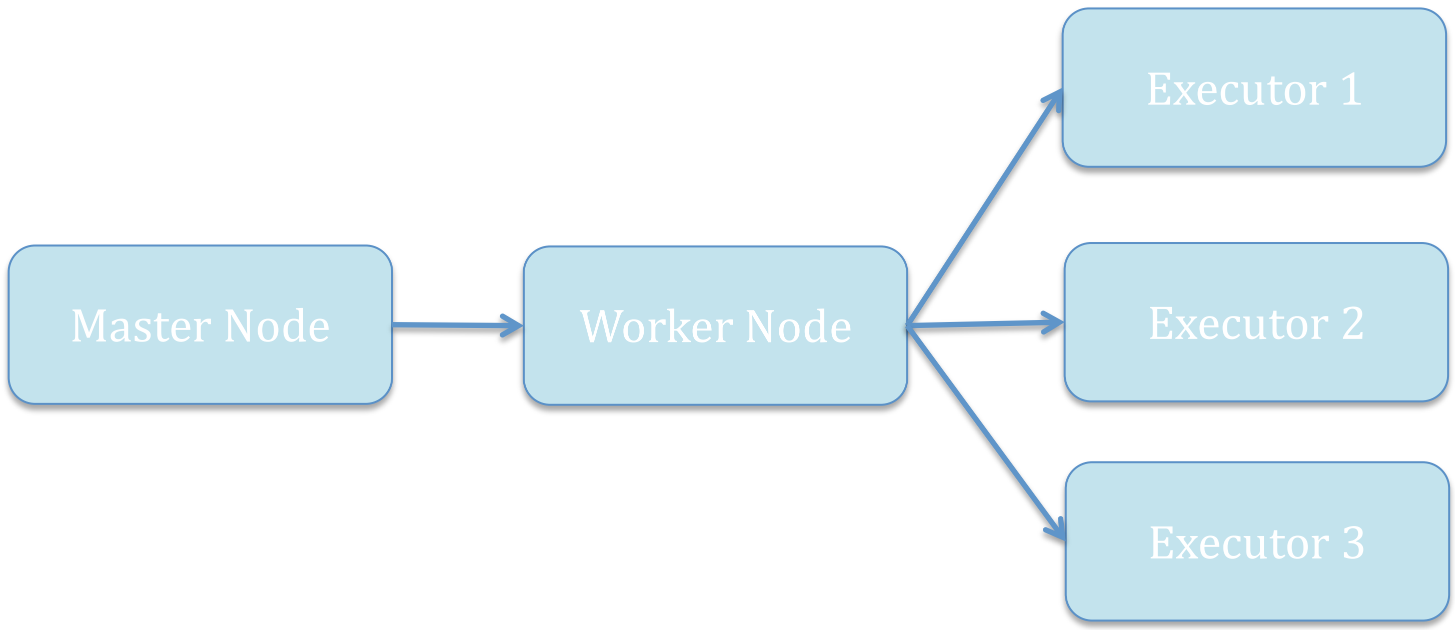
Distributed Stream Processing System

MuthuKutti Raja

CS658 – Spring 2016
Project Report

System's Architecture:

The major components of the system are the master node and the worker nodes. The worker node can spawn multiple executors based on the demand of the topology. On the high level the architecture looks like the following figure.



Worker Nodes:

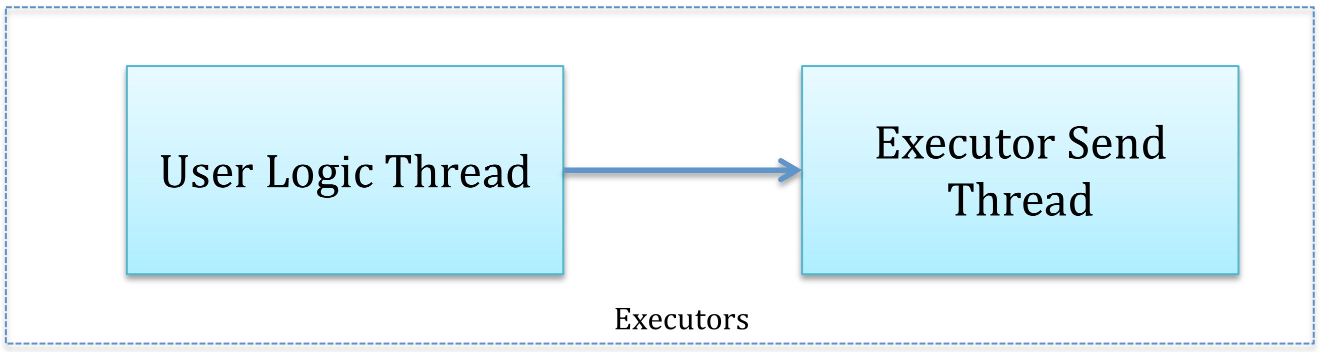
Each worker node will run two threads i.e.) Worker receive thread & Worker Sender Thread. Worker receive thread runs on a separate port. It analyzes the tuple's destination task identifier and place in the In-Queue of the task that is associated with the executors. Worker send thread places the tuple in the global out queue and send the tuples to the next downstream.

