GSoC Midterm Report

Spark Backend Support for Gora (GORA-386) Midterm Report

Furkan KAMACI[1]

[1] *Istanbul Technical University*

Abstract—This reports explains the progress up to midterm evaluations of GSoC 2015.

Index Terms— Apache Gora, GSoC, Spark

I. INTRODUCTION

GORA-386 aims to develop a Spark Backend for Apache Gora. Apache Gora open source framework provides an in-memory data model and persistence for big data. Gora supports persisting to column stores, key value stores, document stores and RDBMSs, and analyzing the data with extensive Apache Hadoop MapReduce support. Even Spark is so powerful compared to Map/Reduce which Gora currently supports; there is no Spark backend for Gora. This proposal aims to develop a solution for Gora to support Spark.

During GSoC period, I've blogged at my personal website (http://furkankamaci.com/) and created a fork from Apache Gora's master branch and worked on it: https://github.com/kamaci/gora

This report explains what is done up to now and what will be the next steps. Currently, project progress is ahead from the proposed timeline. Here is the tag of this report:

Project Name	Spark Backend Support for Gora (GORA-386)
Project URL	https://issues.apache.org/jira/browse/GORA-386
Report #	5
Report Compiled By	Furkan KAMACI
Report Date	30.06.2015

II. PROPOSED SCHEDULE&TIMELINE

Suggested schedule and timeline is as follows:

1) Analyzing The Problem (1 Week)
   a) Problem will be analyzed with more detail.

2) Spark - Map/Reduce Translation (4 weeks)
   a) Behaviors specific to Hadoop’s implementation rather than the idea of MapReduce in the abstract will be recreated according to Spark.

3) Gora – Translated Map/Reduce Integration (4 weeks)
   a) Gora has its own input/output format and Spark uses RDD. After necessary implementation is done for Spark – Map/Reduce translation, Gora Integration will be done. (2 weeks).

4) Generic Abstraction Layer Backend (1 week)
   a) A generic layer backend that let other projects' integration as like Spark.

5) Test (1 week)
   a) Implementation tests will be written and run.

6) Documentation (1 week)
   a) Documentation will be prepared.

III. COMMUNITY BONDING

At community bonding period, Apache Gora documentation is read and Apache Gora source code is checked to be more familiar with project. Related projects are analyzed including Apache Flink and Apache Crunch to implement a Spark backend into Apache Gora.

An issue from Jira is picked up (https://issues.apache.org/jira/browse/GORA-262) and fixed at Apache Gora. Security is always an issue when writing into Solr and resolved issue GORA-262 is about adding support for HTTPClient authentication in gora-solr module.

On the other hand, Apache Gora documentation is organized and introduced with Apache Gora dev team. I've also blogged at my personal website about my acceptance of GSoC (http://furkankamaci.com/gsoc-2015-acceptance-for-apache-gora/)

IV. CODING PERIOD

I've separated this period into three main parts as followed:

Part 1

Due to implementing this project needs an infrastructure about Apache Spark, I've started with analyzing Spark's first papers. I've analyzed *"Spark: Cluster Computing with Working"* (http://www.cs.berkeley.edu/~matei/papers/2010/hotcloud_spark.pdf) and *"Resilient Distributed Datasets: A Fault-Tolerant Abstraction forIn-Memory Cluster Computing"* (https://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf). I've published two posts about *Spark and Cluster Computing* (http://furkankamaci.com/spark-and-cluster-computing/) and *Resilient Distributed Datasets* (http://furkankamaci.com/resilient-distributed-datasets-rdds/) at my personal blog.

I've followed Apache Spark documentation and developed examples to analyze RDDs.

Part 2

Apache Gora's GoraInputFormat class and Spark's newHadoopRDD method are analyzed. An example application implemented to read data from Hbase. Here is the most important line of code for Apache Gora and Spark integration:

```
JavaPairRDD<ImmutableBytesWritable , Result> hbaseRDD = sc.newAPIHadoopRDD(conf, TableInputFormat.class
, ImmutableBytesWritable.class, Result.class);
```

Apache Gora supports reading/writing data from/to Hadoop files. Spark has a method for generating an RDD compatible with Hadoop files. So, an architecture is designed which creates a bridge between GoraInputFormat and RDD due to both of them support Hadoop files.

