CS425, Distributed Systems: Fall 2017
Machine Programming 4 – Sava

Released Date: Nov 9, 2017
Due Date (Hard Deadline): Sunday, Dec 3, 2017 (Code due at 11.59 PM)
Demos on Monday December 4, 2017

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).

SpaceBook needs to fight off competition from their two biggest competitors Hooli Inc. and Pied Piper (a startup). Since they're a social media company they deal with social networks, i.e., graphs. So they need a **distributed graph processing system.** That's what you'll build for this MP.

You have three tasks in this MP. First, build a distributed program called **Sava** (named after a river in East Europe, passing through Belgrade) to process large graphs, for arbitrary functions. Store the input graph and output results in SDFS (MP3). Make your system fault-tolerant on the same number of failures as MP3 by relying on your failure detector from MP2. Second, write (at least) **two applications** using Sava. Third, measure your system's performance, and experimentally compare against GraphX (a graph processing engine based on Spark, readily available but one you will have to deploy on the VMs). 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!

Below, SpaceBook has some requirements about the design of Sava. However, they want you to fill in some gaps in the design, and of course to implement the system. Be prepared to improvise, be prepared to deploy new systems, and be prepared for the unknown.

This MP requires you to use code from MP1, MP2, and MP3.

Design: The basic design of Sava is very similar to the Pregel system that was discussed in class. Read the Pregel paper (also named after a river in East Europe) [https://dl.acm.org/citation.cfm?id=1807184], and make sure you understand how the BSP model/Gather-Apply-Scatter model works. Then implement it. Essentially, your graph processing system uses multiple workers. Each VM-i.e., machine--may run multiple workers--but for simplicity you should

fix the number of workers to 1 worker per VM. For a job, Sava (just like Pregel) parallelizes it into tasks, assigns each task to a worker. The graph's vertices are then partitioned across workers using a partitioning function. In the BSP/GAS model, tasks work in iterations across workers, with a barrier at the end of each iteration when the workers exchange data. Your Sava system must admit various programs for the code that is computed inside each iteration. So you must be able to run PageRank, Single Source Shortest Path, etc. You can reuse design decisions from the Pregel paper if you want (unless it complicates your design.). In any case you have a lot of room to innovate (e.g., think of what partitioning function you want to use).

Designate one of your VMs/machines as a master, another machine as a client, and use the remaining a third machine as a standby master, and use the remaining 7 VMs as worker machines.

The Sava cluster has N (up to max number of VMs) server machines. It accepts one job at a time. You can use a master server to receive commands from clients, and to coordinate activities among the other workers and servers (e.g., sending commands to them to start processing). The master may of course fail (you can assume only 1 master fails at a time)—under failure, the ongoing job must not stop its current iteration, though the failure may prevent the next iteration, and prevent job status being available to the user—**everything else must work just as normal under master failure**. (Recall that SDFS writes and reads should work in spite of a failure). If the master fails, you should create a new master **quickly** (think whether you want to have a hot standby or create a master from scratch post-failure). You can assume no other VM fails between a master failing and it being replaced.

Other than the single master failure assumption, your system should be tolerant to up to **2 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 job itself from the very first iteration, and schedule the tasks quickly. (Notice that you cannot continue the old iterations after failure since we're not checkpointing state in between iterations). Your goal should be to hide the failure's effect and restart the job **automatically** (and not manually). Also when a worker rejoins the system, the master must consider it for new tasks.

(Re-)Use the code for MP1-3 in building the Sava system. Use MP1 for debugging and querying logs, MP2 to detect failures, and MP3's SDFS to store Sava input graphs and output results.

Create logs at each server (so that they are queriable via MP1). You can make your logs as verbose as you want them (for debugging purposes), but at the least a worker must log each time a Sava iteration task is started locally, and the master must log whenever a job is received, each time a Sava 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 MP1'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.

Applications and Experiments: Sava must admit generic graph processing applications. Think about the interface—we suggest you admit a class/interface that specifies the function.

Write **two applications** using Sava. You must choose #1 below (PageRank), and choose the other one from #2-#4 below.

1. **(Mandatory)** PageRank—this is the most popular applications (run for either fixed number of iterations or until convergence). Write a version that outputs the **top 25 vertices with the highest PageRank values.**
2. Single Source Shortest Path (SSSP), where you pick the source.
3. Graph Coloring,
4. Graph Diameter.

You can look these up online (including GraphX tutorials and tutorials in other systems like PowerGraph, GraphLab, and PowerLyra to see how these applications are written).

Dataset/graph information appears later in this document. Try to use graphs that have **at least 100K vertices** in them (if possible, even larger).

Comparison against GraphX: After you have Sava working, make it more efficient. Make it faster than GraphX [https://spark.apache.org/graphx/] . Download Spark and GraphX and run it on your VMs (this step will take some effort, so give it enough time!). **Compare the performance of Sava with GraphX.** Most of the inefficiencies in Sava will be in loading the graph (loading time), so think of which partitioning function is the best. If you want to see a good survey and comparison of partitioning functions, read this paper: [http://dprg.cs.uiuc.edu/docs/vldb2016partitioning/500-verma.pdf] (see especially Sections 5.2, 6.2, 7.2). Don't go overboard with the partitioning function – select the simplest partitioning function that helps you beat GraphX (we recommend starting with Random partitioning and if that's not fast enough, trying something else).

Make sure that you're comparing Sava and GraphX in the same cluster for the same graph. You should be able to beat GraphX in at least 1 out of the above 2 applications (preferably both!).

Datasets/Graphs: Good places to look for datasets are the following (don't feel restricted by these, though the SNAP repo below is a good):

- Stanford SNAP Repository: http://snap.stanford.edu/data/index.html (the LiveJournal and pokec graphs are good ones to use; others are too small).
- Amazon datasets: https://aws.amazon.com/datasets/ . See the fun Marvel Universe Social Graph (though it's small, you can use it for testing).
- KONECT: http://konect.uni-koblenz.de/networks/
- ICON: https://icon.colorado.edu/#!/networks
- Even if these graphs are not big enough for your experiments (> 1M vertices) you can still use them for testing and writing fun applications.

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 usually scheduled on the Monday right after the MP is due. The demos will be on the CS VM Cluster machines. You must use 7-10 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.

For the demo, plan on using the following code (for 20 iterations) and graph (we will compare your answers with the correct ones):
PageRank code:
https://github.com/apache/spark/blob/master/graphx/src/main/scala/org/apache/spark/graphx/lib/PageRank.scala
Dataset: http://snap.stanford.edu/data/com-Amazon.html

Language: Choose your favorite language! We recommend C/C++/Java.

Report: Write a report of less than 3 pages (12 pt font, typed only - no handwritten reports please!). In half a page, briefly describe your design (including architecture and programming framework) for Sava. Be concise and clear.

Using one more page, for both of your chosen applications and a real graph (>

100 K vertices), run Sava using different values for: 1) number of tasks and 2) number of servers (with same number of workers). Draw one plot for each, plotting the job runtime (also called makespan or completion time). Job runtime is the sum of **loading time** and **time to run iterations—separately, show each of these on the plot (along with the sum).** Each plotted data point must be the average of at least 3 experiment runs. Plot both the average (or median) and the standard deviation.

Using the last page, show plots comparing Sava's performance to GraphX, for each of the 2 applications. Can you beat GraphX at least 1 out of the 2 applications? For each data point on the plots, take at least 3 measurements, plot the average (or median) and standard deviation. Run each experiment (for each system) on all the VMs in your allocation.

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

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!)

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. **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 Graph Processing (from us and the fictitious SpaceBook Inc.)!