2015 IEEE 17th International Conference on High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), and 2015 IEEE 12th International Conf on Embedded Software and Systems (ICESS)

Performance Prediction for Apache Spark Platform

Kewen Wang
Department of Computer Science and Engineering
University of Connecticut, Storrs, Connecticut, USA
kewen.wang@uconn.edu

Mohammad Maifi Hasan Khan
Department of Computer Science and Engineering
University of Connecticut, Storrs, Connecticut, USA
maifi.khan@engr.uconn.edu

***Abstract*—Apache Spark is an open source distributed data processing platform that uses distributed memory abstraction to process large volume of data efficiently. However, performance of a particular job on Apache Spark platform can vary significantly depending on the input data type and size, design and implementation of the algorithm, and computing capability, making it extremely difficult to predict the performance metric of a job such as execution time, memory footprint, and I/O cost. To address this challenge, in this paper, we present a simulation driven prediction model that can predict job performance with high accuracy for Apache Spark platform. Specifically, as Apache spark jobs are often consist of multiple sequential stages, the presented prediction model simulates the execution of the actual job by using only a fraction of the input data, and collect execution traces (e.g., I/O overhead, memory consumption, execution time) to predict job performance for each execution stage individually. We evaluated our prediction framework using four real-life applications on a 13 node cluster, and experimental results show that the model can achieve high prediction accuracy.**

***Keywords*-Apache Spark; Performance Modeling; Execution Time Prediction**

I. Introduction

Among many cloud computing platforms, Apache Spark [1] is one of the popular open-source cloud platforms that introduces the concept of resilient distributed datasets (RDDs) [2] to enable fast processing of large volume of data leveraging distributed memory. Its in-memory data operations makes it well-suited for iterative applications such as iterative machine learning and graph algorithms. However, execution time of a particular job on Apache Spark platform can vary significantly depending on the input data type and size, design and implementation of the algorithm, and computing capability (e.g., number of nodes, CPU speed, memory size), making it extremely difficult to predict job performance, which is often needed to optimize resource allocation [3] [4]. Performance prediction can also help to locate execution stages with abnormal resource usage pattern [5]. While prior work exists that looked into the problem of performance prediction for cloud platforms such as Apache Hadoop [6] (an open-source implementation of MapReduce [7] computing framework), these approaches are not suitable for Apache Spark platform due to its different programming model and features such as in-memory data operations. Hence, to address this void, in this paper, we focus on performance modeling for Apache Spark jobs. While various forms of machine learning approaches are often used to predict system performance leveraging past system execution data [8] [9] [10] and can achieve reasonable prediction accuracy, it requires training dataset. In contrast, modeling based approaches predict performance through modeling system behavior [11] [12] [13], and often can provide a better understanding regarding internal execution of a program and resulting performance. Therefore, in this paper, we apply analytical approaches to predict the performance of Apache Spark jobs. Specifically, we leverage the multi-stage execution structure of Apache Spark jobs to develop hierarchical models that can effectively capture the execution behavior of different execution stages. Using these models, we first measure the job performance based on limited scale execution using only a fraction of real data set. Next, we predict the job performance based on the limited scale execution job performance data. We evaluated our framework with four real-world applications. In each case, our model is able to predict execution time for individual stage with high accuracy. Additionally, the model is able to predict memory requirement for RDD creation with high accuracy. However, the accuracy of I/O cost prediction varied for different applications and simulation setup. We discuss our detailed findings in Section IV.

The rest of the paper is organized as follows. Prior efforts that are related to our work is discussed in Section II. Section III explains the models that are used to predict job performance in this work. Experimental results are presented in Section IV. Finally, Section V concludes the paper.

II. Related Work

While significant volume of work exists that looked into various aspects of performance prediction in various distributed settings, in this paper, we focus our discussion on recent efforts that investigated the challenge of performance prediction for various cloud platforms [3]–[5], [9]–[15]. Among numerous efforts, PREDIcT [9] is one of the recent work that looked into the problem of predicting runtime for network intensive iterative algorithms implemented on Hadoop MapReduce platform. Specifically, it aims to predict the number of iterations and runtime for each iteration based on sample run and historical data. However, PREDIcT

978-1-4799-8937-9/15 $31.00 © 2015 IEEE
DOI 10.1109/HPCC-CSS-ICESS.2015.246

focuses on iterative algorithms, and requires representative training dataset to achieve high prediction accuracy, and may lead to poor prediction accuracy for applications with no historical data. To simplify the performance prediction for complex Hadoop application, another recent work presented the idea of using a modeling language (e.g., Hive Query Language (HQL)) [12] that translates big data applications into SQL-like queries on Apache Hive [16]. This provides a convenient way for predicting performance for data processing applications that can be implemented using HQL queries. For Map Reduce jobs running on heterogeneous machines, bound-based performance modeling techniques are tried for predicting job completion time [13] as well. The main idea is to evaluate the upper and lower bounds of job completion time, and use that to predict job performance. Starfish [11] presents a self-tuning framework for MapReduce paradigm to predict job performance under different program configurations for Apache Hadoop jobs. It applies analytical approach to estimate how a job will perform based on job simulation data, and uses that model to predict performance. While this and prior approaches achieve good prediction accuracy, due to the differences between the implementation of other cloud platforms and Apache Spark platform, Starfish and similar approaches are not suitable for predicting job performance running on Apache Spark platform without significant modifications. We address this void in our paper as follows.

