Spark Cyborgs
Integrating IBM Parallel RDBMSs with Spark

Gustavo Arocena
Torsten Steinbach
@torsstei
Legal Disclaimer

• © IBM Corporation 2016. All Rights Reserved.

• The information contained in this publication is provided for informational purposes only. While efforts were made to verify the completeness and accuracy of the information contained in this publication, it is provided AS IS without warranty of any kind, express or implied. In addition, this information is based on IBM’s current product plans and strategy, which are subject to change by IBM without notice. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this publication or any other materials. Nothing contained in this publication is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software.

• References in this presentation to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. Product release dates and/or capabilities referenced in this presentation may change at any time at IBM’s sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way. Nothing contained in these materials is intended to, nor shall have the effect of, stating or implying that any activities undertaken by you will result in any specific sales, revenue growth or other results.
Business Analytics Insights & Value Stages

Descriptive Analytics
What happened?

Predictive Analytics
What will happen?

Prescriptive Analytics
What should I do?

Best Mechanisms Used

-> For analytics continuum both, relational engines + Big Data engines have clear sweet spots
Why Spark?

- Spark provides a **unified framework** to develop hybrid apps that **blend** SQL and non-relational analytics
- Spark becoming **de-facto Big Data processing platform**
- But **not** everyone can **replace** their **SQL** engines with SparkSQL …
  - **Performance** of SparkSQL vs. mature SQL engines
  - Deep **multi tenancy** and **security** SLAs of established SQL engines
  - **Enterprise feature richness** of mature SQL engines
- **Next best** thing: integration of processing engines
RDBMS and Spark MPP Architectures

Architecture of a Spark application

- Spark Driver
- Spark Executor
- Spark Executor
- Spark Executor

Storage layer

Architecture of an IBM MPP RDBMS

- RDBMS Coordinator
- RDBMS Worker
- RDBMS Worker
- RDBMS Worker

Storage layer

Hmm, wait a sec …
IBM MPP RDBMS integrated with Spark
Integration Aspects

- **Externals**
  - “Invoke” Spark from SQL application
    - SQL extensibility constructs (Stored Procedures, UDFs)
  - Spark Submit REST API
  - Spark shells (Notebooks and/or REPL)
  - Read from/Write to RDBMS from Spark application
    - **DataSources** API and beyond

- **Internals**
  - Leverage the similar **MPP** architectures
    - Move data in parallel between executors and RDBMS workers
  - **Colocation** between RDBMS Workers and Spark Executors
    - Transfer most/all the data locally
  - Minimize serialization/deserialization overhead
Usage Scenarios

- **Agile DWH**
  - Spark-based ELT
  - In-Database *Machine Learning* batch processing
  - *Non-relational analytics* (ML, graph) and custom operations (joins, aggregates) invoked from **SELECT** statement
  - **Federation** of relational tables with noSQL & HDFS data (incl. *schema on read*)
  - **Streaming** data landing in RDBMS

- **Operationalization**
  - **Deploy Spark** applications
  - Benefit from **sophisticated** cost-based query **optimization**
  - RDBMS-managed **data access** control
  - **Bring compute to the data**
## Two existing Cyborgs

<table>
<thead>
<tr>
<th>Official attire</th>
<th>Cyborg 1</th>
<th>Cyborg 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage</td>
<td>Inside RDBMS (relational)</td>
<td>Files on HDFS</td>
</tr>
<tr>
<td>Catalog</td>
<td>RDBMS catalog</td>
<td>Hive metastore</td>
</tr>
</tbody>
</table>
RDBMS with Plain SQL Application

con.execute("SELECT * FROM emp_no");

