Integrating Solr & Spark

Timothy Potter

Lucidworks

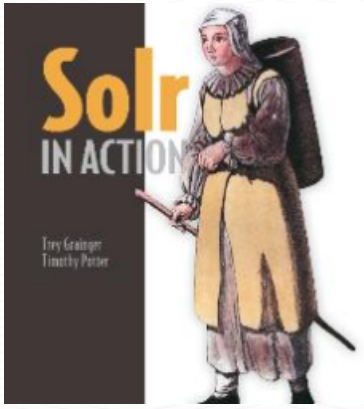
Integrating Solr & Spark

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

- Indexing from Spark
- Reading data from Solr
- Solr data as a Spark SQL DataFrame
- Interacting with Solr from the Spark shell
- Document Matching
- Reading Term vectors from Solr for MLlib

About Me …

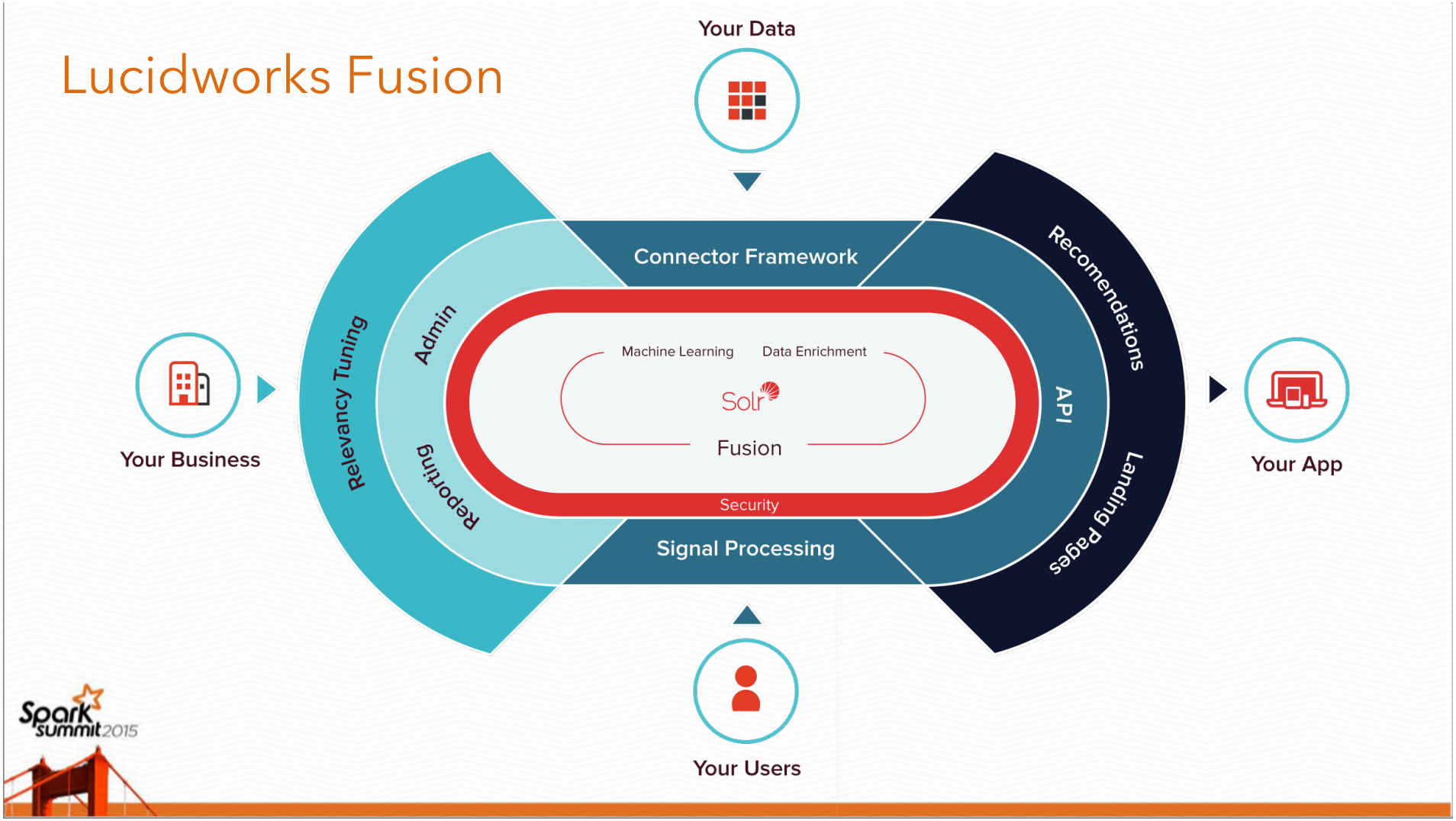
- Solr user since 2010, committer since April 2014, work for Lucidworks, PMC member ~ May 2015
- Focus mainly on SolrCloud features … and bin/solr!
  - ✓ *Release manager for Lucene / Solr 5.1*
- Co-author of Solr in Action
- Other contributions include Solr on YARN, Solr Scale Toolkit, Solr-Storm, and Spark-Solr integration projects on github



