Design:

In our Crane, there are totally four kinds of thread. A checker thread to check whether the introducer has received all other VMs results. A monitor thread to provide interact command for users. A dispatcher thread to sequentially dispatch chunk of data to all VMs. A receiver thread to listen and analyze different message. Initially the dispatcher thread and checker thread are only in introducer VM. Other threads are in each of VMs.

In monitor thread, there are several commands. The 'start <filename file_type >' can launch a stream processing, and start the dispatch thread and checker thread. 'join' and 'leave' are from MP2. 'print' cmd can print the processing results in current VM. 'file result <output_file>' cmd can store the result into file. 'clear result ' can clear all processing result in memory. 'print current host ' can show all VMs whether results have been received by introducer.

The dispatcher thread will start after receive 'start' cmd in monitor thread. Then each data chunk will be dispatched to three VMs, one is primary the other two are replicas. Also a 'dispatch' message will be sent to the VM with primary chunk.

In checker thread of introducer, all VMs are checked periodically to ensure whether the results have been received successfully by introducer. If not, the introducer regard this VM has failed, and then send a message with 'redo' keyword as well as this VM name to next VM, asking it reprocess the replica.

The receiver thread will start processing specific data chunk after receiving 'dispatch' and 'redo' msg. when finish processing, the result will be sent back to introducer with a message identifying by 'dict' keyword.

Fault tolerance.

For each chunk of data, the introducer would send it to specific VM using round robin method. Also the same data is sent to the two successor of that VM. After receiving chunk, the VM would start processing and send the result back to introducer. For each VM, the introducer would maintain the number of the sent chunk and the number of received result from that VM. if two numbers are different over a period of time, then introducer would ask the successor to reprocess that chunk data. If the master fails, the succeeding VM would become new introducer. Those data that already sent would still be processed, and the result would also be stored in local VM and sent to new introducer.

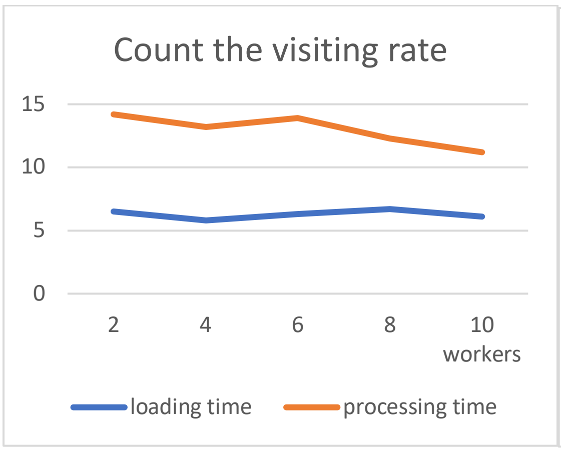
Three applications:

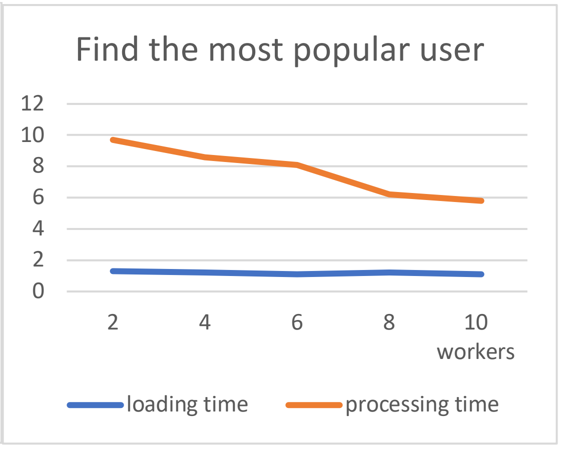
Count the visiting rate of different web object type. The log is 52.4MB.

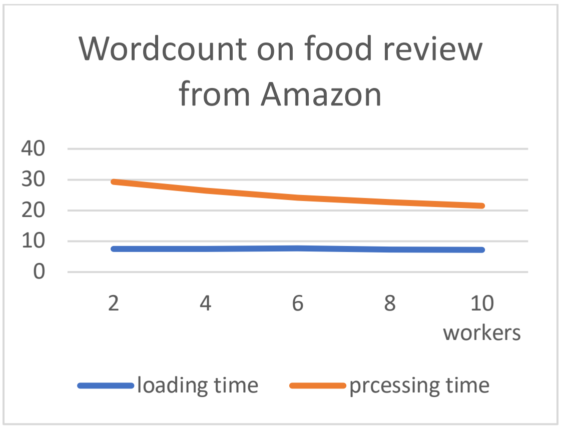
Find the most popular user on Twitter network by counting the node degree. The data is 13.3MB.

