Report

CS425-MP4-Group 9

System Design / Framework

The Crane system consists of a Nimbus and a Worker for each node in the system. User interface wise, like Storm, Crane also has Spout, Bolt and Topology. Instead of Tuple as in Storm, Crane uses Map<String,String> as the standard data structure for passing input/output values between Spouts and Bolts. A crane topology is a tree and not a DAG. Bolts can have multiple tasks being executed in parallel. An important difference of Crane from Storm is that for each tuple input to a Bolt, no more than one tuple can be emitted. We think that this does not add any additional constraints on expressibility (especially because of our map data structure). But, this gives us huge benefits in the ack-ing logic for fault-tolerance and makes Crane faster than Storm.

Topology class, Bolt interface and Spout interface are all Serializable which makes it possible for us to pass these objects across different machines. Implementations of Bolt and Spout can contain any class member, but they all should be serializable. Non-serializable members should be transient. There is a topology builder class which helps user to build a topology like in Storm.

The Nimbus is responsible for getting the topology created by user (CraneSubmitter class acts as a client communicating the topology to nimbus). It is also responsible for breaking down the topology description into a set of tasks which host the spout or bolt logic inside them and then distribute these tasks between different worker nodes. The distribution of tasks is on a purely random basis. If a worker node fails, the Nimbus redistributes the task to other nodes in the system. Again the new workers for the failed tasks are chosen in a random basis. The node failure is checked by using SWIM failure detector which we built as part of MP2. We assume that the Nimbus is always available and does not fail. The nimbus maintains a map of taskIDs which are present in various workers. It also maintains a map of taskID to (hostname,port) of the corresponding task.
Each node in the Crane system has a Worker process running. The Worker process launches Worker Threads on receiving task launch request from Nimbus. A worker thread handles reception of input tuples, uses Spout/Bolt logic to process the input tuple and sends the output tuple to downstream bolts. The Worker process also maintains a mapping to taskID to (hostname,port) for all tasks across the system. Any updates to this mapping is notified by the Nimbus to the Worker. Each Worker Thread receives the destination hostname and port for output tuples from its host worker process.

For fault-tolerance, like Storm, Crane also guarantees at least once processing of each tuple. Which means that no tuple is lost, but multiple processing of a tuple is possible. In Crane, the Spout Thread maintains a buffer of emitted tuples which time-out waiting for acks. On time-out, the tuples are considered failed and hence are re-emitted. For each emitted tuple, the number of acks expected is the number of leaf bolts in the topology (which has a tree structure). Each leaf bolt sends ack of weight 1 to Spout Thread. If a tuple is filtered in an internal bolt, the internal bolt sends ack of weight as the number leaf bolts in its subtree.

Apart from more efficient ack-ing mechanism, we use UDP sockets for tuple passing between spouts/bolts and for ack-ing. These two are the reasons for speedup of Crane over Storm.

Applications

APPLICATION 1:
Converting Document to Bag of Words representation and removing stop words

In this application we take a huge file having a document in each line and then convert that tuple into bag of words representation with each tuple containing

Document ID, Word1:Count, Word2:Count etc…. (with Stop Words removed)

In this MP we used dataset from
"http://snap.stanford.edu/data/bigdata/memetracker9/quotes_2008-08.txt.gz"
Containing news articles from various news articles. Having approx. **18 Million articles**. This application is very useful in many application of text analytics and information retrieval.

APPLICATION 2:
Extract potential inappropriate posts from stream of new posts in Reddit

In this application we take a huge file containing posts from any social media website (in our case Reddit) and we then filter out all potential trouble making posts (inappropriate, explicit, hateful, etc) based on 1) Total number of votes 2) Number of Comments 3) Number of Downvotes. The filtering criteria is based on posts having at-least certain number of votes and comments and having a high ratio of down votes.

In this MP we used dataset from
"http://snap.stanford.edu/data/web-Reddit.html"
Dataset is a collection of **132,308 reddit.com submissions**. For each submission, it contains features such as the number of ratings (positive/negative), the submission title, and the number of comments it received, etc. The application is very useful for many social media companies like Facebook, Reddit etc. Saving time for them to find flagged posts.

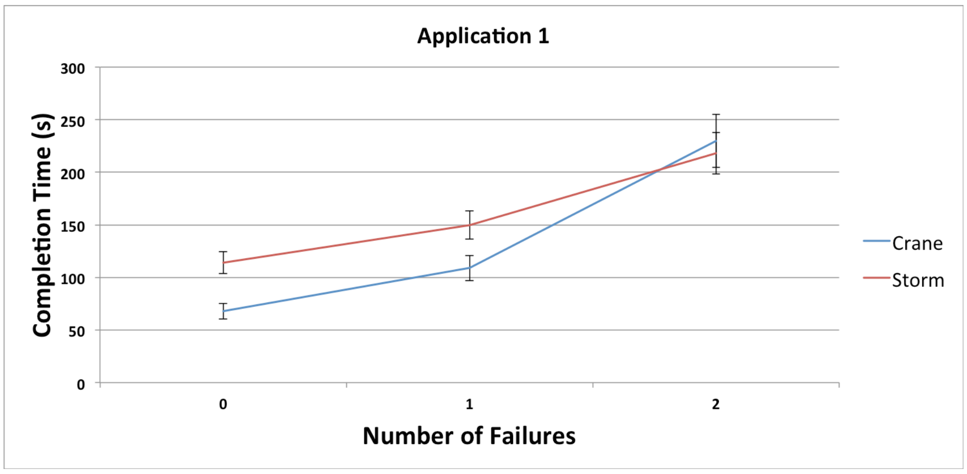
APPLICATION 3:
Extract lines from web-server logs matching a regex with getting error code descriptions by joining to a static file

In this application we take a huge file contains logs from a very busy web-server and filter lines matching a regex and then join an attribute in the line to get description of that attribute.

