# Massively Scalable Parallel KMeans on the HPCC Systems Platform

Lili Xu*†, Amy Apon*, Flavio Villanustre†, Roger Dev† and Arjuna Chala†
*School of Computing
Clemson University, Clemson, USA
Email: lilix@clemson.edu, aapon@clemson.edu
†HPCC Systems
LexisNexis Risk Solutions, Alpharetta, USA
Email: Flavio.Villanustre@lexisnexisrisk.com, Roger.Dev@lexisnexisrisk.com, Arjuna.Chala@lexisnexisrisk.com

***Abstract*—Clustering algorithms are an important part of unsupervised machine learning. With Big Data, applying clustering algorithms such as KMeans has become a challenge due to the significantly larger volume of data and the computational complexity of the standard approach, Lloyd's algorithm. This work aims to tackle this challenge by transforming the classic clustering KMeans algorithm to be highly scalable and to be able to operate on Big Data. We leverage the distributed computing environment of the HPCC Systems platform. The presented KMeans algorithm adopts a hybrid parallelism method to achieve a massively scalable parallel KMeans. Our approach can save a significant amount of time of researchers and machine learning practitioners who train hundreds of models on a daily basis. The performance is evaluated with different size datasets and clusters and the results show a significant scalabilty of the scalable parallel KMeans algorithm.**

***Index Terms*—Scalable KMeans, Hybrid Parallelism, Machine Learning, High Performance Computing, HPCC Systems**

## I. Introduction

As a result of the fast development of data-intensive industries such as social media, online shopping, and Internet of Things, new datasets and sources of data are becoming available at an unprecedented rate. With this growth in size and speed of data, traditional machine learning algorithms such as KMeans [1] face a challenge to process data sizes that can no longer fit into main memory of a single computer.

Compounding the problem of larger data sizes, algorithms like KMeans depend on certain hyper-parameters to identify the most optimal model, which requires the execution of the algorithm multiple times with different values for the hyper-parameters. In the case of KMeans, a pre-defined hyper-parameter $K$, representing the number of centroids, and the starting values for the centroids, are required to train the model. In many cases, machine learning practitioners need to execute the algorithm a large number of times with hundreds of these values, which is very time and resource-consuming.

To meet these challenges, in this paper we introduce a massively scalable parallel KMeans to cluster Big Data that leverages the distributed computing environment of the HPCC Systems platform.

### A. HPCC Systems

HPCC Systems (High Performance Computing Cluster Systems) [2] is an open source, data-intensive computing system platform developed by LexisNexis Risk Solutions in 2000. It is a highly scalable distributed computing system based on commodity hardware. It includes system software that provides a distributed file storage system, job execution environment, online query capability, parallel application processing, and parallel programming development tools [2].

In a manner similar to Hadoop, HPCC Systems uses clusters of commercially available off-the-shelf (COTS) [3] hardware running the Linux operating system. The platform includes additional system software and middleware components to provide a complete and comprehensive job execution environment along with a distributed query and file systems support that are needed for data-intensive computing [2]. In HPCC Systems, datasets are stored in the Dali File System as tabular records, where each record corresponds to a row and the computation is implicitly parallel.

*1) Architecture:* Fig. 1 illustrates the overall software architecture of the HPCC Systems environment. HPCC Systems includes the Thor [2] and Roxie [5] clusters, as well as common middleware components, an external communications layer, client interfaces that provide both end-user services and system management tools, and auxiliary components to support monitoring, and to facilitate loading and storing of file system data from external sources. It includes system configurations to support both parallel batch data processing (Thor) and high-performance online query applications using indexed data files (Roxie) as shown in Fig. 2.

The computing engine Thor is based on a master-slave architecture as shown in Fig 3. Each physical Thor node is capable of running multiple processes to offer configuration flexibility. A Roxie cluster is similar in its function to Hadoop with HBase and Hive capabilities added, but with high-performance online processing that can efficiently

