hosted by Alibaba Group 阿里巴巴集团 APACHE HBASE

Scaling 30 TB's of Data Lake with Apache HBase and Scala DSL at Production

Chetan Khatri

Who Am I

Lead - Data Science, Technology Evangelist @ Accion labs India Pvt. Ltd.
Contributor @ Apache Spark, Apache HBase, Elixir Lang, Spark HBase Connectors.
Co-Authored University Curriculum @ University of Kachchh, India.

Data Engineering @: Nazara Games, Eccella Corporation.
Advisor - Data Science Lab, University of Kachchh, India.
M.Sc. - Computer Science from University of Kachchh, India.

Agenda

01 What is Apache HBase

02 Why Apache HBase

03 Apache Spark and Scala

04 Apache Spark HBase Connector

05 Case Study: Retail Analytics
Architecturing Fast Data Processing Platform to Scale 30 TB Data in Production

What is Apache HBase

Source: https://hbase.apache.org/

- Column-oriented NoSQL
- Non-relational
- Distributed database build on top of HDFS.
- Modeled after Google's BigTable.
- Built for fault-tolerant application with billions/ trillions of rows and millions of columns.
- Very low latency and near real-time random reads and random writes.
- Replication, end-to-end checksums, automatic rebalancing with HDFS.
- Compression
- Bloom filters
- MapReduce over HBase data.
- Best at fetching rows by key, scanning ranges of rows with ordered partitioning.

What is Apache Spark ?

Source: https://spark.apache.org/

Structured Data / SQL - Spark SQL	Graph Processing - GraphX
Machine Learning - MLlib	Streaming - Spark Streaming, Structured Streaming

What is Scala

- Scala is a modern multi-paradigm programming language designed to express common programming patterns in a concise, elegant, and type-safe way.
- Scala is object-oriented
- Scala is functional
- Strongly typed, Type Inference
- Higher Order Functions
- Lazy Computation

Source: www.scala-lang.org

Case Story: Retail Analytics

Architecting Fast Data Processing Platform to Scale 30 TB of Data in Production

Use cases in Retail Analytics:

Business: explain the who, what, when, where, why and how they are doing Retailing.

- What is selling as compared to what was being ordered.
- Effective promotions - right promotions at right outlet and right time.
- What types of Cigarette consumers are shopping in your outlets ?
  - Gives smoking patterns in specific geography, predict demand on supply.
- What are the purchasing patterns of your consumers ?
  - are they purchasing Pizza and Ice cream together ?
  - are they purchasing multiple Instant food products with soda together ?
- Time Series problem - year, month, day of year, week of year to Identify which brands are not getting sold at specific geography, so it can be swap to other store.

Case Story: Retail Analytics - Scale

Challenges

- Weekly Data refresh, Daily Data refresh batch Spark / Hadoop job execution failures with unutilized Spark Cluster.
- Scalability of massive data:
  - ~4.6 Billion events on every weekly data refresh.
  - Processing historical data: one time bulk load ~30 TB of data / ~17 billion transactional records.
- Linear / sequential execution mode with broken data pipelines.
- Joining ~17 billion transactional records with skewed data nature.
- Data deduplication - outlet, item at retail.

Using HBase as a MDM System

MDM - Master Data Management

1. HBase Driven Data Deduplication Algorithms

Example,

- Outlet Matching
- Item Matching
- Address Matching
- Brand Matching

2. Abbreviation Standardization

Example,

- UOM Standardization
- Outlet Name, Address Standardization
- UPC Standardization

