Performance Prediction of Sparse Matrix Multiplication on a Distributed BigData Processing Environment

Jueon Park
Dept. of Computer Science
Kookmin University
Seoul, South Korea
jueon@kookmin.ac.kr

Kyungyong Lee
Dept. of Computer Science
Kookmin University
Seoul, South Korea
leeky@kookmin.ac.kr

***Abstract*—Sparse matrix multiplication (SPMM) is widely used for various machine learning algorithms. With advancements in big-data processing, the importance of distributed SPMM processing becomes important for handling large-scale datasets. We conducted thorough experiments using various distributed SPMM implementations and discovered considerable performance variations for distinct datasets and scenarios. To provide an optimal SPMM execution environment, we propose features that represent SPMM task characteristics. Using these features, we propose building a tree-based nonlinear gradient boosting (GB) regressor model that presents superb prediction accuracy across diverse distributed SPMM implementations and datasets.**

I. Introduction

Large-scale datasets from real-world applications can be represented using a graph format that expresses relationships among nodes. An edge between two nodes means that the nodes have a relation, and an edge can have a weight value based on the datasets. For example, friends connections and topic subscriptions in a social network, product-review ratings in an e-commerce site, user-movie ratings in a movie-streaming service, and hyperlinks from a source to a destination website can be expressed using a graph. To apply several types of machine learning algorithms to extract valuable information from graph-structured datasets, the dataset should be expressed in a computer-friendly format. Many data mining algorithms require input datasets to be represented in a sparse matrix format. For example, the power method implementation of the PageRank algorithm [10] and nonnegative matrix factorization (NMF) [8], which is widely used for various recommendation systems requires input datasets to be represented in a matrix format.

Processing various types of big data requires considerable computing power, and general-purpose distributed computing platforms, such as Apache Hadoop [4] and Spark [17], provide a set of application programming interfaces (APIs) that abstract the complex mechanisms of maintaining fault-tolerance in a distributed environment, task scheduling with heterogeneous resources, and guaranteeing scalability as demand changes. Using the high-level APIs of Apache Spark, MLLib [9] provides an implementation of various machine learning algorithms conducted on a shared-nothing distributed computing environment. It also provides various matrix operations on a distributed environment [1] that include matrix multiplication and factorization, which are core kernels of many machine learning algorithms.

Optimization of matrix multiplication in a HPC environment has well been studied in the literature. Researchers have focused on minimizing communication overhead using a highly optimized MPI library or carefully designing algorithms on multi-core shared memory machines [16], [11], [2]. Despite the importance of an SPMM as a core kernel of many data mining algorithms, the characteristics of the operation in a shared-nothing distributed environment are not well studied, especially for Apache Spark, which is one of the most popular big-data processing engines.

Apache Spark MLLib supports various sparse matrix operations in a distributed computing environment. To investigate the characteristics of SPMM using Spark MLLib, we have conducted experiments with various datasets and distributed implementations; the results are shown in Figure 2. From the experimental results, we observe that the performance of SPMM varies significantly as the datasets and implementation change.

Despite the variability in SPMM performance, barely any guidance exists to optimally operate using Apache Spark. To help users better understand the characteristics of operation and guide optimal implementation, we propose a model to predict the latency of arbitrary SPMM tasks with different numbers of rows and columns, different density, and distinct multiplication implementation methods. We first propose a set of features that represent the characteristics of input matrices. Using the features, we built several models and found that applying a GB regressor [5] with Bayesian optimization [13] to determine the optimal hyper-parameters achieves the best prediction accuracy.

In summary, the main contributions of this paper are as follows.

- We uncovered the performance variability of a distributed SPMM operation on Apache Spark and the lack

of general guidance regarding performance characteristics.
- We propose features that can represent the distributed SPMM well using Spark MLLib.
- We present the practicality of building a model to predict distributed SPMM performance for diverse scenarios.

This paper is organized as follows. Section II discusses various algorithms to implement SPMM with Apache Spark and presents the performance variability. Section III proposes a model to predict SPMM performance, and Section IV thoroughly evaluates the proposed model. Section V concludes this paper with future work.

