Machine Programming 4 **Chen Zhu**
CS425 Distributed System Networking **Weiran Lin**
Prof. Indranil Gupta Date: Dec. 2 2018

1 Objectives

- Build a fault tolerant stream processing system named Crane. Implemented three application on both Crane and Spark Streaming and compare the performance between two systems.

2 System Design

2.1 Topology

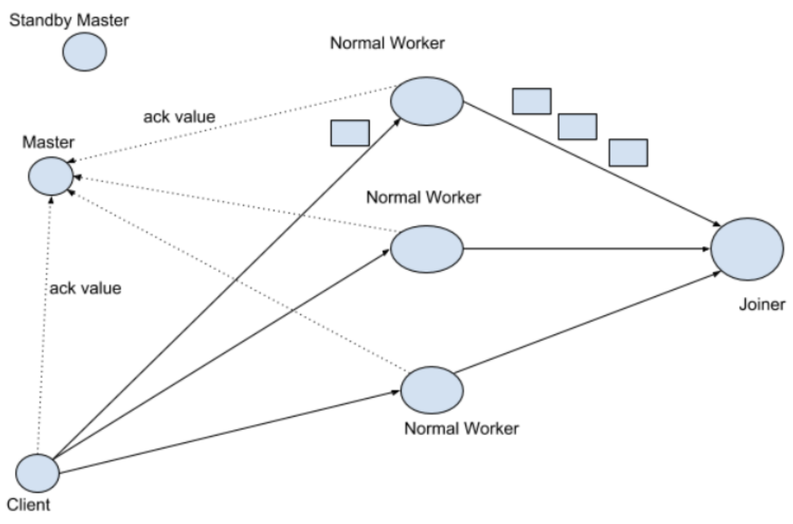

Fig. 1A. Topology of Crane.

Our Spout emits a tuple per 100 millisecond, and we can set the input rate in command line.

We dynamically assign roles for every machine when application is started/restarted and failure is detected. The first alive machine is client and spout which will read data from files and generate a data stream. The second alive machine is master, it is responsible for submitting jobs and also tracking whether a tuple (a line in stream data) is successfully written to the result. The third alive machine is standby master. The last alive machine is worker which joins all results (a.k.a. Joiner). Other alive machines are worker which transform or filter tuple and then output to Joiner.

2.2 Data Structure Design

- **Map in Master**: Master maintains a map. The key is the ID of a tuple, and the value is an integer which called ack value. If the ack value is 0, it means the tuple is successfully processed and the result is written to the map in Joiner.
- **Map in Spount**: Spout maintains a map. The key is the Id of a tuple, and the value is the content in the tuple.
- **Cache in Joiner**: Joiner maintains a cache in joiner which stores all intermediate results. When it receives an acknowledgement from Master, it would write all results into the final result.

2.2.1 Fault Tolerance

We promise fault tolerance by using ack value.

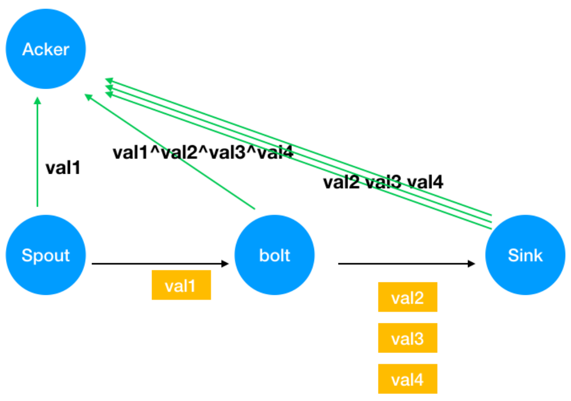

Fig. 1B. Topology of Crane.

Figure 1: In order to make sure that every tuple is completed processes, we used a subtle ack value algorighm to implement the fault tolerance. every message come from the spout will carry a message Id and a random ack value. 1. When Spout emits a tuple, it will also send its ack val (val1 in the figure) to Master and next machine. 2. When worker receives a tuple, it generates n random ack values, where n is number of output tuples. After that, the bolt will xor them together to get the new Ack value, and send the ack value to master. Finally the master will xor all the ack value according to message Id, and send "messageSuccess" to the master and "messageCommit" to the sink node. In the above example, if there are no fault, the acker will calculate $val1^val1^val2^val3^val4^val2^val3^val4 = 0$.

Spout will store the ID (line) and the content into local map. When it receives acknowledgement message from Master, it deletes the corresponding message from the map. After sending all tuples out, it checks the map and re-sends all items again.

When Joiner receives the commit message, it will write intermediate results into final result.

When failure detector detects a failure, it re-assigns new roles to all alive machines in membership list.

- **Normal workers fail**: The normal workers fails to send ack value to Master. Master marks the message processed by the worker as failure, and sends failure message to Spout and Joiner. Hence, the Spout will re-send the message, and Joiner will remove intermediate results.
- **Joiner or Spout fails**: Since all intermediate results are lost, we have to re-start application.
- **Master fails**: Original master losts all ack values. However, since standby master will take over the task, we do not have any operation.

3 Three Applications

3.1 Word Count

We use work of William Shakespeare as data source. We find the number of word and output the top five most frequent words. This application contains transform and join operation. The size of file is 43.7 MB.

