**Program5 & Lab5 Report**

**Name: Chengjun Xi Netid: xi7**

1. **Summary of Application**

**Spark**

To begin with, the program reads the training points from train.csv and test points from test.csv. Then it deals with one test point each time, calculating the distance of all training points to it. After that, the distances are sorted or reduced by a priority queue and the closest k points are selected and do the majority voting to decide the class of a test point.

1. **Parallelization Techniques**

**Spark**

The first version of Spark parallelization follows the figure below:

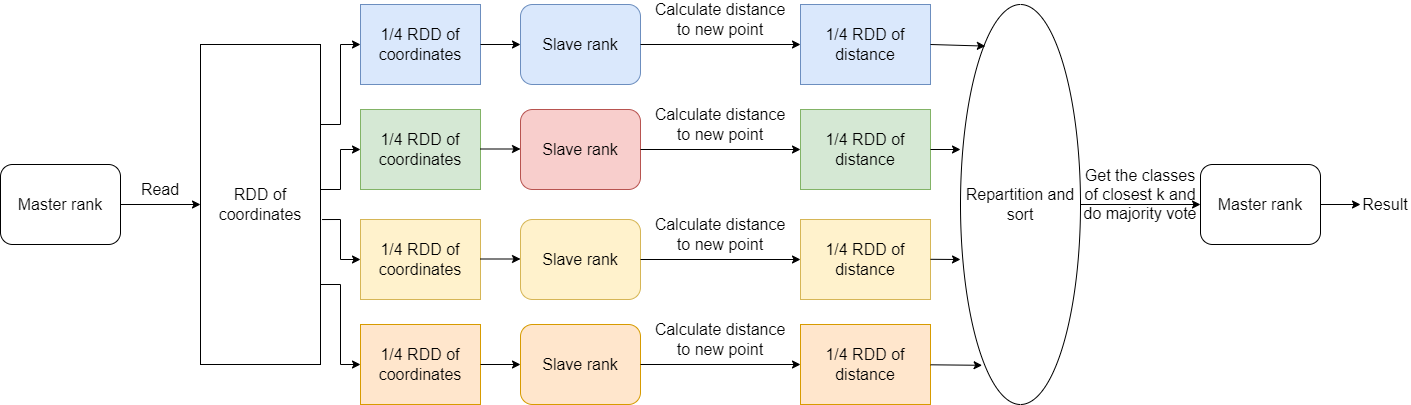


Figure . First version of Spark parallelization strategy

The first 40 points in the about 1400000 points are the test points. These 40 points are in file test.csv. Another file train.csv contains the training set of all rest points. The parameter k is input by command line option, which was set in file run.sh. The master rank reads the files. Spark partitions the training set based on the number of ranks. In the above 4-rank example, each rank receives a quarter of the entire RDD. Then ranks compute the distance to a point in the prediction set parallelly. Each rank computes distances based on only the training set it has. After that, the distances are sorted. However, sorting parallel data in Spark is more complicated than sorting data in shared memory. Spark repartitions the data first and then sorts. Finally, the k points closest to the target point are selected and they decide the class of the target point using majority voting at the master rank.

However, sorting in sequential code is essentially different from sorting in parallel code. Regarding sequential code, since all data are stored on one rank, the sorting can be done locally. Nevertheless, the data in parallel computing are stored on different ranks. Therefore, the order on each rank is local. The data needs inter-rank operations to achieve total order over all data. Fortunately, Spark has wrapped up these operations in its sorting methods. Spark repartitions the data by shuffling. One important difference from sequential code is that the shuffling is expensive and the distributed data is adding extra time cost to sorting. It is uncertain how much this extra expense is. Alternative ways to avoid sorting may improve the performance.

