

From Incubation to Continuous Ingestion The Story of Apache Gora

Renato Marroquin - Lewis John McGibbney

Apache Gora

November 07, 2012

Agenda

- What Apache Gora is
- How Apache Gora works
- Versioning
- GoraCI
- Gora-DynamoDB
- Future work



What Apache Gora is

Apache Gora's Goals:

What Apache Gora is

Apache Gora's Goals:

- Data Persistence : Persisting objects to Column stores, key-value stores, SQL databases and to flat files in local file system of Hadoop HDFS.

What Apache Gora is

Apache Gora's Goals:

- 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.

What Apache Gora is

Apache Gora's Goals:

- 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.

What Apache Gora is

Apache Gora's Goals:

- 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

What Apache Gora is

Apache Gora's Goals:

- 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 Apache Gora is (cont'd)

- Open source framework which provides an in-memory data model and persistence for big data.

What Apache Gora is (cont'd)

- Open source framework which provides an in-memory data model and persistence for big data.
- Gora supports:

What Apache Gora is (cont'd)

- Open source framework which provides an in-memory data model and persistence for big data.
- Gora supports:
 - Column stores:



Cassandra

APACHE
HBASE

What Apache Gora is (cont'd)

- Open source framework which provides an in-memory data model and persistence for big data.
- Gora supports:
 - Column stores:



Cassandra

APACHE
HBASE

- Key Value stores:



What Apache Gora is (cont'd)

- Open source framework which provides an in-memory data model and persistence for big data.
- Gora supports:
 - Column stores:



Cassandra

APACHE
HBASE

- Key Value stores:



- RDBMSs:



HyperSQL

What Apache Gora is (cont'd)

- Open source framework which provides an in-memory data model and persistence for big data.
- Gora supports:
 - Column stores:



Cassandra

APACHE
HBASE

- Key Value stores:



- RDBMSs:



Hyper**SQL**

- Flat files in local file system of Hadoop HDFS

- The Core Gora API
 - Store (`org.apache.gora.store`)
 - Persistency (`org.apache.gora.persistency`)
 - Query (`org.apache.gora.query`)
 - MapReduce (`org.apache.gora.mapreduce`)

How Gora works

- Store API
 - org.apache.gora.store.*

- Store API
 - `org.apache.gora.store.*`
 - Core class is the `DataStore.java` (or `FileBacked...`). `DataStore` handles actual object persistence. Objects can be persisted, fetched, queried or deleted by the `DataStore` methods. `DataStores` can be constructed by an instance of `{@link DataStoreFactory}`

- Store API
 - `org.apache.gora.store.*`
 - Core class is the `DataStore.java` (or `FileBacked...`). `DataStore` handles actual object persistence. Objects can be persisted, fetched, queried or deleted by the `DataStore` methods. `DataStores` can be constructed by an instance of `{@link DataStoreFactory}`
 - Base class for client interaction is `o.a.g.s.impl.DataStoreBase` (or `FileBacked...`) which is a base class for indirectly interacting with the `DataStore`. All `DataStore` implementations extend this class.

How Gora works

- Persistency API
 - `org.apache.gora.persistency.*`

- Persistence API
 - `org.apache.gora.persistence.*`
 - Core classes are `BeanFactory`, `Persistent` and `State`. The former enables the construction of keys and persistent objects. All objects persisted by Gora implement `Persistent` and the latter defines the actual state of an object or field. State is managed through the `StateManager`. Objects can be `NEW`, `CLEAN (UNMODIFIED)`, `DIRTY (MODIFIED)` or `DELETED`

- Persistence API
 - `org.apache.gora.persistence.*`
 - Core classes are `BeanFactory`, `Persistent` and `State`. The former enables the construction of keys and persistent objects. All objects persisted by Gora implement for second and the latter defines the actual state of an object or field. State is managed through the `StateManager`. Objects can be `NEW`, `CLEAN (UNMODIFIED)`, `DIRTY (MODIFIED)` or `DELETED`
 - As with the Store API, persistency has base classes for client interaction e.g. `o.a.g.p.impl.BeanFactoryImpl`, `o.a.g.impl.PersistentBase` and `o.a.g.impl.StateManagerImpl` respectively.

