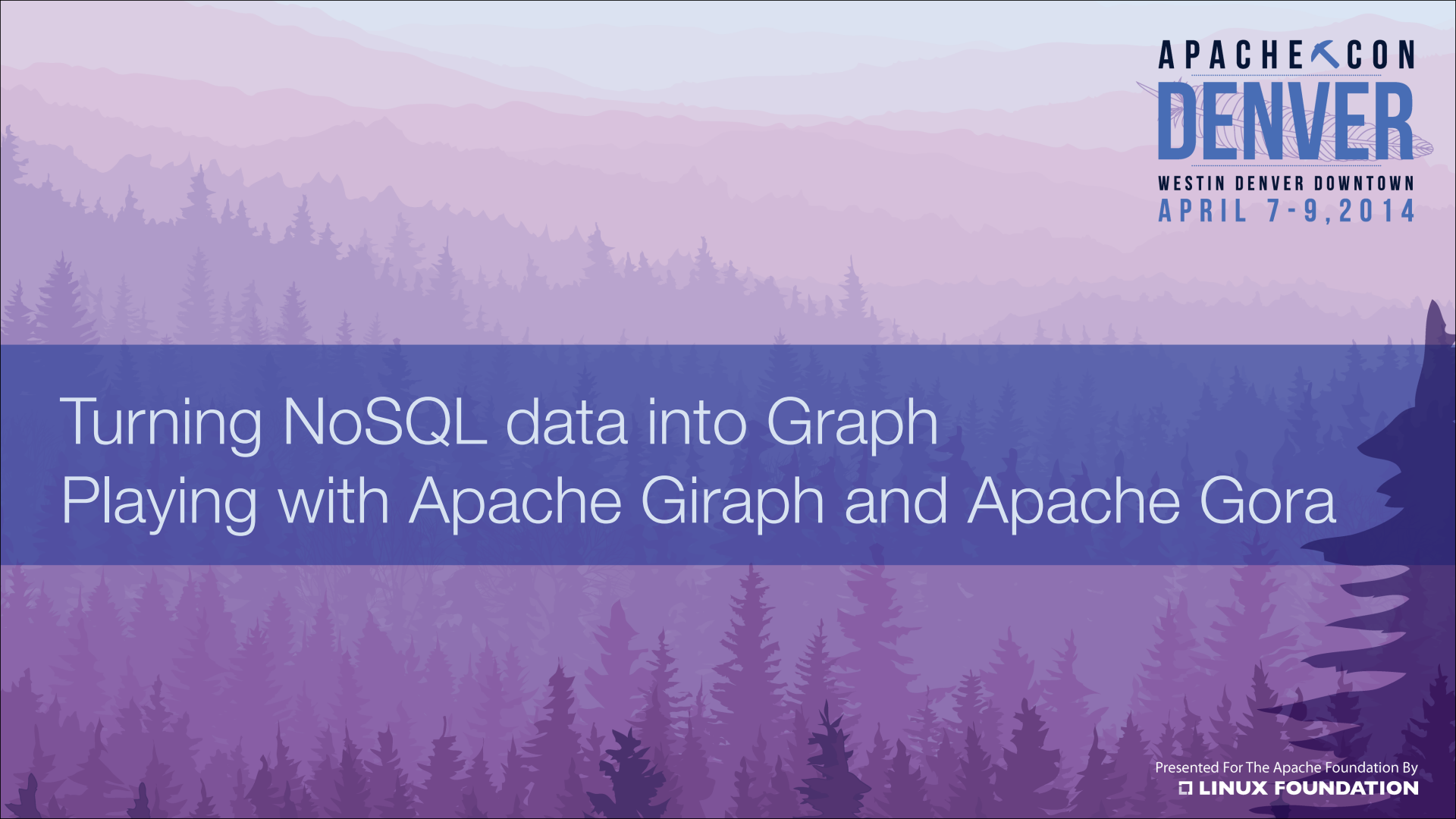


Turning NoSQL data into Graph
Playing with Apache Giraph and Apache Gora

Presented For The Apache Foundation By
LINUX FOUNDATION


APACHECON
DENVER
WESTIN DENVER DOWNTOWN
APRIL 7-9, 2014
Team
Presented For The Apache Foundation By
LINUX FOUNDATION

Renato Marroquín

- PhD student:
- Interested in:
  - Information retrieval.
  - Distributed and scalable data management.
- Apache Gora:
  - PPMC Member
  - Committer.
- rmarroquin [at] apache [dot] org

Claudio Martella

- PhD student: LSDS @VU University Amsterdam.
- Interested in
  - Complex Networks.
  - Distributed and scalable infrastructures.
- Apache Girapher:
  - PPMC Member
  - Committer.
- claudio [at] apache [dot] org

Lewis McGibbney

- Scottish expat fae Glasgow
- Post Doc @Stanford University: Engineering Informatics
- Quantity Surveyor/Cost Consultant by profession
- Cycling mad
- Keen OSS enthusiast @TheASF and beyond
- lewismc [at] apacher [dot] org

APACHECON
DENVER
WESTIN DENVER DOWNTOWN
APRIL 7-9, 2014

APACHECON
DENVER
WESTIN DENVER DOWNTOWN
APRIL 7-9, 2014

Apache Gora

Presented For The Apache Foundation By
LINUX FOUNDATION

What is Apache Gora?

