INTEGRATION OF APACHE MRQL QUERY LANGUAGE WITH APACHE STORM REALTIME COMPUTATIONAL SYSTEM.

By

ACHYUT PAUDEL

THESIS

Submitted in partial fulfillment of the requirements

For the degree of Masters of Computer Engineering at

The University of Texas at Arlington

December, 2016

Arlington, Texas

ABSTRACT

INTEGRATION OF APACHE MRQL QUERY LANGUAGE

WITH APACHE STORM REALTIME COMPUTATIONAL SYSTEM.

ACHYUT PAUDEL, MS

The University of Texas at Arlington, 2016

Supervising Professor: Leonidas Fegaras

Committee member: Dr. Sharma Chakravarty

Committee member: Mr. David Levine

The use of real time processing of data has increased in recent years with the increase of data captured by social media platforms, IOT and other big data applications. The processing of data in real time has been an important aspect of the day from finding the trends over the internet to fraud detection of in the banking transactions. Finding relevant information from large amount of data has always been a difficult problem to solve. MRQL is a query language that can be used on top of different big data platform such as Apache Hadoop, Flink, Hama, and Spark that enables the professionals with Database query knowledge to write queries to run programs on top of these computational systems.

In this work, we have tried to integrate the MRQL query language with a new real time big data computational system called Apache storm. This system was developed by twitter to analyze the trending topics in the social media and is widely used in industry today. The query written in MRQL is converted into a physical plan that involves execution of different functions such as Map Reduce, Aggregation etc. which has to be executed by the platform in its own execution plan. The implementation of Map Reduce has been done in this work for Storm which covers execution for important physical plans of query such as Select and Group By. The implementation of Map Reduce is also an important a part in every big data processing platform. This project will be the starting point in implementation of the MRQL for Apache Storm and the implementation can be extended to support various query plans involving Map Reduce.

ACKNOWLEDGEMENTS

This project would not have been possible without the help and supervision of Prof. Leonidas Fegaras from department of Computer Science and Engineering UTA. Many thanks to Prof Fegaras for helping me in the all possible ways throughout the process. Also thanks to my committee members Prof. Mr David Levine, Prof. Dr. Sharma Chakravarty, and graduate advisor Prof. Braham Khalili. I like to thank Ms. Camille Constiblle for guiding through the process of writing and the submission. I also like to thank Upa Gupta for helping me to better understand the MRQL and its processing.

I would like to thank many friends who have encouraged me and motivated me to work hard and complete the project. Notably, I like to thank Achyut Neupane, Ishwor Timilsina, Gaurav Thapa, Nigesh Shakya and Ashay Rajhimwale for supporting me in the whole process.

I like to thank the open source contributors for Apache Storm, Apache MRQL and Apache storm project and for their clear documentation and examples.

October 16, 2016

TABLE OF CONTENTS

LIST OF FIGURES (or Illustrations)

LIST OF ABBREVIATIONS

MRQL: Map Reduce Query Language

API: Application Programming Interface

HDFS: Hadoop Distributed File System

NFS: Network File System

IOT: Internet of Things

GPS: Global Positioning Satellites

CSE: Computer Science and Engineering

UTA: University of Texas at Arlington

Prof: Professor

Chapter 1

Big data:
Big data is a term for data sets that are so large or complex that traditional data processing applications are inadequate to deal with them. Challenges include analysis, capture, data curation, search, sharing, storage, transfer, visualization, querying, updating and information privacy. [1] Big data can be described by the following characteristics: Volume, Variety, Velocity, Variability, and Veracity.

In recent year there has been a significant progress in the generation of data and the storage of the data due to the boom of internet connected devices, increases in the number of mobile devices, increase in the number of social media users. The reduction of cost of the digital devices, cheap sensors, logs from software, internet connected cameras, wider use of wireless networks are few factors in collecting petabytes and Exabyte of data daily. The large data being generated every day from various platforms are stored and are distributed in various platforms and datacenters around the world.

The analysis of this large amount of data gives and new insight into the business and market to grow and invest. There are numerous use cases where analysis of these data comes handy. In recent years the big data analysis is being used to find the correlations to spot business trends, prevent diseases, combat crimes, finding trending topics in internet and various other fields.[2] Some of these application involves in the storage of data in bulk and processing them later but various of them involves in the processing of the data in the real time and get the insights in the real time so the appropriate decisions can be made.

Data Generation and Storage:
The large amount of data has been generated by various systems. This data comes from everywhere: sensors used to gather climate information, posts to social media sites, digital pictures and videos, purchase transaction records, and cell phone GPS signals to name a few.[3]

The data coming from various sources are stored in servers which is distributed in various data centers. The collected data is being stored in the distributed file systems such as HDFS or NFS. The data processing engines now reads the data from these files systems which are abstracted in such a way that it is transparent to the client accessing the data.

Data Processing and Map Reduce
In 2000, Seisint Inc. developed a C++-based distributed file-sharing framework for data storage and query. The system stores and distributes structured, semi-structured, and unstructured data across multiple servers. Users can build queries in a C++ dialect called ECL for processing and querying the data.[4]

In 2004, Google published a paper on a process called Map Reduce. In this methodology the task is divided into two different stages namely Map and Reduce. With Map reduce, queries are split and distributed across parallel nodes and processed in parallel (the Map step). The results are then gathered and delivered (the Reduce step).[5]

This method is quiet popular and lead to development of very popular big data processing framework called Apache Hadoop. Hadoop included two important aspects of big data. The storage and the processing. HDFS was developed for storing the data and Map Reduce was implemented to process the data by distributing the jobs in thousands of servers across the datacenters.

Data processing Methodologies

Batch Processing
In this methodology, the data is being collected and stored in the batch form which are later processed using the batch processing platform like apache Hadoop for Flink. The dataset is huge and is distributed along large numbers of servers which may be located in one datacenter or multiple data centers in

various locations. It is very efficient way to process the data where the transactions are collected over the time and the result is not time intensive. A typical batch process involves the following steps:

Data Generation -> Data collection over long time -> Data Storage -> Data processing -> batch Results

Real time stream processing:
In contrast to batch processing, there are multiple use cases where the collected data has to be processed in real time. The real time data processing involves a continual input, process and output of data. Data must be processed in a small time period (or near real time) since the result is time incentive. Typical use cases for the real time processing would be the fraud detection in Credit Card transactions, Spam detection in emails, Customer services etc. The real time processing is done by real time frameworks such as apache storm and apache spark.

Importance of Real time stream data processing
Banks, Air traffic, Customer services etc generate real time data streams. Real time data processing and analytics allows an organization the ability to take immediate action for those times when acting within seconds or minutes is significant.

The events and data from multiple sources can be combined to detect patterns and attempt to identify either opportunities or threats. The goal is to identify significant events and respond fast. Sales leads, orders or customer service calls are examples where real time data analysis are helpful. The organizations can benefit from running query against live feeds by obtaining near real time insights of their sales and operations using continuous analytics which allows the organization to take immediate action

Currently the real time data is processed using the real time frameworks such as apache storm, Apache Spark, Apache Flink and others. [6]

Why big data processing is challenging
There are various challenges in processing big data. The challenges vary from efficient processing of data and making sense out of it to efficient storage and abstraction of data. Deploying big data applications on cloud environment is not a trivial or straightforward task. We need to exploit the cloud computing methods to process more areas of big data[7]. The data can be complex and multi varied. The heterogeneity of the data further adds complexity to efficiently store and make sense out of it. The data is coming from different platforms such as sensors to users so the dynamic indexing, analyzing and querying large volumes of high-dimensional spatial big data are major challenges.

Map Reduce for big data processing:
A Map Reduce program is composed of a Map() procedure that applies a function to the data and yield a key and value pair which is distributed along the network. The Reduce method performs a summary operation. i.e. it applies a function over the set of values that corresponds to a key which has been received from the map phase.

The way of writing Map reduce program highly differs from platform to platform. The map reduce program is written differently in Apache Hadoop, Apache Spark, Flink and others. This decreases the efficiency of the programmer since he has to learn all the syntax and library for each platform every time he uses it.

General Architecture of data flow into an enterprise machines

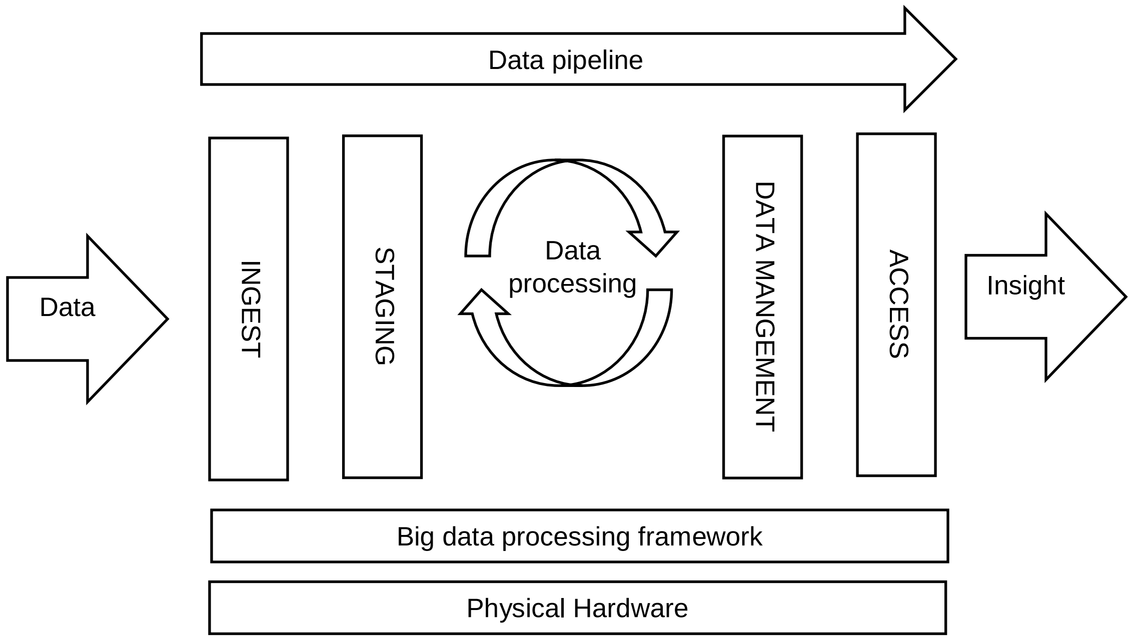


Figure 1 General Architecture of Big data processing

The following figure illustrates the General architecture for collecting and processing big data.

Data Ingest:

After the cluster is setup, the data is fed into the system. In practice there are two main approaches: batch and event-driven. The former is appropriate for file and structured data, while the latter is appropriate for most near-real-time events such as log or transactional data. Apache flume, Sqoop can be used for this process.

Staging:

Once the data has arrived in as a whole, there remains the task of staging it for processing. This is not just about storing it somewhere, but rather storing it in the right format, with the right size, and the right access mask. For batch, container file formats, including the venerable Sequence File and Avro formats, are both useful and popular.[8]

Data Processing:

This step is where the actual processing of the data is done. The processing ranges from analytics to filtering and querying into the data. Various processes and platforms such as Map Reduce, Apache Spark are used in this step.

Chapter 2

Introduction to MRQL:

MRQL is a query processing and optimization system for large-scale, distributed data analysis, built on top of Apache Hadoop, Hama, Spark, and Flink.

MRQL (the MapReduce Query Language) is an SQL-like query language for large-scale data analysis on a cluster of computers. The MRQL query processing system can evaluate MRQL queries in four modes:

- Map-Reduce mode using Apache Hadoop,
- BSP mode (Bulk Synchronous Parallel mode) using Apache Hama,
- Spark mode using Apache Spark, and
- Flink mode using Apache Flink.

In this project, we try to integrate Storm with MRQL so that the queries can be evaluated in storm mode and the data coming from real time data streams can be queried.[9]

Why is MRQL Used:

With MRQL, users are able to express complex data analysis tasks, such as PageRank, k-means clustering, matrix factorization, etc. using SQL-like queries exclusively, while the MRQL query processing system is able to compile these queries to efficient Java code. [9]

Benefits of using MRQL

The MRQL query language is powerful enough to express most common data analysis tasks over many forms of raw in-situ data, such as XML and JSON documents, binary files, and CSV documents. MRQL is more powerful than other current high-level MapReduce languages, such as Hive and PigLatin, since it can operate on more complex data and supports more powerful query constructs, thus eliminating the need for using explicit MapReduce code.[9]

Motivation for integrating MRQL with streaming framework

MRQL is a very powerful query language and storm is a highly used stream processing framework currently exist in the market. Many companies use Storm or streaming engines to do real time data computations. By integrating MRQL with streaming engines, we can make the developers more productive and can write complex queries such as k-means, clustering, matrix multiplication etc in very less time and efficiently.

Architecture of MRQL

MRQL is divided into the core module and modules that are integrated on the top of core modules. The core module consists of all the data structures and generates the evaluation plans that are necessary for the evaluation of query. The logical plan and physical plans are being created and those plans run on top of mode that is selected to run into.