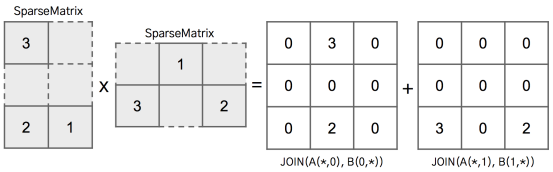
II. Distributed Sparse Matrix Multiplication

To store and access a sparse matrix using Apache Spark, MLLib [9] provides distributed sparse matrix representations: *indexed-row* and *block* matrices. Both approaches use resilient distributed dataset [17] as the underlying mechanism to store sparse matrices while guaranteeing fault-resilience. In the *indexed-row* representation, an input matrix is stored in a row-wise manner so that a row is stored as a sparse vector locally in the distributed servers. To store blocks in a column-major order, we transpose an input matrix and store it in the *indexed-row* format. In the *block* representation, an entire sparse matrix is partitioned into either the row and/or column direction. After partitioning, a block of the sparse matrix is stored in compressed sparse column (CSC) format.

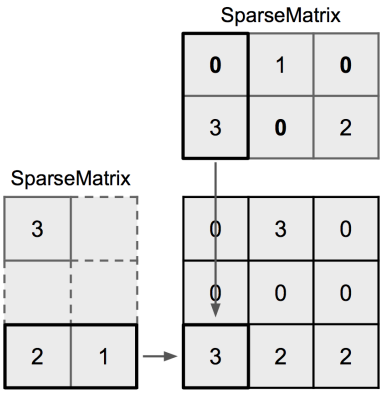
Using the distributed representations of sparse matrices, we highlight four distributed SPMM implementations in Figure 1. For the brevity of the descriptions, we define $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, as the left, right, and the result matrix, which is $\mathbf{C} = \mathbf{A} \times \mathbf{B}$. We use i and j to denote an arbitrary row index of $\mathbf{A}$ and a column index of $\mathbf{B}$. Moreover, we use k to denote an arbitrary column index of $\mathbf{A}$ and row index of $\mathbf{B}$. The capital letters I, K, and J denote the corresponding dimension size. In each matrix, we use subscripts with a parentheses to denote the row and column indices. For example, $\mathbf{A}_{(i,k)}$ indicates an element in the i-th row and k-th column of a matrix $\mathbf{A}$. The $*$ notation in the row or column index of a matrix denote an entire row or column. For example, $\mathbf{A}_{(i,*)}$ indicates the i-th row of a matrix $\mathbf{A}$.

Figure 1a illustrates an SPMM implementation of the outer sparse product. In the method, $\mathbf{A}$ is partitioned into the column ($\mathbf{A}_{*,k}$), and $\mathbf{B}$ is partitioned into the row ($\mathbf{B}_{k,*}$). A *GroupBy* operation is applied for k. In each grouped result, an outer product is conducted ($\mathbf{A}_{*,k} \otimes \mathbf{B}_{k,*}$) that results in an intermediate output matrix of size $I \times J$. During the outer product operation, each vector remains in a sparse status. Element-wise summation is performed to derive the final result, $\mathbf{C}$.

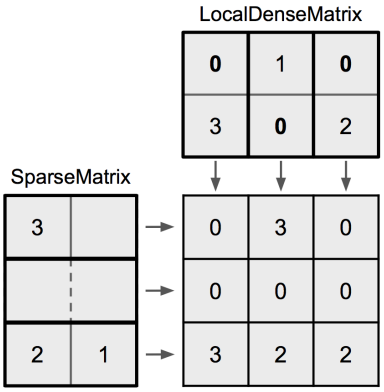
Figure 1b explains an SPMM implementation that uses an inner sparse product. In the method, $\mathbf{C}_{i,j}$ is generated by