- **Data Persistence :** Persisting objects to Column stores, key-value stores, SQL databases and to flat files in local file system of Hadoop HDFS.
- **Data Access :** An easy to use Java-friendly common API for accessing the data regardless of its location.
- **Indexing :** Persisting objects to Lucene and Solr indexes, accessing/ querying the data with Gora API.
- **Analysis :** Accesing the data and making analysis through adapters for Apache Pig, Apache Hive and Cascading
- **MapReduce support :** Out-of-the-box and extensive MapReduce (Apache Hadoop) support for data in the data store.

What is Apache Gora?

- Provides an in-memory data model and persistence for big data.
- Gora supports:

How does Gora work?

1. Define your schema using Apache AVRO.
2. Compile your schemas using Gora's Compiler.
3. Create a *mapping* between logical and physical layout.
4. Update *gora.properties* file to set back-end properties.

Rock the NoSQL world!!!

How does Gora work?

1. Define your schema using Apache AVRO.

```
{
  "type": "record",
  "name": "Employee",
  "namespace": "org.apache.gora.examples.generated",
  "fields" : [
    {"name": "name", "type": "string"},
    {"name": "dateOfBirth", "type": "long"},
    {"name": "ssn", "type": "string"},
    {"name": "salary", "type": "int"},
    {"name": "boss", "type":["null","Employee","string"]},
    {"name": "webpage", "type":["null",
      {
        "type": "record",
        "name": "WebPage",
        "namespace": "org.apache.gora.examples.generated",
        "fields" : [
         {"name": "url", "type": "string"},
         {"name": "content", "type": ["null","bytes"]},
```

How does Gora work?

2. Compile your schemas using Gora's Compiler.

```
java -jar gora-core-XYZ-.jar
    o.a.gora.compiler.GoraCompiler.class
        employee.avsc
        gora-app/src/main/java/
```

How does Gora work?

2. Compile your schemas using Gora's Compiler.

```
2  *Licensed to the Apache Software Foundation (ASF) under one
18 
19 package org.apache.gora.examples.generated;
20 
21 import java.nio.ByteBuffer;
41 @SuppressWarnings("all")
42 public class Employee extends PersistentBase {
43   
44   /**
45    * Variable holding the data bean schema.
46    */
47   public static final Schema _SCHEMA = Schema.parse("{\"type\":\"record\",\"name\":\"Em
48   
49   /**
50    * Enum containing all data bean's fields.
51    */
52   public static enum Field {
53     NAME(0,"name"),
54     DATE_OF_BIRTH(1,"dateOfBirth"),
55     SSN(2,"ssn"),
56     SALARY(3,"salary"),
57     BOSS(4,"boss"),
58     WEBPAGE(5,"webpage");
59     
60     /**
61      * Field's index.
62      */
63     private int index;
```

How does Gora work?

3. Create a *mapping* between logical and physical layout.

```
<gora-orm>
¶
  <table name="Employee"> <!-- optional descriptors for tables -->
    <family name="info"/> <!-- This can also have params like compression, bloom filters -->
  </table>
¶
...
¶
  <class name="org.apache.gora.examples.generated.Employee" keyClass="java.lang.String" table="Employee">
    <field name="name" family="info" qualifier="nm"/>
    <field name="dateOfBirth" family="info" qualifier="db"/>
    <field name="ssn" family="info" qualifier="sn"/>
    <field name="salary" family="info" qualifier="sl"/>
    <field name="boss" family="info" qualifier="bs"/>
    <field name="webpage" family="info" qualifier="wp"/>
  </class>
¶
</gora-orm>
```

How does Gora work?

4. Update *gora.properties* file to set back-end properties.

```
#####################
# Misc properties   #
#####################
gora.datastore.default=org.apache.gora.hbase.store.HBaseStore
gora.datastore.autocreateschema=true

#######################
# HBaseStore properties #
#######################
#gora.datastore.default=org.apache.gora.hbase.store.HBaseStore
gora.hbasestore.scanner.caching=1000

############################
# Default Accumulo properties #
############################
#gora.datastore.default=org.apache.gora.accumulo.store.AccumuloStore
#gora.datastore.accumulo.instance=test14
#gora.datastore.accumulo.zookeepers=localhost
#gora.datastore.accumulo.user=root
#gora.datastore.accumulo.password=secret

############################
# Default SqlStore properties #
############################

#gora.sqlstore.jdbc.driver=org.hsqldb.jdbcDriver
#gora.sqlstore.jdbc.url=jdbc:hsqldb:hsql://localhost/nutchtest
# gora.sqlstore.jdbc.user=
# gora.sqlstore.jdbc.password=

############################
# Default AvroStore properties #
```

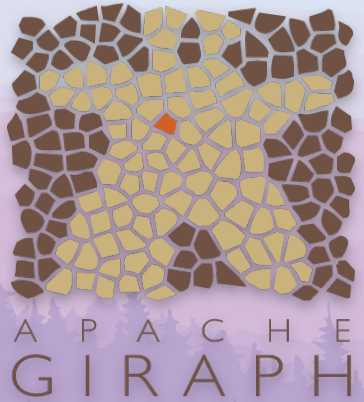
How does Gora work?