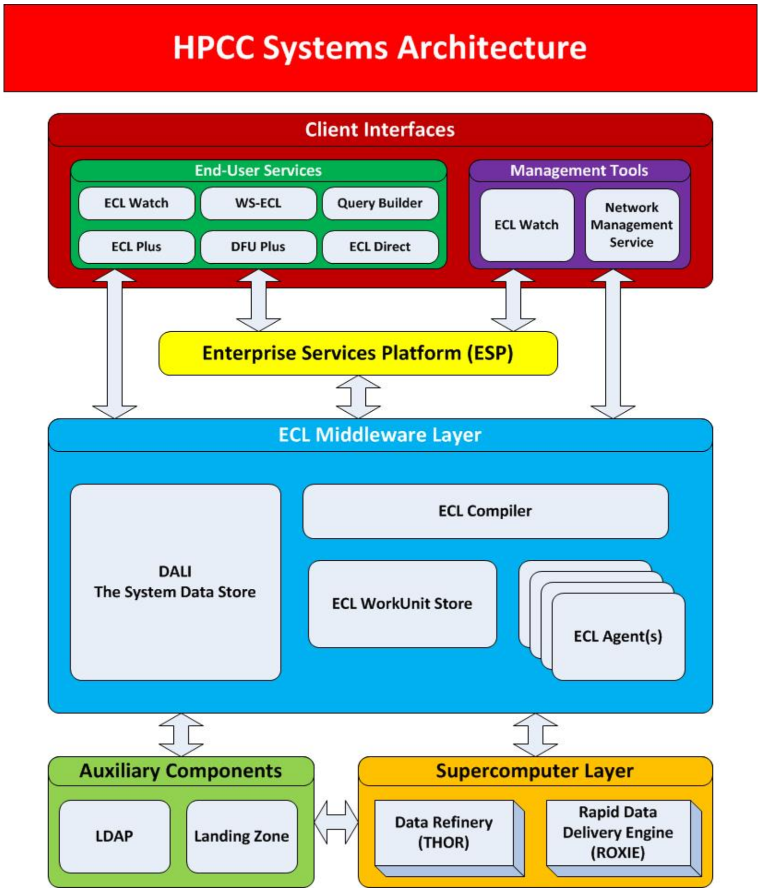


Fig. 1. HPCC Systems Architecture

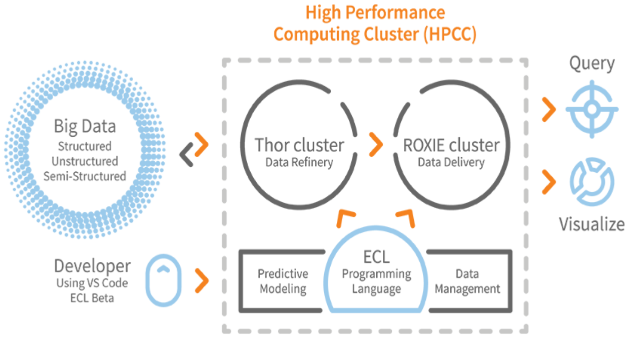


Fig. 2. Use of HPCC Systems for Big Data Processing [4]

execute multiple queries in parallel [2]. Both Thor and Roxie clusters utilize the same ECL programming language to implement applications, which can lead to higher programmer productivity.

HPCC Systems also has an extensible library of native machine learning routines [6]. It is an extensible set of fully scalable parallel Machine Learning and matrix processing algorithms covering supervised and unsupervised learning, such as decision trees, KMeans, regression analysis, statistics and probabilities.

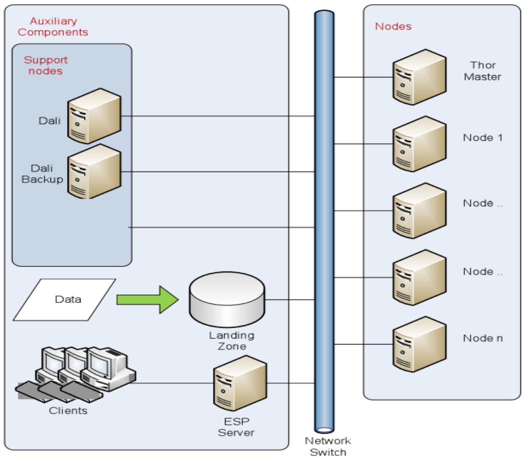


Fig. 3. Thor Computing Cluster

*2) Programming Framework:* Data operations and data queries in the HPCC Systems platform are defined using a declarative dataflow programming language called ECL [7]. ECL is a data-centric parallel processing language designed specifically for the data-intensive computing environment of HPCC Systems. It plays an important role in making HPCC Systems excel at data-intensive Extract, Transform and Load (ETL) tasks and Big Data analytics.

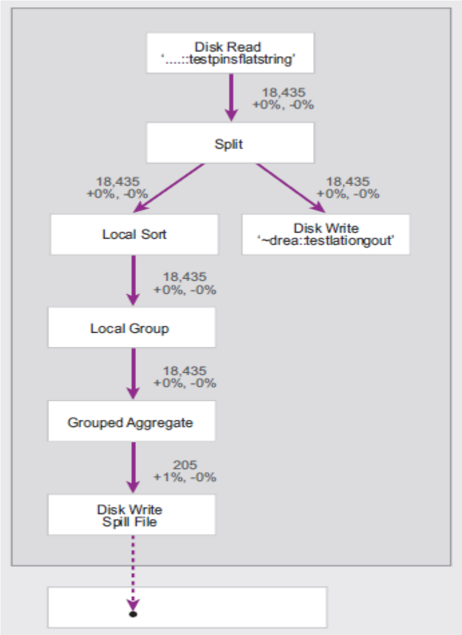
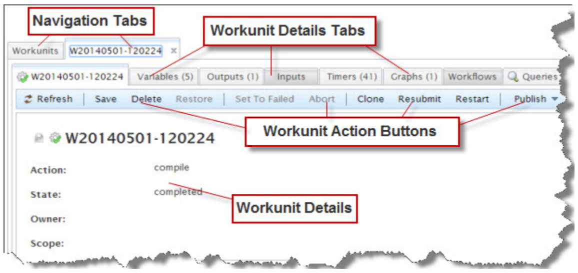


Fig. 4. Dataflow Graph of HPCC Systems [2]

An ECL program, also known as a ”WorkUnit” or ”Query”, declaratively defines activities as nodes in a dataflow graph as shown in Fig. 4, where edges represent the data traversal across these activities. The graph is displayed for the user via the ECL-Watch interface [8].