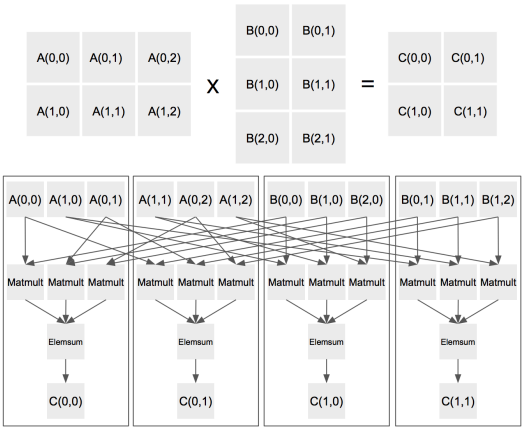
(a) Outer sparse product



(b) Inner sparse product

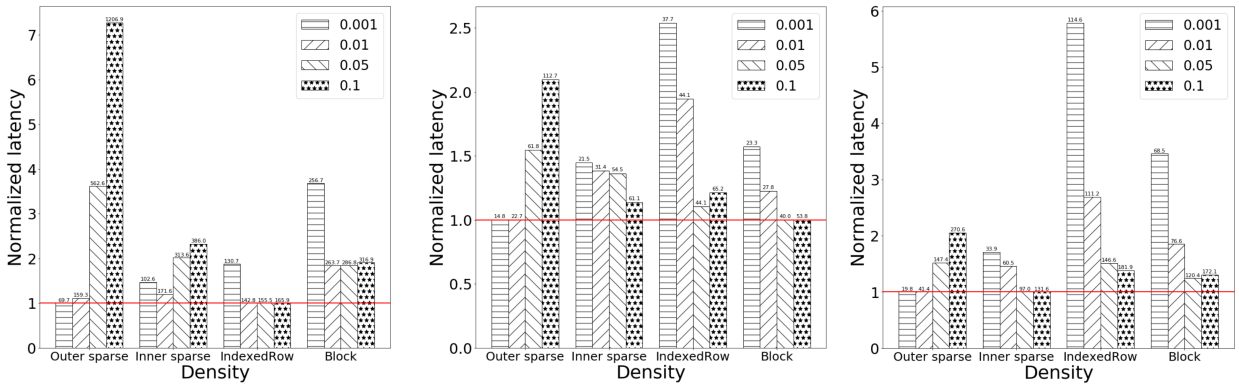


(c) Indexed-Row multiply



(d) Block multiply

Figure 1: Various SPMM implementation with distributed sparse matrices



(a) Orkut dataset (b) DBLP dataset (c) Youtube dataset

Figure 2: Sparse matrix multiplication using Apache Spark with different matrix distribution mechanisms

conducting an inner product of a row from the left matrix and a column from the right matrix, $\mathbf{C}_{i,j} = \mathbf{A}_{i,*} \cdot \mathbf{B}_{*,j}$. In the inner-product operation, each vector is kept in a sparse state. Stitching scalar values from the inner-product operations results in the final output matrix, $\mathbf{C}$.

Figure 1c displays an SPMV implementation using the *indexed-row* data structure provided by Spark [15]. The multiplication method is natively supported by Apache Spark. In the method, the right matrix, $\mathbf{B}$, should exist locally in a Spark driver. A driver broadcasts the matrix to all workers after converting it to a dense matrix. In each worker node with multiple executors, the inner product between a sparse vector ($\mathbf{A}_{i,*}$) and a dense vector ($\mathbf{B}_{*,j}$) calculates $\mathbf{C_{i,j}}$.

Figure 1d represents an SPMM method using a distributed block-partitioning mechanism that is natively supported by Apache Spark [15]. In the block partitioning scheme, a matrix is divided into both the row and column directions. To calculate a block of the output matrix, the corresponding entire rows from the left matrix and columns from the right matrix are fetched to a node responsible for calculation. During multiplication, the left matrix is kept in a sparse CSC format, whereas the right matrix is converted into a dense matrix format.

Among the four methods, Spark does not natively support the outer product and inner sparse product. Thus, we implement these ourselves.

To understand the performance of different SPMM mechanisms presented in Figure 1, we conducted a performance evaluation of different heuristics. In the experiments, we performed SPMMs with a left sparse matrix and a right matrix with various densities. To generate a realistic SPMM workload, we used the multiple-source breadth-first search (BFS) algorithm [6], which is explained in Figure 3. The algorithm repeatedly performs an SPMM operation with a left matrix built from an input sparse dataset and a right matrix indicating the source to destination mappings. A right matrix for a multiply operation is first defined by setting a source ID node to 1 and the others to 0 in each column. The result of the left and right matrix multiplications indicates the path from the source to the destination. During iterative multiplication, a result matrix becomes the right matrix for the next iteration, indicating the path connection in multiple hops. Because a right matrix is updated in every iteration, the density changes in every iteration, and it provides various multiplication scenarios by operating in multiple iterations.