SQL Application

SQL Query Compilation

Query Coordination & Merge

Head Node

Physical Cluster

Local Processing

Part 1

Local Processing

Part 2

Local Processing

Part 3

Local Processing

Part n-1

Local Processing

Part n
RDBMS with Spark App – Range Partitioned Data

```scala
options.put("partitionColumn", "emp_no");
options.put("lowerBound", "10001");
options.put("upperBound", "499999");
options.put("numPartitions", "10");
DataFrame jdbcDF = sqlContext.load("jdbc", options);
```
Spark & RDBMS Partitions Matched Up

```java
options.put("partitionColumn", "DBPARTITIONNUM(..)");
options.put("lowerBound", "0");
options.put("upperBound", "9");
options.put("numPartitions", "10");
DataFrame jdbcDF = sqlContext.load("jdbc", options);
```
Spark & RDBMS Partitions Co-located

```
dbReader = sqlContext.read.format("in-database");
dbReader.option("dbtable", "USER1.MYTAB");
val input: DataFrame = dbReader.load();
  :
dbBWriter = resultDF.write().format("in-database")
dbWriter.mode("Append").partitionBy("Col1, Col2").insertInto("USER1.MYNEWTAB")
```
CALL GLM('model=adults, intable=traintab, id=id, target=age');
CALL PREDICT_GLM('model=adults, intable=testtab, id=id, outtable=adult_predicted');

or

CALL SPARK_SUBMIT('jarfile=myapp/sparkjobs.jar class=com.ibm.dashdb.spark.DemoJob')
RDBMS invoking Spark job from SQL Query

SELECT customer.id, customer.name, customer.lifetime_value
FROM TABLE(EXECSPARK(language => 'scala',
       jarfile => myapp/demoPTF.jar
       class => 'com.ibm.dashdb.spark.DemoPTF',
)) AS customer WHERE customer.country = 'GERMANY'
RDBMS invoking Spark job from SQL Query

```sql
SELECT customer.id, customer.name, customer.lifetime_value
FROM TABLE(EXECSPARK(language => 'scala',
    jarfile => myapp/demoPTF.jar
    class => 'com.ibm.biginsights.bigsql.examples.ReadJsonFile',
    uri => 'hdfs://host.port.com:8020/user/bigsql/demo.json')
)) AS customer WHERE customer.country = 'Canada'
```
Cyborg 1 – Relational Data (stored inside RDBMS)

- Data is **stored** hash-partitioned by the RDBMS
  - Partitioning key can be defined by the user
  - Also possible to use random partitioning (e.g. round robin ingest)
- **Input** data read via RDBMS and **streamed** in parallel to Spark
- **Result** data **streamed** in parallel from Spark to RDBMS
  - When it should be stored relationally again
- Current focus: **Operational** Spark **batch** hosting by Database
  1. Provide Spark **ML** + IBM ML algorithms as database **stored procedures**
  2. Host **custom Spark** logic via **batch** processing, batch job submission via
     - **REST** interface
     - **Stored Procedure**

- **Future:**
  - **Interactive** Spark analytics (shells, Notebooks)
  - Combine/Merge with Cyborg 2
- **Btw, the product is called IBM dashDB** (Cyborg soon available as tech preview there)
Cyborg 2 (Data stored on HDFS)

- Data is randomly partitioned (by HDFS)
  - No user-defined partitioning key
- **Input** data read by **Spark directly**
  - Schema at read
- **Result** data is streamed in parallel from Spark to RDBMS
- Current focus: **Granularly** embedding **Spark** logic in **SQLs**
  - **User-Defined Table Function** Interface to Spark
    - I.e. jobs submitted via custom RPC from a RDBMS SELECT statement
    - Supports **any custom Spark code** (as long as result is a DataFrame)
- **Future:**
  - Execute **native RDBMS queries** from Spark apps & retrieve result in parallel
  - Combine/Merge with Cyborg 1
- Btw, the product is called **IBM BigInsights with BigSQL**
  - Cyborg available as tech preview there
Table UDFs coded in Spark

- **Example: read JSON file from HDFS**

```sql
SELECT *
FROM TABLE (
    SYSHADOOP.EXECSPARK(
        language => 'scala',
        class => 'com.ibm.biginsights.bigsql.examples.ReadJsonFile',
        uri => 'hdfs://host.port.com:8020/user/bigsql/demo.json'
    )
) AS doc
WHERE doc.country IS NOT NULL
```

