Processing RealTime & Streaming Data Using Apache Storm

Assignment Solution Document

Venkatesh Jagannathan

I - Solution

Solution to Load Imbalance Problem

The load imbalance in original solution is caused due to the use of fieldGrouping type of StreamGrouping. This grouping type ensures that a tupe with given value for a field always goes to same instance of bolt. Due to this condition, difference in frequency of words will have unequal distribution. Following was the distribution on trial run: -

Field Grouping

Word Count Bolt Instance 1	2508045
Word Count Bolt Instance 2	3348451
Word Count Bolt Instance 3	5375582
Word Count Bolt Instance 4	5323987

Captured by generating unique instance id and saving tuple count to database on cleanup

To avoid this problem Apache Storm provides another Stream Grouping type called **ShuffleGrouping**. This grouping ensures equal distribution of load across all bolt instances. Following was the distribution on a trial run with this type of grouping: -

Word Count Bolt Instance 1	5008000
Word Count Bolt Instance 2	4820446
Word Count Bolt Instance 3	5528830
Word Count Bolt Instance 4	5228790

Captured by generating unique instance id and saving tuple count to database on cleanup

Above table shows even distribution across tuples.

Following code is used in the solution to implement this grouping:-

```
builder.setBolt(SPLIT_BOLT_ID, splitBolt,3)
            .shuffleGrouping(SENTENCE_SPOUT_ID);
builder.setBolt(COUNT_BOLT_ID, countBolt,4)
            .shuffleGrouping(SPLIT_BOLT_ID);
```

Steps followed to ensure atleast once processing of input tuples

In order to ensure atleast once processing of input tuples, following measures are taken: -

1. Implementation of Reliability API
   a) **Anchoring**: It must be possible for a Tuple tree to be traced from root to child tuples generated from them. This requires copying messageId from parent tuple to child tuple. When we pass input tuple as argument to collector.emit tuple, the messageId gets copied over automatically to generated tuple by storm. This implemented.
   b) **Acknowledgement**: Once tuple is processed, collector.ack is invoked to acknowledge tuple processing to acker bolt. Using above acker bolts are able to aggregate acknowledgement from all bolts instance and confirm back to spout that the tuple is processed successfully. If on some condition the tuple processing needs to be failed collector.fail can be called (indicated in commented code) to indicate the parent tuple processing failure back to spout.
2. Retry of Input tuple processing on Failure
   a) **Retry Logic**: The spout generate a unique message id for each source tuple & assign to tuple for anchoring downstream. In the implementation, Spout maintains a list of messageId for each input value in a hashmap. Upon an input tuple failure from acker bolt, the fail method in spout gets invoked passing the messageId. This messageId is looked up back in the hashmap to retrive the value and the tuple is re-emitted for processing.

Bolt

```
this.collector.emit(tuple,new Values(word, count)); //anchoring
this.collector.ack(tuple);//acknowledge
//this.collector.fail(tuple); if failure needs to be indicated upon some condition
```

Spout

```
//Generate unique message id for each tuple tree root
String messageId = java.util.UUID.randomUUID().toString();
queue.put(messageId, sentences[index]); //track tuples in queue

public void fail(Object msgId) {
        this.collector.emit(new Values(queue.get(msgId))); //retry in case of failure
   }
```

This way the failed tuples are tracked back to source and the source is retried ensuring its successfully processed at least once.

II – Commands

Command to Run Jars in Local Mode

```
bin/storm jar ~/storm/WordCount-0.0.1-SNAPSHOT.jar WordCountTopology
```

Command to Run Jars in Production Mode

```
bin/storm jar ~/storm/WordCount-0.0.1-SNAPSHOT.jar WordCountTopology "WordCountTopology"
```

Name of MySQL Database

```
WCDB
```

TableName used in code

```
wordcount
```

Steps to Create Database and Table

1. Run Following scripts
   a) Create Database

```
CREATE DATABASE WCDB;
```

   b) Create Table

```
USE WCDB;
SET autocommit=1;
CREATE TABLE wordcount (
word nvarchar(255) unique,  #no primary key for fast updates
   counts bigint default 0
);
```