To evaluate the performance of the four different distributed implementations of SPMM in Figure 1, we used Orkut (Figure 2a), DBLP (Figure 2b), and YouTube (Figure 2c) datasets as left matrices. For the right matrix, we set a random source element to 1, and the right matrix is updated in every iteration using a result matrix of a previous SPMM. To investigate the characteristics from various multiplication scenarios, we set the density of the right matrix to 0.001, 0.01, 0.05, and 0.1. Because the right matrix becomes denser in different ratios for a distinct left matrix dataset, we choose cases in which the right matrix density becomes the closest the configured densities. In each figure, the horizontal axis expresses the different distributed SPMM implementations. Each SPMM method has four bars that represent the performance of the distinct right matrix densities. The vertical axis indicates the normalized latency for the best performance of the same right matrix density with different distributed SPMM mechanisms.

To represent the relative performance difference of the SPMM method for different right matrix densities, we group bars of different densities with the same SPMM method. For example, the *outer sparse* in Figure 2a, exhibits the best performance compared to other SPMM methods when the right matrix density is 0.001, but the performance degrades significantly as the density of the right matrices increases. The right matrix density becomes 0.1, and *outer sparse* implementation has about seven times more latency than the best case (*indexed-row*). From the figures, we can

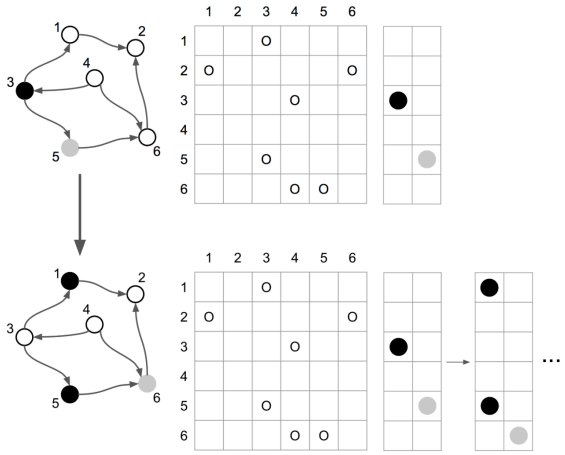


Figure 3: Multi Source BFS algorithm as a workload to evaluate various distributed SPMMs

observe that the performance difference among different SPMM implementations, right matrix densities, and left matrix characteristics are significant. We cannot determine a globally optimal SPMM implementation, which can result in considerable performance variability for various data mining jobs implemented on Apache Spark.

III. Modeling Distributed Sparse MM Performance

The performance of distributed SPMM implementations differs significantly as the input datasets vary, and it is critical to predict the estimated latency of an arbitrary SPMM task because it is a core kernel of many machine learning jobs. To predict the performance of various SPMM tasks, we propose features to represent characteristics of various SPMM workloads. Using the proposed features, we suggest building a prediction model using a GB-regressor [5] that accurately represents non-linear interactions among features. Furthermore, we apply Bayesian optimization [13] to determine the optimal hyper-parameters.

In building a prediction model, determining a representative set of features is the first step. To represent a sparse matrix, we use the dimension of left and right matrices and name them as lr, lc, and rc, which is the number of left matrix rows, left matrix columns, and the right matrix columns, respectively. Because the target workload of the proposed model is a sparse matrix, the density of a matrix is an important factor that must be considered. We call it $l-density$ and $r-density$ for left matrix density and the right matrix density, respectively.