W20140501-120224

Action: compile

State: completed

Owner:

Scope:

Fig. 5. Workunit Detail Page

*3) Performance Monitor:* In HPCC Systems, after an ECL program is submitted, the ECL compiler will, as a side effect, output a full execution plan rendered as a visual dataflow graph, accessible through the ECL-Watch management web tool [8]. This visual dataflow graph plays a key role to further optimize the Big Data applications on the HPCC Systems platform [9].

In addition to accessing the visual execution plans, ECL-Watch provides detailed timing information (see Fig. 5) that can be leveraged to better understand the use of resources and the activities that take the longest, and to direct the optimization work to those areas that can provide the biggest amount of benefit with the least amount of effort [9]. Our work utilizes the built-in ECL distributed computing jobs and the dataflow graph in ECL-watch to optimize Big Data jobs in HPCC Systems.

### *B. KMeans Clustering Algorithm*

Machine learning algorithms can be roughly categorized into two types based on their requirement for training data: supervised learning and unsupervised learning. Unsupervised learning is comprised of machine learning algorithms that do not require labeled training data. Within the unsupervised category, KMeans is by far the most popular clustering algorithm and is extensively used in scientific and industrial applications [10]. KMeans has been widely used in many different fields, including education [11], agriculture [12], fraud detection [13], public transportation [14], IoT management [15].

KMeans groups an unlabeled dataset of $p$ observations into a predefined $K$ number of clusters, so that the resulting clusters have high intra-cluster similarity. If centroids are $m_1, m_2, ...m_k$, and partitions are $c_1, c_2, ...c_k$, then one can calculate the inertia of KMeans as:

$$\sum_{k=1}^{K}\sum_{i\in c_k} ||x_i - m_k||^2 \qquad \text{Euclidean distance} \qquad (1)$$

To initiate the clustering process, the KMeans algorithm takes the number of clusters K and the initial position of the centroids (the center of each cluster) as model parameters. The initial centroids can be randomly chosen or identified using optimization methods to find better initial centroids, such as what is done in K-means++ [16].

After initialization, the KMeans model starts an iterative process which keeps updating the position of the centroids and their associations with the observations. There are two major steps in the iterative process: assignment step and update step.

With a set of $k$ centroids ($\mu$), the assignment steps first calculates the distances from each observation to each centroid via Euclidean Distance [1]. All observations $p$ are assigned to their nearest centroid as described by [17]. The observations assigned to the same centroids are regarded as a cluster (see equation 2).

$$S_i^{(t)} = \left\{x_p : \left\|x_p - \mu_i^{(t)}\right\|^2 \le \left\|x_p - \mu_j^{(t)}\right\|^2 \; \forall j, 1 \le j \le k\right\} \qquad (2)$$

The next step is the update step. The centroids are updated or repositioned by calculating the new mean of the assigned observations, which is the mean of the least squared Euclidean distance (see equation 3).

$$\mu_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j \tag{3}$$

By iterating between assignment and update steps, the KMeans algorithm tries to optimize the objective function.The final position of the centers are generated and the iterations stop when all observations remain at the assigned centroids and therefore the centroids would not be updated anymore.

In practice, a maximum number of iterations and a converge rate or tolerance are defined to avoid an indefinite number of iterations and unnecessary computations. In this case, the KMeans algorithm will stop iterating once it reaches the maximum number of iterations or the movement distance of each center is smaller than the tolerance.

The rest of the paper is organized as follows: Section II provides the related works on the scalable KMeans algorithm. In Section III, we present our approach to implementing a massively scalable parallel KMeans algorithm on the HPCC Systems platform. Section IV introduces the experimental environment and experiment results. Section V is the conclusion of our work. We present our future work in Section VI.

## II. Related Works

Most of the traditional clustering [1] [18] [19] algorithms are designed for centralized computation environments. As data sizes become larger, traditional clustering methods are challenged. To meet this challenge, parallel and distributed clustering algorithms have been implemented, such as [20] [21] [22]. In this session, we introduce some of the parallel and distributed KMeans implementations described in the literature.

With the advent of multi-processor computing machines, the development of parallel KMeans was motivated by these multi-processor architectures such as [20] [23] [21]. In addition to focusing on parallel computing architectures, other variances of parallel KMeans have been published such as the parallel fuzzy KMeans [24] and privacy preserving parallel KMeans [25].

