Mini Batch K-Means

Vikrant Kaushal
Indiana University
E. 7th Street
+1-8123619322
vkaushal@indiana.edu

Shalabh
Indiana University
E. 7th Street
+1-8123182875
sshalabh@umail.iu.edu

1. INTRODUCTION

K-means is one of the simplest algorithms that handle clustering problem. The procedure is to classify a given data set in a certain number of clusters (assume k clusters) fixed from before. The main idea is to define k centers, one for each cluster. These centers should be placed in such a way as different location causes different result. So, the better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest center. When no point is pending, the first step is completed and an initial clustering is done. At this point we need to re-calculate k new centroids as the center of the clusters resulting from the previous step. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new center. A loop has been generated. As a result of this loop we may notice that the k centers change their location step by step until no more changes are done or in other words centers do not move any more. This algorithm aims at minimizing the squared error function given by:-

$$J(V) = \sum_{i=1}^{c}\sum_{j=1}^{c_i}\left(\|x_i - v_j\|\right)^2$$

where,

'||xi – vj||' is the Euclidean distance between xi and vj.
'ci' is the number of data points in ith cluster.
'c' is the number of cluster centers.

Figure 1: Formula to calculate SSE

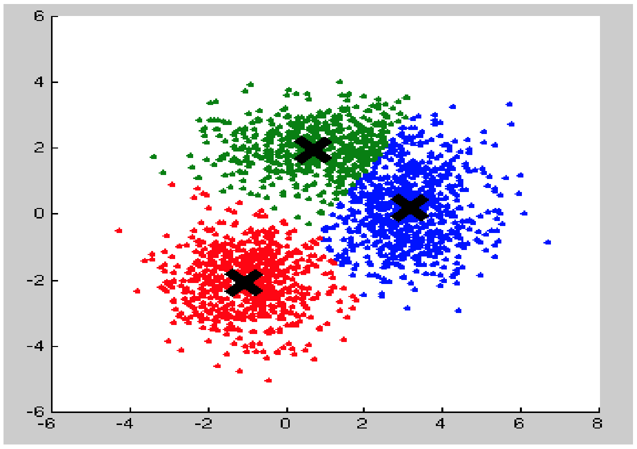


Figure 2: K-Means clustering

2. PROBLEM DEFINITION

K-means is one of the most used clustering algorithms, mainly because of its good time performance. With the size of the data sets being analyzed increasing, this algorithm is losing its attractiveness because of its constraint of needing the whole dataset in main memory. A different approach is the mini batch K-means algorithm. Its main idea is to use small random batches of examples of a fixed size and calculate centroids so they can be stored in memory and thus improve performance with the bigger datasets. Each iteration a new random sample from the dataset is obtained and used to update the clusters and this is repeated until convergence. Each mini batch updates the clusters using a convex combination of the values of the prototypes and the examples, applying a learning rate that decreases with the number of iterations. This learning rate is the inverse of number of examples assigned to a cluster during the process. As the number of iterations increases, the effect of new examples is reduced, so convergence can be detected when no changes in the clusters occur in several consecutive iterations.

The main challenge is to implement K-Means in parallel using harp and multi-threading to gain significant performance advantage. We will have to observe the performance by varying the parameters like batch size, iterations and cluster size to observe the best performance. We will also be fixing the batch size and the number of iterations which be the terminating condition for the program.

3. PROPOSED METHOD

Here are steps we followed for implementation:

1. Split file into a number of mappers parts and move these files to HDFS.
2. Pick initial centroids randomly from data into a file and move it to HDFS.
3. Initiate mapper after configuring the job.
4. Master process will read centroid file and broadcast.
5. We created a new data structure to hold our data and thus wrote combiner for same.
6. Each Iterations does following:
    a. Randomly picks data points equal to size batch.
    b. Each mapper spawns maximum number of thread. Data is divided into size/noOfThreads and thus each thread works on separate data.
    c. For every dataPoint closest centroid is found using Euclidean distance.

d. We are working with Regroup-Allgather and AllReduce.
e. Now every centroid is updated using assigned data points

7. Repeat step 6 for number of iterations.

8. Evaluation: We are using purity for evaluation. We find majority class for every clusters and thus calculate Precision.