UOM	Quantity
PACK	2
2PACK	NA
2PK	

Using HBase as a MDM System

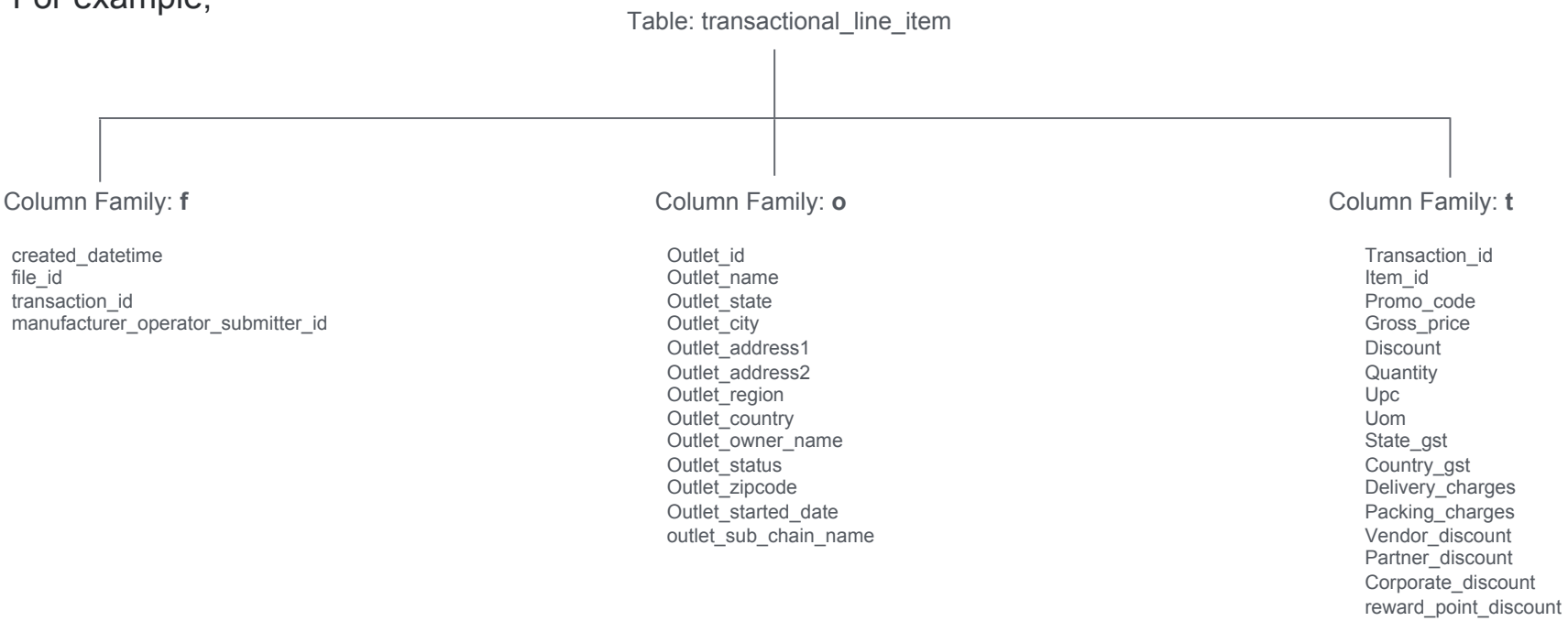
Data Deduplication Problem Retail Analytics !

Examples,

- You may find Item with same UPC code.
- You may find Outlet with same Outlet number.
- What if UPC Code gets upgraded from 10 Digits to 14 Digits. (Update everywhere, needs faster update.)

