streamingRPHash: Random Projection Clustering of High-Dimensional Data in a MapReduce Framework

Jacob Franklin*, Samuel Wenke*, Sadiq Quasem*, Lee A. Carraher* and Philip A. Wilsey*
* Dept of EECS, Univ. of Cincinnati, Cincinnati, OH 45221-0030 Email: wilseypa@gmail.com

The expanding needs for analysis on large datasets has increased as the amount and availability of data continues to grow. The size and format of this data makes manual analysis infeasible and has motivated the drive for automated methods such as data clustering. Among the most commonly used clustering algorithms, K-means has been proven as one of the most popular choice that delivers acceptable results in reasonable time. For many years, K-means has proven to be statistically efficient and easy to implement. While K-means is widely used for clustering streaming data, it has performance issues when it comes to robustness with noise, parallelism and working with very large, high data sets. In particular, K-Means (and other conventional techniques) for data clustering do not parallelize or scale well with the increasing dimensionality of data.

Due to the limited performance and scalability issues of conventional techniques, an algorithm is proposed, called streamingRPHash, that combines approximate and randomized methods from Random Projections [1] and Locality Sensitive Hashing (LSH) [2] to implement a high-performance, scalable parallel and distributed method to cluster high-dimensional streaming data. streamingRPHash can run in parallel on a single node as well as operate as a distributed algorithm in a MapReduce framework. It achieves this by distributing the seeds used to generate the random variables in the approximate methods so that all processing nodes use the same random projection and LSH maps to partition the data. Each source data vector is processed independently into a counted partition (counted using a count-min sketch [3]) that records a vector average for that partition. In the reduce step, the count-min sketches are combined and vector averages combined in a log(n) step reduction. The top K counts and vector averages are then reported as the principle clustering centroids of the data streamed to that time.

The streamingRPHash algorithm is composed of four components, namely: (i) a random projection step, (ii) a spherical decoder LSH step, (iii) a count min sketch, and (iv) a candidate centroid association and clustering step. This allows the algorithm to be used for dense region identification and can be a precursor to optimization based algorithm or can be used as a stand alone clustering algorithm. The random projection step translates high-dimensional vectors to lower dimensions with minimal error [4]. The first and second step, random projection and Locality Sensitive Hash (LSH), is an method for solving approximate or exact near neighbor search in high dimensional spaces. It takes an input data point, which is a vector with many values, and converts it to a single number. The location of the hash still represents its location in relation to the other data points making use of Euclidean geometry. It differs from conventional and cryptographic hash functions in that it maximizes probability of combining similar items.

The third component of the streamingRPHash formula is a Count-Min sketch [3]. In computing, a Count-Min sketch is a probabilistic data structure that serves as a frequency table of the events when data is streamed. It uses hashing to map events to frequencies, but unlike a hash table uses only a sub-linear space, at the expense of over counting some events due to collisions. In a basic version of the Count-Min sketch, it consumes a stream of events one at a time, and counts the frequency of the different types of occurrences. The sketch can be queried at any time for the frequency of a particular event type $i (0 \leq i \leq n$ for some $n)$.

The final step of the streamingRPHash algorithm is to cluster on the reconstructed candidate centroids. For this step we use simple serial version of K-Means clustering. This may seem to go against to the goal of the algorithm, but K-Means has very quick run time on small data sets. Since the algorithm already has vastly reduced our data set to a small number of candidate centroids, the K-Means execution time is insignificant when compared to the overall run time of algorithm. This final K-Means step allows us to reduce from the number of candidate centroids down to the desired number of final centroids. Having more candidate centroids than final centroids increases the accuracy of our results. The number of candidate centroid's chosen is a trade off between accuracy and speed.

The architecture of the streamingRPHash algorithm also lends itself to distribution using a map reduce framework. Incomming data is sent from a head node to multiple slave nodes. On each of these slave nodes the locality sensitive hash step is executed on these slave nodes and the result is sent back to the head node. A count min sketch and centroid priority queue are maintained on the head node. These objects are maintained in the same way on the head node as they as in the non distributed algorithm. This works well for the same reason the algorithm parallelizes well on a single node, the locality sensitive hash step requires no intra process communication and is the most expensive part of the algorithm computationally.

Experimental assessment is performed comparing the implementation of streamingRPHash to an implementation of StreamingKMeans as described in [5]. Experiments were performed using both a labeled data and randomly generated data sets. Both data sets were written to text files and read into the tests before running in order to have consistent data between multiple tests. However, the time used to read the file

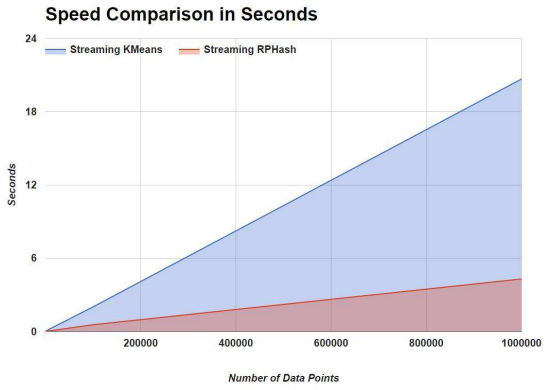

Fig. 1: Runtime comparison of streamingKMeans and streamingRPHash.

