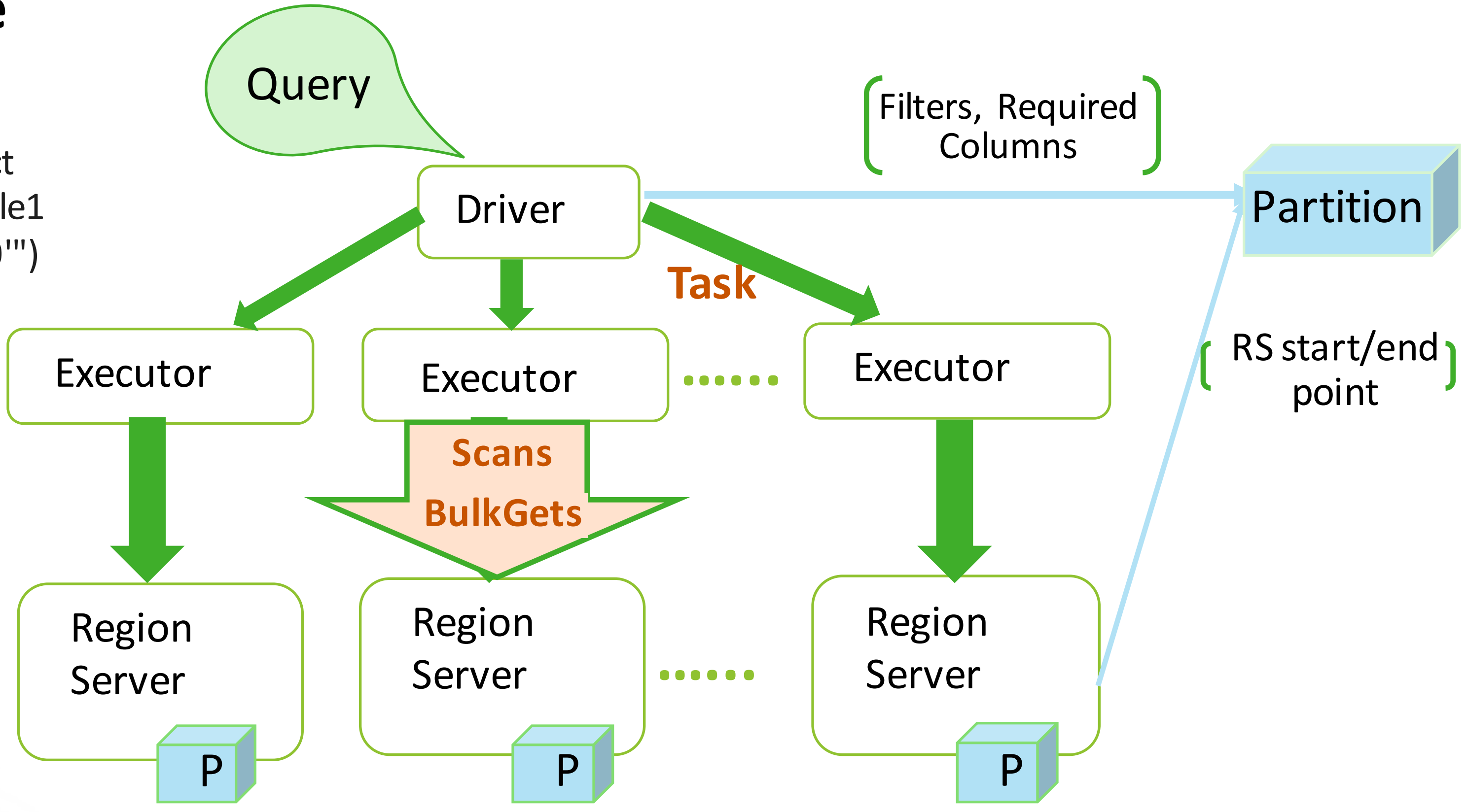
Architecture



Picture 1. SHC architecture

Architecture