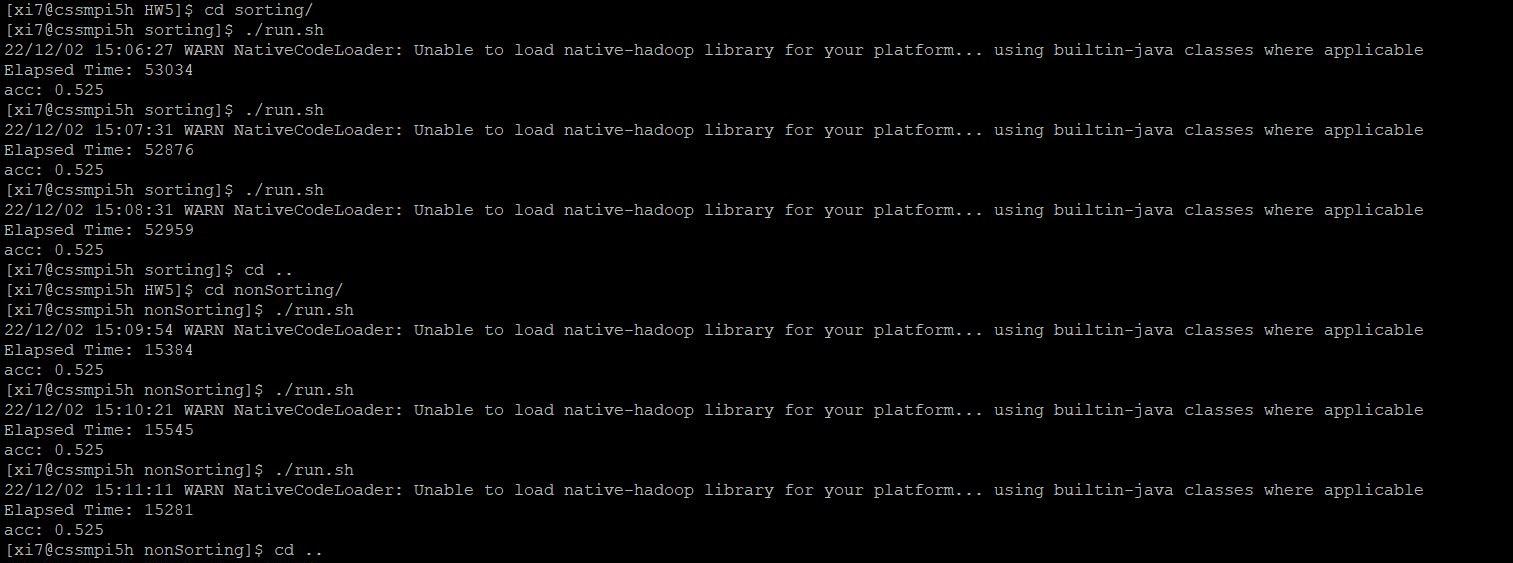
An alternative way to avoid sorting is to maintain a priority queue of k numbers of minimal distances. The distances from all points are reduced by adding the distance to the ordered list if the list is not full, replacing the largest element if the new distance is smaller than the largest element, and skipping the new distance if it is larger than the largest element in the list. Apart from avoiding sorting including expensive shuffling, this algorithm is also of smaller complexity than the algorithm including sorting. Assume the size of all points to be n, the sorting is O(n\*log(n)), whereas the reduce by priority queue is O(n\*k). In most applications, k is much smaller than log(n). In our application, k=10, log(n) is about 20.

1. **Performance Analysis**

**Spark**

**4-ranks and 4-cores**

The experiments are averaged on 3 repeats. They used 4 ranks and 4 cores. The first three are the version using sorting. The later three used priority queue reduction and no sorting.

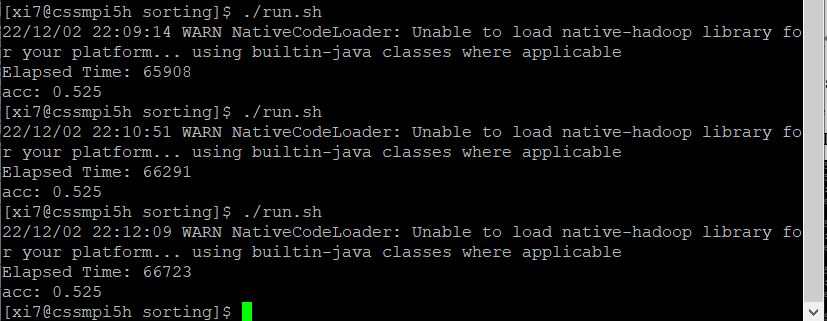


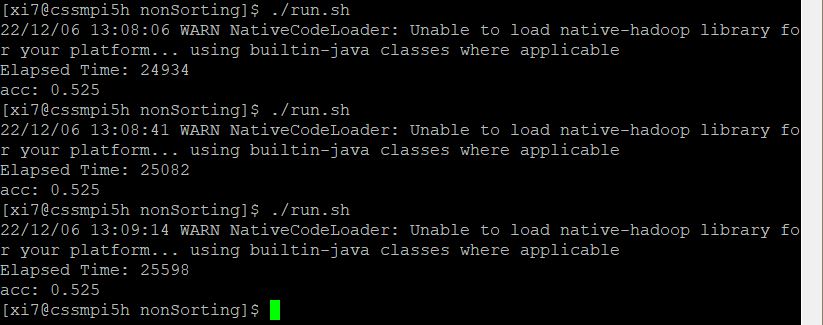
Command line output

The average of performance using sorting is 52956. The average of performance using priority queue reduction is 15403. The improvement is **3.438** times by avoiding sorting. The correctness is proved by unanimous accuracy 0.525.

