High-level Architecture

Indexing from Spark

Reading data from Solr + term vectors & Spark SQL

Document Matching

Q&A
Solr user since 2010, committer since April 2014, work for Lucidworks

Focus mainly on SolrCloud features … and bin/solr!

Release manager for Lucene / Solr 5.1

Co-author of Solr in Action

Several years experience working with Hadoop, Pig, Hive, but only started using Spark about 6 months ago …

Other contributions include Solr on YARN, Solr Scale Toolkit, and Spark-Solr integration project on github

About Me …
• Wealth of overview / getting started resources on the Web
  - Start here -> https://spark.apache.org/

• Faster, more modernized alternative to MapReduce
  - Spark running on Hadoop sorted 100TB in 23 minutes (3x faster than Yahoo’s previous record while using 10x less computing power)

• Unified platform for Big Data
  - Great for iterative algorithms (PageRank, K-Means, Logistic regression) & interactive data mining

• Write code in Java, Scala, or Python … REPL interface too

• Runs on YARN (or Mesos), plays well with HDFS
Spark Components

Spark Core
Spark Streaming
MLlib (machine learning)
GraphX (BSP)
Hadoop
YARN
Mesos
Standalone
HDFS

Execution
Model
The Shuffle
Caching
UI / API
engine
cluster
mgmt

Can combine all of these together in the same app!
Spark Master (daemon)
- Keeps track of live workers
- Web UI on port 8080
- Task Scheduler
- Restart failed tasks

Spark Worker Node (1...N)
- Each task works on some partition of a data set to apply a transformation or action

Losing a master prevents new applications from being executed

Can achieve HA using ZooKeeper and multiple master nodes

Spark Executor runs in separate process than slave daemon

Tasks are assigned based on data-locality

When selecting which node to execute a task on, the master takes into account data locality

RDD Graph
- DAG Scheduler
- Block tracker
- Shuffle tracker
val file = spark.textFile("hdfs://...")
val counts = file.flatMap(line => line.split("\t\n")).map(word => (word, 1)).reduceByKey(_ + _).saveAsTextFile("hdfs://...")
RDD Illustrated: Word count

file RDD from HDFS

val file = spark.textFile("hdfs://...")

HDFS file.

flatMap (line => line.split("\t\n"))

Split lines into words

Map words into pairs with count of 1

map (word => (word, 1))

quick brown fox jumped …

quick brown fox (quick,1) (brown,1) (fox,1)

reduceByKey ( _ + _ )

quick brownie recipe …

quick (quick,1) (quick,1)

Send all keys to same reducer and sum (quick,1) (quick,1) …

quick drying glue …

quick (quick,1) (quick,1) (quick,3)

… … … … …

Shuffle across machine boundaries

Executors assigned based on data-locality if possible, narrow transformations occur in same executor

Spark keeps track of the transformations made to generate each RDD

Partition 1

Partition 2

Partition 3
Understanding Resilient Distributed Datasets (RDD)

- Read-only partitioned collection of records with smart fault-tolerance
- Created from external system OR using a transformation of another RDD
- RDDs track the lineage of coarse-grained transformations (map, join, filter, etc)
- If a partition is lost, RDDs can be re-computed by re-playing the transformations
- User can choose to persist an RDD (for reusing during interactive data-mining)
- User can control partitioning scheme; default is based on the input source
• https://github.com/LucidWorks/spark-solr/

• Streaming applications
  - Real-time, streaming ETL jobs
  - Solr as sink for Spark job
  - Real-time document matching against stored queries

• Distributed computations (interactive data mining, machine learning)
  - Expose results from Solr query as Spark RDD (resilient distributed dataset)
  - Optionally process results from each shard in parallel
  - Read millions of rows efficiently using deep paging

SparkSQL DataFrame support (uses Solr schema API) and Term Vectors too!
Spark Streaming: Nuts & Bolts

• Transform a stream of records into small, deterministic batches
  • Discretized stream: sequence of RDDs
  • Once you have an RDD, you can use all the other Spark libs (MLlib, etc)
  • Low-latency micro batches
    • Time to process a batch must be less than the batch interval time

