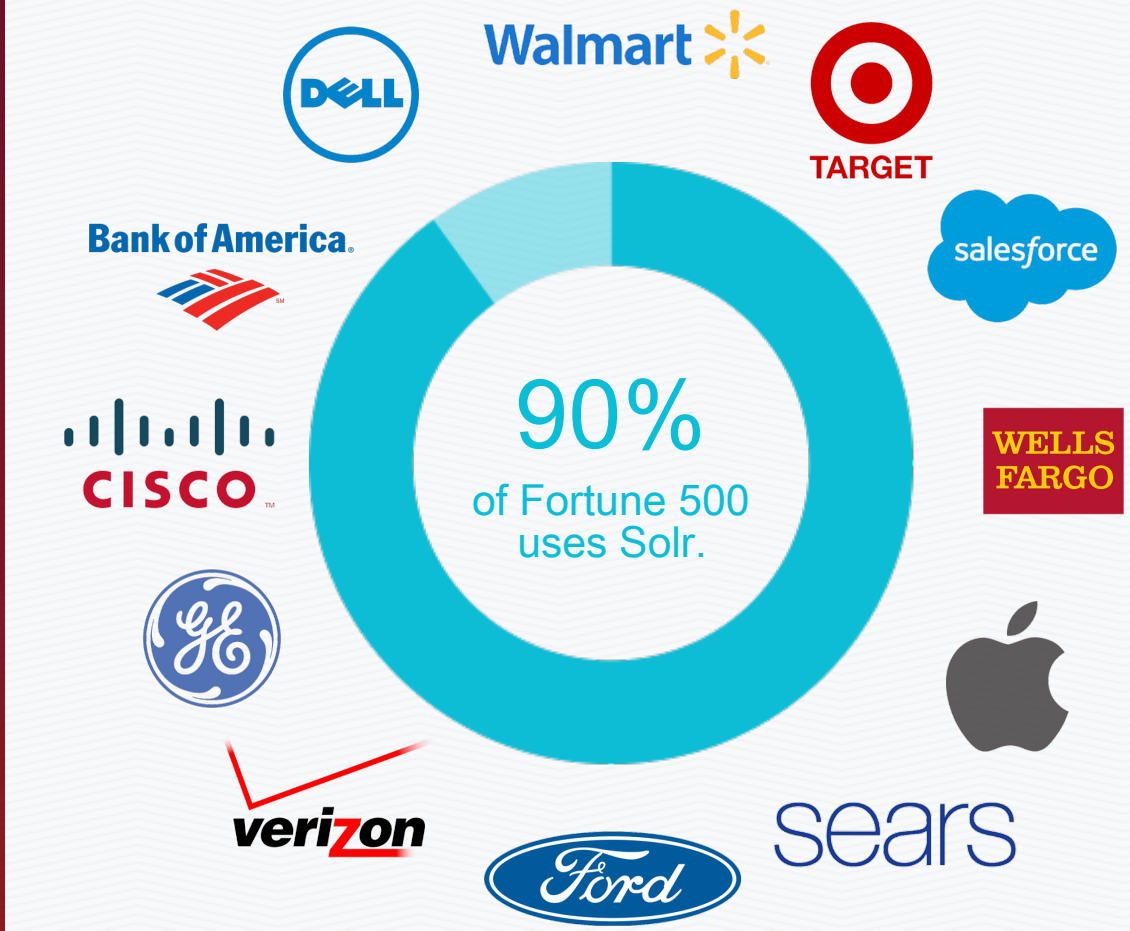
# Solr as a Spark SQL Datasource

Kiran Chitturi
Lucidworks


SPARK SUMMIT 2016
DATA SCIENCE AND ENGINEERING AT SCALE
JUNE 6-8, 2016 SAN FRANCISCO

Solr

The standard for enterprise search.