**1-rank and 1-core**

The experiments are averaged on 3 repeats. They used 4 ranks and 4 cores. The first three are the version using sorting. The later three used priority queue reduction and no sorting.





Command line output

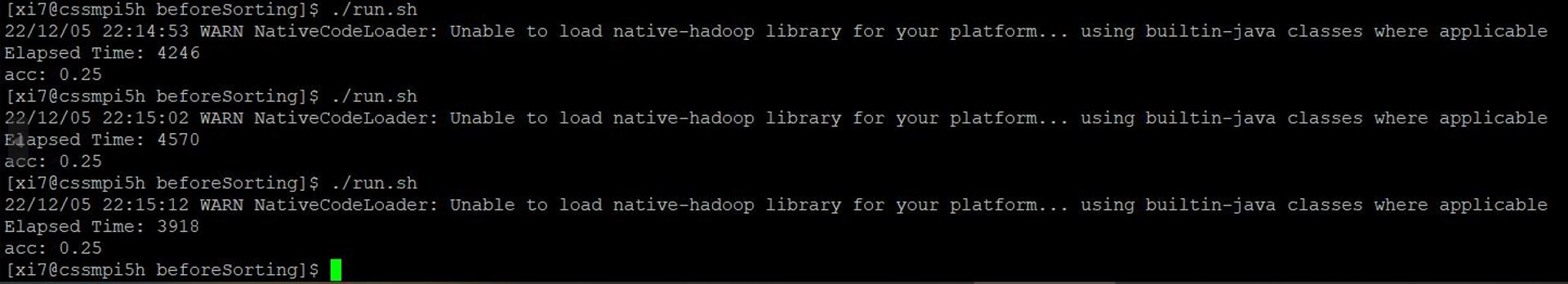
The average of performance using sorting is 66307. The average of performance using priority queue reduction is 25204. The correctness is proved by unanimous accuracy 0.525. The running time statistics is as below table.

**Table 1. Spark running time statistics**

|  |  |  |
| --- | --- | --- |
| Configuration \ Algorithm | Using priority queue reduction | Using sorting |
| 4 ranks and 4 cores | 15403 | 52956 |
| 1 rank and 1 core | 25204 | 66307 |

The improvement of Spark using sorting is 66307/52956=**1.252**. The total improvement of Spark avoiding sorting is **4.305**.

In addition, as a control group for the influence of sorting, I made a special version by simply removing the sortByKey() method in the sorting version. The accuracy is not correct, but the performance is meaningful.



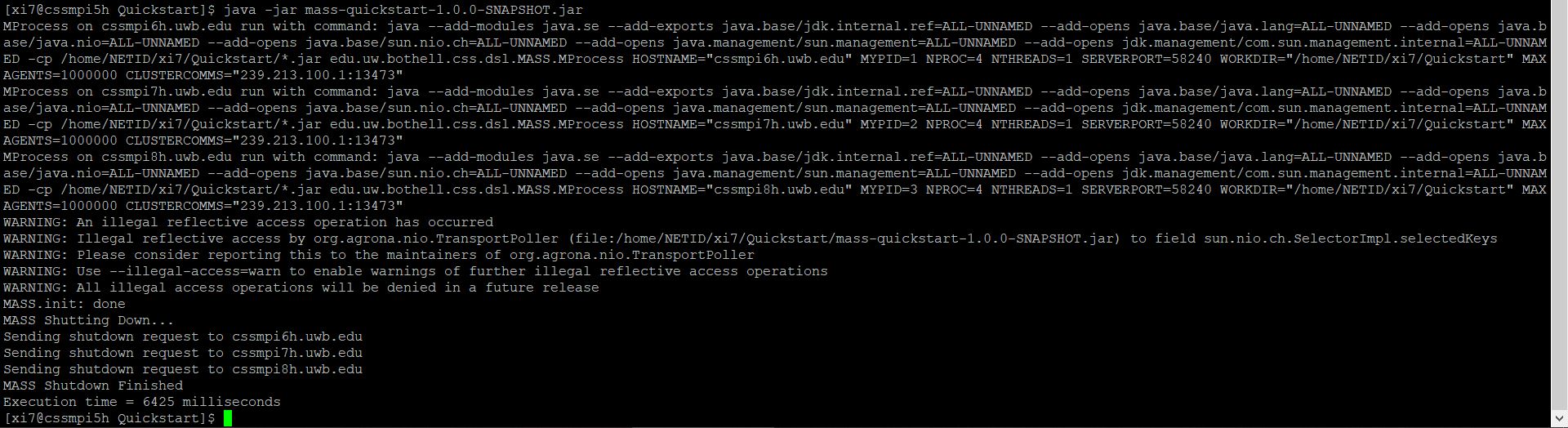
The average running time is only 4245. The sorting caused a slow-down of 12 times. Since only the sortByKey() method was removed, the slow-down was attributed only to the sorting. Therefore, avoiding sorting is crucial to performance when using Spark.