Another important factor that pushed forward the development of parallel Kmeans implementations is the invention of Graphic Processing Unit (GPU) and the use of GPUs in general computing problems (GPGPU). For example, [26] [27] [28] [29] utilize the shared-memory computing structure and high performance GPUs to implement parallel KMeans.

In addition to these parallel implementations of KMeans, scalable implementations of KMeans are available using High Performance Computing (HPC). For example, [30] implements high performance KMeans using Message Passing Interface (MPI) in an HPC distributed computing environment. Other research such as [31] [32] shows scalable implementations based on different distributed frameworks and platforms.

In this paper, we present a massively scalable parallel KMeans implementation to cluster very large datasets on the HPCC Systems platform. There are some works focused on implementing the KMeans algorithm to cluster massive datasets such as [33] [34]. As far as we know, we demonstrate a scalable application of KMeans to datasets that can be much larger than that of previous research.

The common traits between the related works and the current work is that all of them focus on clustering large scale massive datasets. The work presented here can be utilized in a single machine and also can be executed in a distributed High Performance Computing environment. In our experiments, the cluster scale ranges from 1 to 400 nodes. Moreover, the scale of the experimental dataset is much larger than those of previous research. In our experiments, the size of these datasets ranges from a few hundred to hundreds of millions of records. In the following section, we will present the details of our methods.

## III. Approach

To maximize the parallelization and minimize the inter-node communication, our key approach to implementing a highly scalable KMeans on HPCC Systems uses data parallelism. We also show how features of HPCC Systems can be leveraged during specific training phases by allowing it to train many models concurrently on very large datasets on commodity hardware [35].

*1) Data Parallelism Method:* The KMeans algorithm groups $p$ observations into $K$ clusters by iteratively updating the centers until convergence. At each iteration, the computation performs two types of distance calculations: the distance between each observation to each center and the distance movement of each center compared to the last iteration. The computation cost of these two types of distance calculation are $O(nkd)$ and $O(kd)$ respectively, where $n$ is the number of observations, $k$ is the number of cluster and $d$ is the dimensions of the clustering space.

In general, the distance calculations between each observation and each center are more expensive than the distance calculations for the movement of the centers. For Big Data applications in particular, the distance calculation between each observation and each center is significantly more expensive due to the number of observations $n$ being much bigger than $k$ and $d$.

Based on above analysis, our algorithm design aims to use Data Parallelism to maximize the parallelization of the distance computations between observations and centroids and minimize the inter-node communication as much as possible.

**Algorithm 1** Data Parallelism

```
Input: Observations $O$, Centers $C$
Output: Groups $G$
1 Distribute O to each node evenly
2 while Not Converged do
3     Broadcast C to each node;
4     Locally calculate the distance between each observation
        and each center;
5     Locally calculate and assign the closest center to each
        observation;
6     Locally partial update the values of each center;
7     Globally update the number of cluster members;
8     Globally update the values of C;
9 Result clustered Groups $G$
```

Our implementation of the KMeans algorithm calculates all the distances locally via embarrassing parallelism. Most of the assignment and update steps are also executed in parallel, except the global update of the centers. The pseudocode of Data Parallelism of the KMeans algorithm on the HPCC Systems platform is shown in Algorithm 1.

As we can see from the pseudocode, the observations are first evenly distributed to each node. Then the centers are broadcast to each node so that each node has a full copy of all the centers. This step initiates the data parallelism and allows the following steps to be locally calculated without any communication cost, until the global update of the centers.

Note that in Step 4, the observations closest to the same center are grouped as a cluster or a group. In other words, they are all members of the same cluster. In Step 5, the value of centers are locally updated by summing up the values of the local cluster members. Since the members of each cluster can be located in different nodes, the local value of each center needs to sum up and update globally as shown in Step 7 for the accurate final result to go into the next iteration if not converged.

In HPCC Systems, datasets are stored in the Dali File System as records, where each record corresponds to a row. The computation on HPCC Systems is not only implicitly parallel but also allows users to explicitly control the data distribution and customize the distribution methods, such as broadcasting or gathering.

To implement the presented KMeans, we utilize both the implicit and explicit parallelism on HPCC Systems. For example, to distribute the observations evenly on each node and broadcast the centers, we use explicit distribution. As a result of this explicit distribution, the closest center of each observation can be calculated and assigned locally and the values of each center can be partially updated locally, as well. The final step is to globally update the value of each center by gathering the local value of each center.

*2) Notes on Model Parallelism in HPCC Systems:* In addition to the scalability to support clustering of massive datasets, the presented work also enables model parallelism, that is, the ability to run massive number of models simultaneously, leveraging the programming model of HPCC Systems. Model parallelism is when each replica trains over the same or different data but uses different part of the model [36]. Model parallelism also allows for large number of models to run efficiently in parallel to save modeling time.

Model parallelism is essential for Big Data because the training process of Big Data applications usually takes hours or even days. Imagine needing to run a model on 10,000 different datasets with 10,000 different hyperparameters. This would need a total of 100,000,000 models to run sequentially.

