CS425, Distributed Systems: Fall 2018
Machine Programming 4 – Crane
(Lavisha Aggarwal (lavisha2), Vandana Kulkarni (vandana2))

Architecture and Programming framework

In this MP, we implement Crane which similar to Storm is a stream processing system and works on data flowing in the form of streams of tuples. The tasks are divided among spout, bolts and sink. The data is read and streamed from the spout, which in our case is the leader of the group whereas the introducer acts as the sink combining the results from all bolts. The rest of the VM's act as bolts in our design. The spout considers all the alive nodes as bolts and sends them tuples continously. We make use of the distributed file system we implemented in MP3 to store the input data file, intermediate bolt and sink results, task information and log files. We also make use of the get, put, etc functionalities of the distributed file system. Additionally, we use MP2's distributed membership protocol to keep track of new, alive, failed, leaving nodes.

FAILURE HANDLING

In case of leader failure, leader election comes into play, similar to MP3. When the stream processing tasks are being done, we store task information into log files. Hence, during leader failure, the newly elected leader goes through the log to find out if some task needs to be finished identifying which parts of the job have been completed. It reschedules the remaining parts of the job on alive nodes acting as bolts.

DESIGN OF OUR STREAM PROCESSING SYSTEM

We use a tree (DAG) topology for our applications. The spout is given the data source as its input. The spout reads tuples, either one at a time or a few at a time from this data source. This adjustable number can be specified by the programmer. Each bolt gets these fixed number of tuples and perform either of the 3 operations (transform, filter or join). Each bolt saves its output in the sdfs and either forwards it to the next bolt or to the sink. The sink processes the data it receives from the bolts and computes the result. Since, the data is flowing as streams of tuples, the sink also produces multiple outputs and the last result produced by the sink is the output for the given data source collectively. The sink results are forwarded to the client hence, as a user submits her job she can see the result of tuples flowing in the system in real time. We have implemented 3 kinds of bolts: 1. FILTER - These bolts filter tuples based on some user specified criterion. These can be checking for the value in the given tuple and checking if it has to be blocked or passed depending on user's specification. This can be specified by the user in a spec file which the system reads during run time. Hence, the output number of tuples is lesser than than incoming number of tuples for such bolts.2. JOIN - These bolts join the incoming tuples, with a static database hence making the number of output tuples greater than the incoming tuples. Similar to Filter, the details for which database, database information and which column of the tuple to join with can be specified by the user in spec files.3. TRANSFORM - These bolts are responsible for the remaining kind of tasks, like performing arithmetic operations, etc.

APPLICATION 1

Our first application is a word count application which determines the count of each word in the given data source. Hence the output is in the form of tuples of words and their counts in the

given data source. We only use Transform Bolts in this application whose output is the number of times different words appear in the input tuples. The sink, takes this count for different words from all bolts and combines them to yield the combined output. This is done for the incoming stream of data and then final result is displayed to the user and stored in the sdfs as the word count for the given data source.

For application 2 and 3, we work on a reddit log file which contains the information pertaining to the id, unixtime, title, number of votes, category, score, number of comments and username for each entry (or tuple or row).

APPLICATION 2

In this application, we read in tuples as rows from the reddit dataset and find the total number of comments and the score corresponding to a select number of categories which the user can specify as filter operation. Hence, we have two stages of processing through filter bolt followed by transform bolt. The filter bolt, filters out tuples which do not belong to the list of categories specified by the user. These filtered tuples are received by the transform bolt which sums the number of comments and the score for these incoming tuples. Finally, the sink combines the resulting score and comment values from all bolts and produces the combined result. As data comes in the sink determines the count from the starting till that time and the sink result at the last time step is the combined output.

APPLICATION 3

In this application, we ask our users to find out what categories of reddit posts are they interested in reading. We save this information in a static database and run a stream processing job which outputs the number of reddit posts the users would be shown/will be reading depending on the categories they like. Hence, we perform this through a 2 stage : 2 bolts architecture. The database has 2 columns: User, Category. The first bolt is a Join bolt which performs a join of each row in the database with the incoming tuples based on the category. The second bolt is a Transform bolt which counts the number of tuples for each user in the incoming stream it receives from the Join bolt.