About Solr

- Vibrant, thriving open source community
- Solr 5.2 just released!
  - ✓ Pluggable authentication and authorization
  - ✓ ~2x indexing performance w/ replication
  - ✓ Field cardinality estimation using HyperLogLog
  - ✓ Rule-based replica placement strategy (rack awareness)
- Deploy to YARN cluster using Slider

Lucidworks Fusion



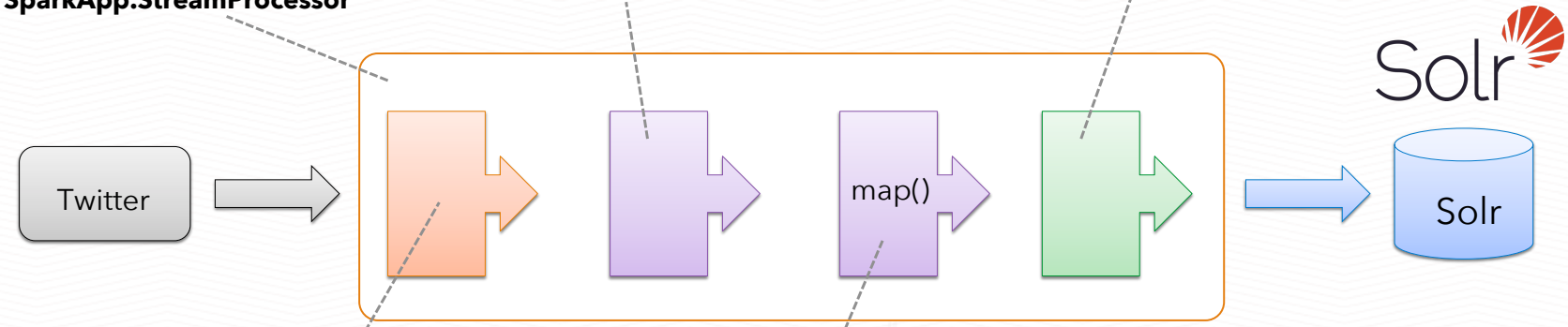
Spark Streaming Example: Solr as Sink

```
./spark-submit --master MASTER --class com.lucidworks.spark.SparkApp spark-solr-1.0.jar \
    twitter-to-solr -zkHost localhost:2181 -collection social
```

class **TwitterToSolrStreamProcessor**
extends **SparkApp.StreamProcessor**

Various transformations / enrichments on each tweet (e.g. sentiment analysis, language detection)

SolrSupport.indexDStreamOfDocs(zkHost, collection, 100, **docs**);



JavaReceiverInputDStream<Status> **tweets** =
TwitterUtils.createStream(jssc, null, filters);