HBase - NoSQL, Denormalized columnar schema model

For example,



Case Story: Retail Analytics - Scale

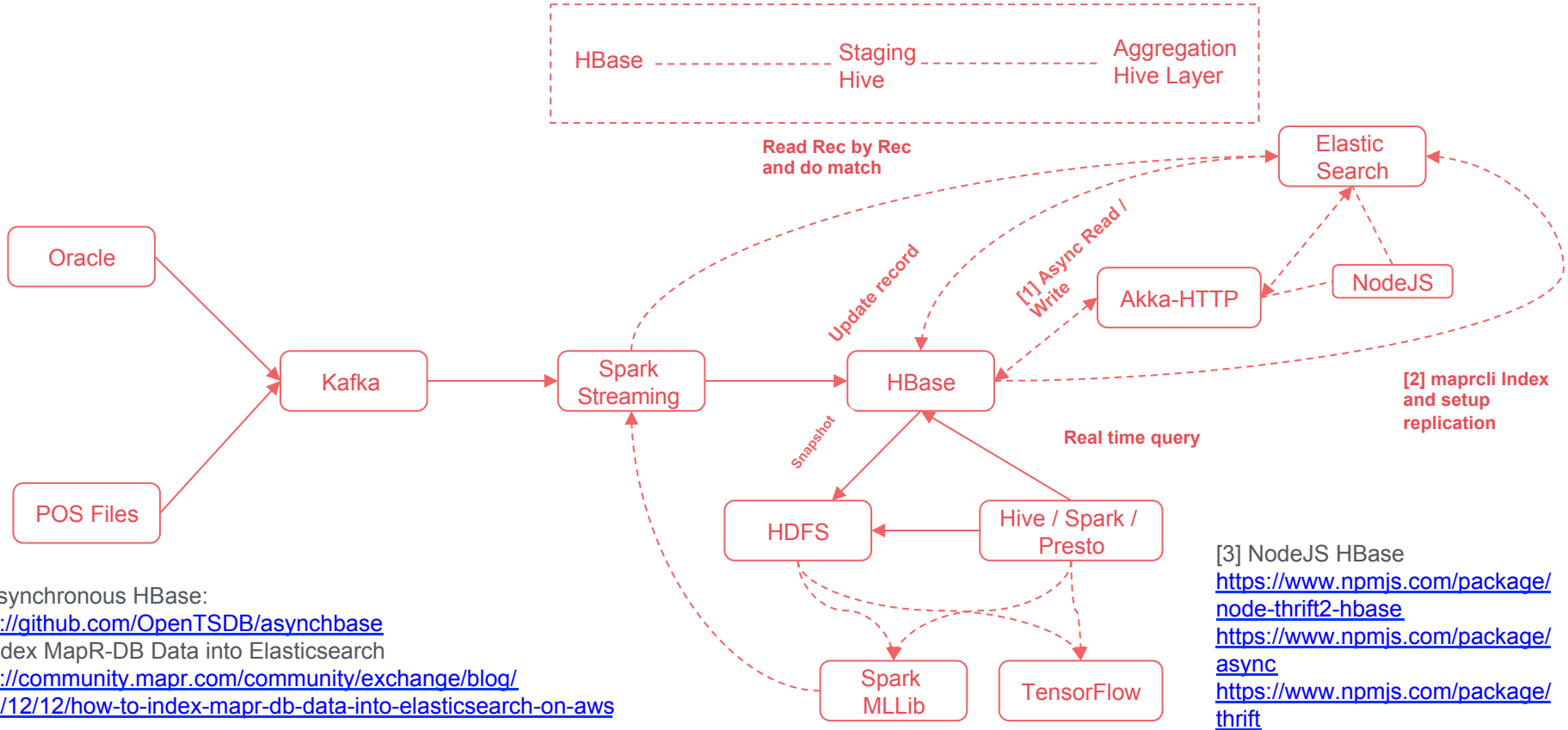
- 5x performance improvements by re-engineering entire data lake to analytical engine pipeline's.
- Proposed highly concurrent, elastic, non-blocking, asynchronous architecture to save customer's ~22 hours runtime (~8 hours from 30 hours) for 4.6 Billion events.
- 10x performance improvements on historical load by under the hood algorithms optimization on 17 Billion events (Benchmarks ~1 hour execution time only)
- Master Data Management (MDM) - Deduplication and fuzzy logic matching on retails data(Item, Outlet) improved elastic performance.
    - Using HBase as a Master Data Management (MDM) System.

Case Story: Retail Analytics

How ?

Data Platform

- Infrastructure Architecture



[1] Asynchronous HBase:
https://github.com/OpenTSDB/asynchbase
[2] Index MapR-DB Data into Elasticsearch
https://community.mapr.com/community/exchange/blog/2016/12/12/how-to-index-mapr-db-data-into-elasticsearch-on-aws

[3] NodeJS HBase
https://www.npmjs.com/package/node-thrift2-hbase
https://www.npmjs.com/package/async
https://www.npmjs.com/package/thrift

AsyncHBase Build.sbt with Akka HTTP

build.sbt

```
name := "async-hbase-demo"

version := "0.1"

scalaVersion := "2.11.8"

libraryDependencies += "com.typesafe.akka" %% "akka-actor" % "2.4.15"
libraryDependencies += "com.typesafe.akka" %% "akka-http-core" % "10.0.2"
libraryDependencies += "com.typesafe.akka" %% "akka-http-spray-json-experimental" % "2.4.11.1"
libraryDependencies += "org.apache.kafka" % "kafka-clients" % "0.10.2.0"
libraryDependencies += "io.spray" %% "spray-json" % "1.3.4"
libraryDependencies += "com.github.fommil" %% "spray-json-shapeless" % "1.4.0"
libraryDependencies += "org.scalamock" %% "scalamock-scalatest-support" % "3.6.0" % Test
libraryDependencies += "org.hbase" % "asynchbase" % "1.7.2"
```

Data Processing Infrastructure

- MapR Distribution - http://archive.mapr.com/releases/ecosystem-5.x/redhat/
- Data Lake - Apache HBase 0.98.12
- EDW / Analytical Data store - Apache Hive 1.2.1
- Unified Execution Engine - Apache Spark 2.0.1
- Distributed File storage - MapR-FS
- Queueing mechanism - Apache Kafka
- Streaming - HTTP Akka + scala 2.11 , Spark Streaming
- Reporting Database - PostgreSQL 9.x
- Legacy Database - Oracle 9
- Workflow Management tool - BMC Control M

Rethink

- Fast Data Architectures. Unify, Simplify.