Rock the NoSQL world!

```
/**
 * Simple GORA example
 */
public class SimpleGora<K, T extends PersistentBase> {

  /**
   * @param args
   * @throws GoraException
   */
  @SuppressWarnings("unchecked")
  public static void main(String[] args) throws GoraException {
    DataStore<String, Employee> dataStore =
        DataStoreFactory.createDataStore(HBaseStore.class,
                                          String.class, Employee.class,
                                          new Configuration());

    Employee emp1 = new Employee();
    emp1.setSsn(new Utf8("43024255"));
    emp1.setName(new Utf8("Renato"));
    dataStore.put(emp1.getSsns(), emp1);
    dataStore.flush();
    Employee sameEmp = dataStore.get(emp1.getSsns());
    System.out.println(sameEmp.toString());
  }
}
```



APACHECON
DENVER
WESTIN DENVER DOWNTOWN
APRIL 7-9, 2014

Apache Giraph

Presented For The Apache Foundation By
LINUX FOUNDATION

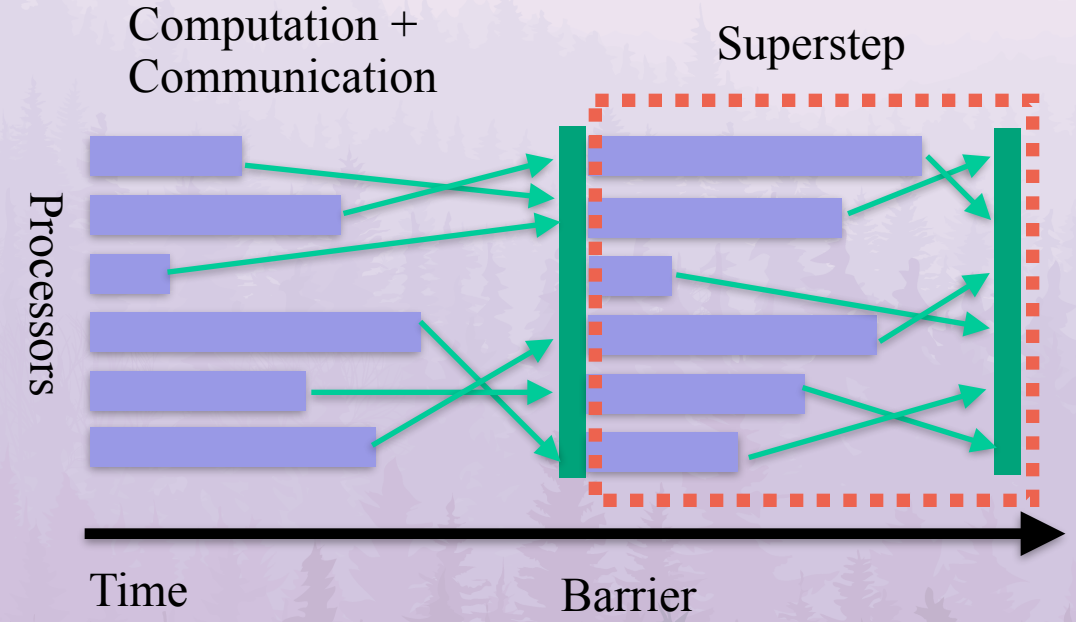
MapReduce and Graphs

- Plain MapReduce is not well suited for graph algorithms because:
  - Graph algorithms are iterative.
  - Not intuitive in MapReduce.
  - Unnecessarily slow
    - Each iteration is a single MapReduce job with too much overhead
    - Separately scheduled
    - The graph structure is read from disk
    - The intermediate results are read from disks
  - Hard to implement

Google's Pregel

- Introduced on 2010
- Based on Valiant's BSP
  - “Think like a vertex” that can send messages to any vertex in the graph using the bulk synchronous parallel programming model.
  - Computation complete when all components complete.
- Batch-oriented processing
- Computation happens in-memory
- Master/slave architecture

Bulk synchronous parallel



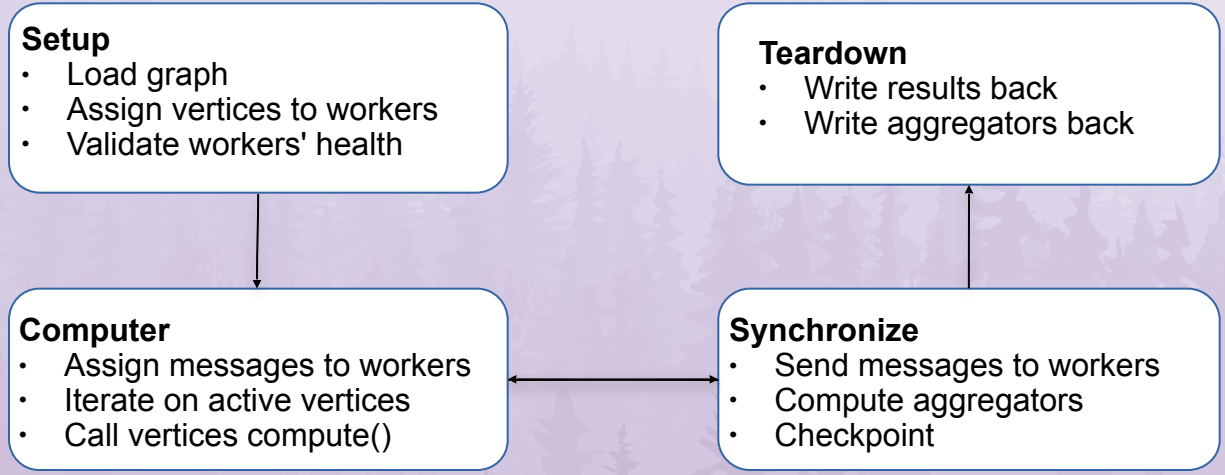
Open source implementations

- There are some such as:
  - **Apache Giraph**
  - Apache Hama
  - GoldenOrb
  - Signal/Collect