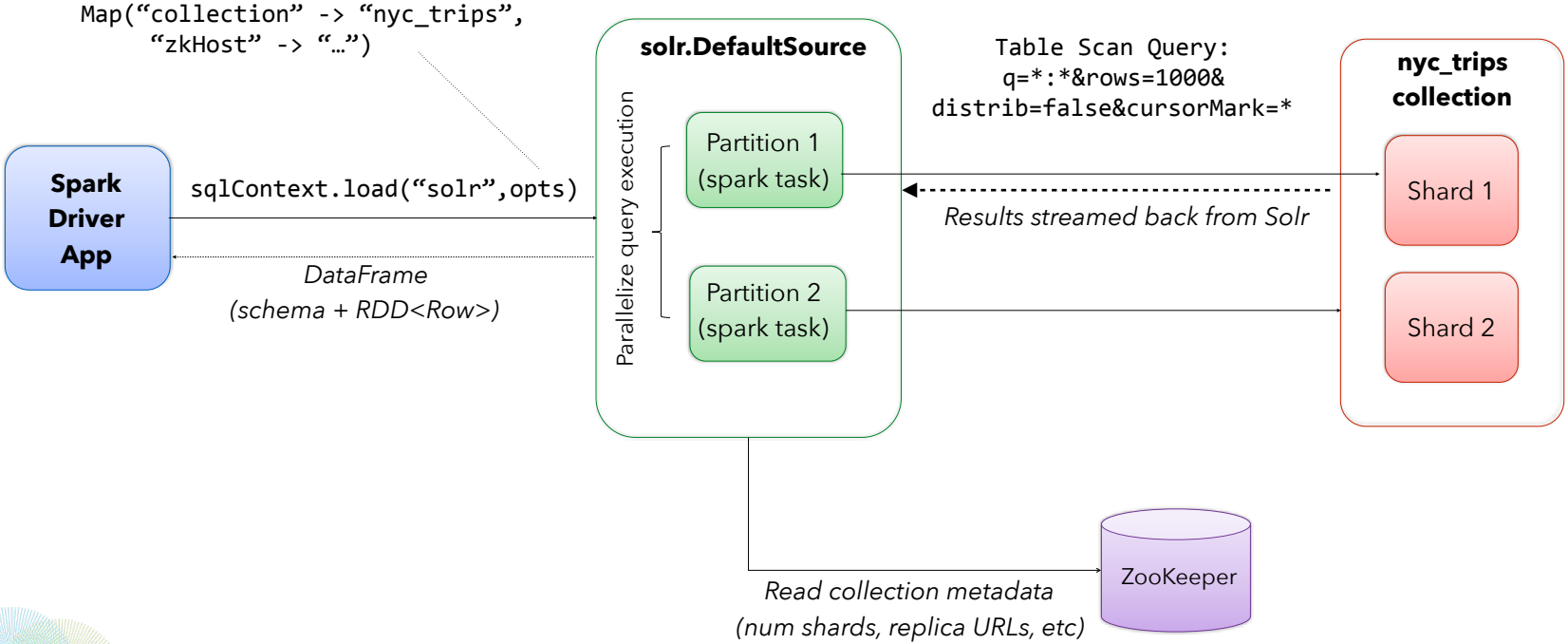


# Solr as a Spark SQL Data Source

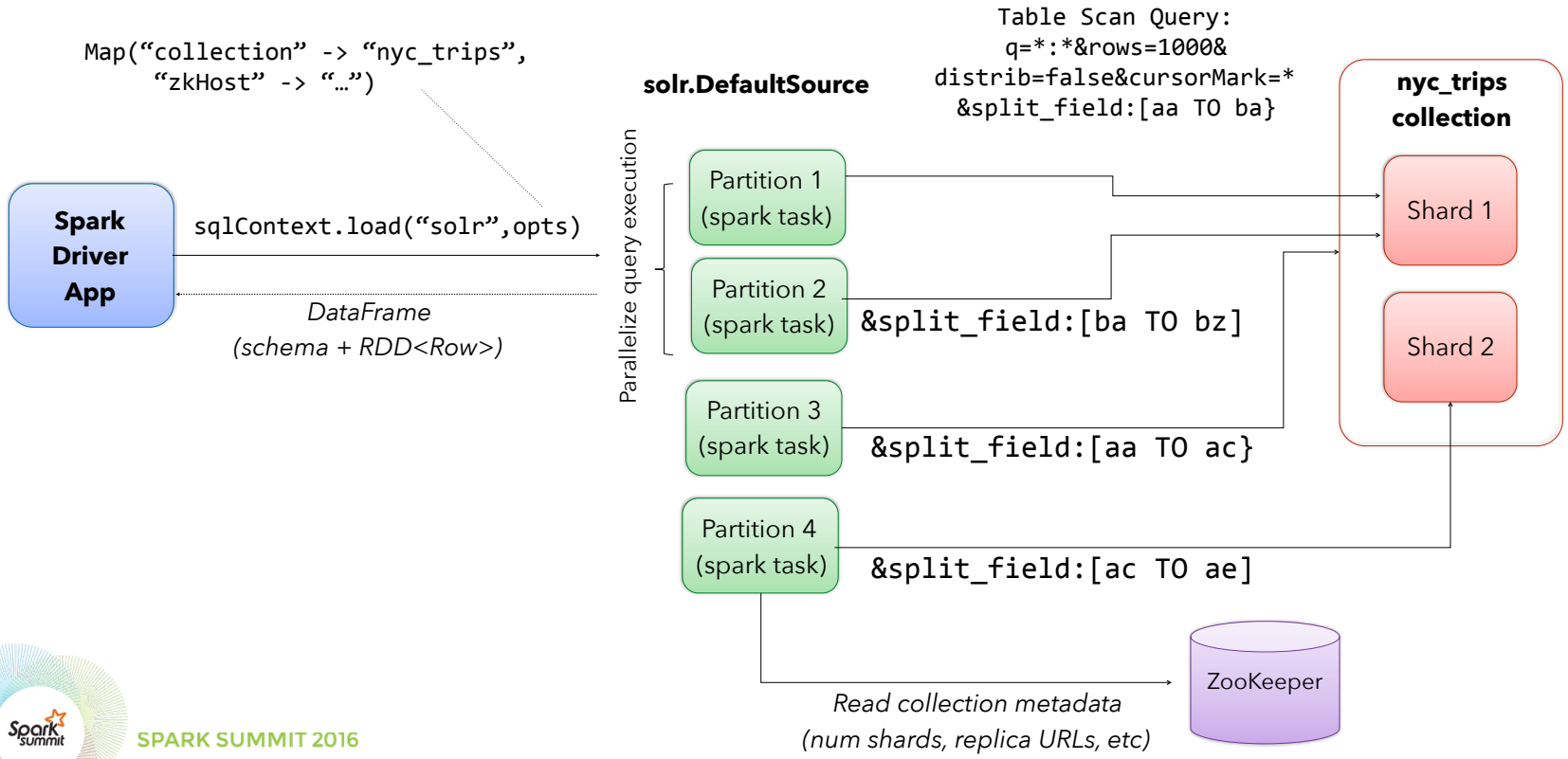
- Read/write data from/to Solr as DataFrame
- Use Solr Schema API to access field-level metadata
- Push predicates down into Solr query constructs, e.g. fq clause
- Shard partitioning, intra-shard splitting, streaming results

```
// Connect to Solr
val opts = Map("zkhost" -> "localhost:9983", "collection" -> "nyc_trips")
val solrDF = sqlContext.read.format("solr").options(opts).load

// Register DF as temp table
solrDF.registerTempTable("trips")

// Perform SQL queries
sqlContext.sql("SELECT avg(tip_amount), avg(fare_amount) FROM trips").show()
```