UNIFIED fast data processing engine that provides:

Spark HBase Connector.

hosted by Alibaba Group 阿里巴巴集团 APACHE HBASE

nerdammer / **spark-hbase-connector**

Watch 31 | Star 237 | Fork 90

Code | Issues 37 | Pull requests 2 | Projects 0 | Wiki | Insights

Connect Spark to HBase for reading and writing data with ease

hbase spark

119 commits | **2** branches | **5** releases | **5** contributors | Apache-2.0

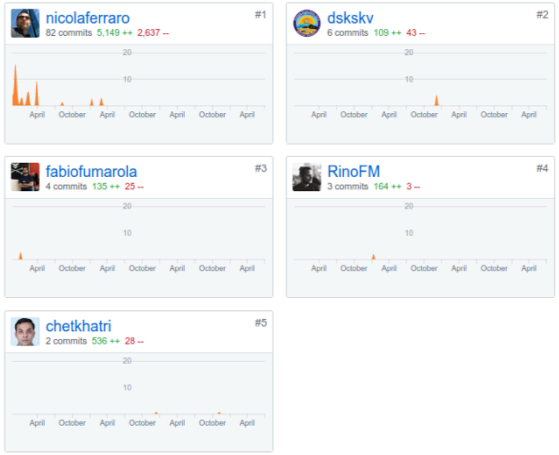
Branch: **master** | **New pull request** | **Create new file** | **Upload files** | **Find file** | **Clone or download**

nicolaferraro Merge pull request #65 from chetkhatri/master ··· Latest commit 92d2507 on 19 Dec 2017

examples/spark-2-0-scala-2-11-sample	Examples are added and coding practice improved	8 months ago
project	Examples are added and coding practice improved	8 months ago
scripts/ci	Upgrade to Spark 1.6.0, Hbase 1.0.3	3 years ago
src	Added tests on precedence	2 years ago
.gitignore	Publish to spark-packages	3 years ago
.travis.yml	Upgrade to Spark 1.6.0, Hbase 1.0.3	3 years ago
LICENSE.txt	Create LICENSE.txt	4 years ago
README.md	README.md markdown was broken	2 years ago
build.sbt	Fixed dependencies for spark-packages	2 years ago

https://github.com/nerdammer/spark-hbase-connector

Credit: Contributors



Spark HBase Connector.

It’s a Spark package connector written on top of Java HBase API. A simple and elegant way to write Spark - HBase Jobs. Powerful Functional Scala DSL integrated for Apache Spark.

Supports:

- Scala > 2.10
- Spark > 1.6
- HBase > 1.0

Spark HBase Connector.

Dependency in build.sbt - libraryDependencies += "it.nerdammer.bigdata" % "spark-hbase-connector_2.10" % "1.0.3"

The HBase Zookeeper quorum host can be set in multiple ways.

(1) Passing the host to the `spark-submit` command:

```
spark-submit --conf spark.hbase.host=thehost ...
```

Setting the HBase Host

(2) Using the hbase-site.xml file (in the root of your jar, i.e. `src/main/resources/hbase-site.xml`):

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
        <property>
                <name>hbase.zookeeper.quorum</name>
                <value>thehost</value>
        </property>

        <!-- Put any other property here, it will be used -->
</configuration>
```

Writing to HBase


```scala
package com.chetan.spark.sample
/**
  * Created by chetan on 28/1/17.
  */
import it.nerdammer.spark.hbase._
import org.apache.spark.sql.SparkSession
object Sample {
  def main(args: Array[String]): Unit = {

    val spark = SparkSession.builder().appName("HBaseReadDemo").getOrCreate()
    // This rdd is made of tuples like ("1", 2, "Hello") or ("27", 28, "Hello").
    // The first element of each tuple is considered the row id, the others will be assigned to columns.
    val rddHBase = spark.sparkContext.parallelize(1 to 100)
      .map(i => (i.toString, i+1, "Hello"))
    rddHBase.toHBaseTable("mytable")
      .toColumns("column1", "column2")
      .inColumnFamily("mycf")
      .save()
```

Reading from HBase