4. DATA SET

We have used Newsgroup dataset. The 20 newsgroups dataset comprises around 18000 newsgroups posts on 20 topics. It contains more than 96k features in inverted index format. Our custom dataset is helping us to parse and work on inverted index efficiently. We wrote a combiner also as required by harp for our class. Here is the structure of our class:

Example data:

Class	Feature:Value
1	10:3000 20:25 30:35

5. DATA-STRUCTURE

We are working on sparse data and thus is in inverted index format. Our data has features more than 96k. We found that Harp doesn't provide any data structure that will help to read sparse data. It would have been very inefficient if we would have used DoubleArray to read provided data. So we wrote a new dataStructure/Class that helps us in reading data in a Int2DoubleOpenHashMap along with other fields: Here is the class:

```
public class MiniBatch extends Writable {
  private int classLabel;
  private int allocatedCentroid;
  private Int2DoubleOpenHashMap featureValueMap;
}
```

As required we override following methods:

1. getNumWriteBytes
2. write
3. read
4. clear

We also wrote a combiner (MiniBatchPlus.java) for our class that works on to combine Int2DoubleOpenHashMap of two data points.

6. FLOW CHART

We have worked with Regroup-Allgather and Allreduce to figure out that which algorithm performs better. Below is the basic flow that we have followed.

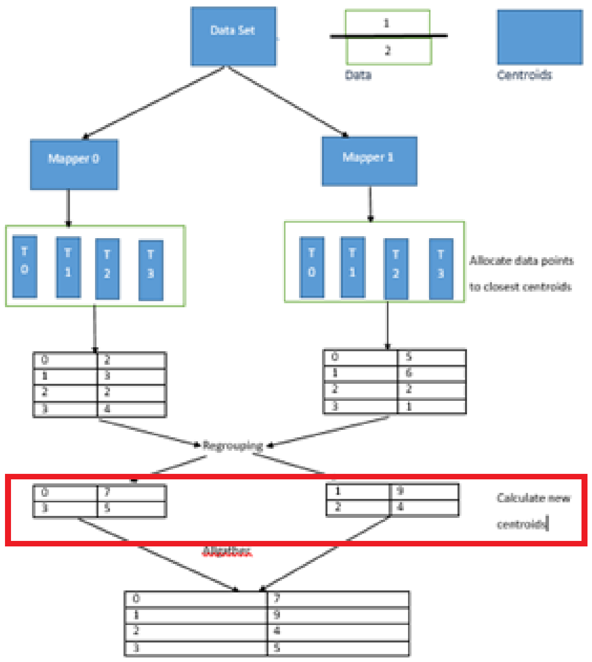


Figure 3: Workflow for Harp

7. EXPERIMENTS

We used six modes of execution namely:-

- regroup-allgather with multi-threading
- allreduce with multi-threading
- regroup-allgather with1 mapper
- allreduce with 1 mapper
- regroup-allgather with 2 mappers
- allreduce with 2 mappers

We experimented by executing harp program with varying various parameters like number of centroids, number of iteration and batch size one at a time and keeping other parameters constant. We recorded the time taken to execute and the positive performance value for each execution mode.

7.1 Varying the no of iterations:-

The batch size was kept fixed at 250 along with the number of centroids as 20.