Query execution steps
The query goes through following steps before getting executed. The general architecture of MRQL can also be described by following figures

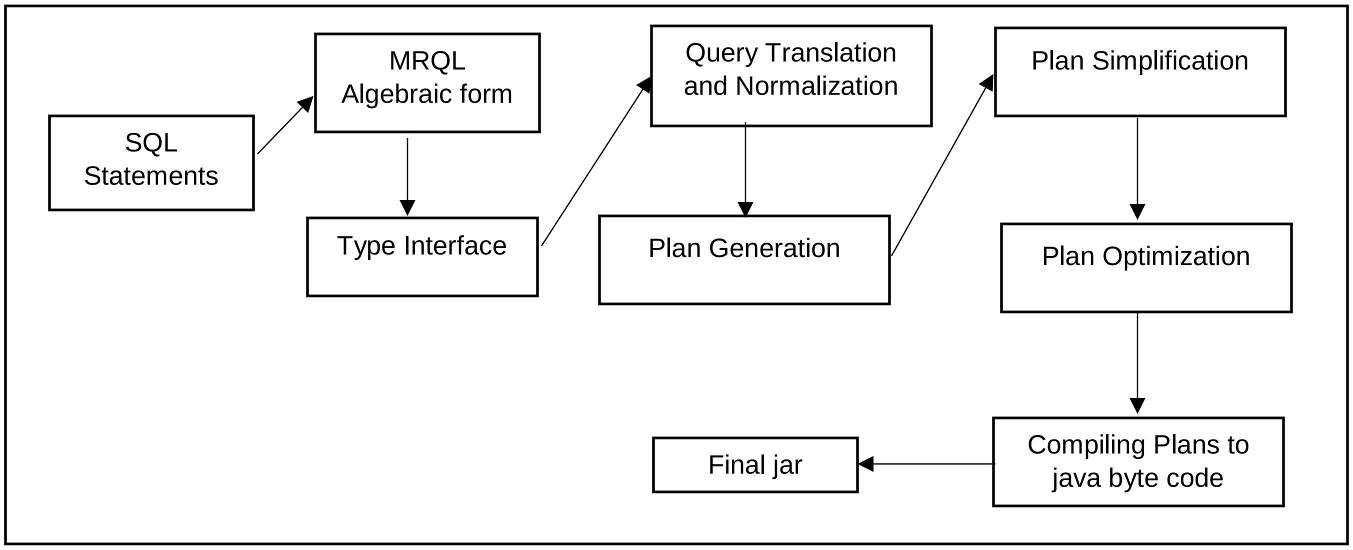


Figure 2 General architecture of MRQL

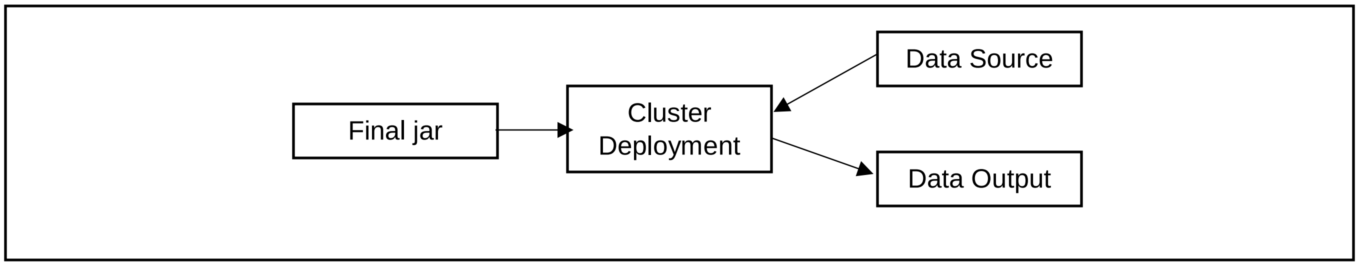
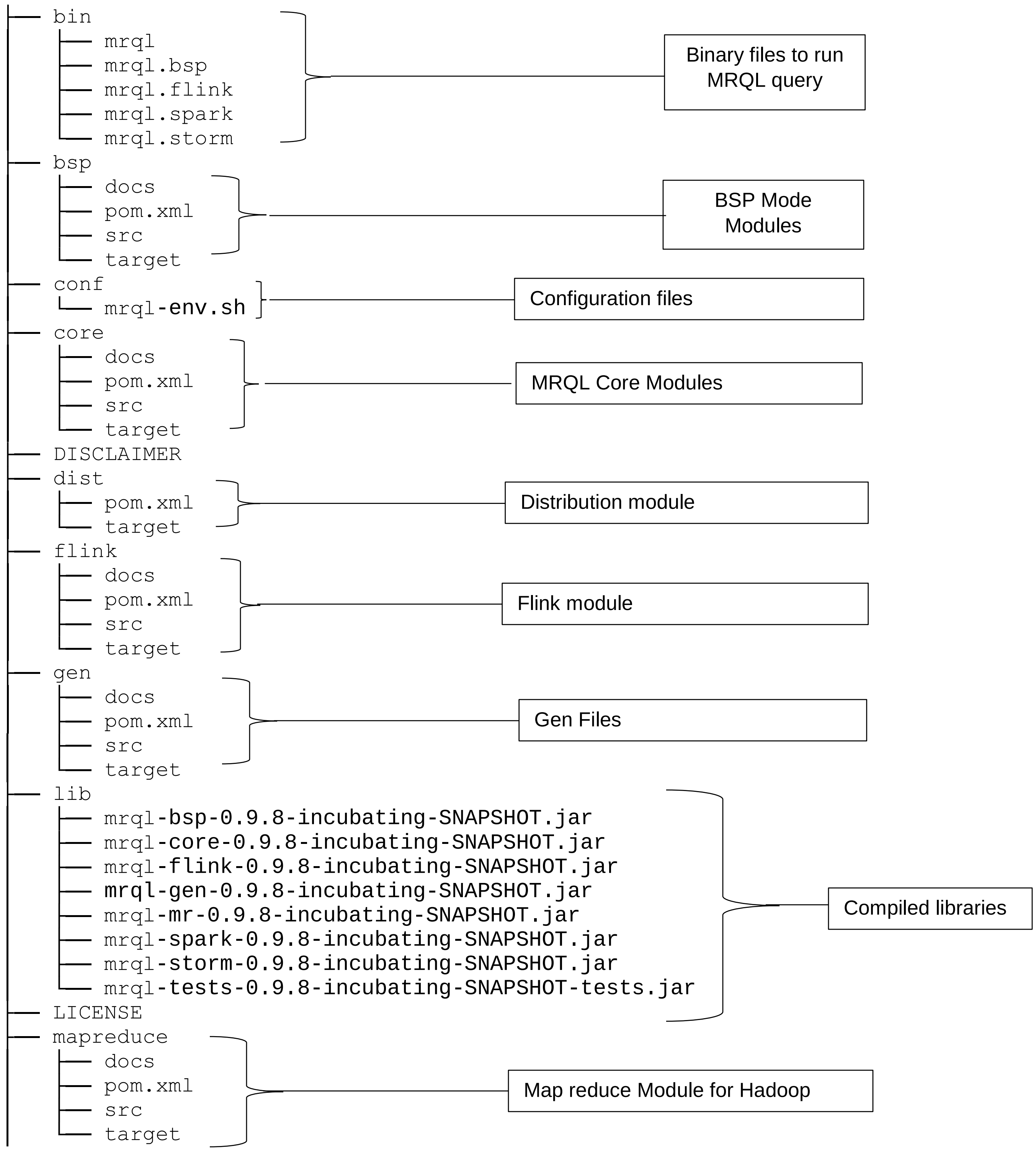


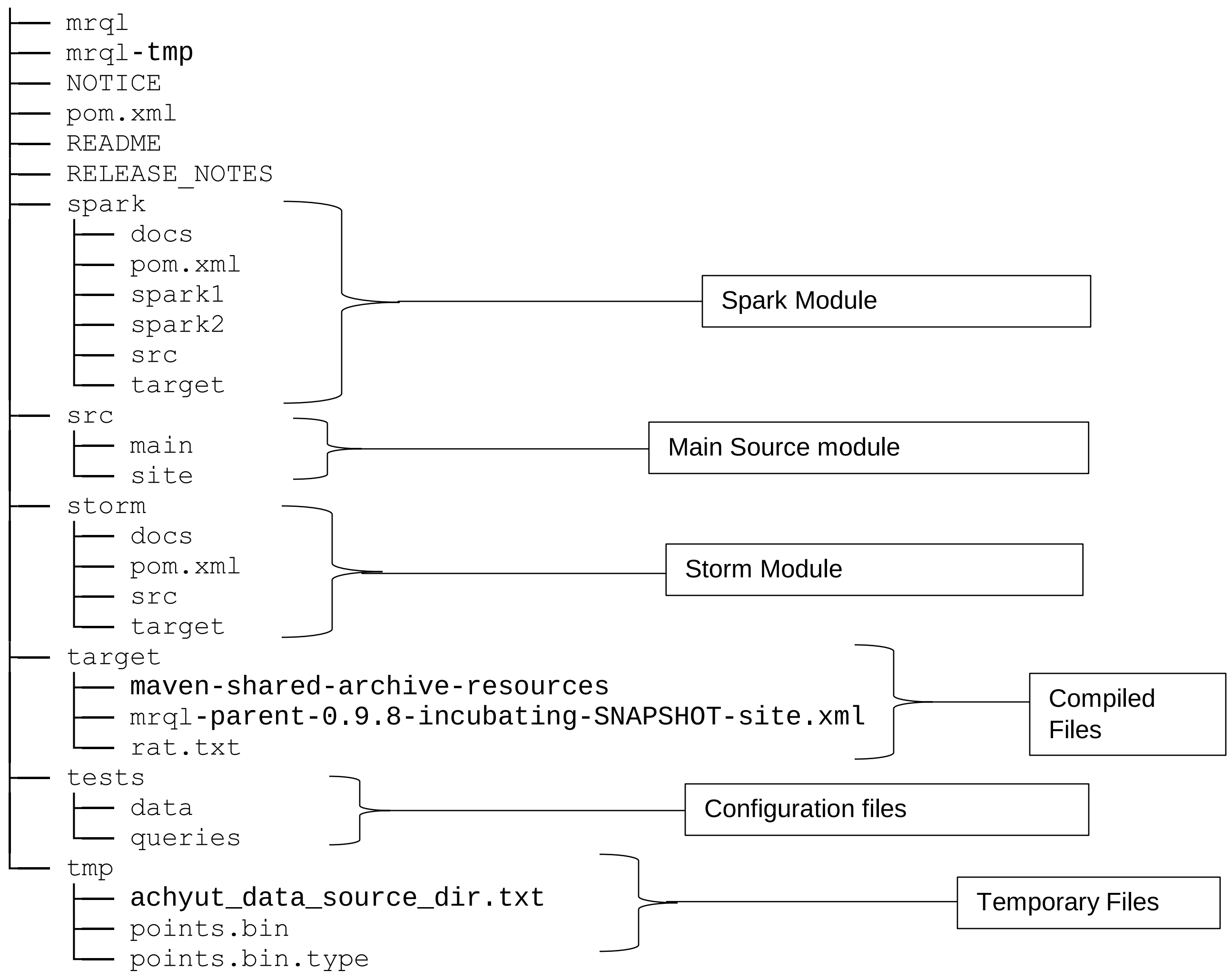
Figure 3 Deployment of MRQL Jar into Cluster

Project structure of MRQL
The MRQL project is divided into its core module and other sub modules. The core module consists of all the data formats and data structures that are used in the project. The supporting modules can extend the core module and can reuse the components.

The project structure is as follows:



```
├── bin
│   ├── mrql
│   ├── mrql.bsp
│   ├── mrql.flink
│   ├── mrql.spark
│   └── mrql.storm
├── bsp
│   ├── docs
│   ├── pom.xml
│   ├── src
│   └── target
├── conf
│   └── mrql-env.sh
├── core
│   ├── docs
│   ├── pom.xml
│   ├── src
│   └── target
├── DISCLAIMER
├── dist
│   ├── pom.xml
│   └── target
├── flink
│   ├── docs
│   ├── pom.xml
│   ├── src
│   └── target
├── gen
│   ├── docs
│   ├── pom.xml
│   ├── src
│   └── target
├── lib
│   ├── mrql-bsp-0.9.8-incubating-SNAPSHOT.jar
│   ├── mrql-core-0.9.8-incubating-SNAPSHOT.jar
│   ├── mrql-flink-0.9.8-incubating-SNAPSHOT.jar
│   ├── mrql-gen-0.9.8-incubating-SNAPSHOT.jar
│   ├── mrql-mr-0.9.8-incubating-SNAPSHOT.jar
│   ├── mrql-spark-0.9.8-incubating-SNAPSHOT.jar
│   ├── mrql-storm-0.9.8-incubating-SNAPSHOT.jar
│   └── mrql-tests-0.9.8-incubating-SNAPSHOT-tests.jar
├── LICENSE
├── mapreduce
│   ├── docs
│   ├── pom.xml
│   ├── src
│   └── target
```



```
├── mrql
├── mrql-tmp
├── NOTICE
├── pom.xml
├── README
├── RELEASE_NOTES
├── spark
│   ├── docs
│   ├── pom.xml
│   ├── spark1
│   ├── spark2
│   ├── src
│   └── target
├── src
│   ├── main
│   └── site
├── storm
│   ├── docs
│   ├── pom.xml
│   ├── src
│   └── target
├── target
│   ├── maven-shared-archive-resources
│   ├── mrql-parent-0.9.8-incubating-SNAPSHOT-site.xml
│   └── rat.txt
├── tests
│   ├── data
│   └── queries
└── tmp
    ├── achyut_data_source_dir.txt
    ├── points.bin
    └── points.bin.type
```

Figure 4 Project Structure of MRQL

Data Models in MRQL

MRQL supports the following types:

a basic type: bool, short, int, long, float, double, string.
a tuple (t1,..., tn),
a record < A1: t1, ..., An: tn >,
a list (sequence) [t] or list(t),
a bag (multiset) {t} or bag(t),
a user-defined type
a data type T
a persistent collection !list(t), ![t], !bag(t), or !{t}[10]

Building a MRQL project:

MQRL can be build as a whole or each modules can be built individually and can be deployed.