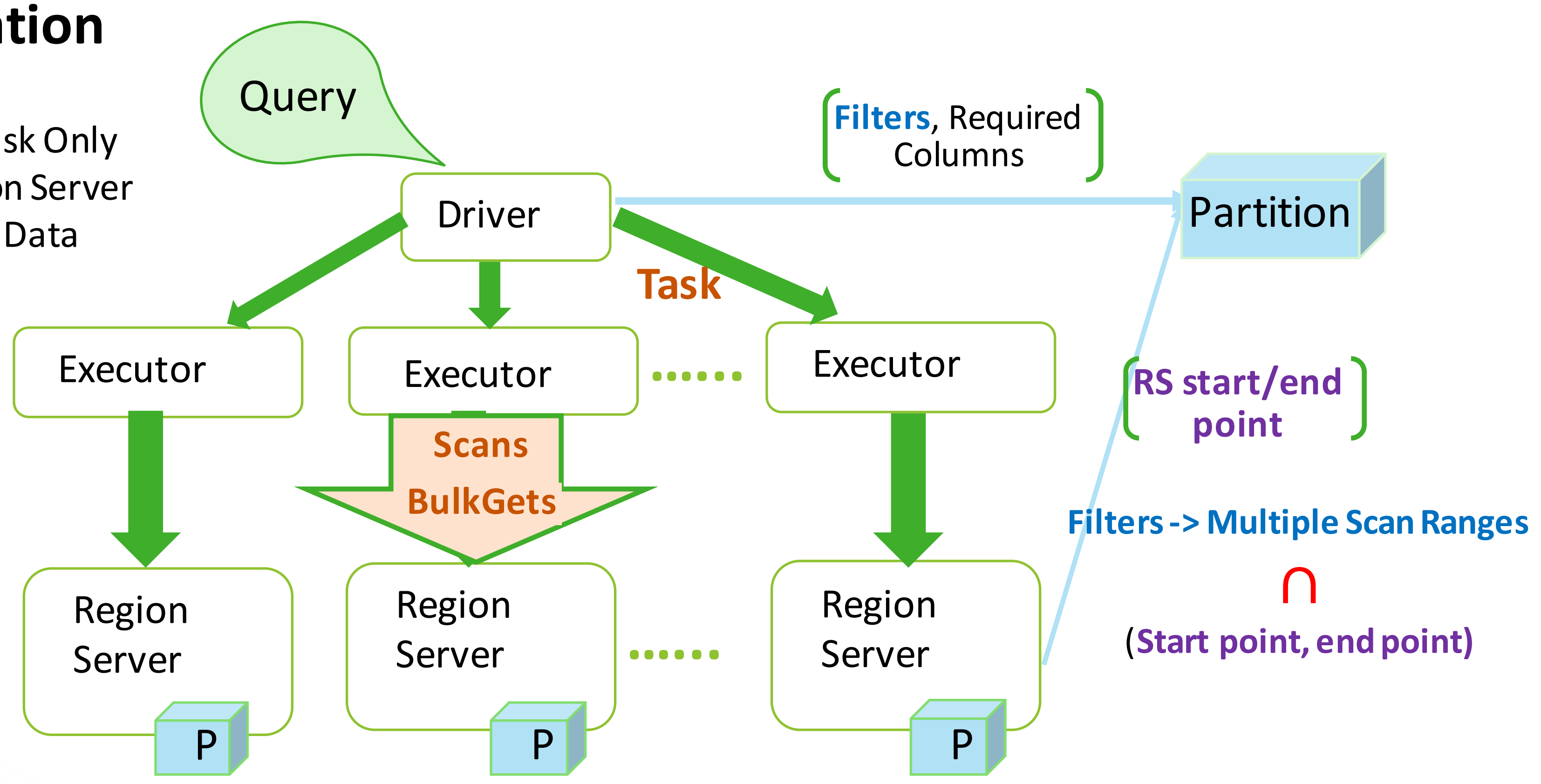
sqlContext.sql("select count(col1) from table1 where key < 'row050'")