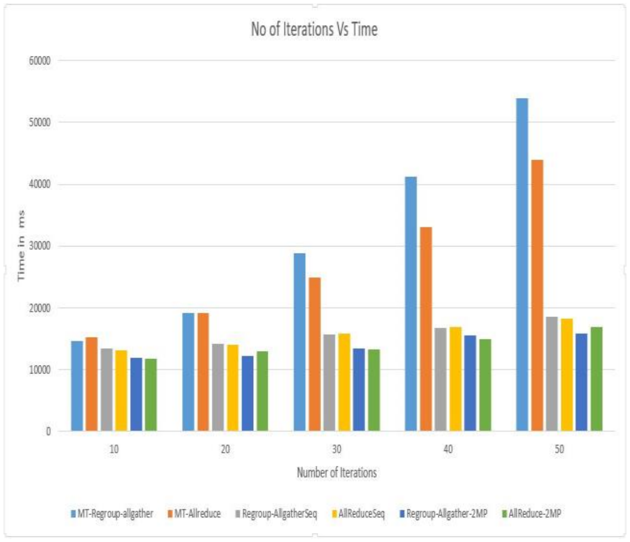


Figure 4: Time Vs No of Iterations

We can observe that the time taken for all reduce with 2 mappers is the least and that with regroup allgather with multi-threading is the most. The reason might be that with the data set being less the overhead in splitting the data set and combining the result in case of multi-threading might be way more than the actual computation work.

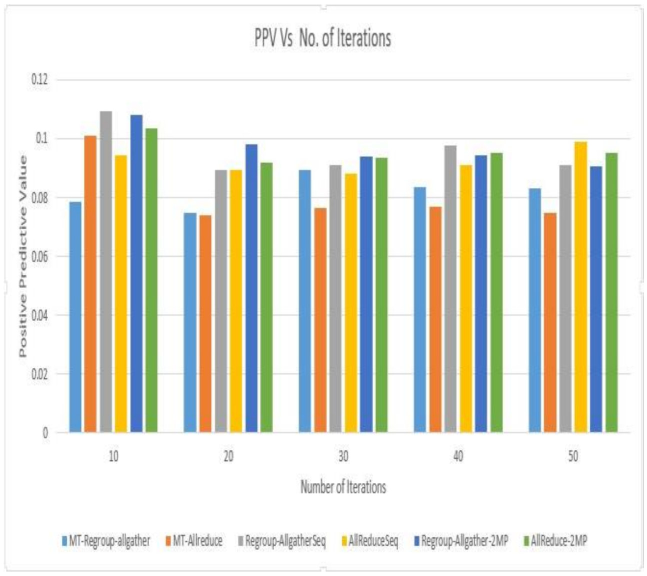


Figure 5: PPV Vs No of Iterations

Overall the accuracy was better for the regroup-allgather with 2 mappers for all iterations.

7.2 Varying the no of clusters:-

The batch size was kept fixed at 250 along with the number of iterations as 10.

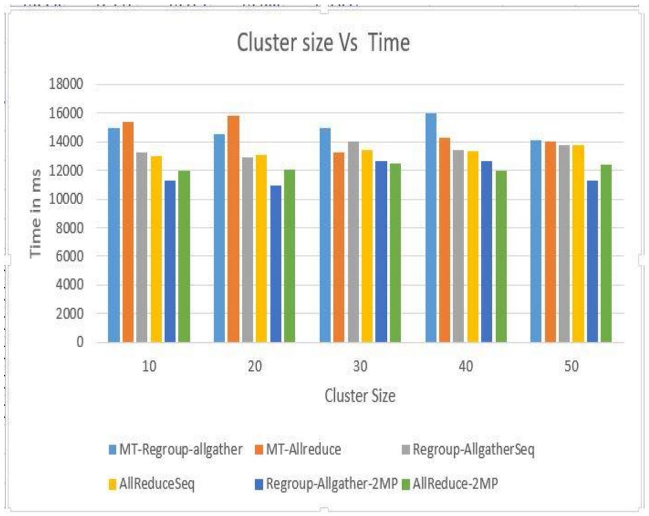


Figure 6: Time Vs Cluster Size

We can observe that the time taken for time taken for all reduce with 2 mappers is the least and that of regroup allgather with multi-threading is the most. One reason which can be mooted upon is that with the data set size being less the overhead in splitting the data set and combining the result in case of multi-threading might be way more than the actual computation work.