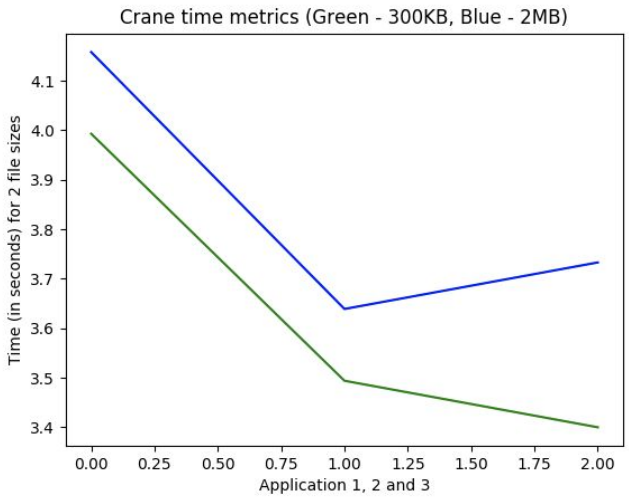
COMPARISON WITH SPARK STREAMING

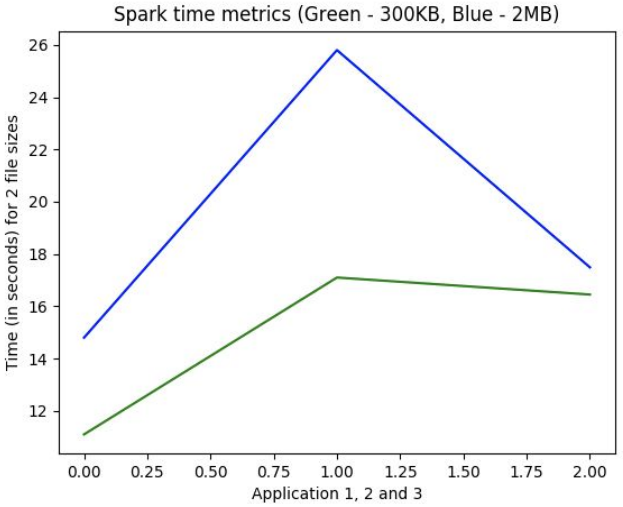
We run all three applications on both Crane and Spark and compare the results of the computation on both. We run the 3 tasks on 2 file sizes of 200 KB’s and 3 MB’s and compare the time and bandwidth between the two.

TIME REQUIREMENTS

We observe that Crane is significantly faster than Spark streaming for all three applications. The possible reasons for this might be the stream processing or a further optimization of it through a group of tuples. More specifically, the time requirements for Crane for applications involving filter are lower than those involving a join as expected. On the other hand due to the batch processing nature of spark, it is the reverse in case of Spark. This also depends on the setup and boot time required by spark as installed the publically available version which comes with a wide range of

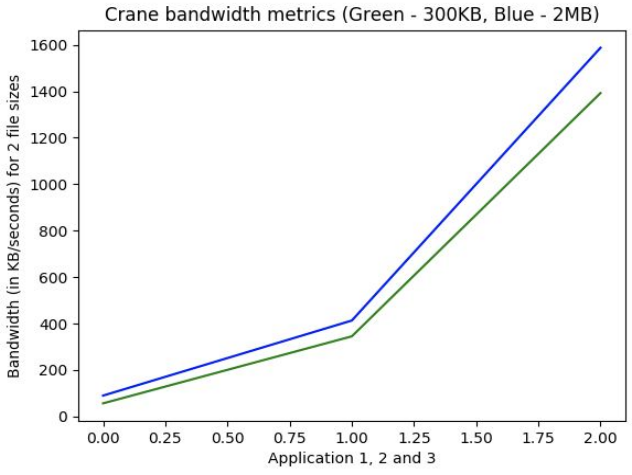
functionalities as opposed to our Crane designed which is just composed of the barebones for stream processing. Overall, this reflects that the design of crane is efficient. As expected the requirements for the smaller file size are lower than the larger file size.

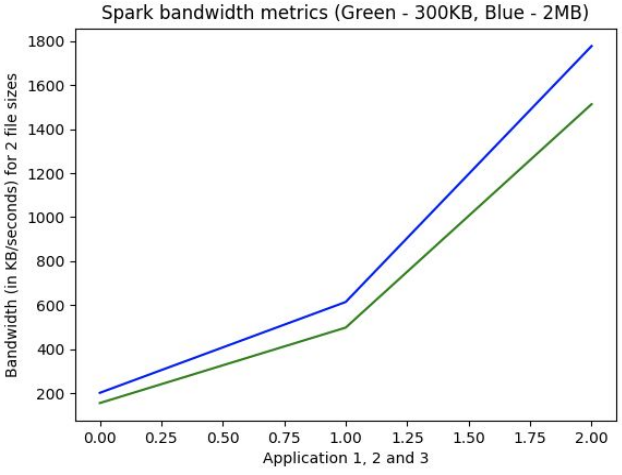




BANDWIDTH REQUIREMENTS

The bandwidth requirements for both Crane and Spark streaming are similar though spark takes in slightly more bandwidth. Again, note that since the last application makes use of joins, hence it has the highest bandwidth and the bandwidth for the larger file size is greater than the smaller file.





Hence, crane is designed efficiently to implement three kinds of bolts - filter, join and transform and uses a DAG topology using spouts, bolts and sinks. We have implemented 3 applications of word count(transform), finding a running sum/average of filtered data (filter and transform), predicting user database join results with reddit posts (join and transform).