In addition to the left and right densities, we also add the number of nonzero (nnz) elements of the left and right matrices, $l-nnz$ and $r-nnz$. The nnz of a matrix is already reflected in the density feature because it is calculated by dividing the nnz ($l-nnz$ for a left matrix) by the total

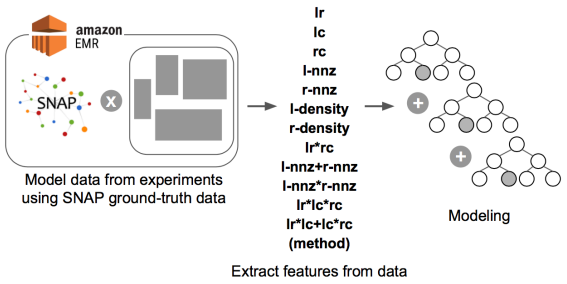


Figure 4: Proposed architecture

number of elements ($lr \times lc$ for a left matrix). Ideally, such a relationship should be captured while building a model. Deep neural-net [12] is good at finding hidden relationships from a very large-scale input datasets. However, it requires a significant number of input datasets to detect hidden characteristics, and it is impractical to generate numerous input datasets for SPMM tasks. Thus, we add the nnz and density feature manually.

We add $lr \times rc$ to represent the dimension of an output matrix from an SPMM task. Because we target sparse input datasets, the size and number of non-zero elements from an output matrix are difficult to estimate before doing the actual computation. We expect that the combination of $l-nnz$, $r-nnz$, and $lr \times rc$ can estimate the overhead of a node that is responsible for storing an output matrix. We add $l-nnz+r-nnz$ to represent the shuffling overhead for all nodes during computation. To consider the actual number of product computations in the sparse matrix format, we add $l-nnz \times r-nnz$.

Referencing the performance estimation of distributed dense matrix multiplication using Spark [7], [14], we add $lr \times lc\ timesrc$ and $lr \times lc + lc \times rc$, which represent the total number of product operations and the shuffle overhead, respectively. To build a unified model that is applicable for the four methods of an SPMM operation, we add a $method$ feature that is categorical.

We build our prediction model using the GB regressor with the features that are mentioned above. The GB regressor produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It produces a classifier based on the accuracy of the classifier generated on the previous step and ensembles those classifiers to generate a more accurate final model.

In modeling using the GB regressor, various hyper-parameters are used, such as learning_rate, max_depth, max_leaf_nodes, and so on. It is difficult to find the best hyper-parameter combination in modeling. Although experimenting with all possible combinations of hyper-parameters provides the best hyper-parameters, it takes a long time. The Bayesian optimization helps to determine the optimal hyper-parameters with extra minimal overhead.