Picture 1. SHC architecture

Implementation

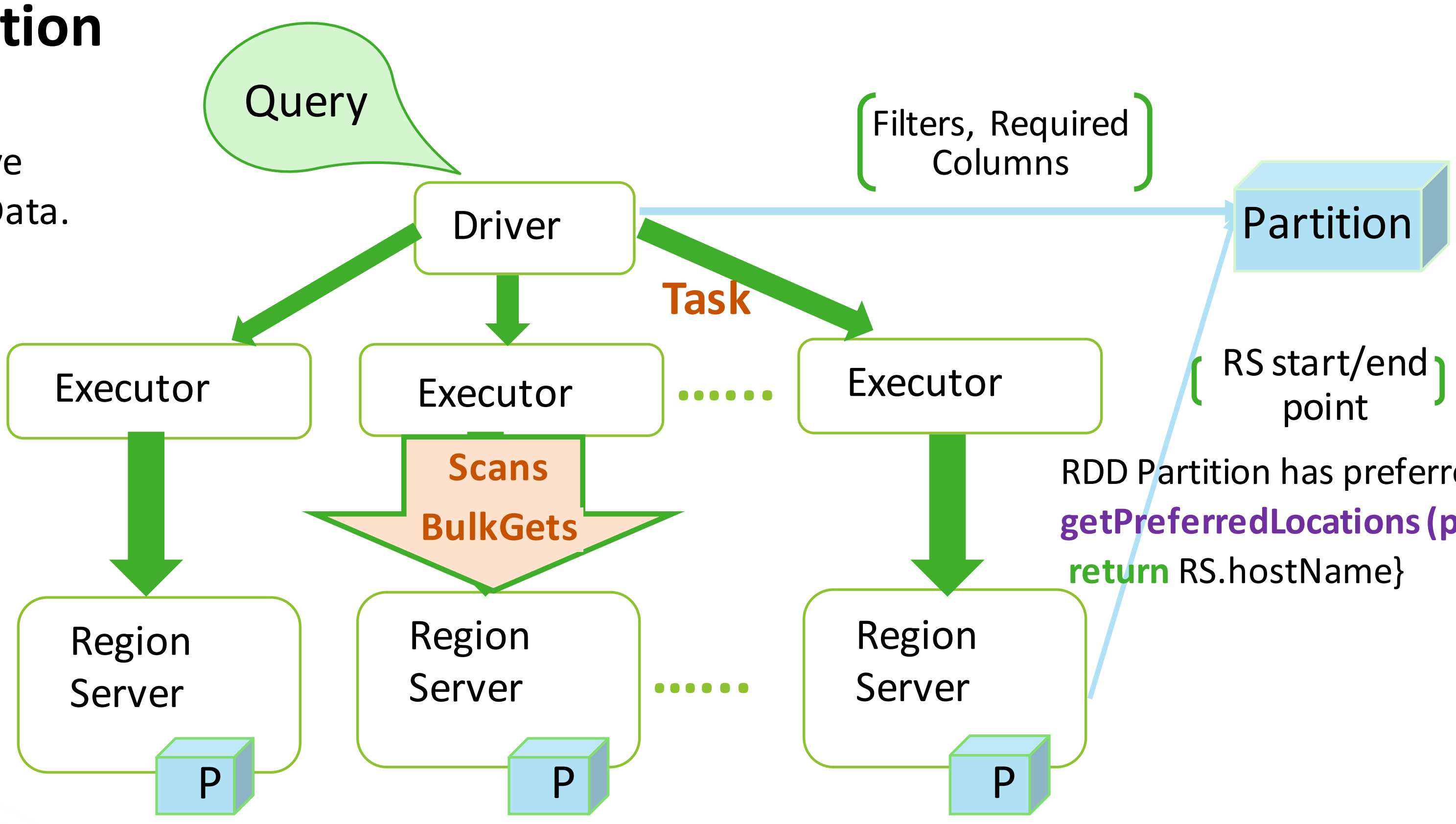
Partition Pruning: Task Only Performed in Region Server Holding Requested Data



Picture 1. SHC architecture

Implementation

Data Locality: Move Computation to Data.