Apache Giraph

- Incubated since summer 2011
- Written in Java
- Implements Pregel's API
- Runs on existing MapReduce infrastructure
- Active community from Yahoo!, Facebook, LinkedIn, Twitter, and more.
- It's a single Map-only job
- It runs on Hadoop in-memory.
- Fault tolerant
    - Zookeeper for state, No SPOF

During execution time



Giraph's components

- Master
  - Application coordinator
  - One active master at a time
  - Assigns partition owners to workers prior to each superstep
  - Synchronizes supersteps
- Worker – Computation & messaging
  - Loads the graph from input splits
  - Performs computation/messaging of its assigned partitions
- Zookeeper
  - Maintains global application state

What is needed then?

- Your algorithm in the Pregel model.
- A VertexInputFormat to read your graph.
  e.g. <vertex><neighbor1><neighbor2>
- A VertexOutputFormat to write back the results.
  e.g. <vertex> <pageRank>
- You could define:
  - A Combiner (for reducing number of messages sent/received)
  - An Aggregator (for enabling global computation)

Running a Giraph job

- It is just like running Hadoop

```
$HADOOP_HOME/bin/hadoop jar
    giraph-examples-1.1.0-XXX-jar-with-dependencies.jar
    o.a.g.GiraphRunner o.a.g.examples.SimpleShortestPathsComputation
    -vif o.a.g.io.formats.JsonLongDoubleFloatDoubleVertexInputFormat
    -vip /user/hduser/input/tiny_graph.txt
    -vof o.a.g.io.formats.IdWithValueTextOutputFormat
    -op /user/hduser/output/shortestpaths
    -w 1
```

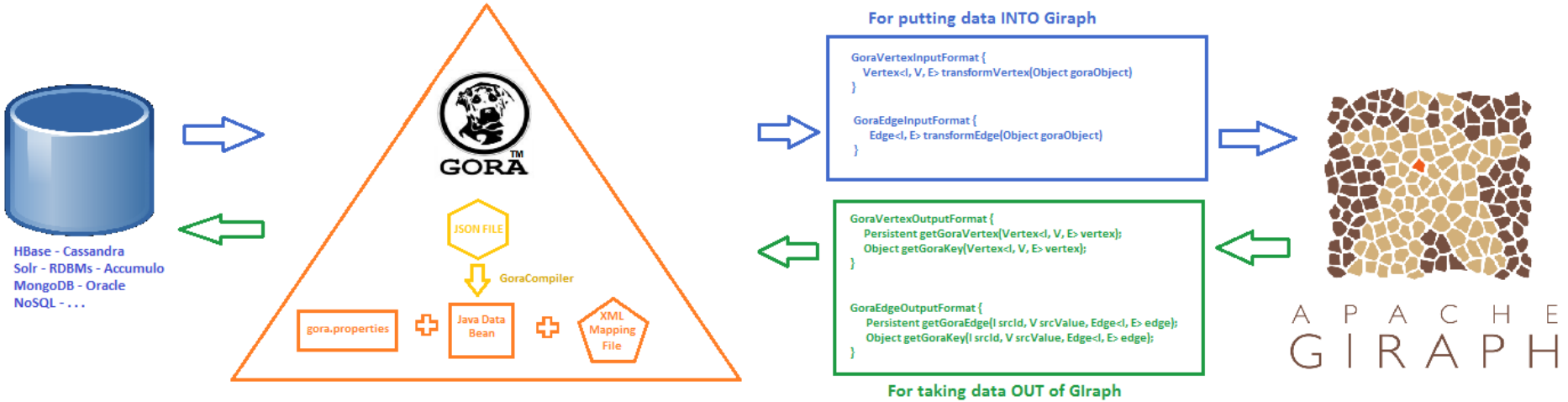
Apache Giraph + Apache Gora

Presented For The Apache Foundation By
LINUX FOUNDATION

The project idea

- Integrating Apache Gora with other cool projects.
- Provide access to different data stores out-of-the-box for Apache Giraph.
- Give users more flexibility when deciding how to run graph algorithms.
- Make the Hadoop Env bigger.
- Apply to for the Google Summer of Code Project.