```scala
// let's read data which is written  in previous example
val hBaseMyTableRDD = spark.sparkContext.hbaseTable[(String, Int, String)]("mytable")
  .select("column1", "column2")
  .inColumnFamily("mycf")

// Now hBaseRDD contains all the data found in the table.
// Each object in the RDD is a tuple containing (in order) the row id,
// the corresponding value of column1 (Int) and column2 (String).

// val you: Nothing = null don't want the row id but, you only want to see the columns,
// just remove the first element from the tuple specs:
// This way, only the columns that you have chosen will be selected.
val hBaseRDD1 = spark.sparkContext.hbaseTable[(Int, String)]("mytable")
  .select("column1", "column2")
  .inColumnFamily("mycf")

// val don: Nothing = null't have to provide column family name as a prefix to column name
// for the provided column family at inColumnFamily(COLUMN_FAMILY_NAME) but for other
// columns you need to provide prefix with :(colon).
val hBaseRDD2 = spark.sparkContext.hbaseTable[(Int, String, String)]("mytable")
  .select("column1","columnfamily2:column2","columnfamily3:column3")
  .inColumnFamily("columnfamily1")
```

Filtering


```
45    // Filtering
46    // It is possible to filter the results by prefixes of row keys.
47    // Filtering also supports additional salting prefixes (see the salting section).
48    val rdd = spark.sparkContext.hbaseTable[(String, String)]("table")
49      .select("col")
50      .inColumnFamily("columnFamily")
51      .withStartRow("00000")
52      .withStopRow("00500")
```

Manage Empty Columns with Option[T]

```
53
54  // The example above retrieves all rows having a row key greater or equal to 00000 and lower than 00500.
55  // The options withStartRow and withStopRow can also be used separately.
56
57  // Managing Empty Columns, Empty columns are managed by using Option[T] types:
58  // You can use the Option[T] type every time you are not sure whether a given
59  // column is present in your HBase RDD.
60  val rdd1 = spark.sparkContext.hbaseTable[(Option[String], String)]("table")
61    .select("column1", "column2")
62    .inColumnFamily("columnFamily")
63
64  rdd1.foreach(t => {
65    if(t._1.nonEmpty) println(t._1.get)
66  })
```

Custom Mapping with Case Classes

```
311 case class Student( studentId: String,
312                     classId: String,
313                     semesterId: String,
314                     grNumber: String,
315                     firstName: String,
316                     middleName: String,
317                     lastName: String,
318                     motherName: String,
319                     address: String,
320                     stdFrom: String,
321                     stdDob: String,
322                     gender: String,
323                     reservationCategory: String,
324                     bloodGrp: String,
325                     contactNo: String,
326                     phNo: String,
327                     mail: String,
328                     addDte: String,
329                     addStd: String,
330                     fatherContact: String,
331                     fatherBusiness: String,
332                     fatherIncome: String,
333                     incomeCerti: String,
334                     casteCerti: String,
335                     lcEntry: String,
336                     resultEntry: String,
337                     entrance: String,
338                     lastSchool: String,
339                     year: String,
340                     remarks: String,
341                     leaveDate: String,
342                     leaveReason: String,
343                     progress: String,
344                     conduct: String,
345                     tryPass: String)
```

Custom Mapping with Case Classes ...

```
349  case class StudentClass (
350    classId: String,
351    classNo: String,
352    classDesc: String
353  )
354
355  case class Semester (
356    semesterId: String,
357    semesterNo: String,
358    semesterDesc: String
359  )
360
361  case class Subject (
362    subjectId: String,
363    semesterId: String,
364    subjectCode: String,
365    subjectName: String,
366    marks: Int,
367    grace: Int
368  )
369
370  case class ExamResult (
371    examId: String,
372    studentId: String,
373    classId: String,
374    subjectId: String,
375    subjectName: String,
376    marks: Int,
377    grace: Int,
378    total: Int,
379    grossMarks: Int,
380    totalGrace: Int,
381    totalMarks: Int
382  )
```

Implicit Reader