Wordcount on food review from Amazon. The data is 78.2MB.

Three charts below shows different running time when worker number varies.



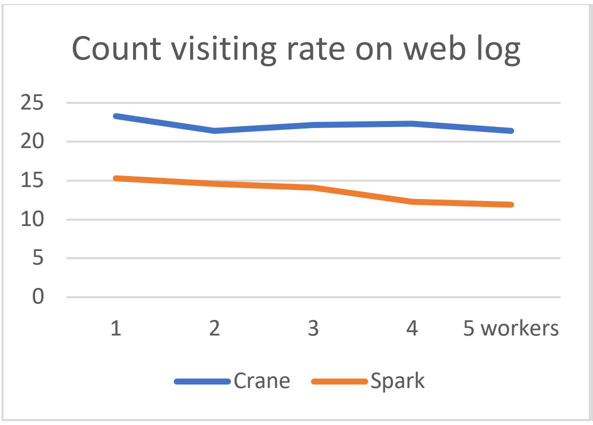


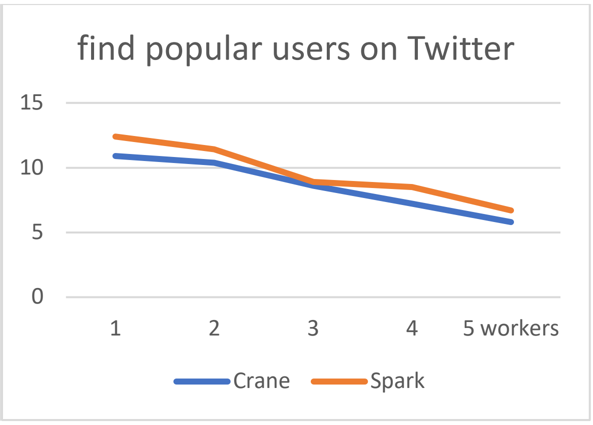


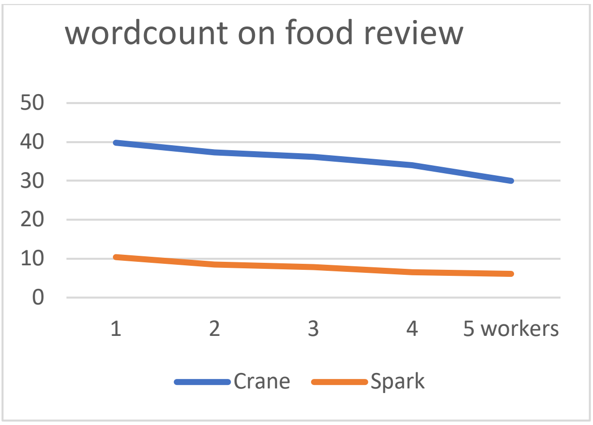
Crane performance on three applications

(1) The loading time for each application is similar among different VM number configurations. This is as expected since the whole file needs to be loaded into memory and then can sequentially send to different VMs.
(2) The processing time is decreased when the number of workers is increasing. This is as expected since different VMs can do the job in parallel. And the introducer only needs to read their results, which is faster than reading raw data. But still, the processing time cannot have dramatic decrease because the communication time between VMs cannot be ignored.

Here, we compare Crane's running time to Spark Streaming, for each of the three applications. Crane is run on 1-5 VMs. Spark is run on 6-10 VMs.







The following ideas can be summarized from three comparison plots above.

(1) With the increase of worker number, the time cost would decrease. Even sometimes the decrease is not obvious or even sometimes the time increases, the total trend on time cost is decreasing.
(2) Both two applications of Counting visiting rate and wordcount on review containing many useless words in raw data. While we only use split method to extract what we want in Crane using python, generating the list when using split many have much time cost. But in Spark, they might have some other method or other language to accelerate the filtering process. Therefore the Spark's time cost on these two applications is much lower.
(3) Twitter network dataset does not contain too much useless words, mostly are node number. Therefore many data split in Crane are useful, and the time cost are lower than Spark.