The following procedures discuss in details about the build process and deployment process.

Downloading latest version of MRQL from git:

```
git clone https://git-wip-us.apache.org/repos/asf/incubator-mrql.git
```

To build MRQL using maven, use

```
mvn clean install
```

To validate the installation, use

```
mvn -DskipTests=false clean install
```

These commands will build the project and installs the libraries into the local maven repository.

Writing Query for MRQL
The query could be written as like in SQL. Following describes the methodology for writing different MRQL queries

Select query:
select [**distinct**] e
from p_1 **in** e_1, ..., p_n **in** e_n
[**where** e_c]
[**group by** p': e' [**having** e_h]]
[**order by** e_0 [**limit** e_l]]

where the syntax enclosed in [ ] is optional and ... means a sequence of similar bindings. Expressions e, e_1,...,e_n, e_c, e', e_h, e_0, and e_l are arbitrary MRQL expressions, which may contain other nested select queries.

Group By query:
The group-by syntax of an MRQL query takes the form group by p': e'. It partitions the query results into groups so that the members of each group have the same e' value.

Order By query:
The order by e0 syntax orders the result of a query (after the optional group-by) by the e0 values. It is assumed that there is a default total order ≤ defined for all data types (including tuples and bags). The special parametric type Inv(T), which has a single data constructor inv(v) for a value v of type T, inverts the total order of T from ≤ to ≥.

Example of MRQL Query
select (sum(x),i)

from (x,i,j) in X

group by i

Where X is a datasource containing attributes x,I and j.[11]

Chapter 3

Apache Storm

Apache Storm is a free and open source distributed real time computation system. Storm makes it easy to reliably process unbounded streams of data, doing for real time processing what Hadoop did for batch processing. Storm is simple, can be used with any programming language, and can process over a million tuples per second per node. It is scalable, fault-tolerant, guarantees your data will be processed, and is easy to set up and operate.[12]

Why choose storm?

Storm is being used widely in industry. The following core attributes make storm stand out from rest of the stream processing systems.

- **Simple API:** The API is very simple to use and program. It uses a concept called tuples which is a list of values where the values can have any types of serializable objects. There are three abstractions namely Spout, bolts and topologies that composes application logics for whole distributed system.
- **Scalable:** Storm can be scaled to hundreds and thousands of machines. The topologies are inherently parallel and run across multiple machines. The command line tool distributes the jar package to different computers in the cluster and the parallism can be adjusted programmatically.
- **Fault Tolerant:** Storm has master nodes called Nimbus node and worker nodes called supervisor nodes. If any of these nodes dies in the process, they will restart like nothing happened. The nimbus node will restart another node and continue processing the data.
- **Guarantees Data processing:** Storm guarantees the data processing by the help of its ack and fail functions. If a tuple is processed successfully it is acknowledged and if it not then it is send as a failed node so that the master keeps tract of the tuples and guarantees the processing of tuples.
- **Easy integration with multiple languages**: Storm can be used with different languages since it is designed from ground up to be usable with multiple languages using thrift. The core of storm contains Thrift definition of topology and the methods for executing and submitting them in the cluster.
- **Easy to deploy and operate**: The topologies can be easily submitted into the cluster using simple command line tool. Minimal configuration is required for the cluster deployment since the default configurations made it easy. It is highly robust running in the machines for months.
- Free and open source: Storm is an Apache project and it is open source distributed under apache license version 2.0. Due to this large number of external development, activity is going on and a large community is created which is helpful for the beginners and adults as a whole.

Architecture of Storm
The storm has a concept of topology, which is a directed graph consisting of spouts and bolts. The Spouts are the data source and the bolts are the executors. The following figure illustrates the topology and its structure.

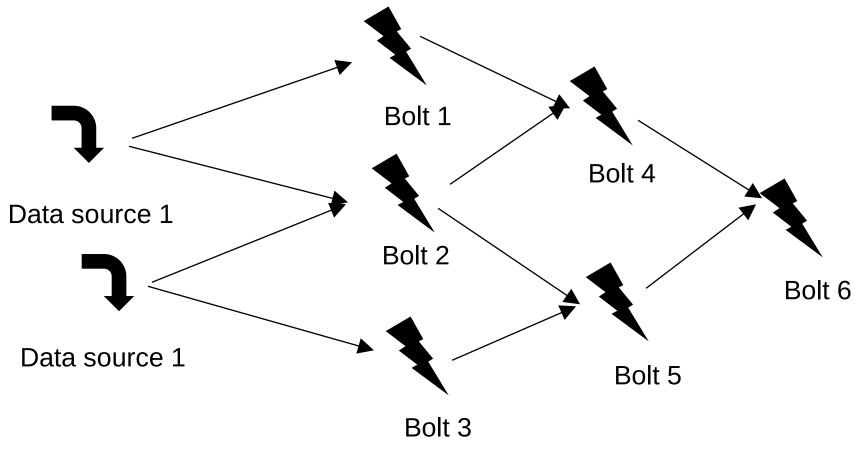


Figure 5 Storm Topology Architecture

Concepts:
Topologies: The logic for a real time application is packaged into a Storm topology. A Storm topology is analogous to a MapReduce job. One key difference is that a MapReduce job eventually finishes, whereas a topology runs forever (or until you kill it, of course). A topology is a graph of spouts and bolts that are connected with stream groupings.[13]

Streams: The stream is the core abstraction in Storm. A stream is an unbounded sequence of tuples that is processed and created in parallel in a distributed fashion. Streams are defined with a schema that names the fields in the stream's tuples. By default, tuples can contain integers, longs, shorts, bytes, strings, doubles, floats, booleans, and byte arrays. [13]

Spouts: A spout is a source of streams in a topology. Generally spouts will read tuples from an external source and emit them into the topology (e.g. a Kestrel queue or the Twitter API). Spouts can either be reliable or unreliable. A reliable spout is capable of replaying a tuple if it failed to be processed by Storm, whereas an unreliable spout forgets about the tuple as soon as it is emitted. [13]
Bolts: All processing in topologies is done in bolts. Bolts can do anything from filtering, functions, aggregations, joins, talking to databases, and more. Bolts can emit more than one stream. [13]

Bolts can do simple stream transformations. Doing complex stream transformations often requires multiple steps and thus multiple bolts. For example, transforming a stream of tweets into a stream of trending images requires at least two steps: a bolt to do a rolling count of retweets for each image, and one or more bolts to stream out the top X images (you can do this particular stream transformation in a more scalable way with three bolts than with two). [13]

Trident
The core data model in Trident is the "Stream", processed as a series of batches. A stream is partitioned among the nodes in the cluster, and operations applied to a stream are applied in parallel across each partition.

There are five kinds of operations in Trident:

- Operations that apply locally to each partition and cause no network transfer

- Repartitioning operations that repartition a stream but otherwise don't change the contents (involves network transfer)
- Aggregation operations that do network transfer as part of the operation
- Operations on grouped streams
- Merges and joins

Functions in Tridents : A function takes in a set of input fields and emits zero or more tuples as output. The fields of the output tuple are appended to the original input tuple in the stream. If a function emits no tuples, the original input tuple is filtered out. Otherwise, the input tuple is duplicated for each output tuple.

Filters: Filters take in a tuple as input and decide whether or not to keep that tuple or not.

Map and FlatMap: Map returns a stream consisting of the result of applying the given mapping function to the tuples of the stream. This can be used to apply a one-one transformation to the tuples.

Peek: Peek can be used to perform an additional action on each trident tuple as they flow through the stream.

PartitionAggergate: PartitionAggregate runs a function on each partition of a batch of tuples.

Projection: The projection method on Stream keeps only the fields specified in the operation.

Group By: The groupBy operation repartitions the stream by doing a partitionBy on the specified fields, and then within each partition groups tuples together whose group fields are equal.[14]

Trident Vs Traditional spout and bolts

Trident is a high-level abstraction for doing realtime computing on top of Twitter Storm, available in Storm 0.8.x. Storm is stateless stream processing framework and Trident provides stateful stream processing.

Assume we want to do filtering + addition of a field to a tuple. If we use storm usually we use 2 bots for filtering, addition of field. So again, we need to send the tuple to new bolt by may be using global grouping. So here, now bandwidth may become bottleneck.By using trident we can use do above on a single machine. So no regrouping is needed in this case. Such use case in addition to "exactly once" or “at least once" can differentiate what to use.[15]

Chapter 4

Integration of MRQL and Apache storm

The Integration of Storm and MRQL started with the creation of the storm module in MRQL project. This is a maven module integrated with the existing MRQL project. A New module has been added in the parent pom.xml file.

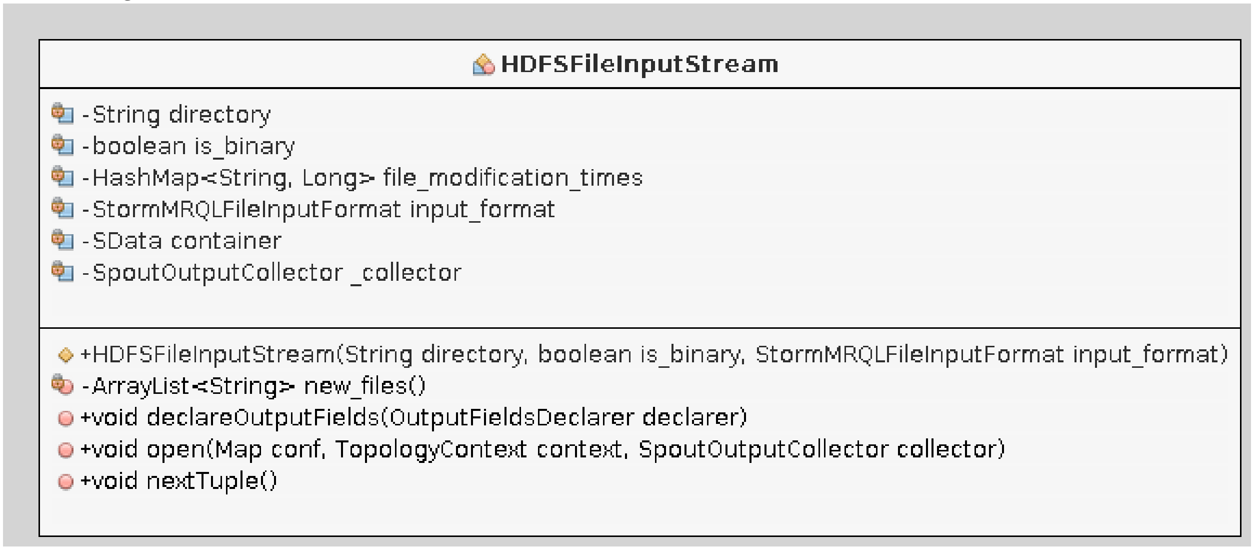
A new storm directory has been added and the submodule is located into that direcotory.