How Gora works

- Query API
 - `org.apache.gora.query.*`

How Gora works

- Query API
 - `org.apache.gora.query.*`
 - Core classes are Query, PartitionQuery and Result.

- Query API
 - `org.apache.gora.query.*`
 - Core classes are Query, PartitionQuery and Result.
 - PartitionQuery divides the results of the Query to multi partitions, so that queries can be run locally on the nodes that hold the data. PartitionQuery's are used for generating Hadoop InputSplits.

- Query API
 - `org.apache.gora.query.*`
 - Core classes are `Query`, `PartitionQuery` and `Result`.
 - `PartitionQuery` divides the results of the `Query` to multiple partitions, so that queries can be run locally on the nodes that hold the data. `PartitionQuery`'s are used for generating Hadoop `InputSplits`.
 - Queries are constructed by the `DataStore` implementation via `{@link DataStore#newQuery() }`

- Query API
 - `org.apache.gora.query.*`
 - Core classes are `Query`, `PartitionQuery` and `Result`.
 - `PartitionQuery` divides the results of the `Query` to multiple partitions, so that queries can be run locally on the nodes that hold the data. `PartitionQuery`'s are used for generating Hadoop `InputSplits`.
 - Queries are constructed by the `DataStore` implementation via `{@link DataStore#newQuery() }`
 - In a common theme clients access core classes through the the classes in `o.a.g.q.impl`

How Gora works

- MapReduce API
 - `org.apache.gora.mapreduce.*`

- MapReduce API
 - `org.apache.gora.mapreduce.*`
 - This package holds ALL of the Gora MR functionality and Utilities required to utilize Gora within such an environment

- MapReduce API
 - `org.apache.gora.mapreduce.*`
 - This package holds ALL of the Gora MR functionality and Utilities required to utilize Gora within such an environment
 - Including GoraMapper, GoraReducer, ALL Record Counter, Reader and Writer implementations.

- MapReduce API
 - `org.apache.gora.mapreduce.*`
 - This package holds ALL of the Gora MR functionality and Utilities required to utilize Gora within such an environment
 - Including GoraMapper, GoraReducer, ALL Record Counter, Reader and Writer implementations.
 - Persistent and String Se/Deserialization is done in the Hadoop serializer using `o.a.g.avro.@link PersistentDatumWriter` (for writing Avro's dirty and readable information)with Avro's `@link BinaryEncoder`.

- The LogManager Tutorial

- The LogManager Tutorial
 - Hadoop has Word Count, Gora has LogManager!

- The LogManager Tutorial
 - Hadoop has Word Count, Gora has LogManager!
 - Tutorial Aim.
 - Implements a system to store web server logs (10K lines) in Apache Hbase, analyze the results using Apache Hadoop and store the results either in HSQLDB or MySQL.

- The LogManager Tutorial
 - Hadoop has Word Count, Gora has LogManager!
 - Tutorial Aim.
 - Implements a system to store web server logs (10K lines) in Apache Hbase, analyze the results using Apache Hadoop and store the results either in HSQLDB or MySQL.
 - Tutorial Objectives.
 - Set the (Gora) Environment
 - Model the Data
 - The Core Gora API
 - MapReduce Support

- Apache Hadoop - 1.0.1
- Apache Cassandra - 1.1.2
- Apache HBase - 0.90.4
- Apache Accumulo - 1.4.0
- Apache Avro - 1.3.3
- Amazon DynamoDB - Amazon SDK 1.3
- MySQL - 5.1.18
- HDBSQL - 2.2.8

- Based on Cloudbase tests
- Scale test
 - Ingest synthetic data for extended periods.
 - Run continuous queries.
- Verify
 - Keeps running.
 - Ingest and queries fast.
 - No data lost.

- Generate linked list of random nodes.

- Generate linked list of random nodes.
- Query processes.
 - Follow linked list.
 - Record timing info.
 - Note missing nodes.

- Generate linked list of random nodes.
- Query processes.
 - Follow linked list.
 - Record timing info.
 - Note missing nodes.
- Write 1 random node.

- Generate linked list of random nodes.
- Query processes.
 - Follow linked list.
 - Record timing info.
 - Note missing nodes.
- Write 1 random node.
- Write 1 random node that references previous node.

