Mohd. Nabil Murshed mnabilmu@qatar.cmu.edu

Project 5 Write up

15-440, Fall 2014

Performance Comparison between Sequential, MPI and MapReduce:
On very large datasets the K means algorithm ran the fastest when using MPI followed by MapReduce and the slowest being the Sequential implementation. However, on a smaller dataset size of 5000 points, Sequential outperformed MapReduce by a large margin and MPI was still the fastest. (Results shown below)

Sequential implementation of K means on 5000 points with 5 clusters took 0.010 Secs

Cluster 0 has 1000 points
Cluster 1 has 907 points
Cluster 2 has 1000 points
Cluster 3 has 1093 points
Cluster 4 has 1000 points

MPI Implementation of K means on 5000 points with 5 clusters took 0.009718

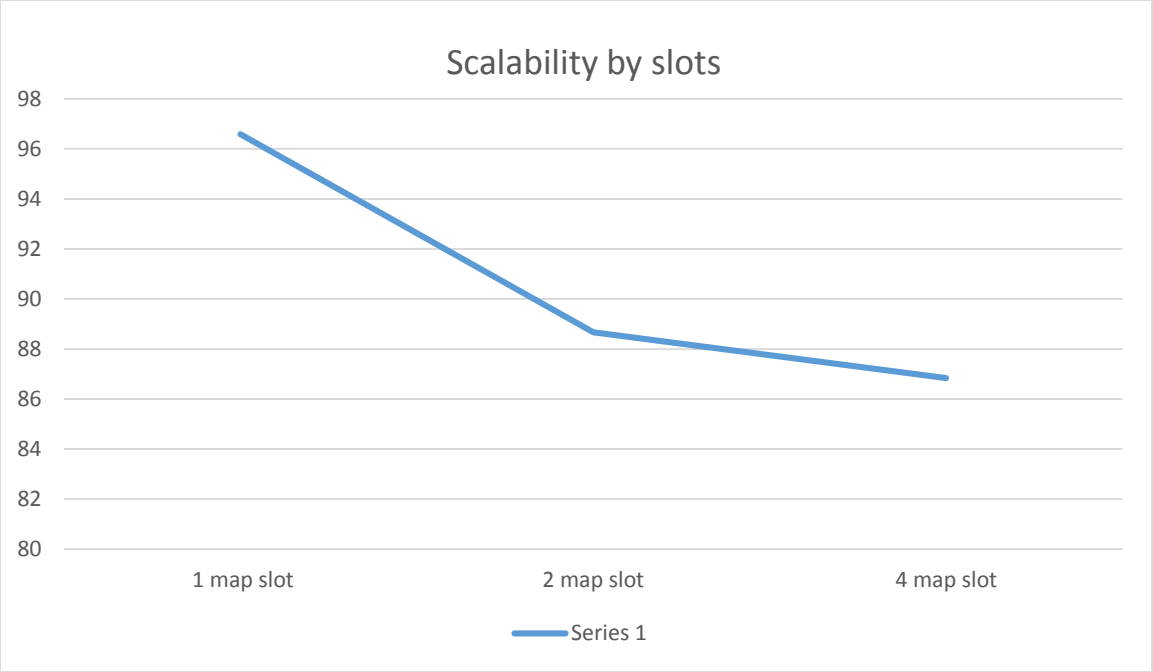
Mapreduce Implementation of K means on 5000 points with 5 clusters in two iterations took 88.662 Sec

Cluster1	1004
Cluster2	1008
Cluster3	1000
Cluster4	528
Cluster5	1460

Development Efforts Comparison between Sequential, MPI and MapReduce
MapReduce was the easiest to develop amongst the three implementations. This is because Hadoop takes care of most of the background implementations and the libraries and tasks are automatically run and need not be implemented by the user. Sequential was the next most easiest to implement due to its straight forwardness. However, a programmer has to implement each and every functions since nothing is premade. MPI was seemingly hard to implement due to its delicate nature. It is very easy to make programming errors due to bad sequence of instructions etc.