The big picture



Integration hooks

- ***Vertices***


```
public abstract class GoraVertexInputFormat<
        I extends WritableComparable,
        V extends Writable,
        E extends Writable>
        extends VertexInputFormat<I, V, E> {

  /** Start key for querying Gora data store. */
  private static Object START_KEY;

  /** End key for querying Gora data store. */
  private static Object END_KEY;
```

```
/** @param conf configuration parameters */
public void checkInputSpecs(Configuration conf) {
  String sDataStoreType =
      GIRAPH_GORA_DATASTORE_CLASS.get(getConf());
  String sKeyType =
      GIRAPH_GORA_KEY_CLASS.get(getConf());
  String sPersistentType =
      GIRAPH_GORA_PERSISTENT_CLASS.get(getConf());
  String sKeyFactoryClass =
      GIRAPH_GORA_KEYS_FACTORY_CLASS.get(getConf());
```

```
/**
 * Gets the splits for a data store.
 * @param context JobContext
 * @param minSplitCountHint Hint for a minimum split count
 * @return List<InputSplit> A list of splits
 */
@Override
public List<InputSplit> getSplits(JobContext context, int minSplitCountHint)
  throws IOException, InterruptedException {
```

```
kFact.setDataStore(getDataStore());
setStartKey(kFact.buildKey(sKey));
setEndKey(kFact.buildKey(eKey));
Query tmpQuery = GoraUtils.getQuery(
    getDataStore(), getStartKey(), getEndKey());
GORA_INPUT_FORMAT.setQuery(tmpQuery);
List<InputSplit> splits = GORA_INPUT_FORMAT.getSplits(context);
return splits;
```

Integration hooks

- ***Vertices***

```
/**
 * Abstract class to be implemented by the user based on their specific
 * vertex input. Easiest to ignore the key value separator and only use
 * key instead.
 */
protected abstract class GoraVertexReader extends VertexReader<I, V, E> {
  /** Current vertex */
  private Vertex<I, V, E> vertex;
  /** Results gotten from Gora data store. */
  private Result readResults;

public void initialize(InputSplit inputSplit, TaskAttemptContext context)
  throws IOException, InterruptedException {
  getResults();
  RECORD_COUNTER = 0;
}
public boolean nextVertex() throws IOException, InterruptedException {
  boolean flg = false;
  try {
    flg = this.getReadResults().next();
    this.vertex = transformVertex(this.getReadResults().get());
    RECORD_COUNTER++;
  } catch (Exception e) {
    LOG.error("Error transforming vertices.");
    LOG.error(e.getMessage());
    flg = false;
  }
  LOG.debug(RECORD_COUNTER + " were transformed.");
  return flg;
}
```

Integration hooks

- ***Edges***


```
public abstract class GoraEdgeInputFormat
  <I extends WritableComparable, E extends Writable>
  extends EdgeInputFormat<I, E> {

  /** Start key for querying Gora data store. */
  private static Object START_KEY;

  /** End key for querying Gora data store. */
  private static Object END_KEY;
```

```
/**
 * @param conf configuration parameters
 */
public void checkInputSpecs(Configuration conf) {
  String sDataStoreType =
      GIRAPH_GORA_DATASTORE_CLASS.get(getConf());
  String sKeyType =
      GIRAPH_GORA_KEY_CLASS.get(getConf());
  String sPersistentType =
      GIRAPH_GORA_PERSISTENT_CLASS.get(getConf());
  String sKeyFactoryClass =
      GIRAPH_GORA_KEYS_FACTORY_CLASS.get(getConf());
```

```
/**
 * Gets the splits for a data store.
 * @param context JobContext
 * @param minSplitCountHint Hint for a minimum split count
 * @return List<InputSplit> A list of splits
 */
@Override
public List<InputSplit> getSplits(JobContext context, int minSplitCountHint)
  throws IOException, InterruptedException {
```

```
kFact.setDataStore(getDataStore());
setStartKey(kFact.buildKey(sKey));
setEndKey(kFact.buildKey(eKey));
Query tmpQuery = GoraUtils.getQuery(
    getDataStore(), getStartKey(), getEndKey());
GORA_INPUT_FORMAT.setQuery(tmpQuery);
List<InputSplit> splits = GORA_INPUT_FORMAT.getSplits(context);
return splits;
```

Integration hooks

- ***Edges***

```
/**
 * Abstract class to be implemented by the user based on their specific
 * vertex input. Easiest to ignore the key value separator and only use
 * key instead.
 */
protected abstract class GoraEdgeReader extends EdgeReader<I, E> {
  /** current edge obtained from Rexster */
  private Edge<I, E> edge;
  /** Results gotten from Gora data store. */
  private Result readResults;
```

```
@Override
public void initialize(InputSplit inputSplit, TaskAttemptContext context)
  throws IOException, InterruptedException {
  getResults();
  RECORD_COUNTER = 0;
}
```

```
public boolean nextEdge() throws IOException, InterruptedException {
  boolean flg = false;
  try {
    flg = this.getReadResults().next();
    this.edge = transformEdge(this.getReadResults().get());
    RECORD_COUNTER++;
  } catch (Exception e) {
    LOG.debug("Error transforming vertices.");
    flg = false;
  }
  LOG.debug(RECORD_COUNTER + " were transformed.");
  return flg;
}
```

Integration hooks

- ***Key factory***

```
/**
 * Class used to convert strings into more complex keys.
 */
public abstract class KeyFactory {

  /**
   * Data store used for creating a new key.
   */
  private DataStore dataStore;

  /**
   * Builds a key from a string parameter.
   * @param keyString the key object as a string.
   * @return the key object.
   */
  public abstract Object buildKey(String keyString);

  /**
   * Gets the data store used in this factory.
   * @return the dataStore
   */
  public DataStore getDataStore() {
    return dataStore;
  }

  /**
   * Sets the data store used in this factory.
   * @param dataStore the dataStore to set
   */
  public void setDataStore(DataStore dataStore) {
    this.dataStore = dataStore;
  }
}
```