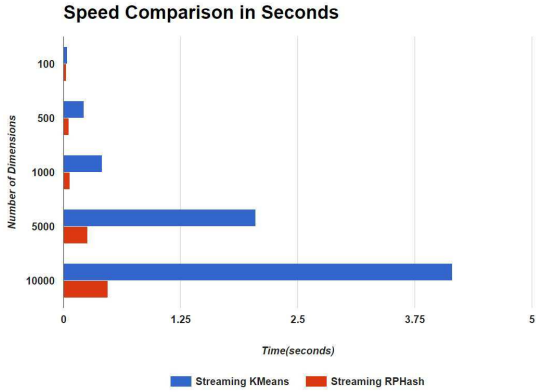

Fig. 2: Runtime Comparison based on Dimensionality.

into memory was not accounted for in the execution time of any experiments. Runtime and the within-cluster sum of squared errors (WCSSE) metric is captured for both streamingRPHash and StreamingKMeans.

Using the randomly generated data, streamingRPHash's performance in relation to data set size was tested and compared to streamingKMeans (Figure 1). The graphs show that streamingRPHash's run time scales linearly when compared to the size of the dataset being clustered. StreamingKMean's run time also scales linearly in relation to the size of the test data set. Both streamingKMeans and streamingRPHash had a constant average squared distance accuracy measurement between 41 and 42 for all tests. This indicates that streamingRPHash created clusters that were similarly accurate to the results produced by streamingKMeans. Other than varying the test data set size 100 and 1,000,000, all other variables were held constant during these tests, the dimensionality of the data set was 500, 10 clusters were requested and the tests were run on a single 4 core computer. If the tests were run on data sets of a larger sizes, the execution time continue to increase in a linear fashion.

streamingRPHash's and streamingKMean's performance in relation to the number of dimensions being clustered was also tested. As with the previous test everything other than the variable being tested was held constant. There data set size was 10,000, the number of clusters requested was 10, and the number of dimensions was varied between 100 and 10,000. The results are show in Figure 2. As the number of dimensions increased streamingRPHash saw an increasingly large ratio in terms of execution speed when comparing it to streamingKmeans. For example at the first measurement, 100 dimensions streamingRPHash was 1.3 times faster than streamingKMeans. By the time the last reading was taken at 10000 dimensions, streamingRPHash was 8.75 times faster. Part of the speed increase can be attributed to the parallel nature of streamingRPHash which allowed it to take advantage of the 4 cores on the testing computer. But, the increasing gains demonstrated by this test indicated that streamingRPHash is well suited to high dimensionality data even outside of its parallel nature. streamingRPHash's and streamingKMeans saw similar WCSSE accuracy measurements at all dimensionality sizes. The memory usage during these tests was asymptotic maxing out at about 1GB of local memory for the streamingRPHash test and about 200MB for the streaming K-Means test

The streamingRPHash algorithm uses qualities of past algorithms which have seen wide success in various fields. It allows us to combine approximate and random projection methods in a unique but effective way to solve scalability issues encountered by other popular solutions. Due to its random nature, it also has the innate ability to provide greater security for clustering of distributed data. Speed ups were observed using streamingRPHash when compared to $Streaming\ k-Means$ as the data's dimensionality increases. $Streaming\ k-Means$ starts to get exponentially slower with higher dimensionality data, whereas streamingRPHash has linear increase in speed, which is constant and makes it a great candidate for high dimensionality live streams of data. The streamingRPHash *golang* implementation is open source and can be found at https://github.com/wilseypa/rphash-golang.

Acknowledgments: Support for this work was provided in part by the National Science Foundation under grant ACI–1440420.

[1] D. Achlioptas, "Database-friendly random projections," in *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2001, pp. 274–281.

[2] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, ser. STOC 98. New York, NY, USA: ACM, 1998, pp. 604–613. [Online]. Available: http://doi.acm.org/10.1145/276698.276876

[3] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[4] Y. Bartal, B. Recht, and L. J. Schulman, "Dimensionality reduction: Beyond the johnson–lindenstrauss bound," in *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, 2011, pp. 868–887.

[5] M. Shindler, A. Wong, and A. W. Meyerson, "Fast and accurate k-means for large datasets," in *Advances in Neural Information Processing Systems*, 2011, pp. 2375–2383.