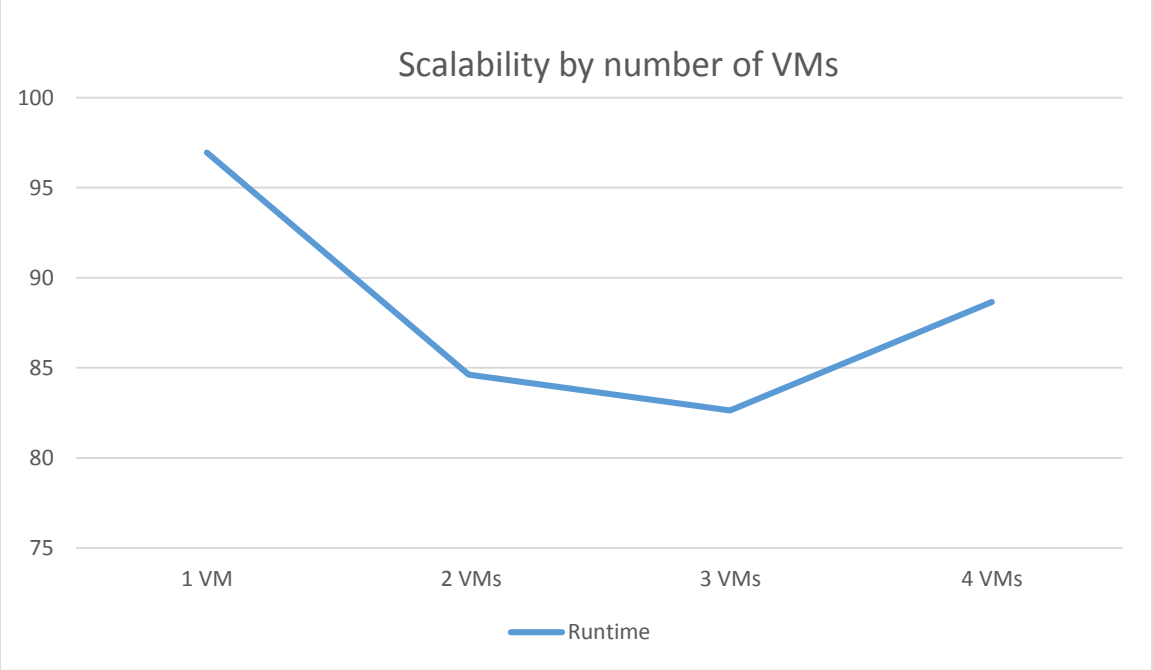
SCALABILITY:

MapReduce Scalability of 2D K means (1,2 and 4 map slots):
Runtime using 1 map slot : 96.584 secs
Runtime using 2 map slot : 88.662 secs
Runtime using 4 map slot: 86.839 secs