3.2 HTTP Hottest Resources

We use a HTTP response log as data source. We find the top five most frequent resources that is gotten with status code 200. This application contains filter, transform and join operation. The size of file is 35.7 MB.

3.3 Twitter Most Popular User

We use a twitter following records as data source. We find top 5 most popular users. The data forms a directed graph. This application contains transform and join operation. The size of the file is 39.4 MB.

4 Performance Analysis

We are going to compare spark-streaming with Crane stream processing system with respect to total latency (data processing time) against data flow rate.

4.1 Wordcount

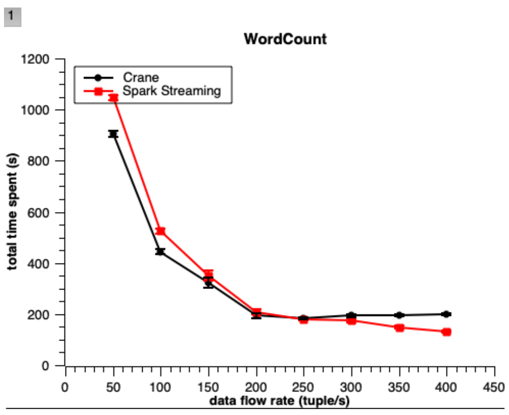

Fig. 2A. word count app without fault

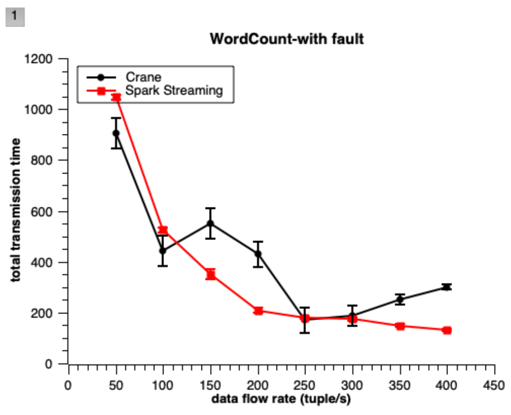

Fig. 2B. word count app with fault

4.2 Twitter hot user app

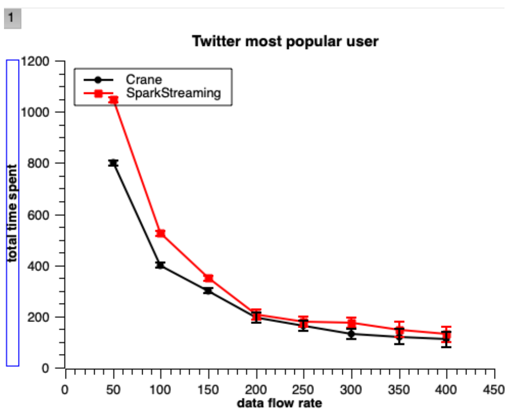

Fig. 2C. twitter hot user app

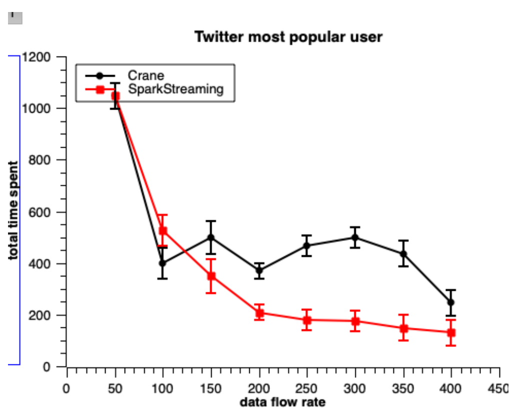

Fig. 2D. twitter hot user app with fault

4.3 Http hot resource app

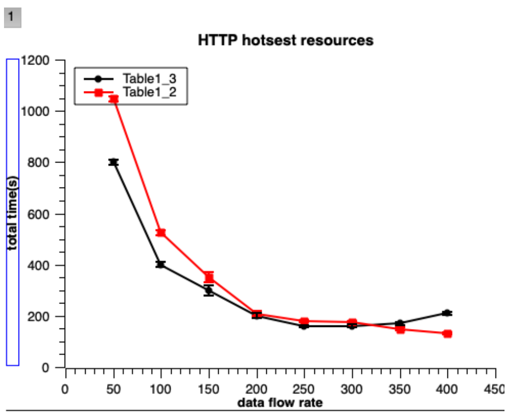

Fig. 2E. Http Hot resource app

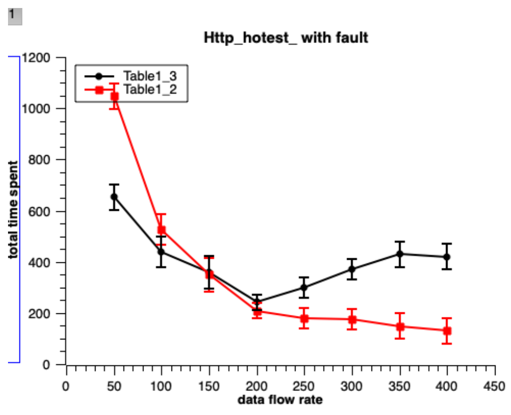

Fig. 2F. Http Hot resource app with fault

5 Conclusion

From the above figure we can see that Crane often performs better when the data rate is small. This is because it usually require less time to start compared to spark streaming. Also, when the data flow rate goes up, the time taken for Crane increased. This is caused by the data drop happens when the data flow rate is higher than the bolt processing rate.