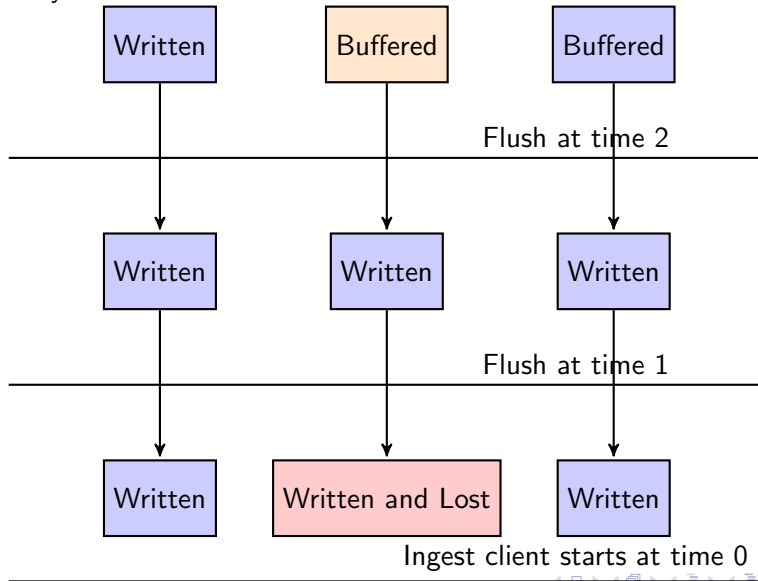
- Client buffer problem.

- Client buffer problem.
 - When client dies some nodes written, others not.
 - Can not easily distinguish data loss from client death.

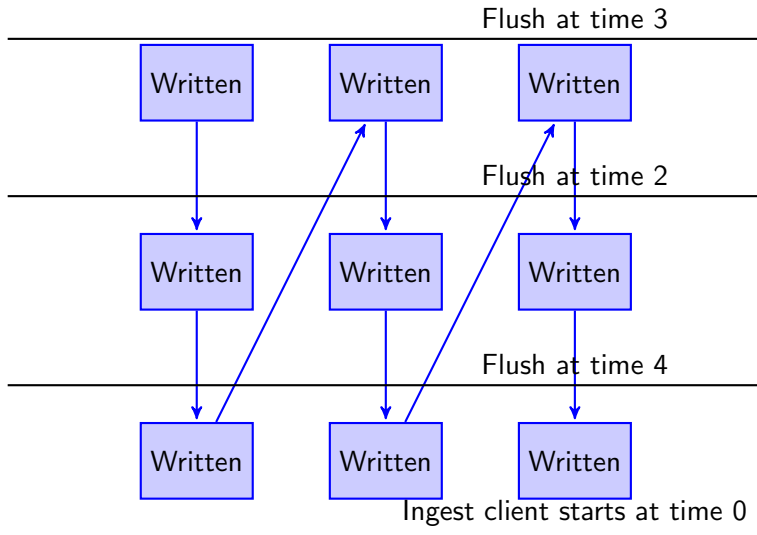
- Client buffer problem.
 - When client dies some nodes written, others not.
 - Can not easily distinguish data loss from client death.



Only reference flushed nodes.



Giant Linked List.



- Verification MapReduce.

- Verification MapReduce.
- Map.
 - `<row>:-1`
 - `<row referenced>:<row>`

- Verification MapReduce.
- Map.
 - `<row>:-1`
 - `<row referenced>:<row>`
- Reduce.
 - Brings references to and definition of node together.
 - Emits references to undefined nodes.

- Related JIRA issues:
 - HADOOP-6945
 - HBASE-5754
 - GORA-XXX

Gora-DynamoDB

- Incubated during last ApacheCon in Vancouver

- Incubated during last ApacheCon in Vancouver
- Product of the GSoC 2012!!!



- Incubated during last ApacheCon in Vancouver
- Product of the GSoC 2012!!!