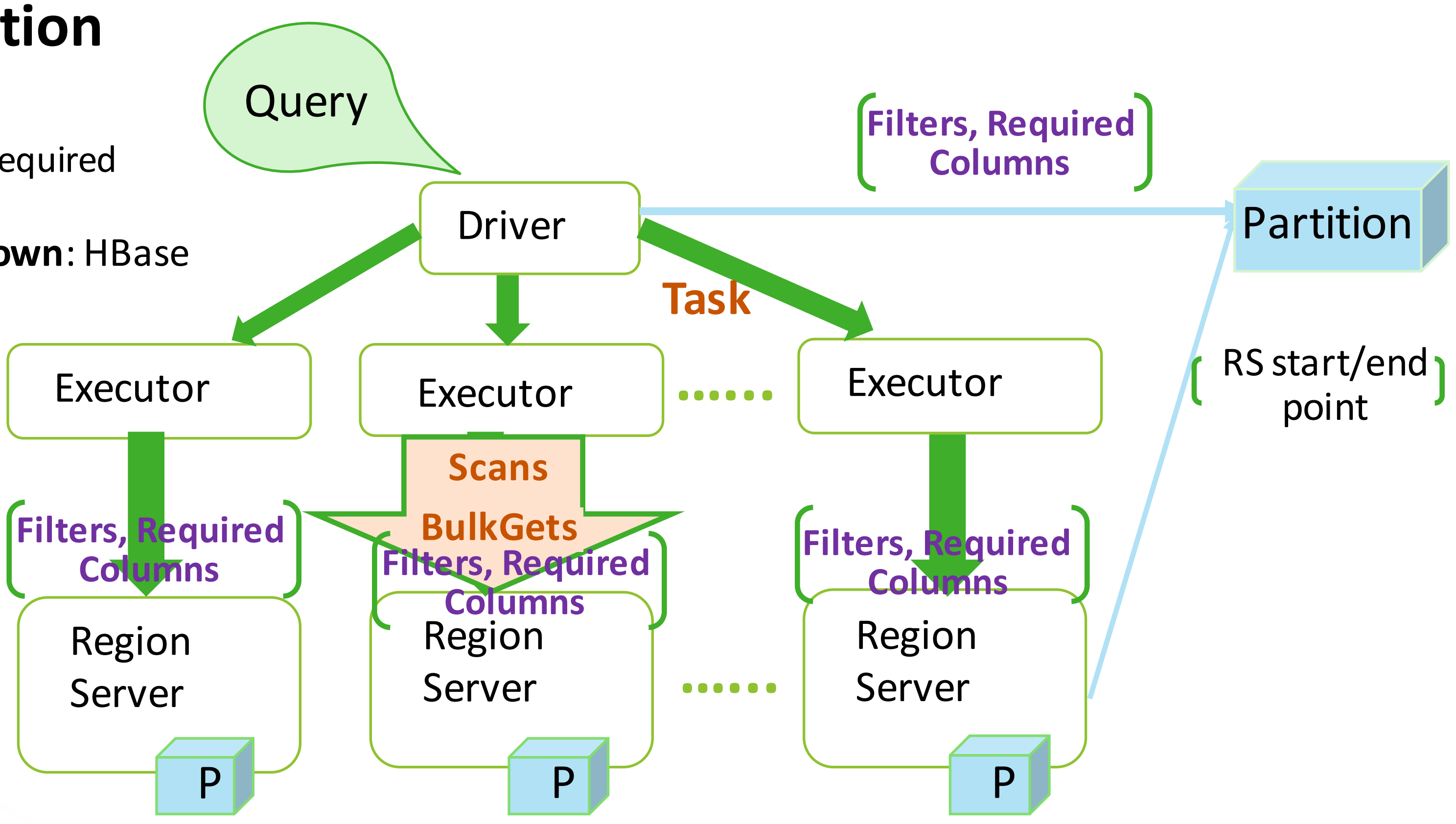


RDD Partition has preferred location:
getPreferredLocations(partition) {
return RS.hostName}