Module Architecture for Storm:

```
├── pom.xml
└── src
    ├── main
    │   ├── java
    │   │   └── org
    │   │       └── apache
    │   │           └── mrql
    │   │               ├── HDFSFileInputStream.java
    │   │               ├── MR_stream.java
    │   │               ├── SData.java
    │   │               ├── StormBinaryInputFormat.java
    │   │               ├── StormEvaluator.gen
    │   │               ├── StormFileInputStream.java
    │   │               ├── StormMRQLFileInputFormat.java
    │   │               ├── StormParsedInputFormat.java
    │   │               ├── StormReducer.java
    │   │               ├── StormStreaming.gen
    │   │               ├── StreamDataSource.java
    │   │               └── utils
    │   │                   └── PrintUtil.java
    │   └── resources
    └── test
        └── java
            └── org
                └── apache
                    └── mrql
                        ├── StormEvaluatorLocalModeTest.java
                        └── StormQueryLocalModeTest.java
```

Figure 6 Module Architecture for Storm

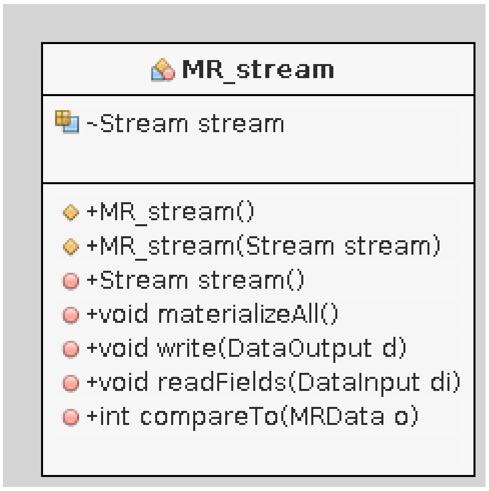
Class diagrams


HDFSFileInputStream

-String directory
-boolean is_binary
-HashMap<String, Long> file_modification_times
-StormMRQLFileInputFormat input_format
-SData container
-SpoutOutputCollector _collector

+HDFSFileInputStream(String directory, boolean is_binary, StormMRQLFileInputFormat input_format)
-ArrayList<String> new_files()
+void declareOutputFields(OutputFieldsDeclarer declarer)
+void open(Map conf, TopologyContext context, SpoutOutputCollector collector)
+void nextTuple()

Figure 7 HDFSFileInputStream Class


MR_stream

~Stream stream

+MR_stream()
+MR_stream(Stream stream)
+Stream stream()
+void materializeAll()
+void write(DataOutput d)
+void readFields(DataInput di)
+int compareTo(MRData o)

Figure 8 MR_Stream Class

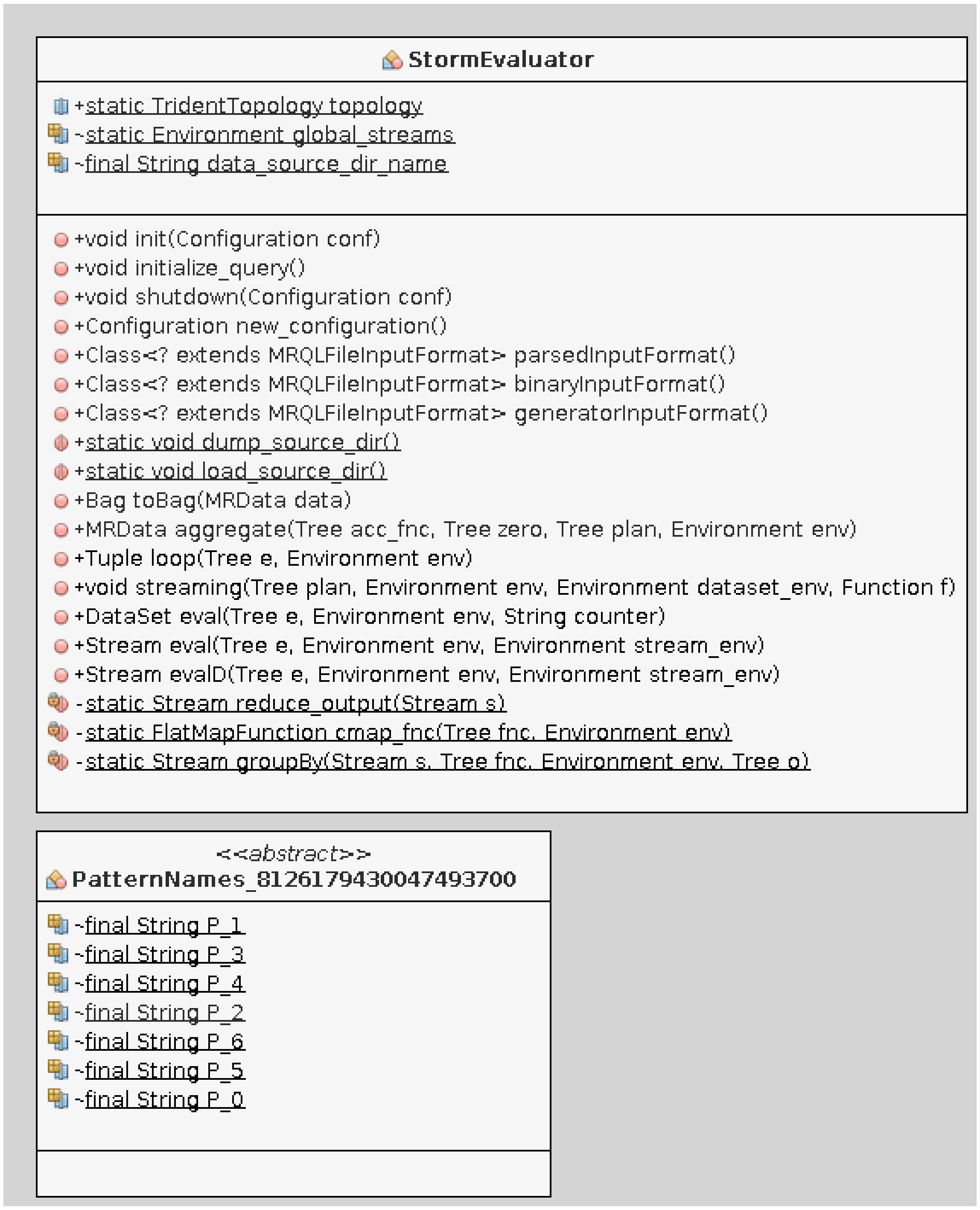

StormEvaluator
+static TridentTopology topology
~static Environment global_streams
~final String data_source_dir_name
+void init(Configuration conf)
+void initialize_query()
+void shutdown(Configuration conf)
+Configuration new_configuration()
+Class<? extends MRQLFileInputFormat> parsedInputFormat()
+Class<? extends MRQLFileInputFormat> binaryInputFormat()
+Class<? extends MRQLFileInputFormat> generatorInputFormat()
+static void dump_source_dir()
+static void load_source_dir()
+Bag toBag(MRData data)
+MRData aggregate(Tree acc_fnc, Tree zero, Tree plan, Environment env)
+Tuple loop(Tree e, Environment env)
+void streaming(Tree plan, Environment env, Environment dataset_env, Function f)
+DataSet eval(Tree e, Environment env, String counter)
+Stream eval(Tree e, Environment env, Environment stream_env)
+Stream evalD(Tree e, Environment env, Environment stream_env)
-static Stream reduce_output(Stream s)
-static FlatMapFunction cmap_fnc(Tree fnc, Environment env)
-static Stream groupBy(Stream s, Tree fnc, Environment env, Tree o)
<<abstract>>
PatternNames_8126179430047493700
~final String P_1
~final String P_3
~final String P_4
~final String P_2
~final String P_6
~final String P_5
~final String P_0

Figure 9 StormEvaluator Class

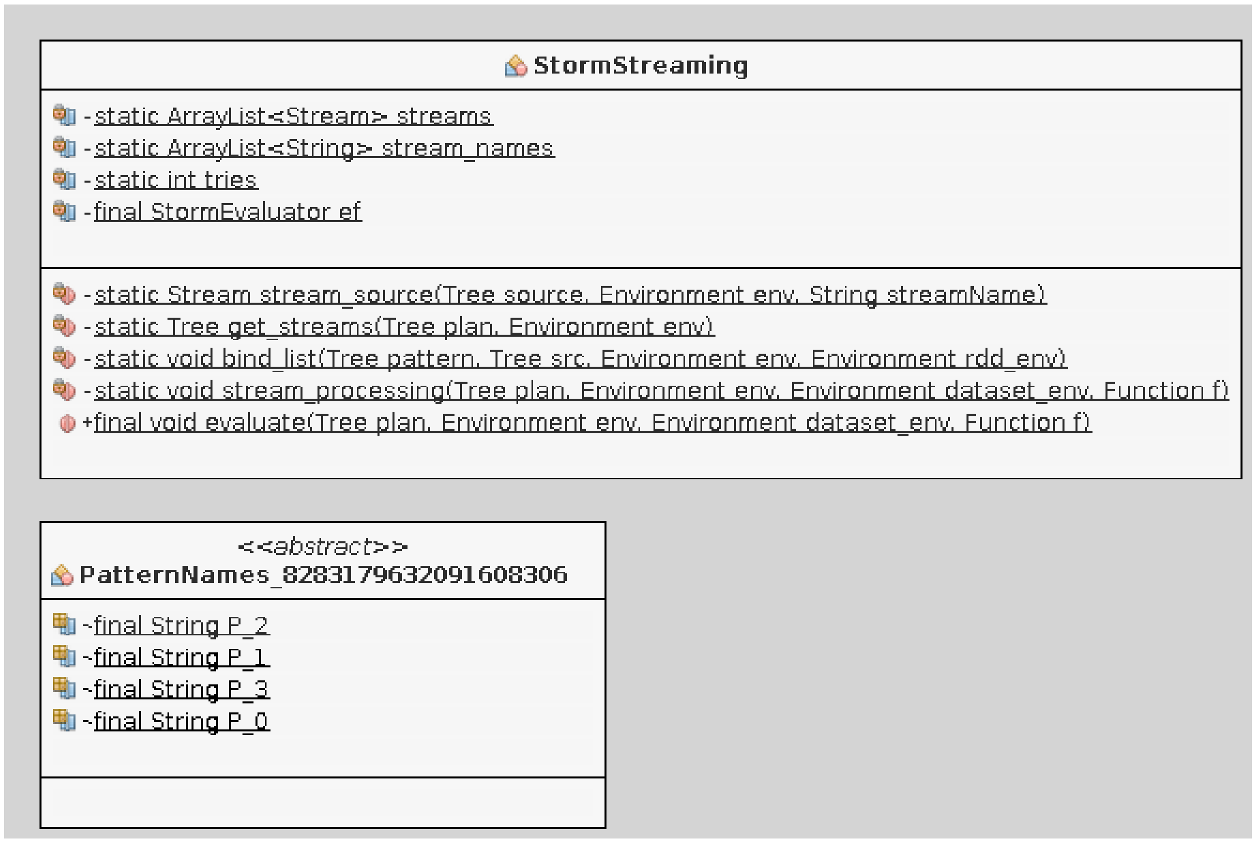

StormStreaming
-static ArrayList<Stream> streams
-static ArrayList<String> stream_names
-static int tries
-final StormEvaluator ef
-static Stream stream_source(Tree source, Environment env, String streamName)
-static Tree get_streams(Tree plan, Environment env)
-static void bind_list(Tree pattern, Tree src, Environment env, Environment rdd_env)
-static void stream_processing(Tree plan, Environment env, Environment dataset_env, Function f)
+final void evaluate(Tree plan, Environment env, Environment dataset_env, Function f)
<<abstract>>
PatternNames_8283179632091608306
~final String P_2
~final String P_1
~final String P_3
~final String P_0

Figure 10 StormStreaming Class

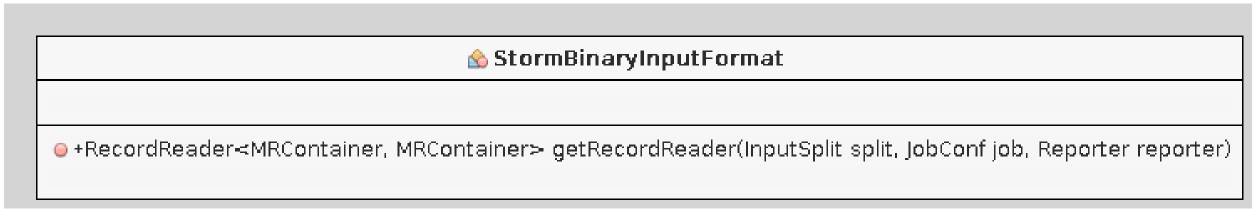

StormBinaryInputFormat
+RecordReader<MRContainer, MRContainer> getRecordReader(InputSplit split, JobConf job, Reporter reporter)