# SolrRDD: Reading data from Solr into Spark



# SolrRDD: Reading data from Solr into Spark



# Solr Streaming API for fast reads

- Contributed to spark-solr by Bloomberg team
- Extremely fast “table scans” over large result sets in Solr
- Relies on a column-oriented data structure in Lucene: docValues
- DocValues help speed up faceting and sorting too!
- Push SQL predicates down into Solr’s Parallel SQL engine available in Solr 6.x (Coming soon)

# Data-locality Hint

- **SolrRDD** extends **RDD**[SolrDocument] (written in Scala)
- Give hint to Spark task scheduler about where data lives

  override def `getPreferredLocations(split: Partition): Seq[String] = {`

  ```
   // return preferred hostname for a Solr partition
  }
  ```

- Useful when Spark executor and Solr replicas live on same physical host, as we do in Fusion
- Query to a shard has a "preferred" replica; can fallback to other replicas if the preferred goes down (will be in 2.1)

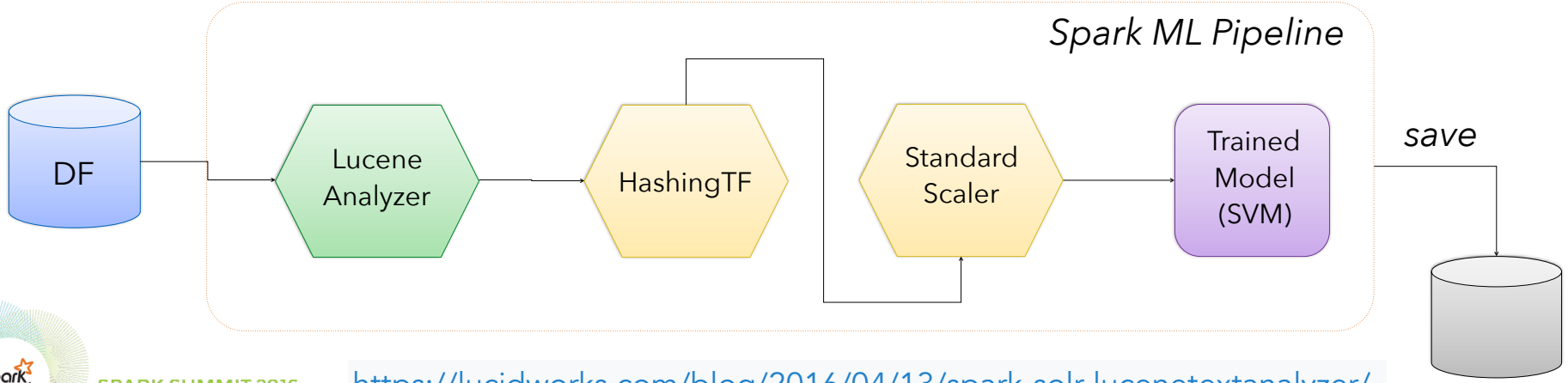
# Writing to Solr (aka indexing)

- Cloud-aware client sends updates to shard leaders in parallel
- Solr Schema API used to create fields on-the-fly using the DataFrame schema
- Better parallelism than traditional approaches like Solr DIH

```
val dbOpts = Map(
  "url" -> "jdbc:postgresql:mydb",
  "dbtable" -> "schema.table",
  "partitionColumn" -> "foo",
  "numPartitions" -> "10")

val jdbcDF = sqlContext.read.format("jdbc").options(dbOpts).load

val solrOpts = Map("zkhost" -> "localhost:9983", "collection" -> "mycoll")

jdbcDF.write.format("solr").options(solrOpts).mode(SaveMode.Overwrite).save
```

