CallMeIshmael Inc. (MP3) just got acquired by the fictitious social media company SpaceBook Inc. SpaceBook Inc. loved your previous work at CallMeIshmael (and they’re also aware of your great work on the mission to Mars in HW3), so they’ve hired you as a Spacebook Fellow. Congratulations!

You must work in groups of two for this MP (yes, Fellows also collaborate).

CallMeIshmael Inc. (MP3) just got acquired by the fictitious company SpaceBook Inc. (because eventually, every company is acquired either by ABCs Inc. or SpaceBook Inc. or Bitter Inc., even though all are fictitious). So they’ve decided to build a real-time stream processing system that will be faster than Spark Streaming (a system we discussed briefly in class).

This MP consists of three parts.

First, your task is to build a system called Crane that is faster than Spark Streaming. Crane has a design that is similar to Apache Storm (a system we studied in class) in that it uses tuples, spouts, bolts, sinks, and topologies. The only differences from Storm are that 1) Crane must be written from scratch (i.e., only uses code from MP1-3 and not from Storm or Spark, though you can look at the code for those systems), and 2) Crane topologies are trees. Crane bolts must support the main types of functionalities supported by Storm bolts, namely: filter, transforms, join of a stream with a static database (database could just be a file). You’re also welcome to implement some stateful bolts (but it’s not mandatory).

Further, Crane must be fault-tolerant: with up to two simultaneous failures of machines, the end result of computing a data set must be identical to the case when there are no failures. When the failure occurs, it must be detected (use MP2 for this), and then parts of the job must be restarted quickly. Any “lost” data must be handled and recomputed appropriately to get the same final result. Think carefully about the mechanisms you use—take inspiration from Storm’s mechanisms, and if you can improve on them, do so!

Second, you need to write three real applications using Crane, each processing a (separate) real dataset. Find a dataset from the web (at least several 10s of MB large, make it streaming, and the input stream running for at least 10 minutes), and stream it through your Crane topology. Build your Crane topology to do something useful, e.g., count the
top trending topics from a Twitter feed. More information about where to look for
datasets appears later below. Other applications include counting ad clicks, or cluster
monitoring e.g., receiving information the health of a cluster and deciding whether some
machines have failed or are up.

Third, compare your Crane system’s performance (both without and with failures)
against Spark Streaming. Make sure that you’re comparing Crane and Spark Streaming
in the same cluster on the same dataset and for the same topology. Draw plots to show
that Crane is better. Can you show that Crane is significantly better? In particular, show
at least one application/dataset for which Spark Streaming is better, and one
application/dataset in which Crane is better. Based on these results, discuss when one
should use Crane vs. when one should use Spark Streaming—if you can design rules of
thumb to help developers with this decision, that would be a great end result!

To measure performance, you could use either throughput, or latency, or both. Decide
which one(s) are important depending on your application.

Use the code for MP1-3 in building the Crane system. Use MP1 for debugging and
querying logs, MP2 to detect failures, and MP3 to store the results from the Crane
topology (or input data if that’s what you want to do).

As usual, don’t overcomplicate your design. SpaceBook Inc. is watching, and if they find
you’ve overcomplicated things, they will throw you into space (or throw the book at you,
depending on how you look at it).

Make sure you design a reasonable UI (command line is ok) to start and stop jobs, see
streaming results (e.g., dashboard, etc.) These will be useful during the demo.

We also recommend (but don’t require) writing tests for basic scheduling operations. In
any case, the next section tests some of the workings of your implementation.

These tasks are sequential, so please start early, and plan your progress with the deadline
in mind. DO NOT start a week before the deadline – at that point you’re already too late!

Designate one of your VMs/machines as a master, another machine as a client, and use a
third machine as a hot standby master. Use the remaining VMs as worker machines. The
master is used to submit and track jobs.