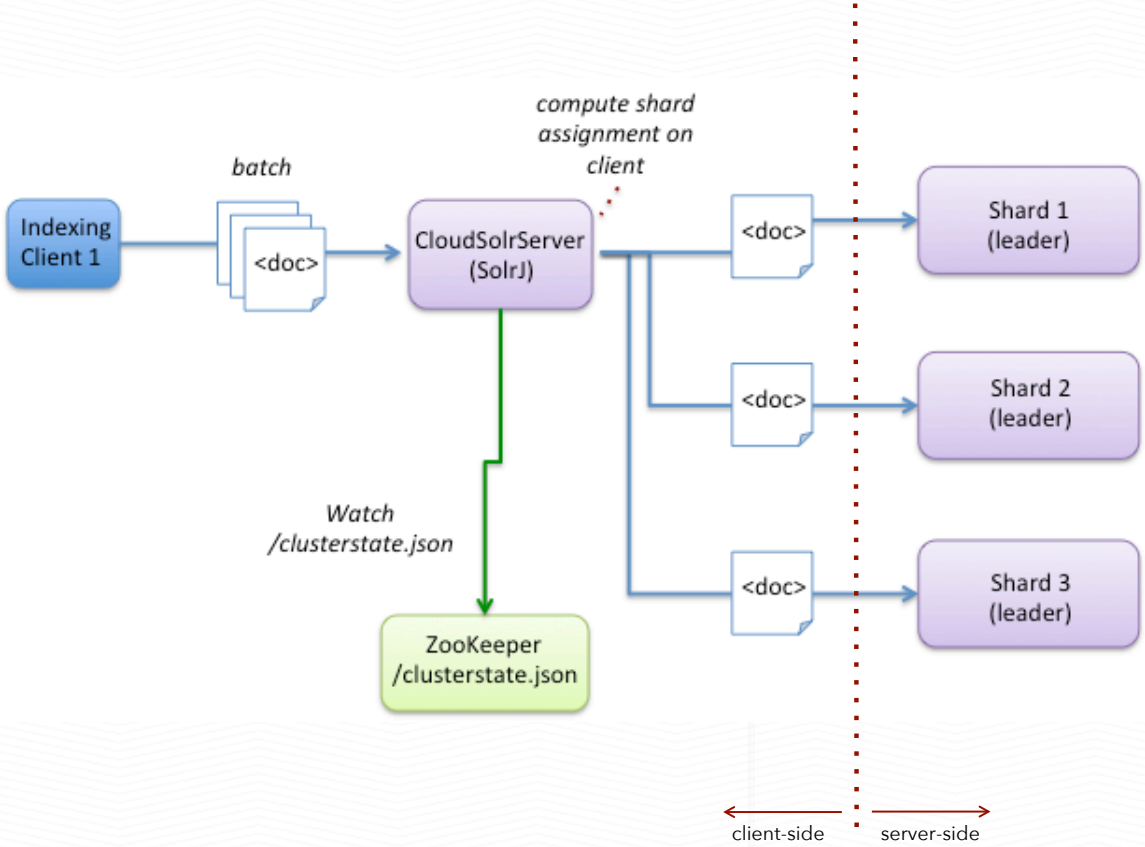
```
JavaDStream<SolrInputDocument> docs = tweets.map(
  new Function<Status,SolrInputDocument>() {
    // Convert a twitter4j Status object into a SolrInputDocument
    public SolrInputDocument call(Status status) {
      SolrInputDocument doc = new SolrInputDocument();
      …
      return doc;
    }});
```

Slide Legend
- Provided by Spark
- Custom Java / Scala code
- Provided by Lucidworks

Spark Streaming Example: Solr as Sink

```
// start receiving a stream of tweets ...
JavaReceiverInputDStream<Status> tweets =
  TwitterUtils.createStream(jssc, null, filters);

// map incoming tweets into SolrInputDocument objects for indexing in Solr
JavaDStream<SolrInputDocument> docs = tweets.map(
  new Function<Status,SolrInputDocument>() {
    public SolrInputDocument call(Status status) {
      SolrInputDocument doc =
        SolrSupport.autoMapToSolrInputDoc("tweet-"+status.getId(), status, null);
      doc.setField("provider_s", "twitter");
      return doc;
    }
  }
);

// when ready, send the docs into a SolrCloud cluster
SolrSupport.indexDStreamOfDocs(zkHost, collection, docs);
```