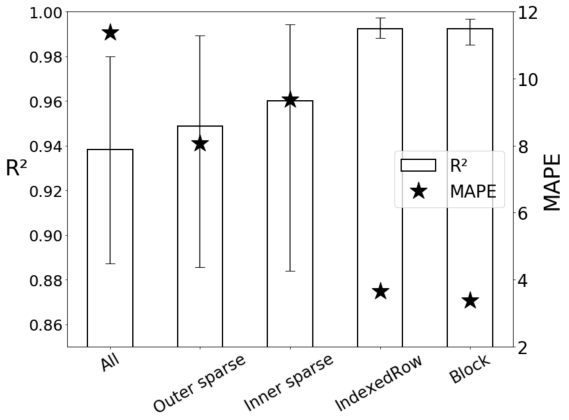


Figure 5: Prediction accuracy of a model built with GB regressor

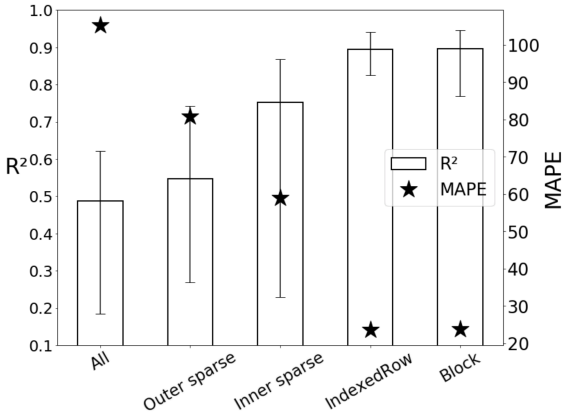


Figure 6: Prediction accuracy of a model built with NNLS

IV. Evaluation

To present the applicability of predicting the performance of SPMM operations with various sparse matrices and distributed implementation, we conducted experiments covering thorough scenarios. To represent various input datasets, we used *Orkut*, *DBLP*, and *Youtube* graph datasets from SNAP[1]. The *Orkut* dataset contains 3,072,441 nodes, 117,185,083 edges, and 234,370,166 nnz. The *DBLP* dataset has 317,080 nodes, 1,049,866 edges, and 2,099,732 nnz. The *Youtube* dataset contains 1,134,890 nodes, 2,987,624 edges, and 5,975,248 nnz. Finally, we could collect 12 distinct SPMM scenarios. For the generated SPMM cases, some implementations could not complete the given workloads due to the memory limitation. Using the input dataset as a left matrix, we conducted multi-source BFS algorithms in multiple iterations using Apache Spark. The experiments were conducted on AWS Elastic MapReduce version 5.27.0 with one master and four worker nodes of *R4.2xlarge* instances.

[1]http://snap.stanford.edu/data/index.html

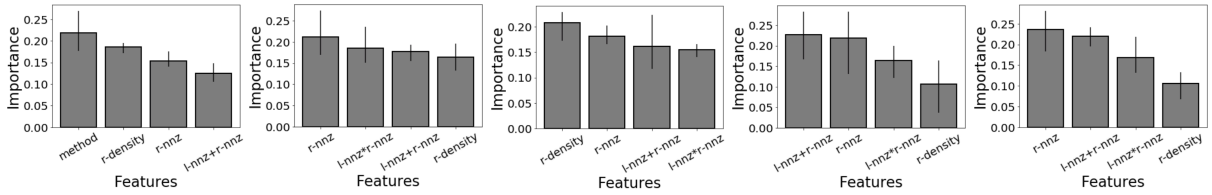
We first evaluated the prediction accuracy of a GB regressor model with the proposed feature sets. We performed K-fold cross validation 10 times dividing the training and test datasets into an 8:2 ratio. We measured the prediction accuracy by using R^2 and the mean absolute percentage error (MAPE) metric. Figure 5 displays the R^2 value (higher is better) in the primary vertical axis whose values are represented in the bar, and the MAPE value (lower is better) is shown in the secondary vertical axis whose values are represented as star marks. The horizontal axis reveals the distributed SPMM implementations. The first four values indicate the prediction accuracy of a model built from training datasets generated from each method explained in Section II exclusively. Other than these, we create a model that uses training datasets generated from four methods and built a single model (*All*).

We observe that the models built from exclusive datasets exhibit better prediction accuracy than the unified model. However, the unified *All* method with the worst performance demonstrates accurate results with R^2 equal to 0.94 and a MAPE of about 11%. The best prediction accuracy is achieved with the *Block* implementation for both R^2 and the MAPE metric.

To present superb prediction accuracy for a model built with a GB regressor algorithm, we built another model using the nonnegative least square (NNLS) algorithm, whose result is presented in Figure 6. The overall prediction accuracy pattern is very similar to that presented with the GB regressor algorithm. However, the degree of accuracy is significantly lower. For instance, the R^2 value of the *All* method is only 0.5, and MAPE is over 100%. From the result, we conclude non-linear interaction exists among the suggested features for various SPMM tasks.

Using a GB regressor model, we accurately modeled the response time of distributed SPMM tasks with the proposed features when they are executed using Apache Spark. To understand which features make considerable contributions during modeling, we calculated the feature importance while building a model. Figure 7 indicates the four most important features for various distributed SPMM implementations. Each figure reveals the relative importance of each feature, and the importance is calculated by counting the number of times a feature is selected during the decision tree build process [3]. In most cases, the right matrix characteristics are more important because the target application generates more diverse right matrices. Other than the right matrix characteristics, the shuffle overhead ($l - nnz + r - nnz$) and computation overhead ($l - nnz \times r - nnz$) influence modeling. Although we added important features for distributed dense matrix multiplication, which were presented in [14], [7], the features do not show a noticeable influence,

which indicates the drastic difference in the characteristics of dense and SPMM.