The master may of course fail (you can assume only 1 master fails at a time). If the master
fails, the hot standby should start kick in quickly. You can assume no other VM fails
between a master failing and it being replaced. While the master is failed or being
replaced, the ongoing jobs must not stop processing data. Recall that our goal is to have
the same end result of processing a data set as if there were no failure. (Recall that SDFS
writes and reads should work in spite of a failure). The only effect of master failure is that new jobs cannot be submitted. It is ok to limit the Crane cluster to accept one job at a time.

Other than the single master failure assumption, your system should be tolerant to up to two simultaneous machine failures (when the master is down you can assume zero further workers fail until a new master comes online quickly). Note that this implies that when workers fail, the master does not. When a machine fails, the master must restart the parts of the job on the remaining workers. Your goal should be to hide the failure’s effect and restart parts of the job automatically (and not manually). Also when a worker rejoins the system, the master must consider it for new tasks. The end result of processing a dataset must be the same, regardless of joins, failures, and leaves of masters and workers (within the above parameters).

Create logs at each server (so that they are queriable via MPI). You can make your logs as verbose as you want them (for debugging purposes), but at least a worker must log each time a Crane task is started locally, and the master must log whenever a job is received, each time a Crane worker task is scheduled or completed, and when the job is completed. Make sure you use unique names/IDs for all workers, servers, etc. We will request to see the log entries at demo time, via the MPI’s querier.

We also recommend (but don’t require) writing tests for basic scheduling operations. In any case, the next section tests some of the workings of your implementation.

Datasets: Good places to look for datasets are the following (don’t feel restricted by these):
- Amazon datasets: https://aws.amazon.com/datasets/
- KONECT: http://konect.uni-koblenz.de/networks/
- ICON: https://icon.colorado.edu/#!/networks

Machines: We will be using the CS VM Cluster machines. You will be using 7-10 VMs for the demo. The VMs do not have persistent storage, so you are required to use git to manage your code. To access git from the VMs, use the same instructions as previous MPs.

Demo: Demos are scheduled on Monday, Dec 3, 2018. The demos will be on the CS VM Cluster machines. You must use 10 VMs for your demo (details will be posted on Piazza closer to the demo date). Please make sure your code runs on the CS VM Cluster machines, especially if you’ve used your own machines/laptops to do most of your coding. Please make sure that any third party code you use is installable on CS VM Cluster. Further demo details and a signup sheet will be made available closer to the date.

Language: Choose your favorite language! We recommend C/C++/Java/Go.
Report: Write a report of less than 3 pages (12 pt font, typed only - no handwritten reports please!). Briefly describe your design (including architecture and programming framework) and chosen applications. Be very clear and very comprehensive. Also, show plots that compare Crane’s performance to Spark Streaming, for each of the three applications. Think about what metrics you wish to plot, and what parameters you wish to vary. Make sure that you’re comparing Crane and Spark Streaming in the same cluster on the same dataset and for the same topology. Discuss your plots, don’t just put them on paper, i.e., discuss trends, and whether they are what you expect or not (why or why not). (Measurement numbers don’t lie, but we need to make sense of them!). Stay within page limits.

Devote sufficient time for doing experiments (this means finishing your system early!).

Submission: There will be a demo of each group’s project code. Submit your report (softcopy) as well as working code. Please include a README explaining how to compile and run your code. Submission instructions are similar to previous MPs (see Piazza).

When should I start? Start now on this MP. Each MP involves a significant amount of planning, design, and implementation/debugging/experimentation work. Learning how to run Spark Streaming is also a time-consuming task, so please devote enough time to do this. Do not leave all the work for the days before the deadline – there will be no extensions.

Evaluation Break-up: Demo [60%], Report (including design and plots) [30%], Code readability and comments [10%].

Academic Integrity: You cannot look at others’ solutions, whether from this year or past years. We will run Moss to check for copying within and outside this class – first offense results in a zero grade on the MP, and second offense results in an F in the course. There are past examples of students penalized in both those ways, so just don’t cheat. You can only discuss the MP spec and lecture concepts with the class students and forum, but not solutions, ideas, or code (if we see you posting code on the forum, that’s a zero on the MP). SpaceBook Inc. is watching!

Happy Stream Processing (from us and the fictitious SpaceBook Inc.)!