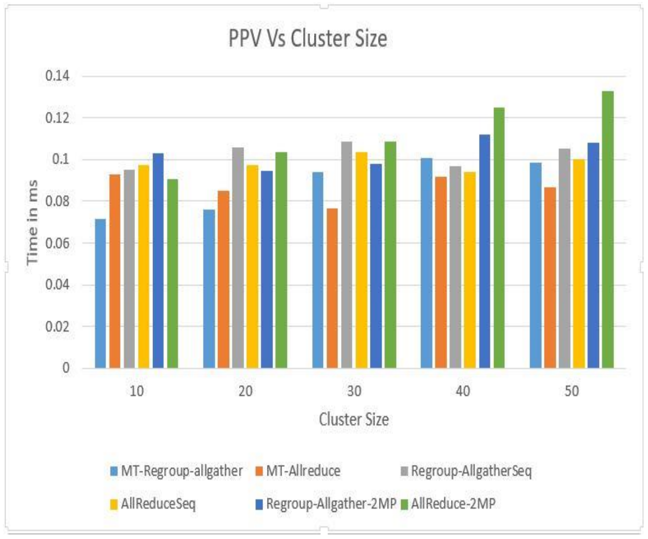


Figure 7: PPV Vs Cluster Size

Overall the accuracy was better for the regroup-allgather with 2 mappers for all iterations is way more pronounced than the other execution mechanisms.

7.3 Varying the size of batch:-

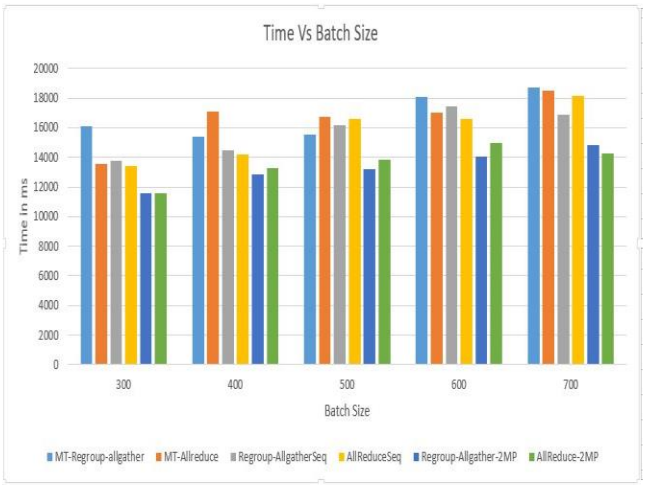


Figure 8: Time Vs Batch Size

We can observe that the time taken for all reduce with 2 mappers is the least and that with regroup allgather with multi-threading is the most. The reason might be that with the data set being less the overhead in splitting the data set and combining the result in case of multi-threading might be way more than the actual computation work.

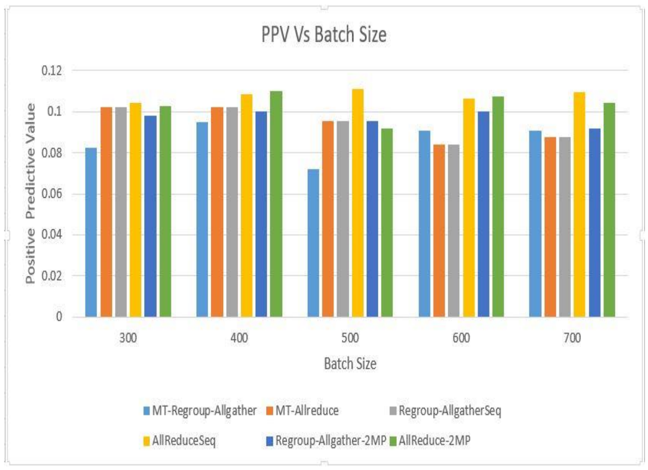
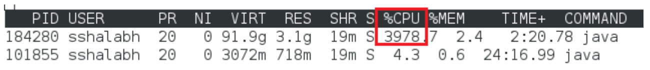


Figure 9: Time Vs Cluster Size

The accuracy was better for the regroup-allgather with 2 mappers for all iterations is way more pronounced than the other execution mechanisms.