In this MP we used dataset from
"*http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html"*
Containing **1.8 Million Lines** of HTTP requests to the NASA Kennedy Space Center WWW server in Florida. We then extract all requests to get .gif images and join the error code of the log to it's description. The application is highly useful to analyze stream of logs coming from remote servers.

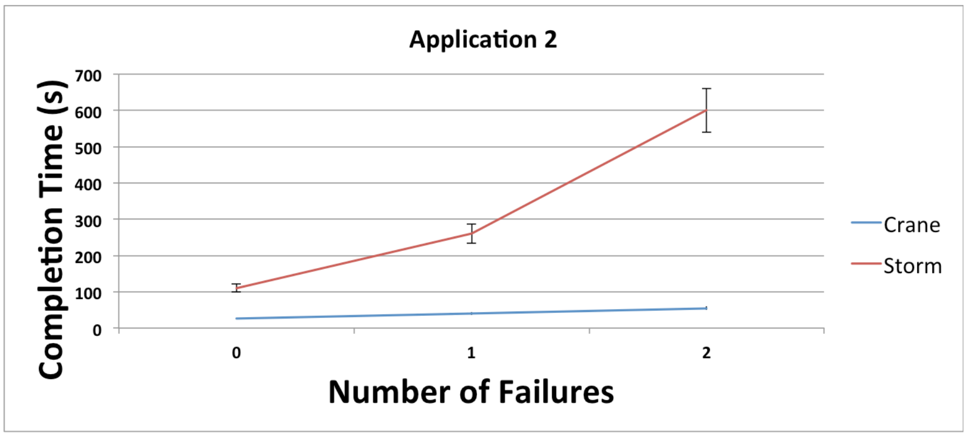
Experiments

Experiment-1



In this experiment we compared the completion time of Storm vs Crane on the following topology.
Storm: 3 Nodes, 3 Supervisors, 3 Workers, 40 tasks
Crane: 3 Nodes, 3 Workers, 40 tasks
We found that in terms of no and single failure Crane was much faster than storm whereas In high failure storm was slightly better than Crane. In terms of 2 failures Crane performed poorer than Storm mainly because of very high task load per worker (40 per worker). In Crane when a worker dies, it doesn't comes back and hence the failed and remaining tasks are launched on the remaining workers increasing the worker load. Whereas in Storm the supervisor relaunches worker in case of failure, keeping task load per worker small and hence the better performance. In 0 and 1 failures the task overload being still worse in Crane is still manageable and thus Crane performed better than Storm.

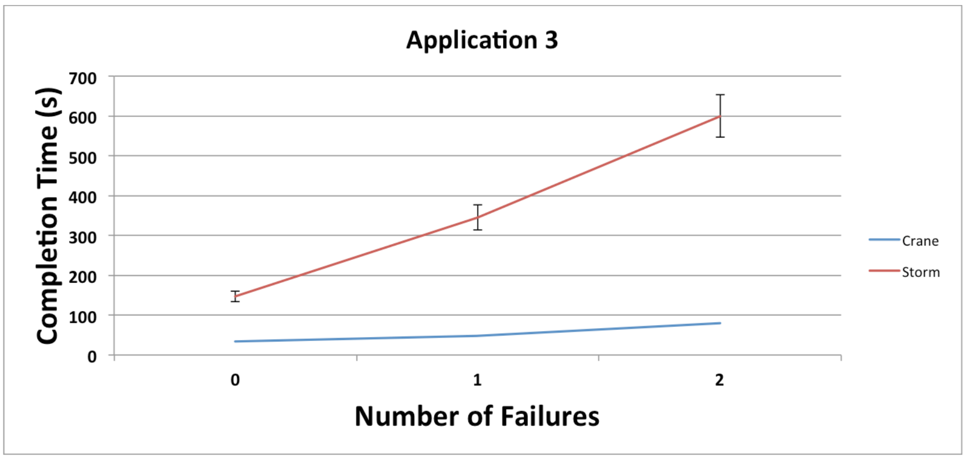
Experiment-2



In this experiment we compared the completion time of Storm vs Crane on the following topology.
Storm: 3 Nodes, 3 Supervisors, 3 Workers, 12 tasks
Crane: 3 Nodes, 3 Workers, 12 tasks
We found that Crane (4 x speedup) was much better than Storm in all conditions. The speedup is mainly caused by 1)lightweight communication using UDP used in Crane as well as 2)better ack-ing mechanism by limiting bolts in crane to emit only 0 or 1 tuple from input (no more need to track the whole tree lifetime of tuple) 3)Simplistic, lightweight design along with generality in framework.

Experiment-3

In this experiment we compared the completion time of Storm vs Crane on the following topology.
Storm: 3 Nodes, 3 Supervisors, 3 Workers, 9 tasks
Crane: 3 Nodes, 3 Workers, 9 tasks
Here also we found that Crane (4 x speedup) was much better than Storm in all conditions. The speedup is ever more than experiment-2 as the tasks per workers is even lower than experiment 2.



Conclusion

From our experiments we conclude that Crane is much faster than Storm in most of the conditions except in case of high task load per worker and very high failure rates. This is mainly due to the possibility of relaunching worker processes in Storm which is not done right now in Crane. In moderate task per worker loads and moderate failure rates Crane outperforms Storm by around 4 times.