# Solr / Lucene Analyzers for Spark ML Pipelines

- Spark ML Pipeline provides nice API for defining stages to train / predict ML models
- Crazy idea ~ use battle-hardened Lucene for text analysis in Spark
- Pipelines support import/export
- Can try different text analysis techniques during cross-validation






https://lucidworks.com/blog/2016/04/13/spark-solr-lucenetextanalyzer/

# Lucene Text Analysis

- Use Lucene text analyzers for text analysis
- Wide selection of tokenizers, filters, Character filters in Lucene
- JSON declared schema analysis definition

```
{ "analyzers": [{ "name": "...",
                  "charFilters": [{ "type": "...", ...}, ... ],
                  "tokenizer": { "type": "...", ... },
                  "filters": [{ "type": "...", ... } ... ] }] },
                { "name": "...",
                  "charFilters": [{ "type": "...", ...}, ... ],
                  "tokenizer": { "type": "...", ... },
                  "filters": [{ "type": "...", ... }, ... ] }] } ],
  "fields": [{"name": "...", "analyzer": "..."},
             { "regex": ".+", "analyzer": "..." }, ... ] }
```

# Lucene Text Analysis (example)

- Word count example of spark-solr README file

```
import com.lucidworks.spark.analysis.LuceneTextAnalyzer
val schema = """{ "analyzers": [{ "name": "StdTokLower",
                |                 "tokenizer": { "type": "standard" },
                |                 "filters": [{ "type": "lowercase" }] }],
                |  "fields": [{ "regex": ".+", "analyzer": "StdTokLower" }] }
              """.stripMargin
val analyzer = new LuceneTextAnalyzer(schema)
val file = sc.textFile("README.adoc")
val counts = file.flatMap(line => analyzer.analyze("anything", line))
                 .map(word => (word, 1))
                 .reduceByKey(_ + _)
                 .sortBy(_._2, false) // descending sort by count
println(counts.take(10).map(t => s"${t._1}(${t._2})").mkString(", "))
```

```
the(158), to(103), solr(86), spark(77), a(72), in(44), you(44), of(40), for(35), from(34)
```

# Lucene Text Analysis (example)

- Word count example of spark-solr README file with stop word filtering

```
import com.lucidworks.spark.analysis.LuceneTextAnalyzer
val schema = """{ "analyzers": [{ "name": "StdTokLower",
                |                 "tokenizer": { "type": "standard" },
                |                 "filters": [{ "type": "lowercase" }] },
                |               { "name": "StdTokLowerStop",
                |                 "tokenizer": { "type": "standard" },
                |                 "filters": [{ "type": "lowercase" },
                |                             { "type": "stop" }] }],
                |  "fields": [{ "name": "all_tokens", "analyzer": "StdTokLower" },
                |             { "name": "no_stopwords", "analyzer": "StdTokLowerStop" } ]}
              """.stripMargin
val analyzer = new LuceneTextAnalyzer(schema)
val file = sc.textFile("README.adoc")
val counts = file.flatMap(line => analyzer.analyze("no_stopwords", line))
                 .map(word => (word, 1))
                 .reduceByKey(_ + _)
                 .sortBy(_._2, false)
println(counts.take(10).map(t => s"${t._1}(${t._2})").mkString(", "))
```

```
solr(86), spark(77), you(44), from(34), source(32), query(25), option(25), collection(24), data(20), can(19)
```

