Disclaimer

This document is for informational purposes only and is subject to change at any time without notice. The information in this document is proprietary to Actian and no part of this document may be reproduced, copied, or transmitted in any form or for any purpose without the express prior written permission of Actian.

This document is not intended to be binding upon Actian to any particular course of business, pricing, product strategy, and/or development. Actian assumes no responsibility for errors or omissions in this document. Actian shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials. Actian does not warrant the accuracy or completeness of the information, text, graphics, links, or other items contained within this material. This document is provided without a warranty of any kind, either express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, or non-infringement.
Spark Connector for VectorH

By Justin Gilbert

Presented by Karl Schendel
Who is Justin Gilbert?

Justin Gilbert is a graduate in Computer Science from the University of Texas at Austin. For the last five years he has worked on various projects involving big data at Pervasive and then Actian. He has experience writing code in a plethora of languages from C and Java to Python and Ruby. After working for several years on the DataRush project he recently has been focused on bridging the gap between Spark and Vector to fully realize their potential.
What will be covered?

➢ What is Spark and why does it matter?
➢ What is Vector and what can it do?
➢ How can we effectively utilize these technologies together?
What is Spark?
What is Spark?

• Fast general engine for large scale data processing and analysis

• An easier to use alternative to Hadoop MapReduce

• Developed at UC Berkeley, open sourced in 2010

• Has libraries that can be seamlessly combined to extend capabilities
  • Multiple libraries supporting everything from SQL and streaming to complex analytics
• Fast distributed data processing system

• Low latency cluster computing system for very large data sets

• Highly modular and customizable based on needs

• Simplified API compared to MapReduce

• Language support for Java, Scala, Python, and R
Why should you care?

• Spark provides a unified platform for Big Data
  ▪ Integrate Batch, Streaming, and Interactive use with Hadoop, Mesos, NoSQL, etc.

• Runs everywhere with multiple deployment options
  ▪ Can run Spark on top of any distributed system Hadoop, Mesos, AmazonEC2, even standalone

• Multi-Language rich APIs – develop using what you want
  ▪ Java, Scala, Python, SQL, R

• Uses in memory processing speeding up computation cycles
  ▪ Potentially anywhere from 10x–100x faster than MapReduce, particularly when using iterative algorithms
How is this possible?

• Uses in memory cluster computing
  ▪ Unlike MapReduce data is cached in memory between processing stages instead of staged to disk
  ▪ The data is more readily available for use in multiple tasks concurrently without waiting for disk I/O
• The directed acyclic graph processing engine Spark uses is run through an optimizer
  ▪ For example the if a filter is detected after a map operation the order will be automatically swapped before execution to reduce the number of records that will need to be mapped and kept in memory
• This also allows Spark to Bring the processing to the data
  ▪ Data processing is done locally whenever possible to minimize network I/O and avoid network bottlenecks
  ▪ The DAG based processing engine helps ensure the data is available and loaded in memory where necessary
• Spark in general can also launch tasks much faster
  ▪ Keeps executor JVMs running on each node so launching new tasks requires much less setup time
  ▪ Keeping a single executor process running helps provide more fault tolerance since the data is kept in memory
Spark and Scala

• Spark is primarily written in Scala, a JVM based language

• Scala is a modern multi-paradigm language that smoothly integrates the features of both an object oriented and functional programming language

• Since the Spark core is written in Scala it is extremely flexible and extensible, and can run almost anywhere since it’s based on the JVM which is OS agnostic

• Scala is a statically typed language so development and deployment can be accelerated since many common type errors can be caught at compile time

• Spark provides an interactive shell with all the same power and functionality available when building a full fledged application in any of the supported languages

• Scala is very powerful and expressive language and provides many useful libraries and advanced features however this also makes the learning curve steeper
Spark WordCount example in Scala

Simple Spark word count example in Scala:

```scala
import org.apache.spark._

object WordCount {
  def main( args: Array[String] ) {
    // Initialize Spark context
    val conf = new SparkConf().setAppName("WordCount")
    val sc = new SparkContext(conf)

    // Count the occurrence of each word in a file and save the result
    val textFile = sc.textFile( "readme.md" )
    val counts = textFile.flatMap( _._split("\s+"))
      .map( word => (word, 1) )
      .reduceByKey( _ + _ )

    counts.saveAsTextFile( "results" )
  }
}
```
Who Uses Spark?