**Algorithm 2** Model Parallelism

```
Input: Groups of Observations $GO$, Groups of Centers $GC$
Output: Groups of Cluster $GG$
1 Distribute $GO$ to each node evenly
2 while Not Converged do
3     Broadcast $GC$ to each node;
4     Locally calculate the distance between each observation
        and each center in each group;
5     Locally calculate and assign the closest center of each
        observation in each group;
6     Locally partial update the values of each center in each
        group;
7     Globally update the number of cluster members in each
        group;
8     Globally update the values of $GC$ of different group;
9 Result clusters of different group $GG$
```

HPCC Systems enables the execution of a large number of models simultaneously on different datasets, leveraging the Myriad Interface and computing framework of HPCC Systems. The Myriad Interface allows a user to perform multiple independent machine learning activities within a single interface invocation [37]. The pseudocode of the massively Scalable Parallel KMeans algorithm on HPCC Systems platform is as shown in Algorithm 2.

The model inputs of Algorithm2 are different from Algorithm1: different groups of observations and different groups of centers instead of a single group of observations and a single group of centers. With the help of the machine learning dataframe of HPCC Systems, we are able to map datasets with its centers in different groups.

*3) Combining Data Parallelism and Model Parallelism:* The implemented KMeans algorithm adopts both data parallelism and model parallelism to enable KMeans calculations on massive datasets and also with a large count of hyper-parameters. In this section, we introduce the methods that we use to realize this hybrid parallelism in HPCC Systems.

HPCC Systems is a Big Data processing platform with implicit parallel computing. It also allows users to control the data distribution method explicitly via the D̂istribute function by specifying the methods in its êxpression. This is the key function to realize data parallelism. Further more, the machine learning library of HPCC Systems provide core dataframes such as the 'AnyField' dataframe, which are essential to realize the myriad interface or model parallelism.

Thus, in HPCC Systems the hybrid parallelism can be realized by utilizing the flexible data distribution control and Myriad Interface. As seen in the base dataframe 1, each data record is assigned a model identifier(wi) and an observation identifier(id). Based on these two identifiers, the D̂istribute function can distribute the data based on the wi and id of each record, allowing data and model parallelism in HPCC Systems.

Listing 1. Hybrid Parallelism Dataframe

```
EXPORT AnyField  := RECORD
    // model identifier
    t_Work_Item wi;
    // observation identifier
    t_RecordID id;
    // feature identifier
    t_FieldNumber number;
END;
```

## IV. Experiment Environment and Results

To evaluate the performance of the Scalable Parallel KMeans on HPCC Systems platform, experiments are conducted on various clusters with different number of nodes. The version of HPCC Systems for performance evaluation is 6.4.0. The size of different clusters are 1 node, 4 nodes, 20 nodes, 25 nodes and 400 nodes, each of which has 2 x Intel(R) Xeon(R), 2.10GHz, 8 cores, 8GB memory, 140GB Micron 5100 SSD and 5Gbps network adapters.

Our experimental data is the public Iris flower dataset [38], which is widely used in the fields of data mining, machine learning, and clustering. The original Iris dataset has 150 records and five attributes. To evaluate the accuracy and scalability of the presented work, the original Iris dataset is replicated to increase the data volume. As a result, the experimental data sizes range from 150 records to 600,000,000 records. We also apply our method to a second test dataset, the public CoverType dataset, which is obtained from [39]. In the CoverType dataset each record represents the cartographic data for one plot of Rocky Mountain forest. The dataset has 54 attributes and 500 million records.

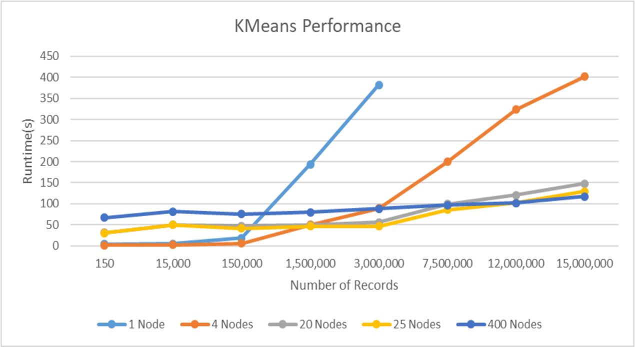
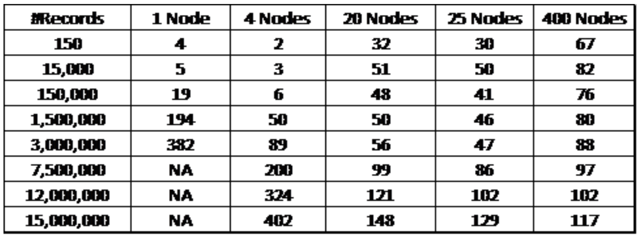