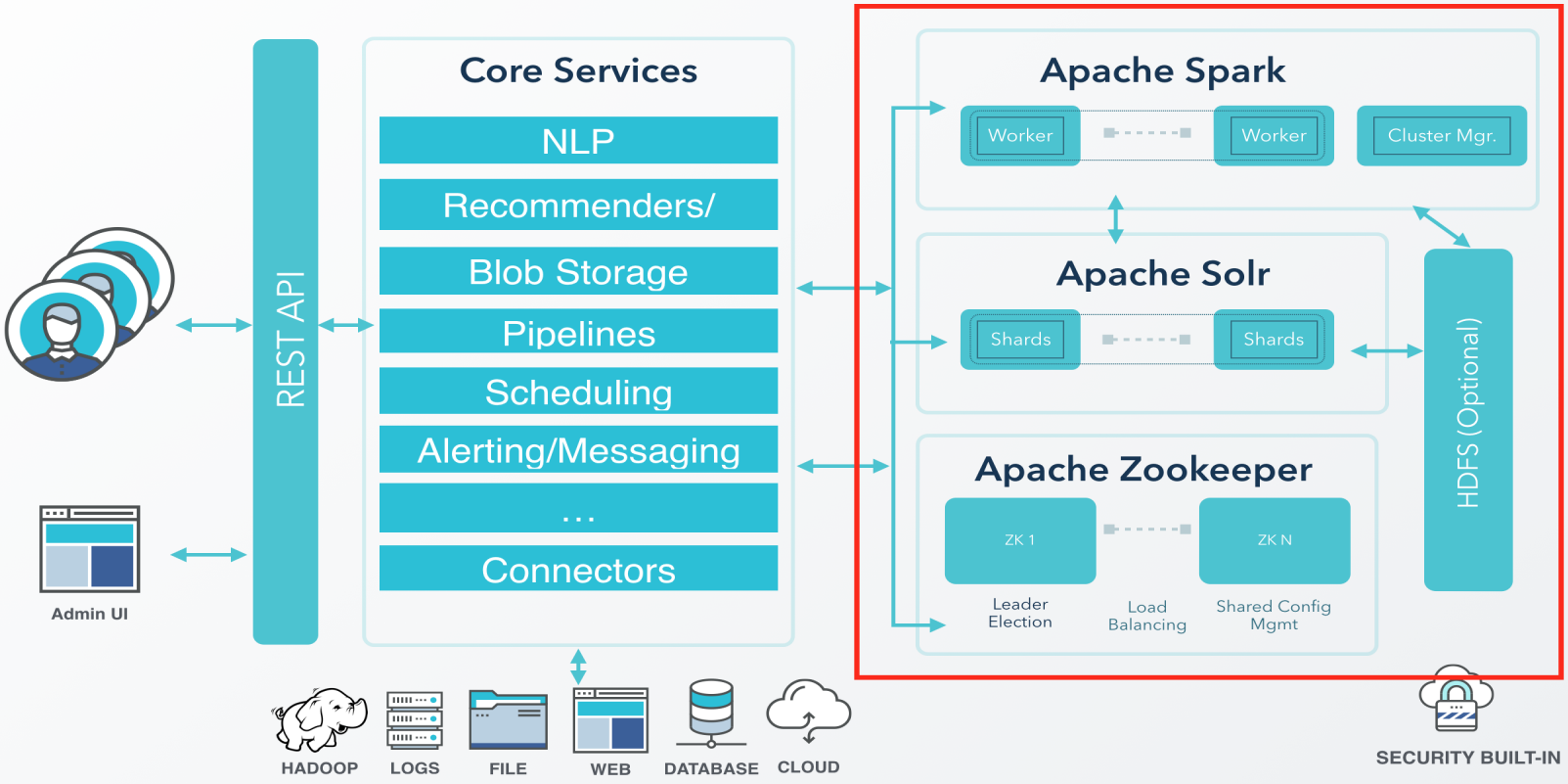
# Lucene ML transformer (example)

- LuceneTextAnalyzerTransformer as an ML pipeline transformer
- Supports multi-valued input columns

```
  val WhitespaceTokSchema =
    """{ "analyzers": [{ "name": "ws_tok", "tokenizer": { "type": "whitespace" } }],
      |  "fields": [{ "regex": ".+", "analyzer": "ws_tok" }] }""".stripMargin
  val StdTokLowerSchema =
    """{ "analyzers": [{ "name": "std_tok_lower", "tokenizer": { "type": "standard" },
      |                  "filters": [{ "type": "lowercase" }] }],
      |  "fields": [{ "regex": ".+", "analyzer": "std_tok_lower" }] }""".stripMargin
[...]
  val analyzer = new LuceneTextAnalyzerTransformer().setInputCols(contentFields).setOutputCol(WordsCol)
[..]
 val pipeline = new Pipeline().setStages(Array(labelIndexer, analyzer, hashingTF, estimatorStage,
labelConverter))
```