• Two types of operators:
  • Transformations (group by, join, etc)
  • Output (send to some external sink, e.g. Solr)

• Impressive performance!
  • 4GB/s (40M records/s) on 100 node cluster with less than 1 second latency

• Haven't found any unbiased, reproducible performance comparisons between Storm / Spark
Spark Streaming Example:

Solr as Sink

twitter-to-solr

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

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

```java
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;
    }
});
```

SolrSupport.indexDStreamOfDocs(zkHost, collection, 100, docs);
Spark Streaming Example:

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

// No 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");
      doc.setField("author_s", status.getUser().getScreenName());
      doc.setField("type_s", status.isRetweet() ? "echo" : "post");
      return doc;
    }
  });

// when ready, send the docs into a SolrCloud cluster
SolrSupport.indexDStreamOfDocs(zkHost, collection, docs);
```
public static void indexDStreamOfDocs(final String zkHost, final String collection, final int batchSize, JavaDStream<SolrInputDocument> docs) {
  docs.foreachRDD(new Function<JavaRDD<SolrInputDocument>, Void>() {
    public Void call(JavaRDD<SolrInputDocument> solrInputDocumentJavaRDD) throws Exception {
      solrInputDocumentJavaRDD.foreachPartition(new VoidFunction<Iterator<SolrInputDocument>>() {
        public void call(Iterator<SolrInputDocument> solrInputDocumentIterator) throws Exception {
          final SolrServer solrServer = getSolrServer(zkHost);
          List<SolrInputDocument> batch = new ArrayList<SolrInputDocument>();
          while (solrInputDocumentIterator.hasNext()) {
            batch.add(solrInputDocumentIterator.next());
            if (batch.size() >= batchSize) 
              sendBatchToSolr(solrServer, collection, batch);
          }
          if (!batch.isEmpty()) 
            sendBatchToSolr(solrServer, collection, batch);
        }
      });
    return null;
  });
}
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.

... need to scale to many queries, check Luwak.
JavaDStream<SolrInputDocument> enriched = SolrSupport.filterDocuments(docFilterContext);

Get queries

...
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.

```scala
Partitioner = new ShardPartitioner(zkHost, collection).foreachPartition(
  Tuple2<String, SolrInputDocument>>>() { 
    Tuple2<String, SolrInputDocument>> tupleIter =
    ConcurrentUpdateSolrClient cuss = null; 
    try {
      concurrentUpdateSolrClient once per partition
```
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)

For reading full result sets where global sort order doesn’t matter, parallelize query execution by distributing requests across the Spark cluster.

```java
val results = solrRDD.queryShards(jsc, solrQuery)
```
Construct RDD<Vector> which can then be processed to:

```scala
val solrRdd = new SolrRdd(zkHost, collection);

val vectors = solrRdd.queryTermVectors(jsc, solrQuery, field, numFeatures);

vectors.cache();

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

// Evaluate clustering by computing Within Set Sum of Squared Errors
val wssse = clusters.computeCost(vectors.rdd());
SolrQuery(...);

solrQuery.setFields("text_t","type_s");

SolrRDD solrRDD(zkHost, collection);

solrJavaRDD = solrRDD.queryShards(jsc, solrQuery);

new SQLContext(jsc);

Context, solrQuery, solrJavaRDD, zkHost, collection

df.applySchema(sqlContext, solrQuery, solrJavaRDD, zkHost, collection)

df.registerTempTable("tweets");

JavaSchemaRDD results = sqlContext.sql("SELECT COUNT(type_s) FROM tweets WHERE type_s='echo'");

List<Long> count = results.javaRDD().map(new Function<Row, Long>() {
    public Long call(Row row) {
        return row.getLong(0);
    }
}).collect();

System.out.println("# of echos: "+count.get(0));
Wrap-up and Q & A

- Reference implementation of Solr and Spark on YARN
- Formal benchmarks for reads and writes to Solr
  - improving replication performance
- Checkout SOLR-6816 – improving replication performance
- Add Spark support to Solr Scale Toolkit
- Integrate metrics to give visibility into performance
- More use cases …

Feel free to reach out to me with questions:
tim.potter@lucidworks.com / @thelabdude