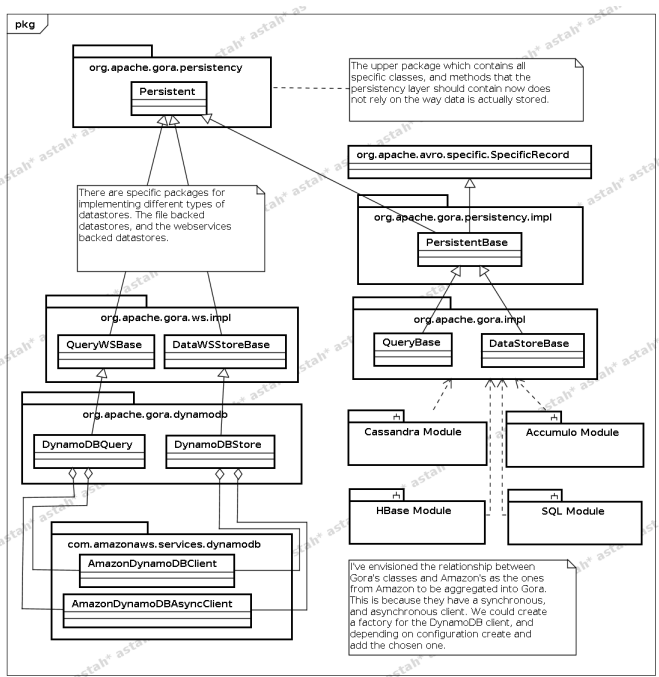


- Motivation
 - Engage with the open source community by using some popular services such as Amazon WebServices.
 - Help n00bs to get familiar more easily with BigData technologies, specially for those who don't have access to big compute clusters ... like me.
 - Make the Gora Project more robust.

Gora-DynamoDB

- Changes within Gora

- Changes within Gora
 - Create new abstraction for Gora internals: One for file backed data stores, and one for web service backed data stores.



Gora-DynamoDB

- Changes within Gora

- Changes within Gora
 - New properties added:
 - Amazon endpoint (Any of the available e.g. us-east-1.amazonaws).
 - Consistent reads (True or False).
 - Client type (Sync or Async).

- Changes within Gora
 - New properties added:
 - Amazon endpoint (Any of the available e.g. us-east-1.amazonaws).
 - Consistent reads (True or False).
 - Client type (Sync or Async).
 - Gora-DynamoDB compiler uses Amazon SDK annotations.
 - @DynamoDBTable
 - @DynamoDBHashKey
 - @DynamoDBRangeKey
 - @DynamoDBAttribute

- Changes within Gora
 - New properties added:
 - Amazon endpoint (Any of the available e.g. us-east-1.amazonaws).
 - Consistent reads (True or False).
 - Client type (Sync or Async).
 - Gora-DynamoDB compiler uses Amazon SDK annotations.
 - @DynamoDBTable
 - @DynamoDBHashKey
 - @DynamoDBRangeKey
 - @DynamoDBAttribute
 - Gora-DynamoDB mapping file.
 - Schema definition.
 - Composite key definition: Hash key and/or range key.
 - Provisioned throughput.
 - Columns specifying their data types.

Future work

- On our web service backed data stores:
 - Add new web based data stores: GAE, Microsoft Data Services.
 - Add batch write functionality to our Gora-DynamoDB module.
 - Add new data types for the Gora-DynamoDB module.
 - Add cost based control for our web based data stores.
 - Fully integrate them with Apache Nutch.

- On our web service backed data stores:
 - Add new web based data stores: GAE, Microsoft Data Services.
 - Add batch write functionality to our Gora-DynamoDB module.
 - Add new data types for the Gora-DynamoDB module.
 - Add cost based control for our web based data stores.
 - Fully integrate them with Apache Nutch.
- On our file backed data stores:
 - Add new file based data stores: HIVE, Riak.
 - Update our AVRO backend which will us new features.

- On our web service backed data stores:
 - Add new web based data stores: GAE, Microsoft Data Services.
 - Add batch write functionality to our Gora-DynamoDB module.
 - Add new data types for the Gora-DynamoDB module.
 - Add cost based control for our web based data stores.
 - Fully integrate them with Apache Nutch.
- On our file backed data stores:
 - Add new file based data stores: HIVE, Riak.
 - Update our AVRO backend which will us new features.
- Add optimistic concurrency control for our data stores.

Questions



Thanks

Thanks!