• Amazon
• eBay Inc.
• Groupon
• NASA
• Nokia Solutions and Networks
• Tencent
• Yahoo!

• For more information go to
  http://spark.apache.org/powered-by.html
What is Vector?
What is Vector?

• A columnar, relational database designed for reporting and analytics

• Delivers extreme high performance even on just a single node

• Easy to install and utilize

• Runs on Linux and Windows, 64 bit

• Excellent concurrency and real-time update characteristics
Vector in Hadoop (VectorH)

VectorH scales single machine Vector up to cluster scales, leveraging the HDFS distributed filesystem and the YARN resource controller. The result is a fully capable Vector DBMS which takes advantage of clustered hardware, for massive performance gains.
Why should you care?

• VectorH is the world’s fastest, most production ready SQL database engine for Hadoop

• Combines the most modern, scalable, high-performing database with the power of Hadoop

• It is built natively to run on Hadoop clusters and support enterprise workloads

• Can process Hadoop data much faster than any other relational database
Why should you care? (cont.)

• Uses standard ANSI SQL:2003 to support standard BI tools

• Supports advanced analytics including CUBE, ROLLUP, GROUPING SETS, and WINDOWING functions to name a few

• Fully ACID compliant with multi-version read consistency for transactional integrity

• Also provides system-wide failover protection

• Has been proven to run up to 900 times faster than other SQL tools on Hadoop
How does it do it?

- Core technology behind the high processing speeds of Vector is vectorized processing.

- Vector is powered by the hardened, patented X100 query engine.

- Can load massive amounts of data in parallel across a cluster.

- Additionally it exploits performance critical features of modern CPUs.

- Utilizes super-scalar execution and Single Instruction, Multiple Data (SIMD) instructions.
How do we bring them together?
How does Actian bring these amazing technologies together?

• Introducing the Spark-Vector connector

• The connector provides the ability to join Vector data with native Hadoop data files
  ▪ Currently supports CSV, ORC, and Parquet formats through the use of the Spark framework

• Can be used to load or unload large data volumes in parallel.

• Send data in a distributed, parallel stream from Spark to a VectorH table

• Retrieve data in a distributed, parallel manner from a VectorH table
The Spark Vector Connector

• Built on the DataStream API that was released with the Vector 4.2.2 release

• Uses Spark’s DataSource API to interface with Spark and SparkSQL

• Allows us to read and write from Spark to VectorH in parallel
Spark-Vector connector example usage

- **Streaming Data**
- **Remote Data Center**
- **Cloud Data & Applications**

> **Data Ingest**: from available Spark Sources

**Spark**

**VectorH**

**SQL**

> **VectorH as a Data Source to Spark Apps**
The Spark RDD and DataFrame

• Spark provides two primary abstractions when working with data
  ▪ The Resilient Distributed Dataset or RDD was the primary object for manipulating data in Spark 1.x
  ▪ Spark 1.6 began adding DataFrames to support SparkSQL among other things
  ▪ Spark 2.0 began replacing RDD usage with DataFrames, which effectively became an alias for a Dataset of Rows

• An RDD is a fault-tolerant collection of elements that can be operated on in parallel
  ▪ Can be created by parallelizing an existing collection object or by referencing an external storage system

• A DataFrame is a generic distributed collection of data that also includes additional information about the columns of the data
  ▪ Can be thought of as a RDD with schema information included
  ▪ Since more information is provided in advance Spark can perform further optimizations with DataFrames

• RDDs and DataFrames are interoperable and can be converted to each other
  ▪ However certain operations are only available with DataFrames, such as direct execution of SQL on the data
Example Scala usage of Spark with Vector

For an interactive Scala session run on command line:

```
#> Spark-shell --jars spark_vector-assembly.jar
```

Or alternatively in your Scala source simply add:

```
import com.actian.spark_vector
```

The spark shell provides some objects automatically but you can also create custom instances:

```
val conf = new SparkConf().setAppName(appName).setMaster(master)
val sc = new SparkContext(conf) //spark-shell provides this object automatically
val sqlContext = new SQLContext(sc) //spark-shell provides this object
```

In Spark 2.0 we can instead use sessions:

```
val spark = SparkSession.builder().appName(appName).getOrCreate()
```
Example Scala usage of Spark with Vector continued