Picture 1. SHC architecture

Implementation

Column Pruning: Required Column
Predicate Pushdown: HBase built-in Filters

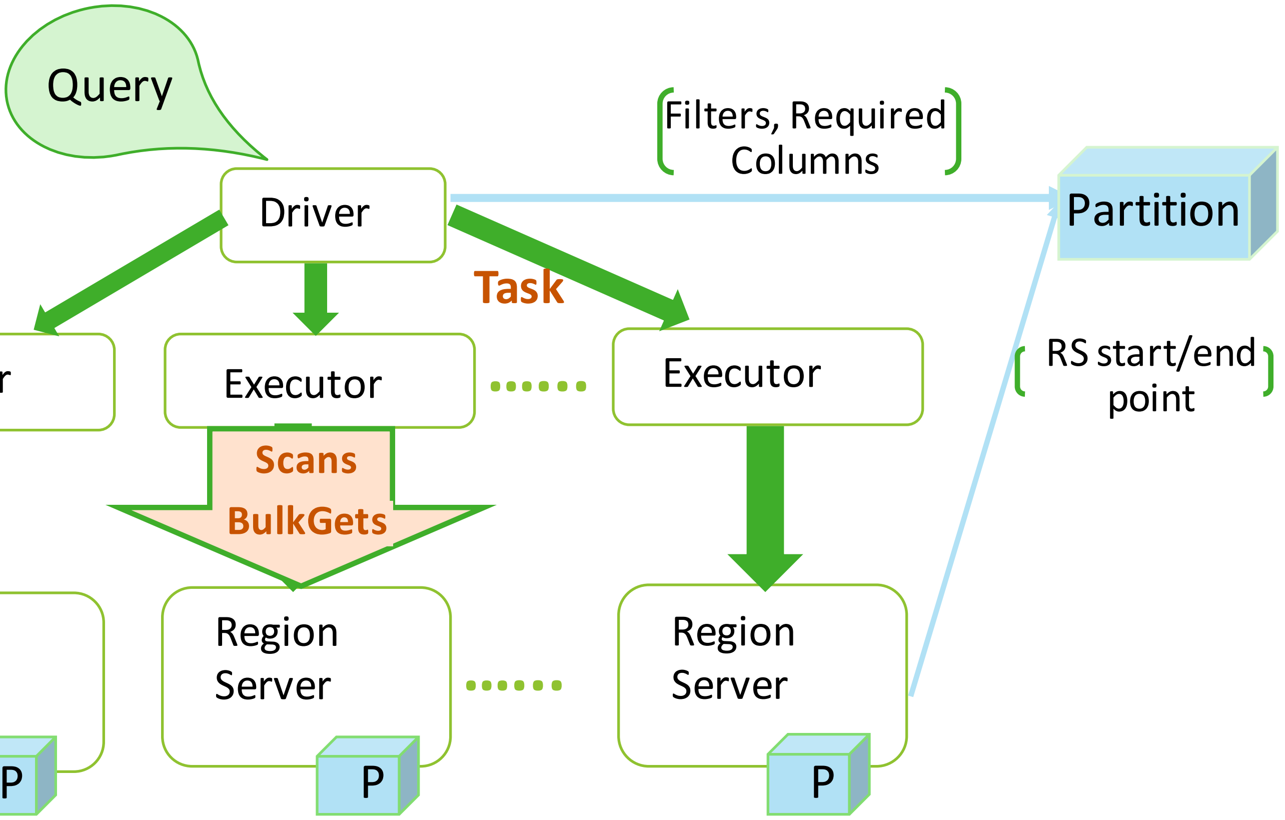


Picture 1. SHC architecture

Implementation

Scan and BulkGets: Grouped by region server.