2. Code uses useSSL=false. If database enforces it, should disable it or remove it

III – Results

Snapshot of Database Table

```
mysql> describe wordcount;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| word   | varchar(255) | YES  | UNI | NULL    |       |
| counts | bigint(20)  | YES  |     | 0       |       |
+--------+-------------+------+-----+---------+-------+
2 rows in set (0.01 sec)

mysql> select * from wordcount;
+-----------+--------+
| word      | counts |
+-----------+--------+
| a         |  12709 |
| ate       |  12534 |
| beverages |  12693 |
| cold      |  12578 |
| cow       |  12612 |
| dog       |  25270 |
| dont      |  25289 |
| fleas     |  25142 |
| has       |  13392 |
| have      |  13324 |
| homework  |  13203 |
| i         |  40144 |
| like      |  26927 |
| man       |  13541 |
| my        |  25203 |
| the       |  12556 |
| think     |  12446 |
+-----------+--------+
17 rows in set (0.00 sec)

mysql>
```

Storm UI Snapshots

Storm UI

Search WordCountTopology-6-1555304434: Search Search Archived Logs:

Topology summary

Name	Id	Owner	Status	Uptime	Num workers	Num executors	Num tasks	Replication count	Assigned Mem (MB)	Scheduler Info
WordCountTopology	WordCountTopology-6-1555304434	ec2-user	ACTIVE	41s	4	12	12	1	3328	

Spouts (All time)

Search:

Id	Executors	Tasks	Emitted	Transferred	Complete latency (ms)	Acked	Failed	Error Host	Error Port	Last error	Error Time
SentenceSpout	1	1	6560	6560	1666.125	160	0				

Showing 1 to 1 of 1 entries

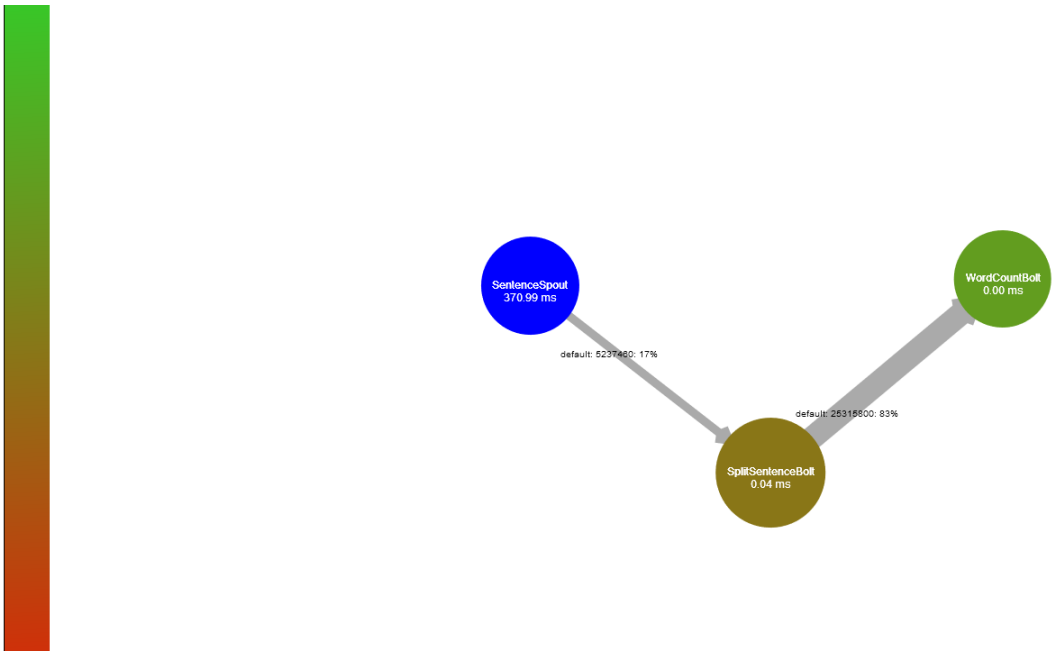
Bolts (All time)

Search:

Id	Executors	Tasks	Emitted	Transferred	Capacity (last 10m)	Execute latency (ms)	Executed	Process latency (ms)	Acked	Failed	Error Host	Error Port	Last error	Error Time
SplitSentenceBolt	3	3	18760	18760	0.368	3.656	3840	2.602	18560	0				
WordCountBolt	4	4	0	0	0.130	0.224	15480	0.218	15440	0				

Showing 1 to 2 of 2 entries

Topology Visualization (upon becoming stable)



IV - Confirmation

1. No bottleneck errors on bolts seen
2. RichBaseBolt Class as per requirement
3. Complete Latency is non zero indicating successful processing
4. Non NAN or Red Bolts on topology visualization
5. The relative distribution of word count matches expected (‘dog’ is roughly double count of ‘a’, ‘i’ roughly 3 times of ‘a’, etc.