Parameters offered

Label	Description
giraph.gora.datastore.class	Gora DataStore class to access to data from - required.
giraph.gora.key.class	Gora Key class to query the datastore - required.
giraph.gora.persistent.class	Gora Persistent class to read objects from Gora - required.
giraph.gora.keys.factory.class	Keys factory to convert strings into desired keys - required.
giraph.gora.output.datastore.class	Gora DataStore class to write data to - required.
giraph.gora.output.key.class	Gora Key class to write to datastore - required.
giraph.gora.output.persistent.class	Gora Persistent class to write to Gora - required.
giraph.gora.start.key	Gora start key to query the datastore.
giraph.gora.end.key	Gora end key to query the datastore.

Rocks in the way

- Dependency issues.
  - Supported versions by each project.
  - Maven war for handling cyclic dependencies.
- Hadoop issues.
  - Not all data stores support MapReduce out of the box.
  - Finding what it is necessary to be in the classpath.
- Providing an API between both projects that is:
  - Flexible.
  - Simple.
  - Pluggable.

So now what?

1. Create your data beans with Gora.

```
{"type": "record",
"name": "Vertex",
"namespace": "org.apache.giraph.gora.generated",
"fields" : [
          {"name": "vertexId", "type": "long"},
          {"name": "value", "type": "float"},
          {"name": "edges",
           "type": {
                   "type":"array", "items": {
                                   "name": "Edge",
                                   "type": "record",
                                   "namespace": "org.apache.giraph.gora.generated",
                                   "fields": [
                                          {"name": "vertexId", "type": "long"},
                                          {"name": "edgeValue", "type": "float"}
                                         ]
                                   }
                   }
         }
         ]
}
```

So now what?

2. Compile them.

java -jar gora-core-XYZ.jar o.a.gora.compiler.GoraCompiler.class vertex.avsc gora-app/src/main/java/

```
/**
 * Class for defining a Giraph-Vertex.
 */
@SuppressWarnings("all")
public class GVertex extends PersistentBase {
  /**
   * Schema used for the class.
   */
  public static final Schema OBJ_SCHEMA = Schema.parse(
      "{\"type\":\"record\",\"name\":\"Vertex\"," +
      "\"namespace\":\"org.apache.giraph.gora.generated\"," +
      "\"fields\":[{\"name\":\"vertexId\",\"type\":\"string\"}," +
      "{\"name\":\"value\",\"type\":\"float\"},{\"name\":\"edges\"," +
      "\"type\":{\"type\":\"map\",\"values\":\"string\"}}]}");

  /**
   * Vertex Id
   */
  private Utf8 vertexId;

  /**
   * Gets vertexId
   * @return Utf8 vertexId
   */
  public Utf8 getVertexId() {
    return (Utf8) get(0);
```

So now what?

3. Get your Gora files set up for passing them to Giraph.

   Gora.properties

   Gora-mapping-{datastore}.xml.

```
######################
# Misc properties    #
######################
gora.datastore.default=org.apache.gora.hbase.store.HBaseStore
gora.datastore.autocreateschema=true

########################
# HBaseStore properties #
########################
#gora.datastore.default=org.apache.gora.hbase.store.HBaseStore
gora.hbasestore.scanner.caching=1000
```

```
<gora-orm>
  <table name="graphGiraph">
    <family name="vertices"/>
  </table>
  <class name="org.apache.giraph.io.gora.generated.GVertex" keyClass="java.lang.String" table="graphGiraph">
    <field name="vertexId" family="vertices" qualifier="vertexId"/>
    <field name="value" family="vertices" qualifier="value"/>
    <field name="edges" family="vertices" qualifier="edges"/>
  </class>
</gora-orm>
```

So now what?

4. Get your hooks in place.

GVertexInputFormat

```
/**¶
 * Example implementation of a specific reader for a generated data bean.¶
 */¶
public class GoraGVertexVertexInputFormat¶
  extends GoraVertexInputFormat<LongWritable, DoubleWritable,¶
          FloatWritable> {¶
¶
```

```
  @Override¶
  public GoraVertexReader createVertexReader(¶
      InputSplit split, TaskAttemptContext context) throws IOException {¶
    return new GoraGVertexVertexReader();¶
  }¶
```

```
protected class GoraGVertexVertexReader extends GoraVertexReader {

  /**
   * Transforms a GoraObject into a Vertex object.
   * @param goraObject Object from Gora to be translated.
   * @return Vertex Result from transforming the gora object.
   */
  @Override
  protected Vertex<LongWritable, DoubleWritable, FloatWritable>
  transformVertex(Object goraObject) {
    Vertex<LongWritable, DoubleWritable, FloatWritable> vertex;
    /* create the actual vertex */
    vertex = getConf().createVertex();
    GVertex tmpGVertex = (GVertex) goraObject;

    LongWritable vrtxId = new LongWritable(
        Long.parseLong(tmpGVertex.getVertexId().toString()));
    DoubleWritable vrtxValue = new DoubleWritable(tmpGVertex.getValue());
    vertex.initialize(vrtxId, vrtxValue);
    if (tmpGVertex.getEdges() != null && !tmpGVertex.getEdges().isEmpty()) {
      Set<Utf8> keyIt = tmpGVertex.getEdges().keySet();
      for (Utf8 key : keyIt) {
        String keyVal = key.toString();
        String valVal = tmpGVertex.getEdges().get(key).toString();
        Edge<LongWritable, FloatWritable> edge;
        if (!keyVal.contains("vertexId") && !keyVal.contains("value")) {
          edge = EdgeFactory.create(
              new LongWritable(Long.parseLong(keyVal)),
              new FloatWritable(Float.parseFloat(valVal)));
          vertex.addEdge(edge);
        }
      }
    }
    return vertex;
  }
}
```

So now what?