Figure 11 StormBinaryInputClass

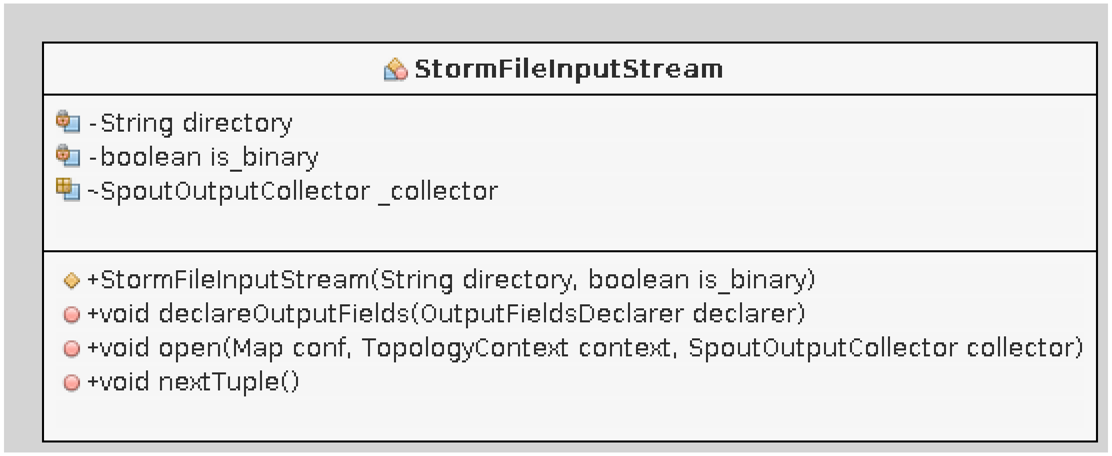
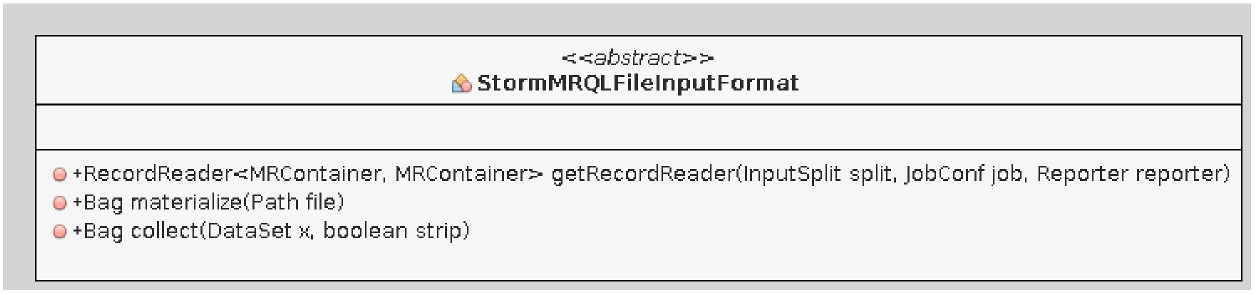

StormFileInputStream
-String directory
-boolean is_binary
~SpoutOutputCollector _collector
+StormFileInputStream(String directory, boolean is_binary)
+void declareOutputFields(OutputFieldsDeclarer declarer)
+void open(Map conf, TopologyContext context, SpoutOutputCollector collector)
+void nextTuple()

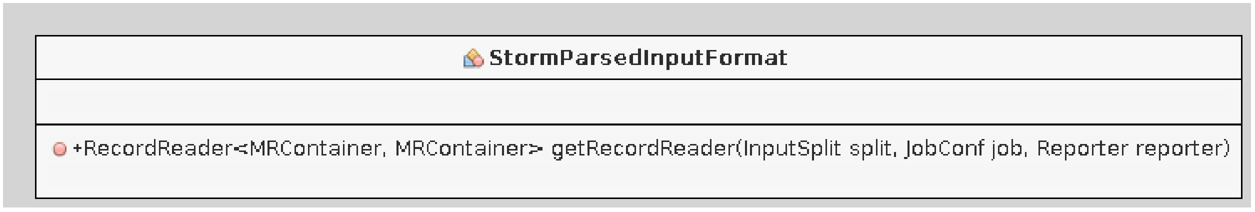
Figure 12 StormFileInputStream Class



```
<<abstract>>
StormMRQLFileInputFormat

+RecordReader<MRContainer, MRContainer> getRecordReader(InputSplit split, JobConf job, Reporter reporter)
+Bag materialize(Path file)
+Bag collect(DataSet x, boolean strip)
```

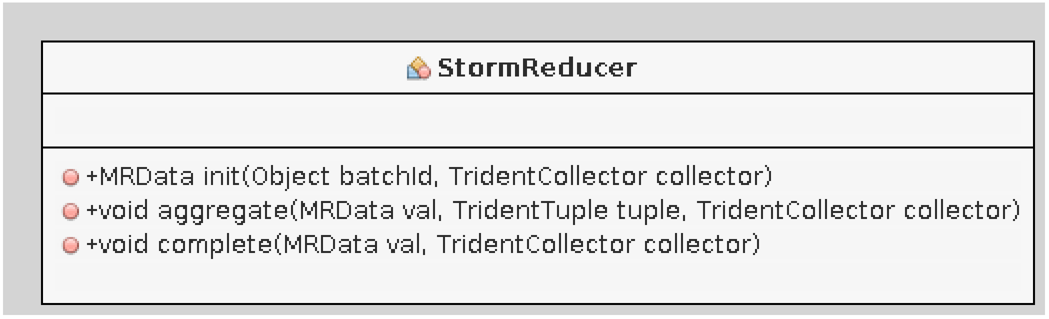
Figure 13 StormMRQLFileInputFormat Class



```
StormParsedInputFormat

+RecordReader<MRContainer, MRContainer> getRecordReader(InputSplit split, JobConf job, Reporter reporter)
```

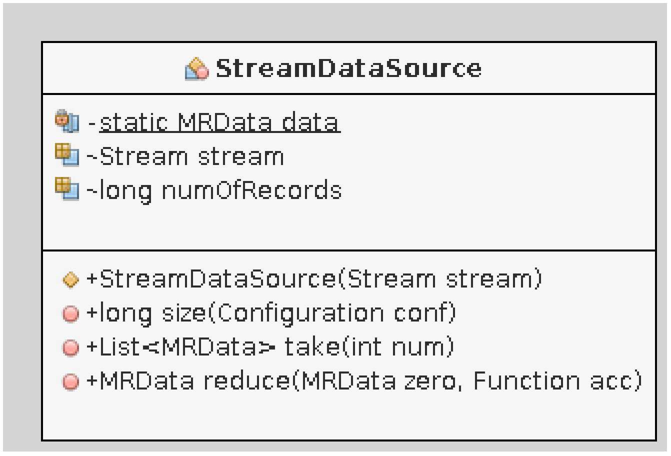
Figure 14 StormParsedInputFormat Class



```
StormReducer

+MRData init(Object batchId, TridentCollector collector)
+void aggregate(MRData val, TridentTuple tuple, TridentCollector collector)
+void complete(MRData val, TridentCollector collector)
```

Figure 15 StormReducer Class



```
StreamDataSource

-static MRData data
~Stream stream
~long numOfRecords

+StreamDataSource(Stream stream)
+long size(Configuration conf)
+List<MRData> take(int num)
+MRData reduce(MRData zero, Function acc)
```

Figure 16 StormDataSource Class

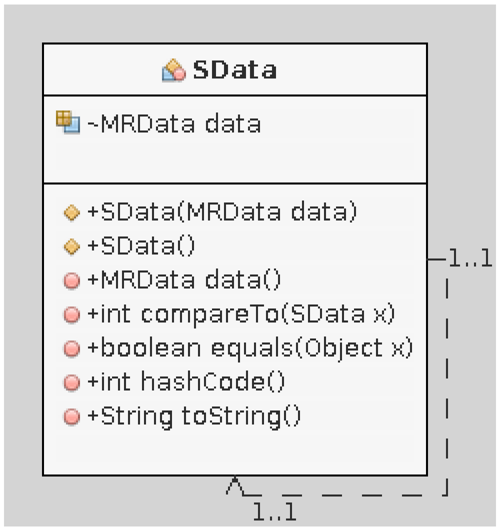


Figure 17 SData Class

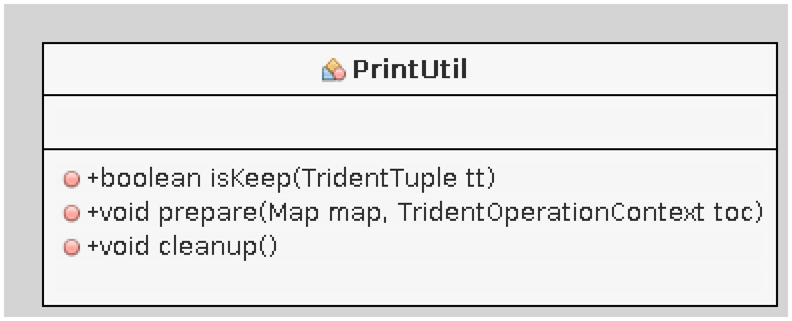

PrintUtil
+boolean isKeep(TridentTuple tt)
+void prepare(Map map, TridentOperationContext toc)
+void cleanup()

Figure 18 PrintUtil Class

Modular Sequence Diagram:

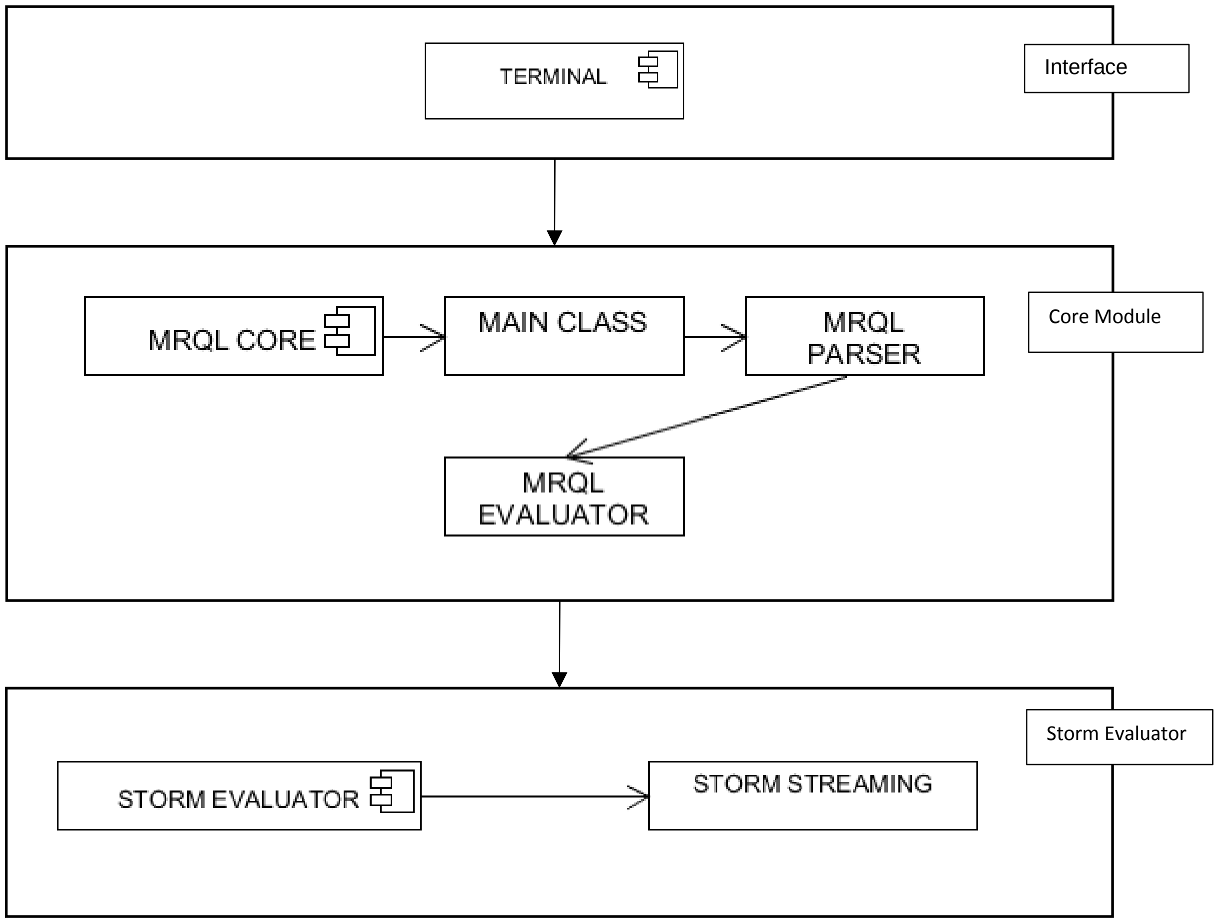


Figure 19 Module Sequence Diagram

Building Storm module:
To build the storm module we should go inside the storm directory and enter mvn clean install.

This command will invoke maven and install the library inside the local repository.