Then we need to create a Vector connection object and get the schema info from the target table:

```scala
val vectorConnectProps = VectorConnectionProperties( "hostname.atsome.domain",
   "VH",
   "testdbname",
   "username",
   "simplepassword" )

val tableColumnMetadata = VectorUtil.getTableSchema( vectorConnectProps,
   "vectortablename" )

val schema = StructType( tableColumnMetadata.map(_.structField) )
```

Or we could create the schema object manually:

```scala
val schema = StructType( Seq( StructField("lineNumber", IntegerType),
   StructField("content", StringType) ) )
```
Spark RDD examples

Reading into an RDD from Vector :

```
val rdd = sc.unloadVector( vectorConnectProps, // The Vector connection properties object
                         "vectortablename", // The Vector table name
                         tableColumnMetadata ) // The Vector table column metadata
```

Writing from an RDD into Vector :

```
rdd.loadVector( schema, // The Vector table schema
                vectorConnectProps, // The Vector connection properties object
                "vectortablename" ) // The Vector table name
```
Spark DataFrame example

Reading into a DataFrame from Vector :

```scala
val dataframe = spark.read.vector( "hostname.atsome.domain", // The primary Vector host
  "VH", // The Vector instance
  "testdbname", // The Vector database name
  "vectortablename", // The Vector table name
  props ) // properties object containing login info, etc.
```

Or alternatively :

```scala
val dataframe = spark.read.vector( vectorConnectProps, // The Vector connect properties
  "vectortablename", // The Vector table name
  props ) // properties object containing login info, etc.
```
Spark DataFrame example

Writing from a DataFrame into Vector:

```python
dataframe.write.vector( "hostname.atsome.domain", // The primary Vector host
    "VH", // The Vector Instance
    "testdbname", // The Vector database name
    "vectortablename", // The Vector table name
    props ) // properties object containing login info, SQL, etc.
```

Or alternatively:

```python
dataframe.write.vector( vectorConnectProps, // The Vector connection properties
    "vectortablename", // The Vector table name
    props ) // properties object containing login info, SQL, etc.
```
SparkSQL

• Originally Apache Shark project

• Allows seamless integration of SQL queries within Spark programs

• Can connect to any supported data source with uniform syntax

• Also supports standard JDBC and ODBC connectivity for everything else

• Utilizes the same DAG processing engine and optimizer that Spark uses

• Requires use of DataFrames and provides extended API for them

• Supported in all the Spark supported APIs: Java, Scala, Python, R
SparkSQL example

```scala
sqlContext.sql( """"CREATE TEMPORARY TABLE sparktablename
    USING com.actian.spark_vector.sql.DefaultSource
    OPTIONS (
        host "hostname.atsome.domain",
        instance "VH",
        database "testdbname",
        table "vectortablename",
        user "username",
        password "simplepassword"
    )"""
)```

© 2017 Actian Corporation
SparkSQL example

```scala
sqlContext.sql("CREATE TEMPORARY TABLE csvtablename
    USING com.databricks.spark.csv  // In 2.0 can simply use csv as source
    OPTIONS (
        path "hdfs://hostname.atsome.domain:8020/Actian/VectorVW/datafile.csv",
        header "true",
        delimiter ",",
        inferSchema "true"
    )")

sqlContext.sql("insert into sparktablename select * from csvtablename")

val results = sqlContext.sql("select * from sparktablename")
```
Spark-Vector Loader utility

- Command line client utility for loading Vector
- Available within Spark-Vector loader module
- Currently supports loading CSV, ORC, and Parquet files directly into Vector

Example usage:

```bash
#> spark-submit --class com.actian.spark_vector.loader.Main --master $SPARK_MASTER $SPARK_LOADER_JAR load csv -sf $HDFS_TMP/data.txt -vh $(hostname -f) -vi $(iigetenv II_INSTALLATION) -vd testdbname -tt vectortablename -sc "\"
```

For more information and command line help run:

```bash
#> spark-submit --class com.actian.spark_vector.loader.Main --master $SPARK_MASTER $SPARK_LOADER_JAR load --help
```
External tables for VectorH

• Functionality added with the Spark-Vector provider module
• Allows you to directly query data stored in Hadoop with VectorH
• Can even join these external tables with native VectorH
• Vector in Hadoop 5.0 currently supports the following file formats: CSV, ORC, & Parquet
• As Spark supports more formats Vector will automatically gain the capabilities
• The data can be stored in HDFS or elsewhere, such as the local filesystem or Amazon S3
External Table Syntax

- To create a new external table:

CREATE EXTERNAL TABLE [schema.]ext_table_name [(col_name [ data_type ], ...)]
  USING spark
  WITH REFERENCE = '<reference>'
  [ FORMAT = '<format>' ]
  [ OPTIONS = ( 'key1' = 'value1', 'key2' = 'value2', ... ) ]

Example:

CREATE EXTERNAL TABLE employees
  (EmployeeID int not null, FirstName varchar(20) not null, Title varchar(10), LaptopID varchar(12))
  USING spark
  WITH REFERENCE = 'hdfs://hostname:8020/Actian/VectorVW/emps.csv',
  OPTIONS = ('delimiter' = ',', 'header' = 'true')
Spark users also benefit

• Can use VectorH like any other source or sink

• Can use SparkSQL to store the results of a query in a VectorH table

• Can run queries directly on data stored in VectorH

• Alternatively can use an RDD or Dataframe object to store data from a Vector table

• The objects can then be used to load the transformed contents back into VectorH
Example usage scenarios for Spark + VectorH

• Finance Industry
  ▪ One financial institution is using Apache Spark on Hadoop to analyze the text inside the regulatory filling of their own reports and also their competitor reports. The firms use the analytic results to discover patterns around what is happening, the marketing around those and how strong their competition is.
  ▪ Financial institutions could implement real time monitoring application that runs on Apache Spark and VectorH. To provide supreme service across its online channels, the applications could helps the bank continuously monitor their client’s activity and identify if there are any potential issues.

• E-commerce Industry
  ▪ eBay uses Apache Spark to provide targeted offers, enhance customer experience, and to optimize overall performance. Apache Spark is leveraged at eBay through Hadoop YARN. YARN manages all the cluster resources to run generic tasks. eBay spark users leverage the Hadoop clusters in the range of 2000 nodes, 20,000 cores and 100TB of RAM through YARN.
  ▪ Shopify wanted to analyze the kinds of products its customers were selling. Its data warehousing platform could not address this problem as it kept timing out while running data mining queries on millions of records. Using Apache Spark Shopify can now process 67 million records in minutes.
Example usage scenarios for Spark + VectorH

• Healthcare
  ▪ Many healthcare providers are using Apache Spark to analyze patient records along with past clinical data to identify which patients are likely to face health issues after being discharged from the clinic. This helps hospitals prevent hospital re-admittance as they can deploy home healthcare services to the identified patient, saving on costs for both the hospitals and patients. By storing patient data in Vector even greater efficiency can be achieved, and these saving can be passed on to the patients.
  ▪ Apache Spark is used in genomic sequencing to reduce the time needed to process genome data. Previously, it took several weeks to organize all the chemical compounds with genes but now with Apache spark on Hadoop it takes just a few hours.

• Media & Entertainment Industry
  ▪ Apache Spark is used in the gaming industry to identify patterns from the real-time in-game events and respond to them to harvest lucrative business opportunities like targeted advertising, auto adjustment of gaming levels based on complexity, player retention and many more. Additionally VectorH could be used to store the large amount of data collected constantly from millions of players worldwide.
The future of Spark and Vector

• Could utilize Spark to provide a whole range of additional functionality within Vector

• Additional formats and databases could be supported for external tables
  - Any table or data source accessible to Spark could be accessed and manipulated with Vector

• In the future could provide extensive UDF functionality to Vector
  - A custom computation could be saved via SQL within Vector
  - This User Defined Function or UDF would be translated into a generic Spark workflow process
  - The UDF could then be utilized by Vector in any typical SQL statement
In summary

• Any file format and storage location supported by Spark can potentially be used as an external table by VectorH

• Any future improvements in Spark either in performance or functionality will translate directly into an improvement in the capabilities of VectorH

• Users can use whatever API they are most comfortable whether SQL or Scala

• Spark and VectorH’s distributed parallel connections can be used to transfer data which can then be processed with the unmatched speed of the VectorH engine combined with the growing analytics capabilities of Spark

• When the technologies are used together data ingestion and processing becomes an almost trivial task

• Freely available at: https://github.com/ActianCorp/spark-vector
Questions?
Thank You!
Coffee Break - 15 minutes

#hybriddataconference