- **SYSHADOOP.EXECSPARK** is a built-in table UDF
- Executes Java/Scala Spark job that produces a DataFrame
- Job runs on a long-running Spark app
- The RDBMS simply scans the result
Example Spark Table UDF

class ReadJsonFile extends SparkPtf {

  override def describe(ctx: BigSQLContext,
                        args: Map[String, Object]): StructType = {
    val path = args.get("URI").asInstanceOf[String].trim()
    ctx.read.json(path).schema
  }

  override def execute(ctx: BigSQLContext,
                       args: Map[String, Object]): DataFrame = {
    val path = args.get("URI").asInstanceOf[String].trim()
    ctx.read.json(path)
  }

  override def destroy(ctx: BigSQLContext,
                       args: Map[String, Object]): Unit = { }

  override def cardinality(ctx: BigSQLContext,
                           args: Map[String, Object]): Long = {
    val cardArg = args.get("CARD")
    if (cardArg == null) 100 else cardArg.asInstanceOf[Int]
  }
}
Spark Table UDF Execution

```
SELECT *
FROM TABLE(
  EXECSPARK(
    class => 'com....PTF', ...))
```
The Spark Gateway

- PTFs are executed on a “slave” Spark app (a.k.a. Spark Gateway)
- Spark gateway is a long-running Spark app, similar to Thrift Server
  - Gateway serves “Spark jobs” represented as class that implements SparkPtf Java interface
  - RDBMS controls Gateway via custom RPC protocol
- Spark Gateway runs in Yarn-client mode
- Interesting problems
  - Co-location
    - Must be prepared for Yarn to place executors on nodes where there is no worker
  - Location of result partitions not known up front (depends on Spark scheduler)
    - Must be prepared to consume partitions originating on any cluster node
  - Determining reasonable value for spark.executor.memory
    - Optimal value depends on the kind of job
  - Potential deadlocks with two Spark PTF executions in single SQL
Having Fun with Operational Aspects

- **Secure Multi-Tenant Execution**
  - Arbitrary custom Spark code inside the DB System needs to be executed in Executors impersonated in OS as connected DB user
  - Different users need different Executors (i.e. JVMs) for their concurrent jobs

- **Authentication**
  - RDBMS calling Spark calling RDBMS with seamless SSO

- **Memory & Colocation** of Spark with RDBMS
  - Both, RDBMS and Spark love to have a lot of memory
  - Avoiding double caching of data

- **Startup Latency**
  - Standing Spark app clusters vs. start/stop with each job

- **Application Deployment**
  - Uploading Spark job code and dependent packages
  - Sharing Spark code and packages – also a multi tenancy problem

- **Monitoring**
  - Giving access to Spark monitoring UI
  - Correlating SQL monitoring in RDBMS with Spark monitoring
Conclusions

- **1+1 > 2**
  - Deep integration between parallel RDBMSs with Spark is mutually beneficial
  - It allows to get best of both worlds:
    - DBs can do ML, Graph & custom analytics, flexible schema and talk more languages
    - Spark gets rich SQL performance & enterprise features

- Bring **compute to the data**
  - High performance data access is possible due to compatible architectures

- By integration into RDBMS **Spark** becomes a **highly operational** multi-tenant & secure deployment framework

- Spark can be used in **coarse-grained batch** fashion or also in **fine-grained UDF** fashion from SQL

- Blending both Cyborgs creates a true hybrid **Big Data Warehouse**
  - Hybrid compute + Hybrid Storage + Hybrid API
Backup
Spark Executor

MyTabDataFrame

SELECT .. WHERE DBPARTITIONNUM(..)
= CURRENT DBPARTITIONNUM

DB Data Node

Filtered read (selection, projection)

Cache

MyTab

Spark Reading DB Data Read via JDBC

© 2016 IBM Corporation
Spark Reading DB Data Read via Shared Memory

INSERT INTO SYSSPARK.MyTabRDD
(SELECT * FROM MyTab
WHERE DBPARTITIONNUM(...) = CURRENT DBPARTITIONNUM)