WHERE column > x and column < y for **scan** and *WHERE column = x* for **get**.



Picture 1. SHC architecture

Usage & Demo

How to Use SHC?

- Github
  - https://github.com/hortonworks-spark/shc

- SHC Examples
  - https://github.com/hortonworks-spark/shc/tree/master/examples

- Apache HBase Jira
  - https://issues.apache.org/jira/browse/HBASE-14789

Demo

- Interactive Jobs through Spark Shell
- Batch Jobs

HORTONWORKS®

Acknowledgement

- HBase Community & Spark Community
- All Spark-HBase Contributors, Zhan Zhang

Reference

- Hortonworks Public Repo
  - http://repo.hortonworks.com/content/repositories/releases/com/hortonworks/
- Apache Spark
  - http://spark.apache.org/
- Apache HBase
  - https://hbase.apache.org/

Thanks

Q & A

Emails:
wyang@hortonworks.com

HORTONWORKS®
POWERING THE FUTURE OF DATA™

BACKUP

HORTONWORKS®
POWERING THE FUTURE OF DATA™

Kerberos Cluster

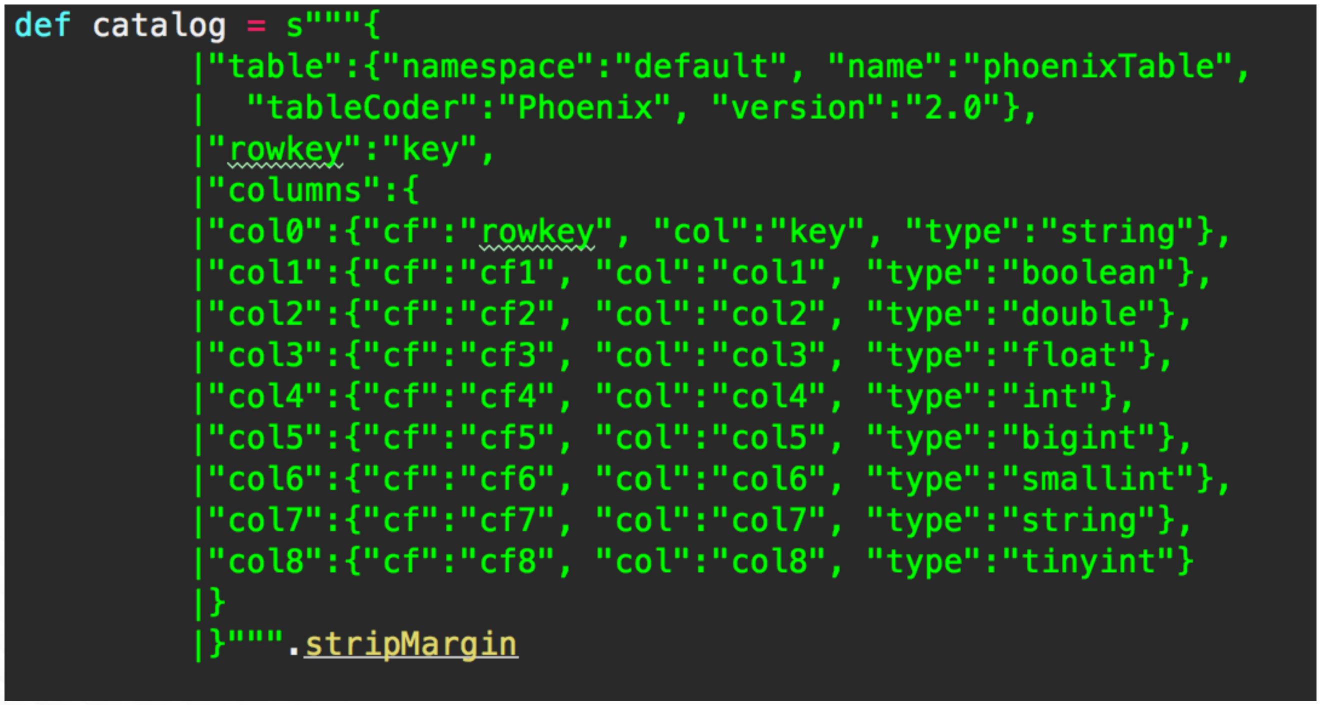
- Kerberos Ticket
  - kinit -kt foo.keytab foouser or Principle/Keytab