Direct updates from Spark to shard leaders



Coming Soon! ShardPartitioner

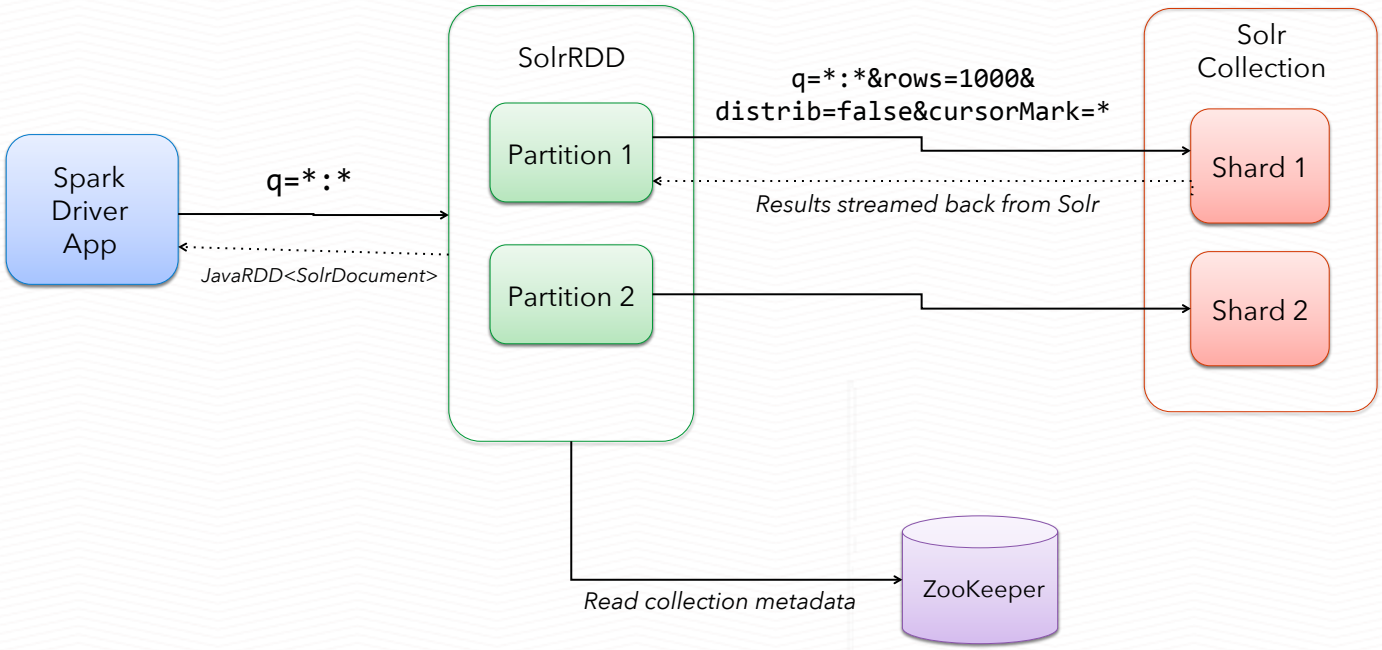
- Custom partitioning scheme for RDD using Solr's DocRouter
- Stream docs directly to each shard leader using metadata from ZooKeeper, document shard assignment, and ConcurrentUpdateSolrClient

```
final ShardPartitioner shardPartitioner = new ShardPartitioner(zkHost, collection);
pairs.partitionBy(shardPartitioner).foreachPartition(
  new VoidFunction<Iterator<Tuple2<String, SolrInputDocument>>>() {
    public void call(Iterator<Tuple2<String, SolrInputDocument>> tupleIter) throws Exception {
      ConcurrentUpdateSolrClient cuss = null;
      while (tupleIter.hasNext()) {
         // ... Initialize ConcurrentUpdateSolrClient once per partition
       cuss.add(doc);
      }
    }
```

SolrRDD: Reading data from Solr into Spark

- Can execute any query and expose as an RDD
- SolrRDD produces `JavaRDD<SolrDocument>`
- Use deep-paging if needed (cursorMark)
- Stream docs from Solr (vs. building lists on the server-side)
- More parallelism using a range filter on a numeric field (_version_)

SolrRDD: Reading data from Solr into Spark



Spark SQL

Query Solr, then expose results as a SQL table

```
JavaSparkContext jsc = new JavaSparkContext(conf);
SQLContext sqlContext = new SQLContext(jsc);

SolrRDD solrRDD = new SolrRDD(zkHost, collection);
DataFrame tweets = solrRDD.asTempTable(sqlContext, queryStr, "tweets");
DataFrame results = sqlContext.sql(
"SELECT COUNT(type_s) FROM tweets WHERE type_s='echo'");

JavaRDD<Row> resultsRDD = results.javaRDD();
List<Long> count = resultsRDD.map(new Function<Row, Long>() { … }).collect();
System.out.println("# of echos : "+count);
```