1. **Programmability analysis**

**Spark**

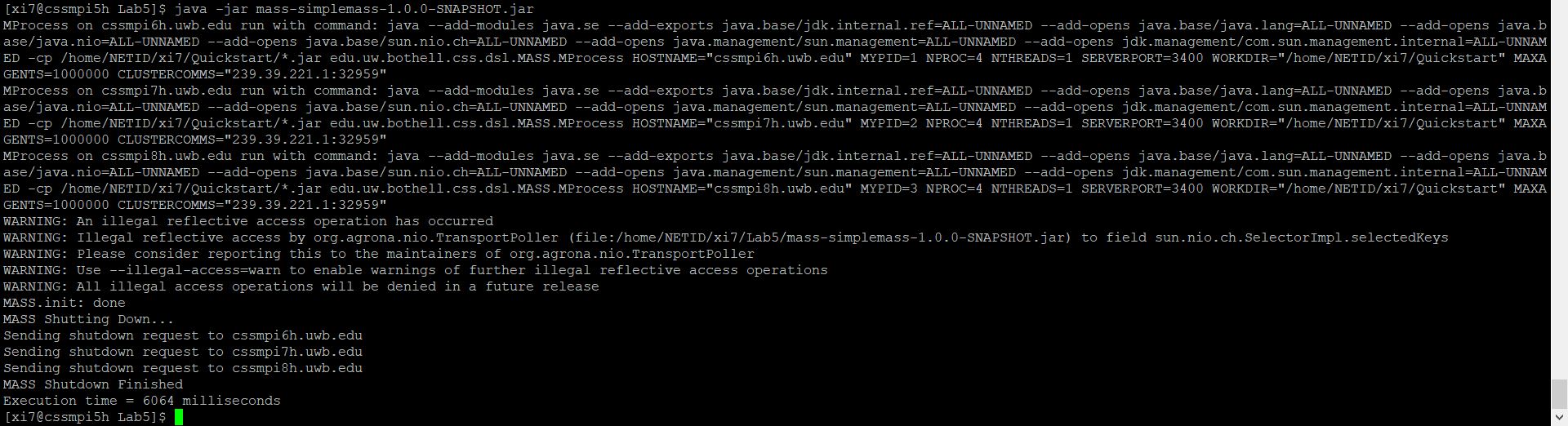
Regarding the programmability of the Spark, it has useful APIs that saves code. However, the APIs are not necessarily optimal in performance, like the sorting method. The map and reduce functions save the codes for the loop control over elements. Nevertheless, the loop control of the map and reduce functions are still written by the user. It gives the user control of the program workflow. Besides, the parallelization part is completely taken care of by the Spark framework. What the user needs to care is just to set the number of cores when running it and set the number of worker machines in configuration files. The communication between ranks and cores are completely wrapped up by Spark. Nevertheless, this may lead to difficulty in making fine-grained control of parallelization. When we meet unbalanced work for each element, it would be hard to customize the parallelization such as dynamic scheduling. Spark is easy to program, but the programmability for complicated communication is low.

1. **Lab 5**

****

Output of Quickstart program

The Quickstart program ran successfully. The setup was correct.



Output of Lab5 program

The Lab5 program that migrates once ran successfully. It was correct.