```
50    // Student : Custom one to one mapping for reading HBase table
51    implicit def studentsReader: FieldReader[Student] = new FieldReader[Student]{
52      override def map(data: HBaseData): Student = Student(
53        studentId = Bytes.toString(data.drop(1).head.getOrElse(null)),
54        classId = Bytes.toString(data.drop(2).head.getOrElse(null)),
55        semesterId = Bytes.toString(data.drop(3).head.getOrElse(null)),
56        grNumber = Bytes.toString(data.drop(4).head.getOrElse(null)),
57        firstName = Bytes.toString(data.drop(5).head.getOrElse(null)),
58        middleName = Bytes.toString(data.drop(6).head.getOrElse(null)),
59        lastName = Bytes.toString(data.drop(7).head.getOrElse(null)),
60        motherName = Bytes.toString(data.drop(8).head.getOrElse(null)),
61        address = Bytes.toString(data.drop(9).head.getOrElse(null)),
62        stdFrom = Bytes.toString(data.drop(10).head.getOrElse(null)),
63        stdDob = Bytes.toString(data.drop(11).head.getOrElse(null)),
64        gender = Bytes.toString(data.drop(12).head.getOrElse(null)),
65        reservationCategory = Bytes.toString(data.drop(13).head.getOrElse(null)),
66        bloodGrp = Bytes.toString(data.drop(14).head.getOrElse(null)),
67        contactNo = Bytes.toString(data.drop(15).head.getOrElse(null)),
68        phNo = Bytes.toString(data.drop(16).head.getOrElse(null)),
69        mail = Bytes.toString(data.drop(17).head.getOrElse(null)),
70        addDte = Bytes.toString(data.drop(18).head.getOrElse(null)),
71        addStd = Bytes.toString(data.drop(19).head.getOrElse(null)),
72        fatherContact = Bytes.toString(data.drop(20).head.getOrElse(null)),
73        fatherBusiness = Bytes.toString(data.drop(21).head.getOrElse(null)),
74        fatherIncome = Bytes.toString(data.drop(22).head.getOrElse(null)),
75        incomeCerti = Bytes.toString(data.drop(23).head.getOrElse(null)),
76        casteCerti = Bytes.toString(data.drop(24).head.getOrElse(null)),
77        lcEntry = Bytes.toString(data.drop(25).head.getOrElse(null)),
78        resultEntry = Bytes.toString(data.drop(26).head.getOrElse(null)),
79        entrance = Bytes.toString(data.drop(27).head.getOrElse(null)),
80        lastSchool = Bytes.toString(data.drop(28).head.getOrElse(null)),
81        year = Bytes.toString(data.drop(29).head.getOrElse(null)),
82        remarks = Bytes.toString(data.drop(30).head.getOrElse(null)),
83        leaveDate = Bytes.toString(data.drop(31).head.getOrElse(null)),
84        leaveReason = Bytes.toString(data.drop(32).head.getOrElse(null)),
85        progress = Bytes.toString(data.drop(33).head.getOrElse(null)),
```

Implicit Reader ...


```
override def columns = Seq(
  "student_id",
  "gr_number",
  "firstname",
  "middlename",
  "lastname",
  "mothername",
  "Address",
  "Stdfrom",
  "Stddob",
  "Gender",
  "reservationCategory",
  "blood_grp",
  "Contactno",
  "Phno",
  "Mail",
  "add_dte",
  "add_std",
  "quota:father_contact",
  "quota:father_business",
  "quota:father_income",
  "quota:income_certi",
  "quota:caste_certi",
  "sling:lc_entry",
  "sling:result_entry",
  "sling:Entrance",
  "sling:last_School",
  "sling:Year",
  "sling:Remarks",
  "sling:leave_dte",
  "sling:leave_rsn",
  "sling:Progress",
  "sling:Conduct",
  "sling:try_pass")
}
```

Do not forget to override the *columns* method.

HBase Read table Data in Spark DataFrame