Query Solr from the Spark Shell

Interactive data mining with the full power of Solr queries

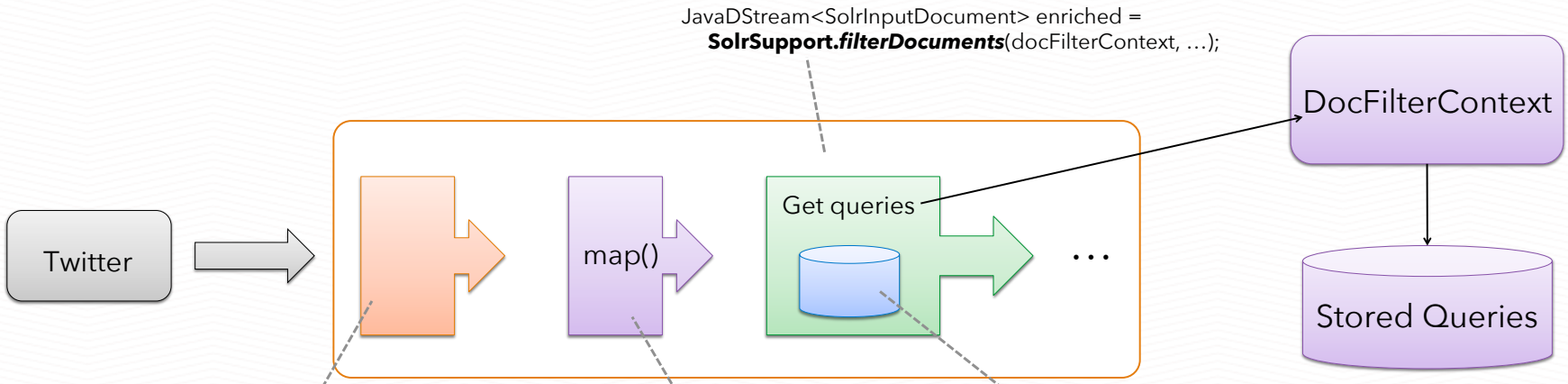
```
ADD_JARS=$PROJECT_HOME/target/spark-solr-1.0-SNAPSHOT.jar bin/spark-shell

import com.lucidworks.spark.SolrRDD;
var solrRDD = new SolrRDD("localhost:9983","gettingstarted");

var tweets = solrRDD.query(sc,"*:*");
var count = tweets.count();

var tweets = solrRDD.asTempTable(sqlContext, "*:*", "tweets");
sqlContext.sql("SELECT COUNT(type_s) FROM tweets WHERE type_s='echo'").show();
```

Document Matching using Stored Queries

- For each document, determine which of a large set of stored queries matches.
- Useful for alerts, alternative flow paths through a stream, etc
- Index a micro-batch into an embedded (in-memory) Solr instance and then determine which queries match
- Matching framework; you have to decide where to load the stored queries from and what to do when matches are found
- Scale it using Spark … need to scale to many queries, checkout Luwak

Document Matching using Stored Queries

Key abstraction to allow you to plug-in how to store the queries and what action to take when docs match



JavaReceiverInputDStream<Status> **tweets** =
TwitterUtils.createStream(jssc, null, filters);

Index docs into an EmbeddedSolrServer Initialized from configs stored in ZooKeeper

```
JavaDStream<SolrInputDocument> docs = tweets.map(
  new Function<Status,SolrInputDocument>() {
    // Convert a twitter4j Status object into a SolrInputDocument
    public SolrInputDocument call(Status status) {
      SolrInputDocument doc = new SolrInputDocument();
      …
      return doc;
    }});
```

ZooKeeper

Slide Legend

- Provided by Spark
- Custom Java / Scala code
- Provided by Lucidworks

Reading Term Vectors from Solr

- Pull TF/IDF (or just TF) for each term in a field for each document in query results from Solr
- Can be used to construct RDD<Vector> which can then be passed to MLLib:

```
SolrRDD solrRDD = new SolrRDD(zkHost, collection);

JavaRDD<Vector> vectors =
  solrRDD.queryTermVectors(jsc, solrQuery, field, numFeatures);
vectors.cache();

KMeansModel clusters =
  KMeans.train(vectors.rdd(), numClusters, numIterations);

// Evaluate clustering by computing Within Set Sum of Squared Errors
double WSSSE = clusters.computeCost(vectors.rdd());
```

Wrap-up and Q & A

Need more use cases …

Feel free to reach out to me with questions:

tim.potter@lucidworks.com / @thelabdude