The output of the command is shown below:

```
achyut@achyut ~/ap/incubator-mrql/storm $ mvn clean install
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------
-----
[INFO] Building Apache MRQL Storm mode 0.9.8-incubating-SNAPSHOT
[INFO] ------------------------------------------------------------------
-----
[INFO]
[INFO] --- maven-clean-plugin:2.4.1:clean (default-clean) @ mrql-storm ---
```

```
[INFO] Deleting /home/achyut/ap/incubator-mrql/storm/target
[INFO] Deleting /home/achyut/ap/incubator-mrql/storm (includes = [**/*~,
lib/**, tests/results/**, tests/error_log.txt, mrql-tmp/**, tmp/**,
**/dependency-reduced-pom.xml, **/docs/**, **/.project, **/.classpath,
**/.settings/**], excludes = [])
[INFO]
[INFO] --- build-helper-maven-plugin:1.8:add-source (default) @ mrql-storm
---
[INFO] Source directory: /home/achyut/ap/incubator-
mrql/storm/target/generated-sources added.
[INFO]
[INFO] --- maven-antrun-plugin:1.7:run (default) @ mrql-storm ---
[INFO] Executing tasks

main:
    [mkdir] Created dir: /home/achyut/ap/incubator-
mrql/storm/target/generated-sources/org/apache/mrql
[INFO] Executed tasks
[INFO]
[INFO] --- maven-remote-resources-plugin:1.4:process (default) @ mrql-
storm ---
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @
mrql-storm ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 0 resource
[INFO] Copying 3 resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ mrql-
storm ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 12 source files to /home/achyut/ap/incubator-
mrql/storm/target/classes
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-
testResources) @ mrql-storm ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /home/achyut/ap/incubator-
mrql/storm/src/test/resources
[INFO] Copying 3 resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @
mrql-storm ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 2 source files to /home/achyut/ap/incubator-
mrql/storm/target/test-classes
[INFO]
[INFO] --- maven-surefire-plugin:2.14.1:test (default-test) @ mrql-storm -
--
[INFO] Tests are skipped.
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ mrql-storm ---
[INFO] Building jar: /home/achyut/ap/incubator-mrql/lib/mrql-storm-0.9.8-
incubating-SNAPSHOT.jar
```

```
[INFO]
[INFO] --- maven-site-plugin:2.1.1:attach-descriptor (attach-descriptor) @
mrql-storm ---
[INFO]
[INFO] --- apache-rat-plugin:0.8:check (default) @ mrql-storm ---
[INFO] Exclude: .git/**
[INFO] Exclude: .gitignore
[INFO] Exclude: .idea/**
[INFO] Exclude: **.iml
[INFO] Exclude: .classpath/**
[INFO] Exclude: .project/**
[INFO] Exclude: tests/data/**
[INFO] Exclude: tests/results/**
[INFO] Exclude: tests/error_log.txt
[INFO] Exclude: **/dependency-reduced-pom.xml
[INFO] Exclude: **/docs/apidocs/**
[INFO]
[INFO] --- maven-javadoc-plugin:2.9:jar (attach-javadocs) @ mrql-storm ---
[INFO] Building jar: /home/achyut/ap/incubator-mrql/storm/target/mrql-
storm-0.9.8-incubating-SNAPSHOT-javadoc.jar
[INFO]
[INFO] --- maven-shade-plugin:2.1:shade (default) @ mrql-storm ---
Excluding oro:oro:jar:2.0.8 from the shaded jar.
[INFO] Excluding org.eclipse.jdt:core:jar:3.1.1 from the shaded jar.
[INFO] Excluding org.codehaus.jackson:jackson-mapper-asl:jar:1.8.8 from
the shaded jar.
.
.
.
.
.
[INFO] Replacing original artifact with shaded artifact.
[INFO] Replacing /home/achyut/ap/incubator-mrql/lib/mrql-storm-0.9.8-
incubating-SNAPSHOT.jar with /home/achyut/ap/incubator-
mrql/storm/target/mrql-storm-0.9.8-incubating-SNAPSHOT-shaded.jar
[INFO]
[INFO] --- maven-install-plugin:2.3.1:install (default-install) @ mrql-
storm ---
[INFO] Installing /home/achyut/ap/incubator-mrql/lib/mrql-storm-0.9.8-
incubating-SNAPSHOT.jar to
/home/achyut/.m2/repository/org/apache/mrql/mrql-storm/0.9.8-incubating-
SNAPSHOT/mrql-storm-0.9.8-incubating-SNAPSHOT.jar
[INFO] Installing /home/achyut/ap/incubator-mrql/storm/pom.xml to
/home/achyut/.m2/repository/org/apache/mrql/mrql-storm/0.9.8-incubating-
SNAPSHOT/mrql-storm-0.9.8-incubating-SNAPSHOT.pom
[INFO] Installing /home/achyut/ap/incubator-mrql/storm/target/mrql-storm-
0.9.8-incubating-SNAPSHOT-javadoc.jar to
/home/achyut/.m2/repository/org/apache/mrql/mrql-storm/0.9.8-incubating-
SNAPSHOT/mrql-storm-0.9.8-incubating-SNAPSHOT-javadoc.jar
[INFO] ------------------------------------------------------------------
-----
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------
-----
```

```
[INFO] Total time: 13.490 s
[INFO] Finished at: 2016-10-18T12:13:57-05:00
[INFO] Final Memory: 51M/523M
[INFO] ------------------------------------------------------------------------
```

After the build has been successful, a build success message will appear and the libaray is installed in the local repository.

Chapter 5

Changes made in core module

Main.java

Line 66:

```
 (Evaluator)Class.forName("org.apache.mrql.StormEvaluator").newInstance();
        else // when Config.map_reduce_mode but also the default


Line 92:
Config.storm_mode |= arg.equals("-storm");
        };
        Config.map_reduce_mode = !Config.bsp_mode && !Config.spark_mode &&
!Config.flink_mode  && !Config.storm_mode;


Line 124:
            else if (Config.storm_mode)
                System.out.println("Storm mode using "+Config.nodes+"
tasks)");
```

Config.java

```
Line 43:
// true, for Storm mode
    public static boolean storm_mode = false;


Line 198:

}else if (args[i].equals("-storm")) {
                storm_mode = true;
                i++;
            }
```

Chapter 6

Code for Storm module in MRQL:

Mrql.storm

```
#!/bin/bash
#--------------------------------------------------------------------------------
----
#
# Licensed to the Apache Software Foundation (ASF) under one
# or more contributor license agreements.  See the NOTICE file
# distributed with this work for additional information
# regarding copyright ownership.  The ASF licenses this file
# to you under the Apache License, Version 2.0 (the
# "License"); you may not use this file except in compliance
# with the License.  You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing,
# software distributed under the License is distributed on an
# "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
# KIND, either express or implied.  See the License for the
# specific language governing permissions and limitations
# under the License.
#

#cd storm
#mvn clean install
#cd ..
#exit(0)

MRQL_HOME="$(cd `dirname $0`/..; pwd -P)"

. "$MRQL_HOME/conf/mrql-env.sh"

GEN_JAR=`ls "$MRQL_HOME"/lib/mrql-gen-*.jar`
CORE_JAR=`ls "$MRQL_HOME"/lib/mrql-core-*.jar`
MRQL_JAR=`ls "$MRQL_HOME"/lib/mrql-storm-*.jar`
FULL_JAR="/tmp/${USER}_mrql_storm.jar"
CLASS_DIR="/tmp/${USER}_mrql_classes"

if [[ -z ${STORM_JARS} ]]; then
   echo "*** Cannot find the Storm jar file. Need to edit mrql-env.sh"; exit
-1
fi
HADOOP_JARS=${HADOOP_HOME}/share/hadoop/mapreduce/hadoop-mapreduce-client-
core-${HADOOP_VERSION}.jar:${HADOOP_HOME}/share/hadoop/common/hadoop-common-
${HADOOP_VERSION}.jar:${HADOOP_HOME}/share/hadoop/hdfs/hadoop-hdfs-
${HADOOP_VERSION}.jar:${HADOOP_HOME}/share/hadoop/common/lib/*
LAMDA_JARS="/tmp/mrql_jar_${USER}/*"

export JAVA_HOME FS_DEFAULT_NAME BSP_MASTER_ADDRESS STORM_ZOOKEEPER_QUORUM
BSP_SPLIT_INPUT
```

```
if [[ ($MRQL_JAR -nt $FULL_JAR) ]]; then
   rm -rf $CLASS_DIR
   mkdir -p $CLASS_DIR
   pushd $CLASS_DIR > /dev/null
   $JAVA_HOME/bin/jar xf $CUP_JAR
   $JAVA_HOME/bin/jar xf $JLINE_JAR
   $JAVA_HOME/bin/jar xf $GEN_JAR
   $JAVA_HOME/bin/jar xf $CORE_JAR
   $JAVA_HOME/bin/jar xf $MRQL_JAR
   cd ..
   $JAVA_HOME/bin/jar cf $FULL_JAR -C $CLASS_DIR .
   popd > /dev/null
fi

if [ "$1" == "-local" ]; then
   export STORM_CLASSPATH=$FULL_JAR  # FOR LOCAL CLUSTER IN STORM
   $STORM_HOME/bin/storm --config $STORM_CONFIG jar $FULL_JAR
org.apache.mrql.Main -storm $*
else if [ "$1" == "-dist" ]; then
   $STORM_HOME/bin/storm --config $STORM_CONFIG jar $FULL_JAR
org.apache.mrql.Main -storm $*
else
   #$JAVA_HOME/bin/java -classpath "$FULL_JAR:$STORM_JARS:$HADOOP_JARS" -
Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=1044
org.apache.mrql.Main -storm $*
   $JAVA_HOME/bin/java -classpath
"$FULL_JAR:$STORM_JARS:$HADOOP_JARS:$LAMDA_JARS" org.apache.mrql.Main -storm
$*
fi
fi
```

StormStreaming.gen

```
/**
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License.  You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.apache.mrql;


import java.util.ArrayList;
import java.io.*;
import org.apache.mrql.gen.*;
import org.apache.storm.trident.Stream;

/*
import org.apache.storm.util.Collector;
import org.apache.storm.core.fs.Path;
import org.apache.storm.core.fs.FileSystem;
import org.apache.storm.api.common.io.FileInputFormat;
import org.apache.storm.api.common.functions.*;
import org.apache.storm.api.common.operators.*;
import org.apache.storm.api.java.*;
import org.apache.storm.api.java.DataSet;
import org.apache.storm.api.java.tuple.Tuple2;
import org.apache.storm.api.java.io.LocalCollectionOutputFormat;
import org.apache.storm.api.java.io.DiscardingOutputFormat;
import org.apache.storm.api.java.functions.*;
import org.apache.storm.api.java.operators.*;
import org.apache.storm.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.storm.client.program.ContextEnvironment;

*/


/** Evaluates physical plans in Apache Storm mode */
public class StormStreaming extends StormEvaluator implements Serializable {

    private static ArrayList<Stream> streams;
    private static ArrayList<String> stream_names;
    private static int tries = 0;
    private final static StormEvaluator ef =
(StormEvaluator)Evaluator.evaluator;

    private static Stream stream_source ( Tree source, Environment env,String
streamName) {
        System.out.println("Steam source");
        match source {
        case BinaryStream(`file,_):
            String path = ((MR_string)evalE(file,env)).get();
            new BinaryDataSource(path,Plan.conf);
            return topology.newStream(streamName, new
HDFSFileInputStream(path, is_dataset, new StormBinaryInputFormat()));

        case ParsedStream(`parser,`file,...args):
            System.out.println("Inside parsed streaming");

            String path = ((MR_string)evalE(file,env)).get();
            Class<? extends Parser> p =
DataSource.parserDirectory.get(parser.toString());
            if (p == null)
                throw new Error("Unknown parser: "+parser);
            new ParsedDataSource(path,p,args,Plan.conf);
            try {
                dump_source_dir();
              } catch (IOException ex) {
                  throw new Error("Cannot dump source directory");
              };
              return topology.newStream(streamName, new
HDFSFileInputStream(path, is_dataset, new StormParsedInputFormat()));

        };
        throw new Error("Unknown stream source: "+print_query(source));
    }

    private static Tree get_streams ( Tree plan, Environment env ) {
        System.out.println("Inside get streams");
        match plan {
        case Stream(lambda(`v,`b),`source):
            streams.add(stream_source(source,env,v.toString()));
            stream_names.add(v.toString());
            return get_streams(b,env);
        };
        return plan;
    }

    /** bind the pattern variables to values */
    private static void bind_list ( Tree pattern, Tree src, Environment env,
Environment rdd_env ) {
        System.out.println("Inside bind_list ");
    }

     private static void stream_processing ( final Tree plan, final
Environment env,
                                            final Environment dataset_env,
final Function f ) {

        System.out.println("stream stream_processing for storm");

        if (streams.size() == 0)
            throw new Error("No input streams in query");
        ArrayList<Stream> rdds = new ArrayList<Stream>();
        for ( Stream jd: streams )
            rdds.add(jd);


        final ArrayList<String> vars = stream_names;


        long t = System.currentTimeMillis();
        Environment rdd_env = dataset_env;
        int i = 0;
        /*
        for ( RDD<?> rdd: JavaConversions.seqAsJavaList(rdds) ) {
            try {
                if (rdd.count() == 0)
                    return
SparkEvaluator.spark_context.sc().emptyRDD(classtag);
            } catch (Exception ex) {
                return SparkEvaluator.spark_context.sc().emptyRDD(classtag);
            };

        };
        */

        for(String name:stream_names){
            MRData d = new MR_stream(streams.get(i));
            global_streams = new Environment(vars.get(i),d,global_streams);
            rdd_env = new Environment(vars.get(i++),d,rdd_env);
        }

        match plan {
        case lambda(`pat,`e):
            bind_list(pat,e,env,rdd_env);
            f.eval(null);
        case _: // non-incremental streaming
            f.eval(ef.evalE(plan,env));
        };

    }

    /** evaluate plan in stream mode: evaluate each batch of data and apply
the function f */
    final public static void evaluate ( Tree plan, Environment env,
Environment dataset_env, Function f ) {
        System.out.println("inside evalue of stream");
        streams = new ArrayList<Stream>();
        stream_names = new ArrayList<String>();
        match plan {
        case lambda(`p,`b):
            b = get_streams(b,env);
            stream_processing(#<lambda(`p,`b)>,env,null,f);
        case _:  // non-incremental streaming
            plan = get_streams(plan,env);
            stream_processing(plan,env,dataset_env,f);
        };

        try {
            System.out.println("Starting a topology");
            try{


                Thread.sleep(10000);
            }
            catch(Exception e){
                System.out.println("error while sleeping");
            }
            System.out.println("Topology execution completed");
            //stream_context.start();
            //stream_context.awaitTermination();
        } catch (Exception ex) {
            throw new Error(ex);
        }

    }
}
```