```
248   // Reading StudentClass table from HBase
249   val studentClassDF = spark.sparkContext.hbaseTable[StudentClass](HBASE_TABLE_CLASS_MASTER.get)
250     .inColumnFamily(HBASE_TABLE_CLASS_MASTER_DEFAULT_COLUMN_FAMILY.get)
251     .map(record => {
252       StudentClass(
253         record.classId,
254         record.classNo,
255         record.classDesc
256       )
257     }).toDS()
258   // Reading semester table from HBase
259   val semesterDF = spark.sparkContext.hbaseTable[Semester](HBASE_TABLE_SEMESTER_MASTER.get)
260     .inColumnFamily(HBASE_TABLE_SEMESTER_MASTER_DAFAULT_COLUMN_FAMILY.get)
261     .map(record => {
262       Semester(
263         record.semesterId,
264         record.semesterNo,
265         record.semesterDesc
266       )
267     }).toDS()
268
269   // Reading subject table from HBase
270   val subjectDF = spark.sparkContext.hbaseTable[Subject](HBASE_TABLE_SUBJECT_MASTER.get)
271     .inColumnFamily(HBASE_TABLE_SUBJECT_MASTER_DEFAULT_COLUMN_FAMILY.get)
272     .map(record => {
273       Subject(
274         record.subjectId,
275         record.semesterId,
276         record.subjectCode,
277         record.subjectName,
278         record.marks,
279         record.grace
280       )
281     }).toDS()
```

HBase Implicit Field Writer


```scala
// ExamResult: Custom one to one mapping for writing to HBase table

implicit def examResultWriter: FieldWriter[ExamResult] = new FieldWriter[ExamResult]
{
  override def map(exam: ExamResult): HBaseData =
    Seq(
      // here when you insert to HBase table you need to pass 1 extra argument, as compare to HBase table reading mapping.
      Some(Bytes.toBytes(s"${exam.examId}${exam.examId}")),
      Some(Bytes.toBytes(exam.examId)),
      Some(Bytes.toBytes(exam.studentId)),
      Some(Bytes.toBytes(exam.classId)),
      Some(Bytes.toBytes(exam.subjectId)),
      Some(Bytes.toBytes(exam.subjectName)),
      Some(Bytes.toBytes(exam.marks)),
      Some(Bytes.toBytes(exam.grace)),
      Some(Bytes.toBytes(exam.total))
    )
  override def columns = Seq(
    "exam_id",
    "student_id",
    "class_id",
    "subject_id",
    "subject_name",
    "marks",
    "grace",
    "total"
  )
}
```

Save DataFrame to HBase


```
291   // Iterating DataFrame and saving data to HBase table
292   examResultDF.map(record =>
293     ExamResult(
294       record.getAs[String](0),
295       record.getAs[String](1),
296       record.getAs[String](2),
297       record.getAs[String](3),
298       record.getAs[String](4),
299       record.getAs[Int](5),
300       record.getAs[Int](6),
301       record.getAs[Int](7),
302       record.getAs[Int](8),
303       record.getAs[Int](9),
304       record.getAs[Int](10))
305   ).rdd.toHBaseTable(HBASE_TABLE_EXAM_RESULT.get).inColumnFamily(HBASE_TABLE_EXAM_RESULT_DEFAULT_COLUMN_FAMILY.get).save()
```

Reference

[1] Apache HBase – Apache HBase™ Home
URL: https://hbase.apache.org/
[2] Architecting HBase Applications: A Guidebook for successful development and design by Jean-Marc Spaggiari & Kevin O'Dell.
[3] AsyncHBase
URL: https://github.com/OpenTSDB/asynchbase
[4] Spark HBase Connector
URL: https://github.com/nerdammer/spark-hbase-connector
[5] NodeJS Thirft2 HBase package
URL: https://www.npmjs.com/package/node-thrift2-hbase
[6] NodeJS Async
URL: https://www.npmjs.com/package/async
[7] Akka Actor
URL: https://mvnrepository.com/artifact/com.typesafe.akka/akka-actor
[8] Akka HTTP Core
URL: https://mvnrepository.com/artifact/com.typesafe.akka/akka-http-core_2.11/10.0.1
[9] Akka HTTP Spray JSON
URL: https://mvnrepository.com/artifact/com.typesafe.akka/akka-http-spray-json-experimental

Reference

[10] Kafka Clients
URL: https://mvnrepository.com/artifact/org.apache.kafka/kafka-clients
[11] Spray JSON
URL: https://mvnrepository.com/artifact/io.spray/spray-json
[12] Spray JSON Shapeless
URL: https://mvnrepository.com/artifact/com.github.fommil/spray-json-shapeless
[13] Scalamock scalatest
URL: https://mvnrepository.com/artifact/org.scalamock/scalamock-scalatest-support

hosted by Alibaba Group 阿里巴巴集团 APACHE HBASE

Thanks