(a) All methods (b) Outer sparse (c) Inner sparse (d) Indexed-Row (e) Block

Figure 7: The most important 4 features for each methods and all methods aggregated

V. Conclusion and Future Work

This work presents a tree-based non-linear GB regressor model to predict the performance of distributed SPMM tasks on Apache Spark. After summarizing the SPMM implementations in a distributed environment, we demonstrated performance variability for diverse SPMM implementations and input datasets that necessitates a performance predictor for optimal performance. We proposed feature sets to build a model and demonstrated superb prediction performance.

The performance of the proposed model of distinct SPMM algorithms is better than that of the unified model built for all implementations. Building separate models for distinct implementations requires a substantial number of experiments to generate training datasets, and we are actively working on improving the performance of the unified model. We are also actively working on building a model in a cloud computing environment that provides various instance types. In such an environment, controlling the experimental scenario to decrease experimental cost is crucial, and we are investigating opportunities in that direction.

Acknowledgement

This work is supported by the National Research Foundation of Korea (NRF) Grant funded by the Korean Government (MSIP) (No. NRF-2015R1A5A7037615), the ICT R&D program of IITP (2017-0-00396), and Research Credits provided by AWS.

References

[1] R. Bosagh Zadeh, X. Meng *et al.*, "Matrix computations and optimization in apache spark," ser. KDD '16. ACM, 2016, pp. 31–38.

[2] J. Choi, J. J. Dongarra *et al.*, "Scalapack: a scalable linear algebra library for distributed memory concurrent computers," in *[Proceedings 1992] The Fourth Symposium on the Frontiers of Massively Parallel Computation*, 1992, pp. 120–127.

[3] J. Elith, J. R. Leathwick, and T. Hastie, "A working guide to boosted regression trees," *Journal of Animal Ecology*, vol. 77, no. 4, pp. 802–813, 2008.

[4] A. S. Foundation, "Apache hadoop," 2004. [Online]. Available: http://hadoop.apache.org/

[5] J. H. Friedman, "Greedy function approximation: A gradient boosting machine." *Ann. Statist.*, vol. 29, no. 5, pp. 1189–1232, 10 2001. [Online]. Available: https://doi.org/10.1214/aos/1013203451

[6] J. Kepner and J. Gilbert, *Graph Algorithms in the Language of Linear Algebra*, J. Kepner and J. Gilbert, Eds. Society for Industrial and Applied Mathematics, 2011. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9780898719918

[7] J. Kim, M. Son, and K. Lee, "Mpec: Distributed matrix multiplication performance modeling on a scale-out cloud environment for data mining jobs," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2019.

[8] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *In NIPS*. MIT Press, 2000, pp. 556–562.

[9] X. Meng, J. Bradley *et al.*, "Mllib: Machine learning in apache spark," *J. Mach. Learn. Res.*, vol. 17, no. 1, p. 1235â1241, Jan. 2016.

[10] L. Page, S. Brin *et al.*, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Technical Report 1999-66, November 1999, previous number = SIDL-WP-1999-0120. [Online]. Available: http://ilpubs.stanford.edu:8090/422/

[11] M. M. A. Patwary, N. R. Satish *et al.*, "Parallel efficient sparse matrix-matrix multiplication on multicore platforms," in *High Performance Computing*, J. M. Kunkel and T. Ludwig, Eds. Cham: Springer International Publishing, 2015, pp. 48–57.

[12] O. Russakovsky, J. Deng *et al.*, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[13] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'12. USA: Curran Associates Inc., 2012, pp. 2951–2959. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999325.2999464

[14] M. Son and K. Lee, "Distributed matrix multiplication performance estimator for machine learning jobs in cloud computing," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, vol. 00, Jul 2018, pp. 638–645. [Online]. Available: doi.ieeecomputersociety.org/10.1109/CLOUD.2018.00088

[15] A. Spark, "Apache spark mllib distributed matrix computation," https://goo.gl/Vnii2M, 2017, [Online; accessed 20-Nov-2017].

[16] R. A. van de Geijn and J. Watts, "Summa: Scalable universal matrix multiplication algorithm," Austin, TX, USA, Tech. Rep., 1995.

[17] M. Zaharia, M. Chowdhury *et al.*, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. San Jose, CA: USENIX, 2012, pp. 15–28.