HASHTAG COUNT TOPOLOGY

TUPLE LOAD BALANCING SOLUTION

The un-even distribution of tuples using **Fields Grouping** is a problem when a particular word or few words are very high in frequency. This leads to severe load on those workers where the tuple containing the high frequency words are emitted for a bolt to process. This reduces parallelism and introduces latency.

The solution to this problem is to use **Shuffle Grouping** as a Stream Grouping technique. Shuffle Grouping distributes tuples in a uniform, random way acorss all tasks across all workers. An equal number of tuples will be processed by each task.

The following is a code snippet from the **HashTagCountTopology.java** file, which is the main class file where a topology is orchestrated and submitted to the storm cluster.

```
builder.setBolt(HASHTAG_COUNT_BOLT_ID, countBolt,
3).shuffleGrouping(TWEET_STREAM_SPOUT_ID);
```

The entire stream of tuples coming from the **TweetStreamSpout** gets uniformly distributed among all tasks of the **HashTagCountBolt**.

The integer **3** passed as argument to the above setBolt() method is a parallelism hint for the topology. It represents the number of tasks that should be assigned to execute the **HashTagCountBolt**. Each task will run on a thread in a process in workers on the cluster.

GUARANTEED MESSAGE PROCESSING (ATLEAST ONCE) USING RELIABILITY API

By default, tuple processing in Storm is not fault tolerant i.e. if a tuple fails to be processed completely throughout the Storm topology then such tuples go un-noticed. The storm engine does not know about them and does not re-process them and this situation leads to data loss i.e. data which did not got processed due to failure.

In use cases where fault tolerance is key and latency is not a big concern i.e. where we want to ensure guaranteed message / tuple processing i.e. atleast once tuple processing is introduced using the Storm Reliability API.

In order to design a topology with fault tolerance i.e using Reliability API, the below are the key points to remember:

- ✓ Inform Storm whenever we are creating a new link in the tree of tuples – (**Anchoring**)
- ✓ Inform Storm when we have finished processing an individual tuple – (**Ack or Fail**)

Storm can detect when the tree of tuples is fully processed and can ack or fail the spout tuple appropriately.

The following is a code snippet from the nextTuple() method implementation in the **TweetStreamSpout.java** file:

```
this.collector.emit(tupleValue, msgId);
```

While the tuple is emitted to be processed by a chain of Bolts, a msgId is associated with the tuple. The tuple tree will have msgId as an association of a unique id to uniquely identify the tuple.

```
private ConcurrentHashMap <UUID, Values> pendingMsgQueue;

msgId = UUID.randomUUID();

this.pendingMsgQueue.put(msgId, tupleValue);
```

A ConcurrentHashMap (which is thread-safe) is defined and initialised in the **open()** method of the Spout. Every time a tuple is received in the **nextTuple()** method, the tuple and the associated msgId is put in the hashmap to be later used.

The msgid is generated using the **randomUUID()** utility method of the **UUID** class.
Please refer to JavaDocs for more details on this method.

The **ack()** and **fail()** method are implemented in **TweetStreamSpout.java** file.

In the **ack()** method, the tuple added to the hashmap is removed as the **ack()** is called when the entire tuple tree processing for a tuple is successfull.

```
this.pendingMsgQueue.remove(msgID);
```

In the **fail()** method, the tuple for which processing is failed is replayed i.e. again emitted to the Bolt for processing. Calling the emit method again in the **fail()** method is a tuple processing replay technique which gaurantees atleast once processing.

```
this.collector.emit(this.pendingMsgQueue.get(msgID), msgID);
```

In all the Bolts i.e. user-defined Bolt classes, the **ack()** method is called within a try block in the **execute()** method at the end of data processing to notify to the storm Acker bolts that the tuple is successfully processed in that particular bolt.

```
this.collector.ack(tuple);
```

The **fail()** method is called incase of **02** scenarios:

- ✓ If a message time-out occurs i.e. a tuple does not get processed within the time-out interval.

  The default time-out interval is 30 seconds which is set in the

  Config.TOPOLOGY_MESSAGE_TIMEOUT_SECS

  The **fail()** is called automatically by the Storm framework.

- ✓ If an error occurs in processing the tuple. It is a good practice to call this method in the catch block of the **execute()** method.

```
            this.collector.fail(tuple);
```

EXECUTE STORM TOPOLOGY IN LOCAL MODE

Go to the <**APACHE_STORM_HOME**> and execute the following command:

➔ bin/storm jar /root/apache-storm-1.2.1/Data/HashTagCountTopology-0.0.1-SNAPSHOT.jar HashTagCountTopology

EXECUTE STORM TOPOLOGY IN PRODUCTION MODE ON STORM CLUSTER

Go to the <**APACHE_STORM_HOME**> and execute the following command to start Nimbus

➔ bin/storm nimbus

Go to the <**APACHE_STORM_HOME**> and execute the following command to start Supervisor

➔ bin/storm supervisor

Go to the <**APACHE_STORM_HOME**> and execute the following command to start Supervisor

➔ bin/storm jar /root/apache-storm-1.2.1/Data/HashTagCountTopology-0.0.1-SNAPSHOT.jar HashTagCountTopology "HashTagCountTopology"

Go to the <**APACHE_STORM_HOME**> and execute the following command to start Storm UI

➔ bin/storm ui

Note:

- ➔ On the EC2 instance, the Cloudera Hadoop Cluster should be installed and started.
- ➔ The ZooKeeper service should be running on the CDH cluster.
- ➔ The Apache Storm should be installed and configured.
- ➔ Execute all the above commands on seperate terminals on the EC2 instance.
- ➔ The <**APACHE_STORM_HOME**> is the folder where the Apache Storm is installed.
- ➔ In this case, /root/apache-storm-1.2.1 is the location of Apache Storm Home.
- ➔ **JAR file Name:** HashTagCountTopology-0.0.1-SNAPSHOT.jar
- ➔ **Main Class Name:** HashTagCountTopology
- ➔ **Topology Name (In Production Mode):** HashTagCountTopology

MYSQL DATABASE DETAILS

Database Name: STORM

Table Name: HASHTAGCOUNTS

DB User Name: root

DB Password: password

Invoke the MySQL command line tool with root credentials and execute the following commands:

- ✓ CREATE DATABASE STORM;
- ✓ USE STORM;
- ✓ CREATE TABLE HASHTAGCOUNTS (hashtag varchar(1024), count bigint(20));