Fig. 6. KMeans Experimental results, Iris dataset



| #Records | 1 Node | 4 Nodes | 20 Nodes | 25 Nodes | 400 Nodes |
|---|---|---|---|---|---|
| 150 | 4 | 2 | 32 | 30 | 67 |
| 15,000 | 5 | 3 | 51 | 50 | 82 |
| 150,000 | 19 | 6 | 48 | 41 | 76 |
| 1,500,000 | 194 | 50 | 50 | 46 | 80 |
| 3,000,000 | 382 | 89 | 56 | 47 | 88 |
| 7,500,000 | NA | 200 | 99 | 86 | 97 |
| 12,000,000 | NA | 324 | 121 | 102 | 102 |
| 15,000,000 | NA | 402 | 148 | 129 | 117 |

Fig. 7. Single Node Configuration Runs Show Out of Memory Compared to Runtimes on Various-Sized Clusters (Time in Seconds, Iris dataset)

The performance of the KMeans implementation is shown in Fig. 6 and Fig. 7. From the results we can see that:

1) The single node configuration shows better performance only when the data size is less than 150,000 records. When the data size is too small, other configurations' performance is heavily affect by the communication overhead. However, when the data size is larger than 150,000 records, all of the configurations outperform the single node configuration. Moreover, the single node configuration runs out of memory when the data size increases to 7,500,000 records as we can see in Fig. 7.
2) As the data sizes increase, the runtime of the single node and 4 nodes configurations increase rapidly. In contrast, the 20, 25 and 400 nodes configurations show good

scalability with much less run time and lower run time increase rate as show in Fig. 6.

If we remove single and four-node configurations from the graph, the results are as shown in Fig. 8. We can see that all nodes scale reasonably well and 400 nodes eventually outperform other clusters when the data is large enough.

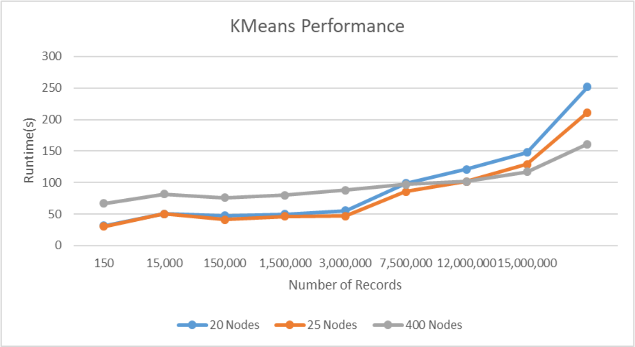


Fig. 8. KMeans Runtime on Larger Cluster, Iris dataset

To further evaluate the performance of our method, we conduct experiments on massive datasets, as shown in Fig. 9. The record count ranges from 15,000,000 to 600,000,000. The testing clusters have 4, 20 and 400 nodes respectively.

Fig. 10 shows that the 400-node cluster configuration has the best performance for all massive datasets. As data sizes increase, the 400-node cluster configuration has outstanding performance as compared to the smaller cluster configurations, with a lower run time increase rate as shown in Fig. 9.

We validate the accuracy of our approach by comparing the results of clustering experimental data to the KMeans implementation in scikit-learn [40]. The comparison shows that the output of the methods is the same.

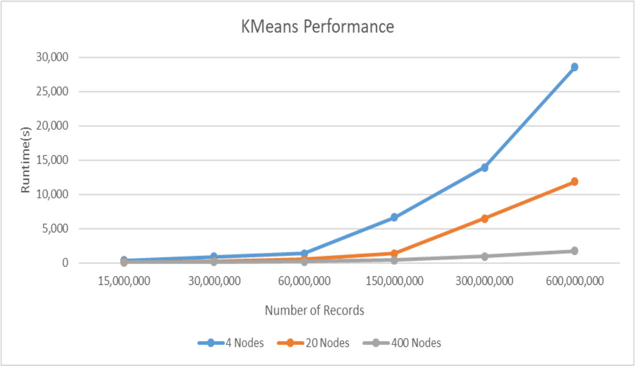
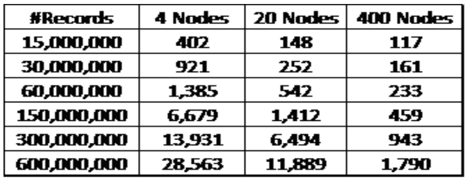


Fig. 9. KMeans Runtime on Massive Datasets

To further demonstrate the scalabiliy of the presented work, another set of experiments are conducted with the CoverType dataset with higher dimensions. We randomly sampled 5,000 records as base dataset and test the scalability as the same manner as testing Iris dataset. The result also shows impressive scalability as seen in Fig. 11.



| #Records | 4 Nodes | 20 Nodes | 400 Nodes |
|---|---|---|---|
| 15,000,000 | 402 | 148 | 117 |
| 30,000,000 | 921 | 252 | 161 |
| 60,000,000 | 1,385 | 542 | 233 |
| 150,000,000 | 6,679 | 1,412 | 459 |
| 300,000,000 | 13,931 | 6,494 | 943 |
| 600,000,000 | 28,563 | 11,889 | 1,790 |