Part 3

A base class is created for Apache Gora and Spark integration named as: GoraSparkEngine. It has initialize methods that takes Spark context, data store, optional Hadoop configuration and returns an RDD. One of its initialize method's signature and JavaDoc is as follows:

```
/**
 * Initializes a {@link JavaPairRDD} from given Spark context and data store.
 * If given data store is {@link Configurable} and has not a configuration
 * than a Hadoop configuration is created otherwise existed configuration is
 * used.
 *
 * @param sparkContext
 *          Spark context
 * @param dataStore
 *          Data store
 * @return initialized rdd
 */
public JavaPairRDD<K, V> initialize(JavaSparkContext sparkContext,
    DataStore<K, V> dataStore) {}
```

Apache Gora has a tutorial module and there is an example for map reduce jobs which is named as LogAnalytics. It is documented as follows:

LogAnalytics is the tutorial class to illustrate Gora MapReduce API. The analytics mapreduce job reads the web access data stored earlier by the LogManager, and calculates the aggregate daily pageviews. The output of the job is stored in a Gora compatible data store.

After implementing a base for GoraSpark engine, I've developed a new example aligned to LogAnalytics named as: LogAnalyticsSpark. I've developed map and reduce parts (except for writing results into database) which does the same thing as LogAnalytics and also something more i.e. printing number of lines in tables.

Here is the part of how to run GoraSparkEngine:

```
SparkConf sparkConf = new SparkConf().setAppName("Gora Integration Application").setMaster("local");

Class[] c = new Class[1];
c[0] = Pageview.class;
sparkConf.registerKryoClasses(c);

JavaSparkContext sc = new JavaSparkContext(sparkConf);
Configuration hadoopConf = new Configuration();
DataStore<Long, Pageview> dataStore = DataStoreFactory.getDataStore(inStoreClass, Long.class, Pageview.class,
hadoopConf);
JavaPairRDD<Long, Pageview> goraRDD = goraSparkEngine.initialize(sc, dataStore);
```

When we get an RDD, we can do the operations over it as like making operations on any other RDDs which is not created over Apache Gora. Here is the map function to be used at calculation:

```
/**
 * map function used in calculation
 */
private static Function<Pageview, Tuple2<Tuple2<String, Long>, Long>> mapFunc = new Function<Pageview,
Tuple2<Tuple2<String, Long>, Long>>() {
  @Override
  public Tuple2<Tuple2<String, Long>, Long> call(Pageview pageview)
      throws Exception {
    String url = pageview.getUrl().toString();
    Long day = getDay(pageview.getTimestamp());
    Tuple2<String, Long> keyTuple = new Tuple2<>(url, day);
    return new Tuple2<>(keyTuple, 1L);
  }
};
```

Here is the way that map function is called on an RDD which is created via GoraSparkEngine with GoraInputFormat:

```
JavaRDD<Tuple2<Tuple2<String, Long>, Long>> mappedGoraRdd = goraRDD.values().map(mapFunc);
```

Whole code can be checked from code base: https://github.com/kamaci/gora

V. ALIGNMENT WITH TIMELINE

Project progress is ahead from the proposed timeline up to now. GoraInputFormat and RDD transformation is done and it is shown that map, reduce and other methods can properly work on that kind of RDDs.

VI. CONCLUSION & NEXT STEPS

Spark jobs can be run via Gora and next step will be support of persisting data via Gora. My previous reports can be found here: https://cwiki.apache.org/confluence/display/GORA/Spark+Backend+Support+for+Gora+%28GORA-386%29+Reports

Before the next steps, I am planning to design an overall architecture according to feedbacks from community (there are some prerequisites when designing an architecture: i.e. configuration of a context at Spark cannot be changed after context has been initialized).

When necessary functionalities are implemented examples, tests and documentations will be done. After that if I have extra time, I'm planning to make a performance benchmark of Apache Gora with Hadoop MapReduce, Hadoop MapReduce, Apache Spark and Apache Gora with Spark as well.

VII. REFERENCES

https://issues.apache.org/jira/browse/GORA-386
http://furkankamaci.com/
https://github.com/kamaci/gora
https://issues.apache.org/jira/browse/GORA-262
http://furkankamaci.com/gsoc-2015-acceptance-for-apache-gora/
http://www.cs.berkeley.edu/~matei/papers/2010/hotcloud_spark.pdf
https://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf
http://furkankamaci.com/spark-and-cluster-computing/
http://furkankamaci.com/resilient-distributed-datasets-rdds/
https://cwiki.apache.org/confluence/display/GORA/Spark+Backend+Support+for+Gora+%28GORA-386%29+Reports
http://gora.apache.org/current/tutorial.html#introduction
http://en.wikipedia.org/wiki/Apache_Gora