StormEvaluator.gen

```
/*
 * Copyright 2016 The Apache Software Foundation.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.apache.mrql;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.io.Serializable;
import java.net.URL;
import java.net.URLClassLoader;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.mrql.gen.Node;
import org.apache.mrql.gen.Tree;
import org.apache.mrql.gen.Trees;
import org.apache.mrql.gen.VariableLeaf;
import org.apache.storm.LocalCluster;
import org.apache.storm.trident.Stream;
import org.apache.storm.trident.TridentTopology;
import org.apache.storm.trident.operation.FlatMapFunction;
import org.apache.storm.trident.tuple.TridentTuple;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Values;
import org.apache.mrql.utils.PrintUtil;
import org.apache.storm.trident.fluent.GroupedStream;
import org.apache.storm.trident.operation.BaseAggregator;
import org.apache.storm.trident.operation.BaseFunction;
import org.apache.storm.trident.operation.TridentCollector;

/**
 *
 * @author achyut
 */
public class StormEvaluator extends Evaluator implements Serializable {

    public static TridentTopology topology;
    static Environment global_streams = null;
    final static String data_source_dir_name =
"tmp/"+System.getenv("USER")+"_data_source_dir.txt";

    @Override
    public void init(Configuration conf) {
        global_streams = null;
        System.out.println("Inside the init");
        if (Config.stream_window > 0 && (Config.local_mode ||
Config.hadoop_mode)) {
            System.out.println("Creating a new topology");
        }
        topology = new TridentTopology();
        Plan.conf = new Configuration();
        if (Config.hadoop_mode && Config.local_mode) {
            FileSystem.setDefaultUri(Plan.conf,"file:///");
        } else if (Config.hadoop_mode) {
            if (!System.getenv("FS_DEFAULT_NAME").equals(""))

FileSystem.setDefaultUri(Plan.conf,System.getenv("FS_DEFAULT_NAME"));
        };
    }

    @Override
    public void initialize_query() {
        System.out.println("-----------------------------");
        System.out.println("-----------------------------");
        System.out.println("-----------------------------");
        System.out.println("-----------------------------");
        System.out.println("initialize_query");
        Plan.distribute_compiled_arguments(Plan.conf);
//        System.setProperty("storm.jar", Plan.conf.get("mrql.jar.path"));
        String jarPath =  Plan.conf.get("mrql.jar.path");
        System.out.println(jarPath);
        try{
            URLClassLoader child = new URLClassLoader (new URL[] {new
URL("file://"+jarPath)}, StormEvaluator.class.getClassLoader());
        }
        catch(Exception e){
            e.printStackTrace();
        }
    }

    @Override
    public void shutdown(Configuration conf) {

    }

    @Override
    public Configuration new_configuration() {
        return new Configuration();
    }

    @Override
    public Class<? extends MRQLFileInputFormat> parsedInputFormat() {
        return StormParsedInputFormat.class;
    }
    @Override
    public Class<? extends MRQLFileInputFormat> binaryInputFormat() {
        return StormBinaryInputFormat.class;
    }

    @Override
    public Class<? extends MRQLFileInputFormat> generatorInputFormat() {
        return null;
    }

    /** used by the master to send parsing details (eg, record types) to
workers */
    public static void dump_source_dir () throws IOException {
        if (Config.local_mode)
            return;
        DataSource.dataSourceDirectory.distribute(Plan.conf);
        Path path = new Path(data_source_dir_name);
        FileSystem fs = path.getFileSystem(Plan.conf);
        PrintStream ps = new PrintStream(fs.create(path,true));
        ps.println(Plan.conf.get("mrql.data.source.directory"));
        ps.close();
    }

    /** executed by a worker when reading parsed input (see
SparkParsedInputFormat) */
    public static void load_source_dir () throws IOException {
        if (Plan.conf == null) {
            if (evaluator == null)
                evaluator = new StormEvaluator();
            Plan.conf = evaluator.new_configuration();
            Config.read(Plan.conf);
        };
        if (Config.local_mode)
            return;
        // the name of the file that contains the source directory details is
read from an HDFS file by workers
        Path path = new Path(data_source_dir_name);
        FileSystem fs = path.getFileSystem(Plan.conf);
        BufferedReader ftp = new BufferedReader(new
InputStreamReader(fs.open(path)));
        Plan.conf.set("mrql.data.source.directory",ftp.readLine());
        DataSource.dataSourceDirectory.read(Plan.conf);
        ftp.close();
    }

    @Override
    public Bag toBag(MRData data) {
        System.out.println("Inside bag");
        return null;
    }

/*
    private final static MapFunction get_first
        = new MapFunction() {
           public Values execute(TridentTuple input) {
               MRData value = (MRData)input.get(0);
               return new Values(((Tuple)value).first());
            };
    };
*/

    @Override
    public MRData aggregate(Tree acc_fnc, Tree zero, Tree plan, Environment
env) throws Exception {
        System.out.println("Inside aggregate");
        return null;
    }

    @Override
    public Tuple loop(Tree e, Environment env) throws Exception {
        return null;
    }

    @Override
    public void streaming(Tree plan, Environment env, Environment
dataset_env, Function f) {
        StormStreaming.evaluate(plan, env, dataset_env, f);
    }

    @Override
    public DataSet eval(Tree e, Environment env, String counter) {
        Stream res = eval(e, env,(Environment)null);

        res.filter(new PrintUtil());


        org.apache.storm.Config conf = new org.apache.storm.Config();
        conf.setFallBackOnJavaSerialization(false);
        try{

            conf.setMaxSpoutPending(20);
            LocalCluster cluster = new LocalCluster();
            cluster.submitTopology("mrqlstormevaluator", conf,
topology.build());
            Thread.sleep(15000);
            cluster.shutdown();

        }
        catch(Exception e2){
            e2.printStackTrace();
        }

        return null;
    }

    final public Stream eval(final Tree e, final Environment env,final
Environment stream_env){
        if (Config.trace_execution) {
            tab_count += 3;
            System.out.println(tabs(tab_count) + print_query(e));
        };
        final Stream res = evalD(e, env,stream_env);
        return res;
    }
    final public Stream evalD ( final Tree e, final Environment env,final
Environment stream_env ) {
        System.out.println("Inside eval1D");
        try {
            match e {
            case MapAggregateReduce(`m,`r,null,_,`s,`o):
                System.out.println("MapAggregateReduce");
                return null;
            case cMap(`f,`s):
                System.out.println("cMap");
                return eval(s,env,stream_env).flatMap(cmap_fnc(f,env));
            case MapReduce(`m,`r,`s,true):
                System.out.println("inside the map reduce true");
            //    return sortBy(eval(s,env).flatMap(cmap_fnc(m)),null,r);
                return null;
            case MapReduce(`m,`r,`s,`o):
                System.out.println("inside the map reduce object");
            //    return groupBy(eval(s,env).flatMap(cmap_fnc(m)),null,r);
                return null;
            case MapCombineReduce(`m,`c,`r,`s,true):
                System.out.println("inside the map combine reduce true");
                //return sortBy(eval(s,env).flatMap(cmap_fnc(m)),c,r);
                return null;
            case MapCombineReduce(`m,`c,`r,`s,`o):
                System.out.println("inside the map combine reduce object");
            //    return eval(s,env,stream_env).flatMap(cmap_fnc(m,env));
                Stream mappedStream =
eval(s,env,stream_env).flatMap(cmap_fnc(m,env));
                return groupBy(mappedStream,r,env,o);
            case ParsedSource(`parser,`file,...args):
                System.out.println("inside the parsed source");
                return null;

               // return null;
            // case MapCombineReduce(`m,`c,`r,`s,`o):
            //     System.out.println("Inside map combine reduce");
            //    return
groupBy(eval(s,env,stream_env).flatMap(cmap_fnc(m,env))
            //                    .map(combiner_fnc(evalF(c,env))),r,env,o);

            case `v:
                System.out.println("v");
                 if (!v.is_variable())
                    fail;
                MRData x = variable_lookup(v.toString(),global_streams);
                if (x != null && x instanceof MR_stream)
                    return ((MR_stream)x).stream();

                x = variable_lookup(v.toString(),global_env);
                System.out.println(x);
                x = variable_lookup(v.toString(),env);

                System.out.println(x);

                throw new Error("Variable "+v+" is not bound");
            };
            throw new Error("Unrecognized Storm plan: "+e);
        } catch (Error msg) {
            if (!Config.trace)
                throw new Error(msg.getMessage());
            System.err.println(msg.getMessage());
            throw new Error("Evaluation error in: "+print_query(e));
        } catch (Exception ex) {
            System.err.println(ex.getMessage());
            ex.printStackTrace();
            throw new Error("Evaluation error in: "+print_query(e));
        }
    }

    private static Stream reduce_output(Stream s){
        return null;
    }
    private static FlatMapFunction cmap_fnc ( final Tree fnc,Environment env
) {
       final Function f = evalF(fnc,null);
       return new FlatMapFunction() {
           @Override
           public Iterable<Values> execute(TridentTuple input) {
            //    System.out.println("input in cmap is "+input);

               List<Values> out = new ArrayList<Values>();
                MRData value = (MRData)input.get(0);
                for (MRData e: (Bag)f.eval(value) ){
                    out.add(new Values(e));
                }
            //    System.out.println("output from cmap is "+out);

                return out;
           }
       };
    }

    private static Stream groupBy ( Stream s, Tree fnc, Environment env, Tree
o ) {
       final Function reducer = evalF(fnc,null);

        /*
        Stream newstream = s.each(s.getOutputFields(),new BaseFunction() {
           @Override
           public void execute(TridentTuple tuple, TridentCollector
collector) {
               System.out.println(tuple);
           }
       },new Fields("outputdata"));
        s.filter(new PrintUtil());
        return newstream;
        */



        Stream keyValueStream = s.each(s.getOutputFields(),new BaseFunction()
{
```

```
            @Override
            public void execute(TridentTuple input, TridentCollector output)
{
                //System.out.println("Input from each function "+input);
                Tuple value = (Tuple)input.get(0);
                MRData key = value.first();
                //System.out.println("key --- "+key);
                MRData values = (MRData)value.second();
                //System.out.println("values --- "+values);

                output.emit(new Values(key,values));
            }
        },new Fields("keyfield","valuefield"));




        GroupedStream groupedStream = keyValueStream.groupBy(new
Fields("keyfield"));
        //System.out.println(groupedStream.getOutputFields());

      // keyValueStream.filter(new PrintUtil());

        //return keyValueStream;


       Stream finalStream = groupedStream.aggregate(new
Fields("keyfield","valuefield"), new BaseAggregator<Map<MRData,Bag>>() {

            @Override
            public Map<MRData,Bag> init(Object batchId, TridentCollector
collector) {
                return new HashMap<MRData,Bag>();
            }

            @Override
            public void aggregate(Map<MRData,Bag> val, TridentTuple tuple,
TridentCollector collector) {
                //System.out.println("in aggregate:"+tuple);

                MRData key = (MRData)tuple.get(0);
                Bag values;
                if(!val.containsKey(key)){
                    values = new Bag();
                }
                else{
                    values = val.get(key);
                }

                // System.out.println("The key is:--"+key);

                MRData value = (MRData)tuple.get(1);
                //System.out.println(key);
                //System.out.println(value);

                //Tuple t = (Tuple)value;
                //key = t.first();
                values.add(value);
                val.put(key,values);

            }

            @Override
            public void complete(Map<MRData,Bag> val, TridentCollector
collector) {
                for (Map.Entry<MRData, Bag> entry : val.entrySet()) {
                    MRData key = entry.getKey();
                    Bag value = entry.getValue();
                    Bag reducedValues = (Bag)reducer.eval(new
Tuple(key,value));

                    //System.out.println("---- reduced values ----");
                    //System.out.println(reducedValues);
                    //System.out.println("---- end of reduced values ----");

                    for ( MRData v: reducedValues){
                        collector.emit(new Values(v));
                    }
                }

            }
        }, new Fields("outputdata"));
        System.out.println(finalStream.getOutputFields());
        return finalStream.project(new Fields("outputdata"));

    }


}
```