4. Get your hooks in place.

GVertexOutputFormat

```
public class GoraGVertexVertexOutputFormat
  extends GoraVertexOutputFormat<LongWritable, DoubleWritable,
  FloatWritable> {
```

```
  @Override
  public VertexWriter<LongWritable, DoubleWritable, FloatWritable>
  createVertexWriter(TaskAttemptContext context)
      throws IOException, InterruptedException {
      return new GoraGVertexVertexWriter();
  }
```

```
protected class GoraGVertexVertexWriter extends GoraVertexWriter {

  @Override
  protected Persistent getGoraVertex(
      Vertex<LongWritable, DoubleWritable, FloatWritable> vertex) {
    GVertexResult tmpGVertex = new GVertexResult();
    tmpGVertex.setVertexId(new Utf8(vertex.getId().toString()));
    tmpGVertex.setValue(Float.parseFloat(vertex.getValue().toString()));
    Iterator<Edge<LongWritable, FloatWritable>> it =
        vertex.getEdges().iterator();
    while (it.hasNext()) {
      Edge<LongWritable, FloatWritable> edge = it.next();
      tmpGVertex.putToEdges(
          new Utf8(edge.getTargetVertexId().toString()),
          new Utf8(edge.getValue().toString()));
    }
    return tmpGVertex;
  }

  @Override
  protected Object getGoraKey(
      Vertex<LongWritable, DoubleWritable, FloatWritable> vertex) {
    String goraKey = String.valueOf(vertex.getId());
    return goraKey;
  }
}
```

So now what?

4. Get your hooks in place.

KeyFactory

```
/**
 * Example class for defining a way to construct Gora keys.
 * Uses strings as keys inside Gora.
 */
public class DefaultKeyFactory extends KeyFactory {

  /**
   * Builds a key from a string parameter.
   * @param keyString the key object as a string.
   * @return the key object.
   */
  @Override
  public Object buildKey(String keyString) {
    Object key = null;
    if (getDataStore() == null) {
      throw new RuntimeException(
          "DataStore must be defined before using a key Builder.");
    } else {
      key = getDataStore().newKey();
      // Do specific transformation
      key = keyString;
    }
    return key;
  }
```

So now what?

5. Run Giraph!

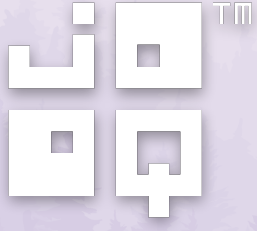
```
    hadoop jar $GIRAPH_EXAMPLES_JAR org.apache.giraph.GiraphRunner
-files ../conf/gora.properties,../conf/gora-hbase-mapping.xml,../conf/hbase-site.xml
-Dio.serializations=o.a.h.io.serializer.WritableSerialization,o.a.h.io.serializer.JavaSerialization
          -Dgiraph.gora.datastore.class=org.apache.gora.hbase.store.HBaseStore
          -Dgiraph.gora.key.class=java.lang.String
          -Dgiraph.gora.persistent.class=org.apache.giraph.io.gora.generated.GEdge
          -Dgiraph.gora.start.key=0 -Dgiraph.gora.end.key=10
          -Dgiraph.gora.keys.factory.class=org.apache.giraph.io.gora.utils.KeyFactory
          -Dgiraph.gora.output.datastore.class=org.apache.gora.hbase.store.HBaseStore
          -Dgiraph.gora.output.key.class=java.lang.String
          -Dgiraph.gora.output.persistent.class=org.apache.giraph.io.gora.generated.GEdgeResult
          -libjars $GIRAPH_GORA_JAR,$GORA_HBASE_JAR,$HBASE_JAR
          org.apache.giraph.examples.SimpleShortestPathsComputation
          -eif org.apache.giraph.io.gora.GoraGEdgeEdgeInputFormat
          -eof org.apache.giraph.io.gora.GoraGEdgeEdgeOutputFormat
           -w 1
```

Future work

More complex schemas

```
{
  "type": "record",
  "name": "WebPage","default":null,
  "namespace": "org.apache.gora.examples.generated",
  "fields" : [
    {"name": "url", "type": ["null","string"], "default":null},
    {"name": "content", "type": ["null","bytes"],"default":null},
    {"name": "parsedContent", "type": {"type":"array", "items": "string"}, "default":null},
    {"name": "outlinks", "type": {"type":"map", "values":"string"}, "default":{}},
    {"name": "headers","type":["null",{"type":"map","values":["null","string"]}],"default":null},
    {"name": "metadata", "default":null, "type": {
      "name": "Metadata",
      "type": "record",
      "namespace": "org.apache.gora.examples.generated",
      "fields" : [
        {"name":"version","type":"int","default":0},
        {"name":"data","type":{"type":"map","values":"string"}, "default":null}
      ]
    }}
  ]
}
```