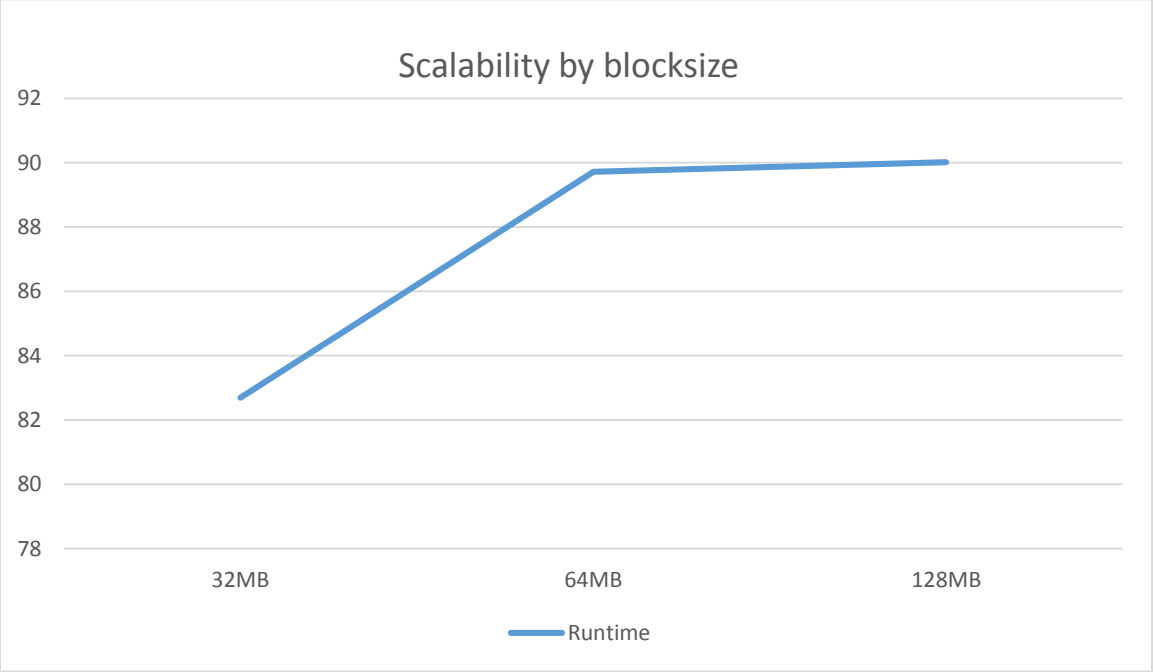
We can see that performance improves with the increase in map slots. This is because more map slots are available to increase the number of parallel tasks. Therefore, increase in map slots increases parallelism and hence performance.

MapReduce Scalability of 2D K means (1,2 3 and 4 VMs):
1VM time : 96.953
2VM time : 84.619
3VM time : 82.638
4VM time : 88.662



We can see a gradual increase in performance as we add virtual machines except for when we added the 4th one. The 4th virtual machine is not used but it still contributes to overhead time. Using 3 virtual machines produced optimum performance and the 4th machine was not needed but just added overhead cost..

MapReduce Scalability of 2D K means (32MB, 64MB and 128MB):
32 block size time : 82.69800000000001 secs
64 block size time : 89.718 secs
128 block size time : 90.021 secs



Having too few tasks can degrade the performance typically when number of tasks are less than the number of nodes. Therefore, there will be idle nodes at a time when other nodes will be overloaded with large data to process. Therefore, this will create and unbalance of load.

However, too small block size can also reduce performance because of the production of a large number of tasks. This will result in too much metadata to processes and hence more time. Also, each process will have its own overhead which will be multiplied by the number of tasks.

Ideally, if there are as many blocks as there are tasks, this means that no node is remaining idle and no block has to wait for a node to become free. This is when maximum parallelism can be achieved.

Performance trade-offs between MPI and MapReduce:
One major trade-off between MPI and MapReduce is overhead cost. MapReduce manages to amortize most of the overhead cost but for regular size datasets, MPI is certainly faster. However, MPI requires a lot of communication between tasks and,

therefore, is network dependent whereas MapReduce uses shared memory and is less likely to depend on communications, and hence, less prone to network issues.

Thoughts on applicability of K-Means to MapReduce:
One issue with the k means algorithms is its non-transitivity. What ends up happening in the reducer is that there is wait for all the map tasks to finish and send their output so that the average can be calculated for the centroid recalculation. Calculating average is not a transitive operation and therefore requires all the tasks' output. MapReduce would be best for an algorithm whose tasks are independent and can be shuffled easily.

Recommendations regarding usage of MapReduce for similar algorithms:
Reducing the number of I/O accesses time per map task or per reduce task will significantly reduce the run time of the program. Also, MapReduce sometimes creates a huge volume of intermediate files that may take up a lot of memory. Therefore, it is advisable to be careful with using memory to store states, metadata etc. Furthermore, it must be notes that MapReduce generally has a large overhead and this cost is amortized only if the dataset is large. MapReduce is unnecessarily slow for small datasets and therefore should not be used for such tasks.