HDFSFileInputStream.java

```
/*
 * Copyright 2016 The Apache Software Foundation.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.apache.mrql;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.fs.PathFilter;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.storm.spout.SpoutOutputCollector;
import org.apache.storm.task.TopologyContext;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.base.BaseRichSpout;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Values;

public class HDFSFileInputStream extends BaseRichSpout{
    private final String directory;
    private final boolean is_binary;
    private HashMap<String,Long> file_modification_times;
    private StormMRQLFileInputFormat input_format;
    final private SData container = new SData();
    private SpoutOutputCollector _collector;

    public HDFSFileInputStream(String directory, boolean
is_binary,StormMRQLFileInputFormat input_format) {
        this.directory = directory;
        this.is_binary = is_binary;
        this.input_format = input_format;
    }

    private ArrayList<String> new_files () {
        try {
            long ct = System.currentTimeMillis();
            Path dpath = new Path(directory);
            final FileSystem fs = dpath.getFileSystem(Plan.conf);
            final FileStatus[] ds
                = fs.listStatus(dpath,
                                new PathFilter () {
                                    public boolean accept ( Path path ) {
                                        return
!path.getName().startsWith("_")
                                            &&
!path.getName().endsWith(".type");
                                    }
                                });
            ArrayList<String> s = new ArrayList<String>();
            for ( FileStatus d: ds ) {
                String name = d.getPath().toString();
                if (file_modification_times.get(name) == null
                       || d.getModificationTime() >
file_modification_times.get(name)) {
                    file_modification_times.put(name,new Long(ct));
                    s.add(name);
                }
            };
            return s;
        } catch (Exception ex) {
            throw new Error("Cannot open a new file from the directory
"+directory+": "+ex);
        }
    }

    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("inputdata"));
    }

    @Override
    public void open(Map conf, TopologyContext context, SpoutOutputCollector
collector) {
        this.file_modification_times =  new HashMap<String,Long>();
        _collector = collector;
    }

    @Override
    public void nextTuple() {
        try{
           // System.out.println("--- Checking for new files in hdfs");
            for ( String path: new_files() ) {
                Path filePath = new Path(path);
                Bag value = input_format.materialize(filePath);
                for(MRData val : value){
                    _collector.emit(new Values(val));
                }
            }
        }
        catch(Exception e){
            e.printStackTrace();
        }

    }

}
```

MR_Stream.java:

```
/*
 * Copyright 2016 The Apache Software Foundation.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.apache.mrql;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import org.apache.storm.trident.Stream;

public class MR_stream extends MRData{
    Stream stream;

    public MR_stream() {
    }

    public MR_stream(Stream stream) {
        this.stream = stream;
    }

    public Stream stream(){
        return this.stream;
    }

    @Override
    public void materializeAll() {
        throw new UnsupportedOperationException("Not supported yet."); //To
change body of generated methods, choose Tools | Templates.
    }

    @Override
    public void write(DataOutput d) throws IOException {
        throw new UnsupportedOperationException("Not supported yet."); //To
change body of generated methods, choose Tools | Templates.
    }

    @Override
    public void readFields(DataInput di) throws IOException {
        throw new UnsupportedOperationException("Not supported yet."); //To
change body of generated methods, choose Tools | Templates.
    }

    @Override
    public int compareTo(MRData o) {
        throw new UnsupportedOperationException("Not supported yet."); //To
change body of generated methods, choose Tools | Templates.
    }


}
```

SData.java

```
/*
 * Copyright 2016 The Apache Software Foundation.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.apache.mrql;

import java.io.Serializable;

/**
 *
 * @author achyut
 */
public class SData implements Comparable<SData>,Serializable{
     MRData data;

    public SData(MRData data) {
        this.data = data;
    }

    public SData() {
    }

    public MRData data () { return data; }


    @Override
    public int compareTo ( SData x ) { return data.compareTo(x.data); }

    @Override
    public boolean equals ( Object x ) {
        return x instanceof SData && data.equals(((SData)x).data);
    }

    @Override
    public int hashCode () { return data.hashCode(); }

    @Override
    public String toString () { return data.toString(); }
}
```

StormBinaryInputFormat.java:

```
/*
 * Copyright 2016 The Apache Software Foundation.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.apache.mrql;

import java.io.IOException;
import org.apache.hadoop.mapred.FileSplit;
import org.apache.hadoop.mapred.InputSplit;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.SequenceFileRecordReader;

public class StormBinaryInputFormat extends StormMRQLFileInputFormat{
    public static class BinaryInputRecordReader extends
SequenceFileRecordReader<MRContainer,MRContainer> {
        final MRContainer result = new MRContainer();

        public BinaryInputRecordReader ( FileSplit split,
                                         JobConf job ) throws IOException {
            super(job,split);
        }

        @Override
        public synchronized boolean next ( MRContainer key, MRContainer value
) throws IOException {
                boolean b = super.next(key,result);
                value.set(result.data());
                return b;
        }
    }

    @Override
    public RecordReader<MRContainer,MRContainer>
              getRecordReader ( InputSplit split,
                                JobConf job,
                                Reporter reporter ) throws IOException {
        String path = ((FileSplit)split).getPath().toString();
        BinaryDataSource ds = (BinaryDataSource)DataSource.get(path,job);
        return new BinaryInputRecordReader((FileSplit)split,job);
    }

}
```

StormFileInputStream.java
```
/*
 * Copyright 2016 The Apache Software Foundation.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.apache.mrql;

import java.util.List;
import java.util.Map;
import org.apache.storm.spout.SpoutOutputCollector;
import org.apache.storm.task.TopologyContext;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.base.BaseRichSpout;
import org.apache.storm.trident.operation.TridentCollector;
import org.apache.storm.trident.spout.IBatchSpout;
import org.apache.storm.trident.testing.FixedBatchSpout;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Values;

/**
 *
 * @author achyut
 */
public class StormFileInputStream extends BaseRichSpout{
     private String directory;
     private boolean is_binary;

    SpoutOutputCollector _collector;
    public StormFileInputStream(String directory, boolean is_binary) {
        this.directory = directory;
        this.is_binary = is_binary;
    }


    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("data"));
    }

    @Override
    public void open(Map conf, TopologyContext context, SpoutOutputCollector
collector) {
           _collector = collector;
    }

    @Override
    public void nextTuple() {
        System.out.println("--------------------************************");
        System.out.println(directory);
        System.out.println(is_binary);
        Bag val =  (Bag)MapReduceAlgebra.read_binary(directory);
        _collector.emit(new Values(val));
    }

}
```

StormMRQLFileInputFormat

```
/*
 * Copyright 2016 The Apache Software Foundation.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.apache.mrql;

import java.io.IOException;
import java.io.Serializable;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.InputSplit;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.Reporter;

/**
 *
 * @author achyut
 */
public abstract class StormMRQLFileInputFormat extends
FileInputFormat<MRContainer,MRContainer> implements
MRQLFileInputFormat,Serializable {

    abstract public RecordReader<MRContainer,MRContainer>
        getRecordReader ( InputSplit split, JobConf job, Reporter reporter )
throws IOException;

    @Override
    public Bag materialize(final Path file) throws IOException {
        final JobConf job = new JobConf(Plan.conf,MRQLFileInputFormat.class);
        setInputPaths(job,file);
        final InputSplit[] splits = getSplits(job,1);
        final Reporter reporter = null;
        final RecordReader<MRContainer,MRContainer> rd =
getRecordReader(splits[0],job,reporter);
        return new Bag(new BagIterator () {
                RecordReader<MRContainer,MRContainer> reader = rd;
                MRContainer key = reader.createKey();
                MRContainer value = reader.createKey();
                int i = 0;
                public boolean hasNext () {
                    try {
                        if (reader.next(key,value))
                            return true;
                        do {
                            if (++i >= splits.length)
                                return false;
                            reader.close();
                            reader = getRecordReader(splits[i],job,reporter);
                        } while (!reader.next(key,value));
                        return true;
                    } catch (IOException e) {
                        throw new Error("Cannot collect values from an
intermediate result");
                    }
                }
                public MRData next () {
                    return value.data();
                }
            });
    }

    @Override
    public Bag collect(DataSet x, boolean strip) throws Exception {
        throw new UnsupportedOperationException("Not supported yet.");
    }

}
```

StormParsedInputFormat

```
/*
 * Copyright 2016 The Apache Software Foundation.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.apache.mrql;

import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapred.FileSplit;
import org.apache.hadoop.mapred.InputSplit;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.Reporter;
import org.apache.mrql.gen.Trees;

/**
 *
 * @author achyut
 */
public class StormParsedInputFormat extends StormMRQLFileInputFormat{
    public static class ParsedRecordReader implements 
RecordReader<MRContainer,MRContainer> {
        final FSDataInputStream fsin;
        final long start;
        final long end;
        Iterator<MRData> result;
        Parser parser;

        public ParsedRecordReader ( FileSplit split,
                                    Configuration conf,
                                    Class<? extends Parser> parser_class,
                                    Trees args ) throws IOException {
            start = split.getStart();
            end = start + split.getLength();
            Path file = split.getPath();
            FileSystem fs = file.getFileSystem(conf);
            fsin = fs.open(split.getPath());
            try {
                parser = parser_class.newInstance();
            } catch (Exception ex) {
                throw new Error("Unrecognized parser:"+parser_class);
            };
            parser.initialize(args);
            parser.open(fsin,start,end);
            result = null;
        }

        public MRContainer createKey () {
            return new MRContainer();
        }

        public MRContainer createValue () {
            return new MRContainer();
        }

        public synchronized boolean next ( MRContainer key, MRContainer value
) throws IOException {
            while (result == null || !result.hasNext()) {
                String s = parser.slice();
                if (s == null)
                    return false;
                result = parser.parse(s).iterator();
            };
            value.set((MRData)result.next());
            key.set(new MR_long(fsin.getPos()));
            return true;
        }

        public synchronized long getPos () throws IOException { return
fsin.getPos(); }

        public synchronized void close () throws IOException { fsin.close();
}

        public float getProgress () throws IOException {
            if (end == start)
                return 0.0f;
            else return Math.min(1.0f, (getPos() - start) / (float)(end -
start));
        }
    }


    @Override
    public RecordReader<MRContainer, MRContainer> getRecordReader(InputSplit
split, JobConf job, Reporter reporter) throws IOException {
        StormEvaluator.load_source_dir();  // load the parsed source
parameters from a file
        String path = ((FileSplit)split).getPath().toString();
        ParsedDataSource ds =
(ParsedDataSource)DataSource.get(path,Plan.conf);
        return new
ParsedRecordReader((FileSplit)split,job,ds.parser,(Trees)ds.args);
    }

}
```

StreamDataSource

```
 /*
 * Copyright 2016 The Apache Software Foundation.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.apache.mrql;

import java.io.Serializable;
import java.util.List;
import org.apache.hadoop.conf.Configuration;
import org.apache.storm.trident.Stream;

/**
 *
 * @author achyut
 */
public class StreamDataSource extends DataSource implements Serializable {
    Stream stream;

    public StreamDataSource(Stream stream) {
        super();
        this.stream = stream;
    }

    @Override
    public long size(Configuration conf) {
        return super.size(conf);
    }

    @Override
    public List<MRData> take(int num) {
        return super.take(num); //To change body of generated methods, choose
Tools | Templates.
    }

    @Override
    public MRData reduce(MRData zero, Function acc) {
        return super.reduce(zero, acc); //To change body of generated
methods, choose Tools | Templates.
    }




}
```

Chapter 7

Challenges Faced:
There were some challenges during the implementation of the project. Few of those are listed below.

- Distribution of the compiled jar file to all the distributed servers
- Integration with HDFS and getting data from it
- Understanding the code base and core module since it requires understanding the compiler theory, parser and DBMS implementation knowledge.

Future works
There are a lot of future works that can be done. Few of them are listed below:

- Implementation of functions other than map reduce so that more diverse queries could be run.
- Integration of more data source like web sockets.
- Integration of messaging queues like Kafka in the future.
- Integration of more data storage options like export to databases and files.
- Integration of more data parser formats to parse data types other than XML.

Improvements

- The project structure can be further improved and code quality can be improved with proper documentation

Conclusion:
The real time data streams are becoming increasingly popular day by day and the processing of the data from those streams are really valuable in least time. The development and integration of query language for this whole task will be helpful for the industry for perform analytics on these streams much easily and efficiently. It will allow a DBAs with SQL query knowledge to perform the analytics on large amount of data and can be more productive.

Appendix A

References

[1] https://en.wikipedia.org/wiki/Big_data

[2] http://www.economist.com/node/15557443

[3] https://www-01.ibm.com/software/data/bigdata/what-is-big-data.html

[4] https://www.datanami.com/2012/10/01/quantcast_opens_exabyte_ready_file_system/

[5] http://research.google.com/archive/mapreduce.html

[6] http://www.datasciencecentral.com/profiles/blogs/batch-vs-real-time-data-processing

[7] https://www.researchgate.net/publication/236263585_Big_data_processing_Big_challenges

[8] http://blog.cloudera.com/blog/2014/09/getting-started-with-big-data-architecture/

[9] http://mrql.incubator.apache.org/index.html

[10] http://wiki.apache.org/mrql/LanguageDescription#The_MRQL_Data_Model

[11] http://wiki.apache.org/mrql/LanguageDescription#The_Query_Syntax

[12] http://storm.apache.org/index.html

[13] http://storm.apache.org/releases/2.0.0-SNAPSHOT/Concepts.html

[14] http://storm.apache.org/releases/2.0.0-SNAPSHOT/Trident-API-Overview.html

[15] http://stackoverflow.com/questions/15520993/storm-vs-trident-when-not-to-use-trident