Fig. 10. Runtimes on Massive Datasets (Time in Seconds, Iris dataset)

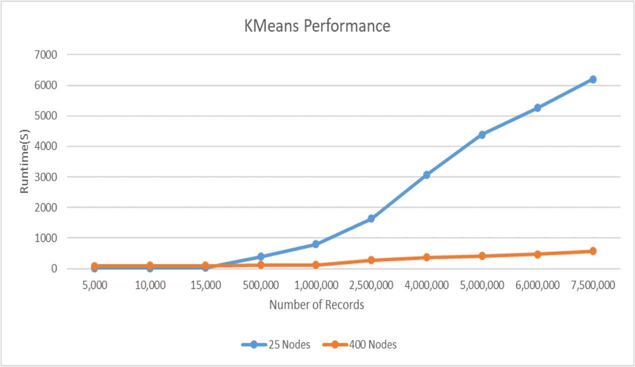


Fig. 11. KMeans Runtimes on CoverType Dataset (Time in Seconds)

Our implementation [41] is open source and can be accessed via https://github.com/hpcc-systems/KMeans.

## V. Conclusions

To tackle the challenge of clustering big datasets we present a massively scalable parallel KMeans method that leverages the distributed computing environment of the HPCC Systems platform. Our key contribution is to leverage data parallelism in the distributed environment and an efficient method for global update of KMeans centroids. We also call out the model parallelism available in HPCC Systems. We report the performance of the implementation on a large range of dataset sizes on several clusters with different node counts. The experimental results show that massive datasets that cannot be processed on a single node environment can be processed effectively in parallel using a distributed cluster. The results show significant speedup and scalability up to a cluster of size 400 nodes on a dataset of 600,000,000 records.

## VI. Future Work

In future work we will first explore optimization of the positions of the initial centroids. The current implementation does not consider optimal starting positions. Secondly, we will study the performance on additional types of datasets and on additional sizes of clusters.

# References

[1] S. Lloyd, "Least squares quantization in pcm," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

[2] M. Anthony and C. Arjuna. (2011) Introduction to HPCC (High-Performance Computing Cluster). [Online]. Available: http://cdn.hpccsystems.com/whitepapers/wp_introduction_HPCC.pdf

[3] N. A. GENERAL SERVICES ADMINISTRATION, DEPARTMENT OF DEFENSE and S. ADMINISTRATION, "2.1-3 federal acquisition regulation," p. 46, 2019.

[4] LexisNexis. (2017) HPCC Systems. [Online]. Available: https://hpccsystems.com

[5] HPCC Systems. (2017) Roxie: The Rapid Data Delivery Engine. [Online]. Available: http://cdn.hpccsystems.com/releases/CE-Candidate-6.4.2/docs/RoxieReference-6.4.2-1.pdf

[6] ——. (2019) HPCC Systems Machine Learning Library. [Online]. Available: https://hpccsystems.com/download/free-modules/machine-learning-library

[7] ——. (2017) ECL Language Reference. [Online]. Available: http://cdn.hpccsystems.com/releases/CE-Candidate-6.4.2/docs/ECLLanguageReference-6.4.2-1.pdf

[8] ——. (2017) Using ECL-Watch. [Online]. Available: http://cdn.hpccsystems.com/releases/CE-Candidate-6.4.2/docs/The_ECL_Watch_Manual-6.4.2-1.pdf

[9] L. Xu, E. Muharemagic, and A. Apon, "Ecl-watch: A big data application performance tuning tool in the hpcc systems platform," in *Big Data (Big Data), 2017 IEEE International Conference on*. IEEE, 2017, pp. 2941–2950.

[10] P. Berkhin, "A survey of clustering data mining techniques," in *Grouping multidimensional data*. Springer, 2006, pp. 25–71.

[11] O. Oyelade, O. Oladipupo, and I. Obagbuwa, "Application of k means clustering algorithm for prediction of students academic performance," *arXiv preprint arXiv:1002.2425*, 2010.

[12] M. R. Badnakhe and P. R. Deshmukh, "An application of k-means clustering and artificial intelligence in pattern recognition for crop diseases," in *International Conference on Advancements in Information Technology*, 2011.

[13] A. Srivastava, A. Kundu, S. Sural, and A. Majumdar, "Credit card fraud detection using hidden markov model," *IEEE Transactions on dependable and secure computing*, vol. 5, no. 1, pp. 37–48, 2008.

[14] R. A. Kadir, Y. Shima, R. Sulaiman, and F. Ali, "Clustering of public transport operation using k-means," in *2018 IEEE 3rd International Conference on Big Data Analysis (ICBDA)*. IEEE, 2018, pp. 427–432.

[15] J. Stewart, R. Stewart, and S. Kennedy, "Dynamic iot management system using k-means machine learning for precision agriculture applications," in *Proceedings of the Second International Conference on Internet of things and Cloud Computing*. ACM, 2017, p. 142.

