CS425 MP4 Report

Group 43: Siti Zhang (sitiz2) & Mingrui Yin (mingrui4)

Design:

The architecture of our distributed stream processing system consists of a master and several worker nodes. The master is responsible for receiving jobs from clients, creating topology and communicating the worker nodes to assign tasks on a random order. And it is automatically selected as the spout for the input stream. The master maintains the membership list and it has knowledge of all the worker nodes and the tasks that they are handling. The worker node will listening to the role assignment to work as the filter bolt or join bolt. They are responsible for receiving the input tuples, conducting the corresponding task and finally sending the output to the sink bolt. The final result could return to the master node or save as a file in our distributed file system developed in MP3.

The failure detector based on the SWIM style we built in MP2. Each machine has a sorted and full membership list which is used to monitor the next three machines in the system ring for failure detector by using PING->ACK messages.Once one worker fails, the master will stop the thread of input streams, send stop message to every bolts, re-assign the whole job to other live worker nodes and restart every task. As for the bolt, once it receive the stop message, it will clear the queue of input and output, and wait re-assign. Considering master failure, when one worker node notices the failure of master, it will send the membership list and the information of corresponding tasks to the a hot standby master. Then the hot standby master will take charge the responsibilities of the old master and restart the whole tasks to guarantee the same end result of processing data.

Because TCP is too slow in python, we used UDP for the communication from spout to bolt or bolt to bolt and adopted the ack mechanism for accurate data transferring. Every time a tuple is sent from spout to bolt or bolt to bolt, the receiver will also send an ack message to the sender for each tuple to ensure no tuple is lost. Also, we use multi-thread and buffer to make sure the system will not broken down. In spout, we read the input streams by lines, put each line into a queue and use each line as a stream. In filter bolt, we use two thread, one is to receive the data from spout and another is to filter the data and send it to join bolt. To send data in two thread, we also use queue as

a buffer. In join bolt, we receive the data and aggregate it, then send it to the master to show the answer.

The whole system is designed by Python.

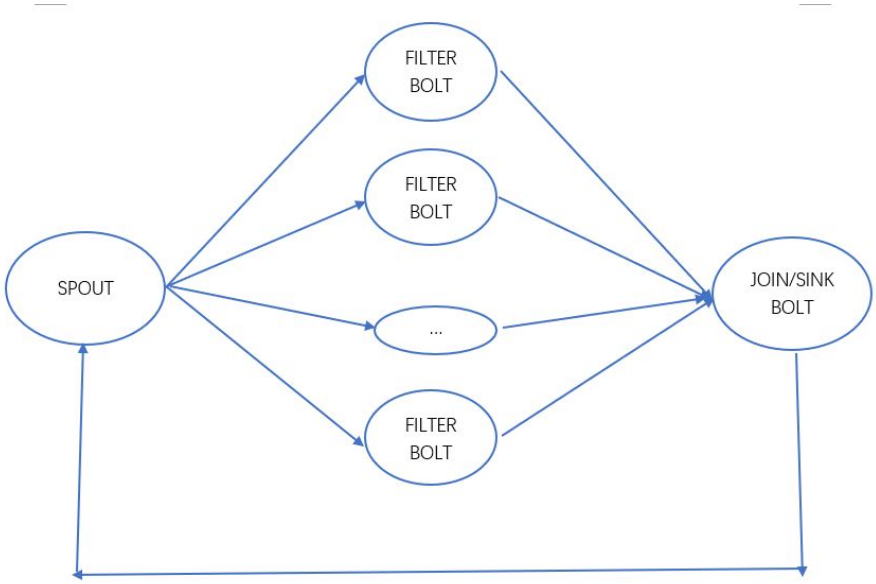
Applications:

The first application is to find the number of specified word: “Google”, “Twitter”, “Amazon”, “Test”, “Apple” and “Facebook” .

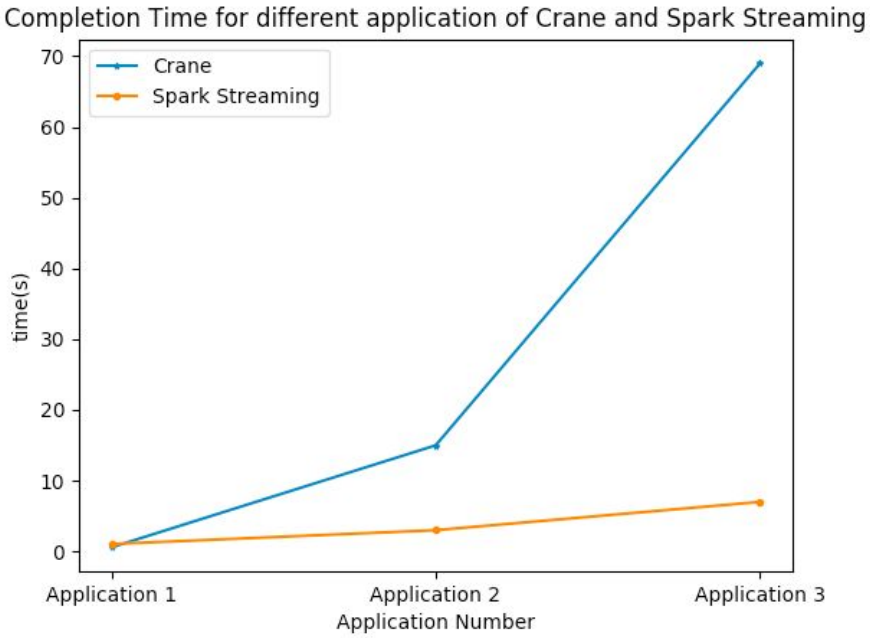
The second application is count the number of every word appears in one file.

The third application is filter the words twice, the first filter is to count number of every word appears in one file, then the second filter is to choose specific word and count the number of it. The topology is : SPOUT->FILTER BOLT->FILTER BOLT->JOIN BOLT

The first and the second application have same topology shows above:



During the test, the file size in each application is: several lines, 5,000 lines about 1M and 20,000 lines about 5M. Run these application in both Crane and Spark Streaming, the time to complete the application shows in the plot:



In this plot, we can clear see that in application 1, the Crane have better performance than Spark but in application 2 and 3, the Spark performs better. The most time-consuming work in Crane is to read and send stream data. Relatively, in Spark Streaming, creating the topology will cost more time than Crane. Thus, in application 1, when the input stream is really small, crane will cost little time to finish it. However, when the input stream becomes larger, the stream that sending in spout in Crane will cause a lot of time which make the whole system take much time.