Adding more data stores

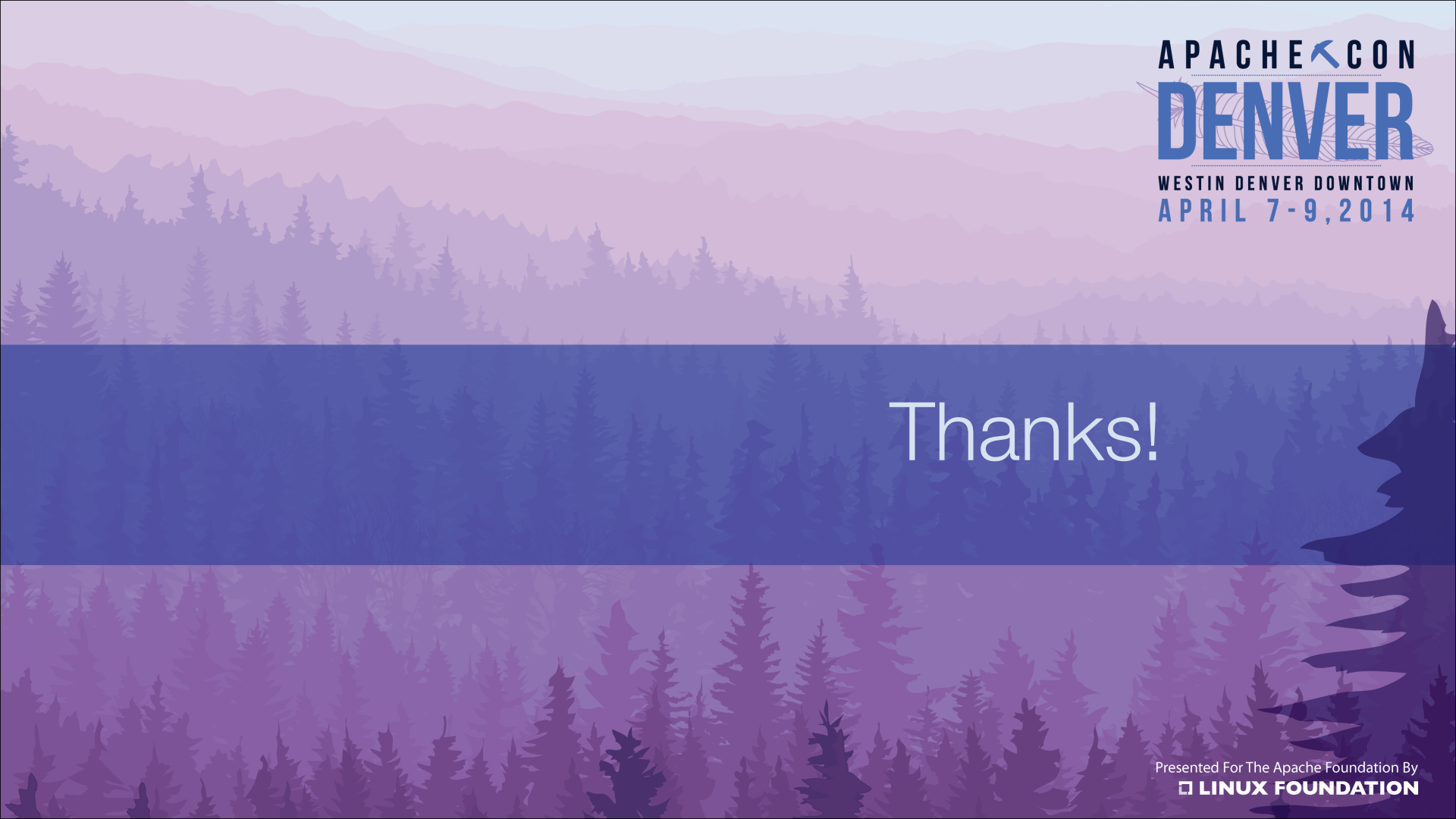


Send us an email on the mailing lists

New serialization formats

- Different serialization formats beside Apache Avro.

- And others that could be interesting for handling different use cases.


APACHECON
DENVER
WESTIN DENVER DOWNTOWN
APRIL 7-9, 2014
Thanks!
Presented For The Apache Foundation By
LINUX FOUNDATION


APACHECON
DENVER
WESTIN DENVER DOWNTOWN
APRIL 7-9, 2014
Q&A
Presented For The Apache Foundation By
LINUX FOUNDATION

References

- http://prezi.com/9ake_klzwrga/apache-giraph-distributed-graph-processing-in-the-cloud/
- http://de.slideshare.net/sscdotopen/large-scale
- http://www.slideshare.net/Hadoop_Summit/processing-edges-on-apache-giraph

Bulk synchronous parallel model

- Computation consists of a series of “supersteps”
  - Supersteps are an atomic unit of computation where operations can happen in parallel
  - During a superstep, components are assigned to tasks and receive unordered messages from previous supersteps.
- Point-to-point messages
  - Sent during a superstep from one component to another and then delivered in the following supersteps.
- Computation completes when all components complete