[16] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[17] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14. Oakland, CA, USA, 1967, pp. 281–297.

[18] D. Birant and A. Kut, "St-dbscan: An algorithm for clustering spatial–temporal data," *Data & Knowledge Engineering*, vol. 60, no. 1, pp. 208–221, 2007.

[19] D. E. Gustafson and W. C. Kessel, "Fuzzy clustering with a fuzzy covariance matrix," in *1978 IEEE conference on decision and control including the 17th symposium on adaptive processes*. IEEE, 1979, pp. 761–766.

[20] M. K. Ng, "K-means-type algorithms on distributed memory computer," *International Journal of High Speed Computing*, vol. 11, no. 02, pp. 75–91, 2000.

[23] D. Arthur, B. Manthey, and H. Röglin, "K-means has polynomial smoothed complexity," in *2009 50th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, 2009, pp. 405–414.

[21] I. S. Dhillon and D. S. Modha, "A data-clustering algorithm on distributed memory multiprocessors," in *Large-scale parallel data mining*. Springer, 2002, pp. 245–260.

[22] Y. Zhang, Z. Xiong, J. Mao, and L. Ou, "The study of parallel k-means algorithm," in *2006 6th World Congress on Intelligent Control and Automation*, vol. 2. IEEE, 2006, pp. 5868–5871.

[24] L. Vendramin, R. J. G. B. Campello, L. F. Coletta, and E. R. Hruschka, "Distributed fuzzy clustering with automatic detection of the number of clusters," in *International Symposium on Distributed Computing and Artificial Intelligence*. Springer, 2011, pp. 133–140.

[25] S. Patel, V. Patel, and D. Jinwala, "Privacy preserving distributed k-means clustering in malicious model using zero knowledge proof," in *International Conference on Distributed Computing and Internet Technology*. Springer, 2013, pp. 420–431.

[26] E. Kijsipongse, U. Suriya *et al.*, "Dynamic load balancing on gpu clusters for large-scale k-means clustering," in *2012 Ninth International Conference on Computer Science and Software Engineering (JCSSE)*. IEEE, 2012, pp. 346–350.

[27] P. Mackey and R. R. Lewis, "Parallel k-means++ for multiple shared-memory architectures," in *2016 45th International Conference on Parallel Processing (ICPP)*. IEEE, 2016, pp. 93–102.

[28] B. Hong-Tao, H. Li-li, O. Dan-tong, L. Zhan-shan, and L. He, "K-means on commodity gpus with cuda," in *2009 WRI World Congress on Computer Science and Information Engineering*, vol. 3. IEEE, 2009, pp. 651–655.

[29] R. Farivar, D. Rebolledo, E. Chan, and R. H. Campbell, "A parallel implementation of k-means clustering on gpus." in *Pdpta*, vol. 13, no. 2, 2008, pp. 212–312.

[30] J. Zhang, G. Wu, X. Hu, S. Li, and S. Hao, "A parallel k-means clustering algorithm with mpi," in *2011 Fourth International Symposium on Parallel Architectures, Algorithms and Programming*. IEEE, 2011, pp. 60–64.

[31] V. R. Eluri, M. Ramesh, A. S. M. Al-Jabri, and M. Jane, "A comparative study of various clustering techniques on big data sets using apache mahout," in *2016 3rd MEC International Conference on Big Data and Smart City (ICBDSC)*. IEEE, 2016, pp. 1–4.

[32] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii, "Scalable k-means++," *Proceedings of the VLDB Endowment*, vol. 5, no. 7, pp. 622–633, 2012.

[33] M. Capó, A. Pérez, and J. A. Lozano, "An efficient k-means algorithm for massive data," *arXiv preprint arXiv:1605.02989*, 2016.

[34] ——, "An efficient k-means clustering algorithm for massive data," *arXiv preprint arXiv:1801.02949*, 2018.

[35] R. K. L. Kennedy, T. M. Khoshgoftaar, F. Villanustre, and T. Humphrey, "A parallel and distributed stochastic gradient descent implementation using commodity clusters," *Journal of Big Data*, vol. 6, no. 1, p. 16, Feb 2019. [Online]. Available: https://doi.org/10.1186/s40537-019-0179-2

[36] B. T. Polyak and A. B. Juditsky, "Acceleration of stochastic approximation by averaging," *SIAM Journal on Control and Optimization*, vol. 30, no. 4, pp. 838–855, 1992.

[37] D. Roger. (2018) Understanding the Myriad Interface feature of HPCC Systems Machine Learning. [Online]. Available: https://hpccsystems.com/blog/understanding-myriad-interface

[38] A. Asuncion and D. Newman, "Uci machine learning repository," 2007.

[39] S. Hettich and S. Bay, "Uci kdd archive," 1999.

[40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python ," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[41] HPCC Systems. (2019) HPCC Systems KMeans Bundle. [Online]. Available: https://github.com/hpcc-systems/KMeans