8. HIGH RESOURCE UTILIZATION DURING MULTI-THREADING

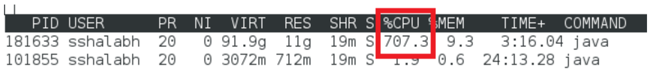
While Using multi-threading during the project CPU usage exceeded more than 100% on Juliet and thus it leads to low performance. Below is the snap from Juliet for same.



```
   PID USER      PR  NI   VIRT   RES   SHR S %CPU %MEM     TIME+  COMMAND
184280 sshalabh  20   0 91.9g 3.1g   19m S 3978.7  2.4   2:20.78 java
101855 sshalabh  20   0 3072m 718m   19m S   4.3  0.6  24:16.99 java
```

Figure 10: CPU Usage while execution

Since the maximum number of threads i.e. 48 were spawned and used for computation, the CPU usage for the each thread was bordering 100%.



```
   PID USER      PR  NI   VIRT   RES   SHR S %CPU %MEM     TIME+  COMMAND
181633 sshalabh  20   0 91.9g   11g   19m S 707.3  9.3   3:16.04 java
101855 sshalabh  20   0 3072m 712m   19m S   1.9  0.6  24:13.28 java
```

9. Expectation

We experimented with Regroup-Allgather and Allreduce. We used three procedures each with Regroup-Allgather and Allreduce. Below are the procedures:

Regroup-Allgather

1. 2 Mappers with Muti-Threading
2. 2 Mappers without Multi-Threading
3. 1 Mapper without MultiThreading

Allreduce

1. 2 Mappers with Muti-Threading
2. 2 Mappers without Multi-Threading
3. 1 Mapper without MultiThreading

We expected Regroup-Allgather with multi-threading to outperform all other versions because calculation of centroids is parallel in case of Regroup-Allgather (Highlighted in figure 2) but sequential in Allreduce. But our experiments has surprised us as Allreduced has performed better.

10. CONCLUSION

We can see the that the quality of cluster(positive predictive value) is much better for the all-reduce without multi-threading paradigm as compared to others when the batch size is being altered. Also the clustering process is surprisingly fastest for the all reduce without multi threading. Surprisingly time taken by allreduce without multi-threading and with two mappers is out performing others. We feel that as our data set was quite small and we spawned maximum number of threads to do the computation, the overhead involved in the communication between the threads might have caused the performance of multi thread to suffer.

As suggested to us by BingZhang and Professor Judy Qiu, we reduced the number of threads being used. So now in the submitted code we are using 70% of the total threads that can be spawned. But we experienced that Allreduce is taking more time than it was taking before in case of 100% percent usage of threads. So it is expected that the performance might improve for multi-threading if we toggle the number of threads to find the sweet range where the performance might be optimum and if we use large data-set with high batchsize.

11. ACKNOWLEDGMENTS

Our thanks to Professor Judi Qiu, Po Beng and Bingjing Zhang for guiding and helping us during the term project.

12. REFERENCES

[1] L. Bottou and Y. Bengio. Convergence properties of the k means algorithm. In Advances in Neural Information Processing Systems. 1995.

[2] Paul S. Bradley, Usama M. Fayyad, and Cory Reina. Scaling clustering algorithms to large databases. In Rakesh Agrawal, Paul E. Stolorz, and Gregory Piatetsky-Shapiro, editors, KDD,pages 9–15. AAAI Press, 1998.
[3] Charles Elkan. Using the triangle inequality to accelerate k-means. In Tom Fawcett and Nina Mishra, editors, ICML, pages 147–153. AAAI Press, 2003
[4] J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra. Efficient projections onto the l1-ball for learning in high dimensions. In ICML '08: Proceedings of the 25th international conference on Machine learning, 2008.
[5] C. Elkan. Using the triangle inequality to accelerate k-means. In ICML '03: Proceedings of the 20th international conference on Machine learning, 2003.
[6] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. J. Mach. Learn. Res., 5, 2004.
[7] D. Witten and R. Tibshirani. A framework for feature selection in clustering. To Appear: Journal of the American Statistical Association, 2010.
[8] X. Wu and V. Kumar. The Top Ten Algorithms in Data Mining. Chapman & Hall/CRC, 2009.