- Long Running Service
  - --principal, --keytab

- Multiple Secure HBase Clusters
  - Spark only Supports Single Secure HBase Cluster
  - Use SHC Credential Manager
  - Refer *LRJobAccessing2Clusters Example in github*

Usage

Define the catalog for the schema mapping:



```
def catalog = s"""{
         |"table":{"namespace":"default", "name":"phoenixTable",
         |  "tableCoder":"Phoenix", "version":"2.0"},
         |"rowkey":"key",
         |"columns":{
         |"col0":{"cf":"rowkey", "col":"key", "type":"string"},
         |"col1":{"cf":"cf1", "col":"col1", "type":"boolean"},
         |"col2":{"cf":"cf2", "col":"col2", "type":"double"},
         |"col3":{"cf":"cf3", "col":"col3", "type":"float"},
         |"col4":{"cf":"cf4", "col":"col4", "type":"int"},
         |"col5":{"cf":"cf5", "col":"col5", "type":"bigint"},
         |"col6":{"cf":"cf6", "col":"col6", "type":"smallint"},
         |"col7":{"cf":"cf7", "col":"col7", "type":"string"},
         |"col8":{"cf":"cf8", "col":"col8", "type":"tinyint"}
         |}
         |}""".stripMargin
```

Usage

- **Prepare the data and populate the HBase table**

```
val data = (0 to 255).map { i =>  HBaseRecord(i, “extra”)}

sc.parallelize(data).toDF.write.options(
   Map(HBaseTableCatalog.tableCatalog -> catalog, HBaseTableCatalog.newTable -> “5”))
   .format(“org.apache.spark.sql.execution.datasources.hbase”)
   .save()
```

Usage

- **Load the DataFrame**

```
def withCatalog(cat: String): DataFrame = {
   sqlContext
       .read
       .options(Map(HBaseTableCatalog.tableCatalog->cat))
       .format(“org.apache.spark.sql.execution.datasources.hbase”)
       .load()
}

val df = withCatalog(catalog)
```

HORTONWORKS®

Usage

- **Query**

Language integrated query:

```
val s = df.filter((($"col0” <= “çrow050” && $”col0” > “row040”) ||
   $”col0” === “row005” && ($”col4” === 1 || $”col4” === 42))
   .select(“col0”, “col1”, “col4”)
```

SQL:

```
val s = df.filter((($”col0” <= “row050” && $”col0” > “row040”)
df.registerTempTable(“table”)
sqlContext.sql(“select count(col1) from table”).show
```

HORTONWORKS®

Usage

- **Work with different data sources**

```
// Part 1: write data into Hive table and read data from it
val df1 = sql("SELECT * FROM shcHiveTable")

// Part 2: read data from Hbase table
val df2 = withCatalog(cat)

// Part 3: join the two dataframes
val s1 = df1.filter($"key" <= "40").select("key", "col1")
val s2 = df2.filter($"key" <= "20" && $"key" >= "1").select("key", "col2")
val result = s1.join(s2, Seq("key"))
result.show()
```

HORTONWORKS®