Executors:

It runs two threads

User Logic thread – Runs the user logic i.e) analysis part written with in the code of the relavant tasks

Executor Send thread – After generating the tuples from the user logic thread, this thread sends the tuple to the global out queue.



Results:

Experiment Setup 1:

Spout: Continuous stream of empty packets
Bolts: 3 Processing bolts, where each processing time is T(milliseconds).
There are two tables which have the values populated based on different combinations of executors of bolts.

Table 1:

The total number of tuples transferred in the network in a minute for different combinations of bolts executor is populated. Whenever a tuple is transferred, the message responsible is, the message request to nimbus, message response from nimbus and the message sent to downstream. i.e) 3 messages involved for each tuple transfer.

Bolt 1 (No of Executors)	**Bolt 2 (No of Executors)**	**Bolt 3 (No of Executors)**	**T (millisecond)**	**Tuple Count (Per Minute) (1000's)**	**Total messages (1000's)**
1	1	1	0	156	418
3	3	1	0	240	720
10	10	2	0	365	1095
3	3	1	2	185	555
5	5	5	5	167	498
10	10	10	10	165	495
20	20	20	0	520	1560

Observation:
With executors of all bolts tasks set to 1, the number of tuples that has flown within the network per minute is 156000.
As we parallelize the bolts i.e.) more number of executors running the instances, the number of tuples processed per minute will be more. From the results we can infer the above statement. With 3 bolts running 10 executors each, the number of tuples processed is 365,000 tuples. And with 3 bolts running 20 executors each 520,000 tuples are generated. But we have to be careful in picking the number of executors, because as we can see the total number of messages that's created for the bolts with 20 executors, it had 1.5 million message exchange within the network i.e) 500,000 requests alone served by nimbus, which is a overkill.
When the processing time for each bolt increases, the number of tuples processed per minute is also reduced with a big margin. With 3 bolts running 10 exceutors each with processing time 0 produced 365,000 tuples within the network, but with the same combination except the processing time which is 10 milliseconds per request produced 165,000 tuples per minute.

Table 2:
Complete processing time of a tuple is the time taken to complete its cycle i.e.) Time where the spout generated a tuple until the time it has reached the last bolt. The Average processing time of all generated tuples are computed and the values are populated in the above table.

Bolt 1 (No of Executors)	**Bolt 2 (No of Executors)**	**Bolt 3 (No of Executors)**	**T (millisecond)**	**Average Processing time of tuple (millisecond)**
1	1	1	0	4
3	3	1	0	2
10	10	2	0	1.78
3	3	1	2	8.3
5	5	5	5	16.9
10	10	10	10	31.6
20	20	20	0	1.67

Observation:

With executors of all bolts tasks set to 1, the average processing time of tuple within the experiment is 4 millisecond

As we parallelize the bolts i.e.) more number of executors running the instances, the number of tuples processed per minute will be more. From the results we can infer the above statement. With 3 bolts running 10 executors each, the average processing time is 1.78 milliseconds. And with 3 bolts running 20 executors each, the average processing time 1.68 millisecond.

When the processing time for each bolt increases, the average processing time of tuple to complete also increases. With 3 bolts running 10 executors each with processing time 0 had an average 1.78 milliseconds for each tuple, and with the same combination except, where the processing time is 10 milliseconds, the average processing time is 31.6 milliseconds. This is because, since each of the task's processing time is 10 millisecond and each tuple needs to complete 3 tasks with the experiment i.e.) 3*10 => 30 milliseconds are added for the processing time alone.

Experiment Setup 2:
Spout: Continuous stream of lines read from a file.
Bolts:

Word Splitter Bolt – Splits the sentence into words and sends to next bolt task.
Word Merger Bolt – Generate the count for the words generated from the stream

Table:

Overall completion time to process all tuples in the network is populated with different executors count of word splitter bolt.

Word Splitter Bolt (No of Executors)	**Word Merger Bolt (No of Executors)**	**File Size (MB)**	**Overall Completion Time (Sec)**
1	1	0.17	32
5	1	0.17	28
10	1	0.17	22
10	1	1.68	232
10	1	6.89	878

Observation:

With executors of word splitter bolt tasks set to 1 & the file size being 176 kb, the overall completion time is 32 seconds.

As we parallelize the bolts i.e.) more number of executors running the instances, the overall completion time of the analysis will be lesser. From the results we can infer the above statement. With Word splitter bolt running 5 executors each, the completion time is 28 seconds for the file of same size. And with Word splitter bolt running 5 executors each, the completion time is 22 seconds for the file of same size.

Conclusion:

From the experiments we have seen above, we can conclude that as the executors are parallelized

- Number of tuples processed in the network will be more.
- Completion time of a analysis will be reduced.
- Also the number of messages needs to be communicated will be more, which is a downside of this streaming system