# Fusion Architecture

Lucidworks



# Fusion & Spark

- spark-solr 2.0.1 released, built into Fusion 2.4
- Users leave evidence of their needs & experience as they use your app
- Fusion closes the feedback loop to improve results based on user “signals”
- Train and serve ML Pipeline and mllib based Machine Learning models
- Run custom Scala “script” jobs in the background in Fusion
  - ➢ Complex aggregation jobs
  - ➢ Unsupervised learning (LDA topic modeling)
  - ➢ Re-train supervised models as new training data flows in

# Getting started with spark-solr

- Import package via maven. Available at spark-packages.org

```
./bin/spark-shell --packages "com.lucidworks.spark:spark-solr:2.0.1"
```

- Build from source

```
git clone https://github.com/Lucidworks/spark-solr
cd spark-solr
mvn clean package -DskipTests
./bin/spark-shell --jars 2.1.0-SNAPSHOT.jar
```

# Example : Deep paging via shards

```
// Connect to Solr
val opts = Map(
  "zkhost" -> "localhost:9983",
  "collection" -> "nyc_trips")

val solrDF = sqlContext.read.format("solr").options(opts).load

// Register DF as temp table
solrDF.registerTempTable("trips")

sqlContext.sql("SELECT * FROM trips LIMIT 2").show()
```

# Example : Deep paging with intra shard splitting

```
// Connect to Solr
val opts = Map(
  "zkhost" -> "localhost:9983",
  "collection" -> "nyc_trips",
  "splits" -> "true")
val solrDF = sqlContext.read.format("solr").options(opts).load

// Register DF as temp table
solrDF.registerTempTable("trips")

sqlContext.sql("SELECT * FROM trips").count()
```

# Example : Streaming API (/export handler)

```
// Connect to Solr
val opts = Map(
  "zkhost" -> "localhost:9983",
  "collection" -> "nyc_trips")
val solrDF = sqlContext.read.format("solr").options(opts).load

// Register DF as temp table
solrDF.registerTempTable("trips")

sqlContext.sql("SELECT avg(tip_amount), avg(fare_amount) FROM
trips").show()
```

# Performance test

- NYC taxi data (30 months - **91.7M** rows)
- Dataset loaded in to AWS RDS instance (Postgres)

- 3 EC2 nodes of r3.2x large instances
- Solr and Spark instances co-located together

- Collection 'nyc-taxi' created with 6 shards, 1 replication
- Deployed using solr-scale-tk (https://github.com/LucidWorks/solr-scale-tk)

- Dataset link: https://github.com/toddwschneider/nyc-taxi-data
- More details: https://gist.github.com/kiranchitturi/0be62fc13e4ec7f9ae5def53180ed181

# Index performance

- **91.4M** rows indexed to Solr in 49 minutes
- Docs per second: **31K**
- JDBC batch size: 5000
- Indexing batch size: 50000
- Partitions: 200

# Query performance

- Query - simple aggregation query to calculate averages
- Streaming expressions took **2.3** mins across 6 tasks
- Deep paging took **20** minutes across 120 tasks

# Roadmap

- Implement Datasource with Catalyst scan.
- Fallback to different replica if the preferred replica is down
- HashQParserPlugin for intra-shard splits
- Output PreAnalyzedField compatible JSON for Solr

# THANK YOU.

kiran.chitturi@lucidworks.com / @chitturikiran

https://github.com/Lucidworks/spark-solr/

Download Fusion: http://lucidworks.com/fusion/download/


SPARK SUMMIT 2016
DATA SCIENCE AND ENGINEERING AT SCALE
JUNE 6-8, 2016 SAN FRANCISCO