III. Overview

In Apache Spark platform, each job consists of multiple execution stages implementing distinct operations of an application program where stages are executed sequentially. To facilitate parallel processing, input data set is partitioned into multiple sets and are distributed over multiple worker nodes. Within each worker node, batches of tasks are launched to process the corresponding partition of the input data. The number of tasks within each node is determined based on the size of the input data and configuration settings of the program. To illustrate the main idea behind Apache Spark job execution, let us consider the Apache Spark PageRank job running on two worker nodes: A and B, where Node A has 8 CPU cores and Node B has 12 CPU cores as shown in Figure 1. This PageRank job will have 13 stages if the iteration number is set to 10, where stage 1 and stage 2 execute the distinct() operation. In the iteration part from stage 3 to stage 12, the operation reduceByKey() is executed. The final stage performs the saveAsTextFile() operation. As shown in Figure 1, each box in the Figure represents one stage, and each line in the box represents one task. Different colors are used to differentiate tasks running on different worker nodes. If the input data size of this PageRank job is 2.5 GB, the total number of input blocks will be 40 for a default block size of 64 MB. As the number of tasks is same as the input block number, there are 40 lines in each stage. In addition, the number of tasks in each stage is same within one Spark job. Therefore, for this example, in each stage, 40 tasks will be executed. However, different CPU core may complete different number of tasks due to the difference in computing ability and uncertainty during the program execution. Given the above model of execution, next, we present the developed hierarchical models that can be used to predict job execution time, memory footprint for RDD creation, and I/O overhead as follows.

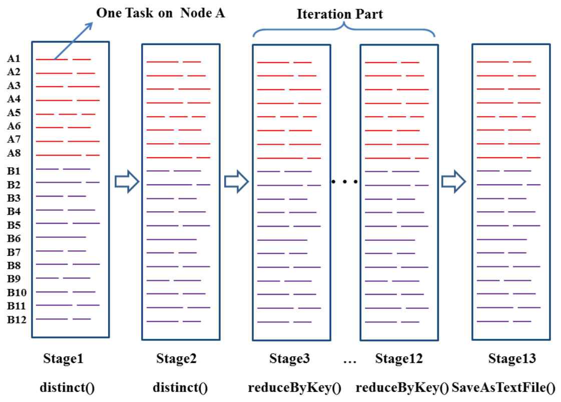

Figure 1: Apache Spark PageRank Job

A. Model for Estimating Execution Time

As a Spark job is executed in multiple stages where each stage contains multiple tasks, we use the following notation to represent an Apache Spark job:

$$Job = \{Stage_i \mid 0 \leq i \leq M\} \tag{1a}$$

$$Stage_i = \{Task_{i,j} \mid 0 \leq j \leq N\} \tag{1b}$$

Here M is the number of stages in a job and N is the number of tasks in a stage. Next, as different stages within a job are executed sequentially, we represent the execution time of a job as the sum of the execution time of each stage plus the job startup time and the job cleanup time as follows:

$$JobTime = Startup + \sum_{s=1}^{M} StageTime_s + Cleanup \tag{2}$$

Next, within each stage, as one CPU core executes one task at a time, in a cluster with H worker nodes, the number of tasks P that can run in parallel can be calculated as follows:

$$P = \sum_{i=1}^{H} CoreNum_i \tag{3}$$

Here, $CoreNum_i$ is the number of CPU cores of working node i and H is the number of working nodes in the cluster. Hence, within an execution stage, tasks in each stage are executed in batches where each batch consists of P tasks running in parallel. However, due to the differences in computing capabilities among different worker nodes in a heterogeneous cluster and inherent uncertainty in program execution, the execution time of different tasks may vary significantly. Therefore, the time spent in a particular stage can be calculated as the maximum of the sum of all the sequential tasks' time within a stage plus the stage startup time and the stage cleanup time as follows:

$$StageTime = Startup + \max_{c=1}^{P} \sum_{i=1}^{K_c} TaskTime_{c,i} \\ +Cleanup \tag{4}$$

Here P is the number of total CPU cores, and K_c is the number of sequential tasks executed on CPU core c. Finally, as different tasks in a stage follow the same execution pattern, the execution time of a task can be computed as follows:

$$TaskTime = DeserializationTime + RunTime \\ +SerializationTime \tag{5}$$

Here $DeserializationTime$ is the time taken to deserialize the input data, $SerializationTime$ is the time taken to serialize the result, and $RunTime$ is the actual time spent performing operations on data such as data mapping, filtering, calculating, and analyzing.

B. Memory Consumption

As the Spark platform takes the advantage of in-memory processing to improve the computing efficiency, it is important to allocate sufficient memory needed to create initial RDD to avoid possible slowdown of the execution. Moreover, under certain system configurations, lack of enough memory for initial RDD creation can lead to unexpected program termination. To avoid such adverse events, we develop a simple model to estimate the minimum memory requirement for RDD creation. Specifically, if there are N tasks in the system, we can express the total memory requirement for the job as follows:

$$JobRddMem = \sum_{i=1}^{N} TaskRddMem_i \tag{6}$$

C. Model for Estimating I/O Cost

Finally, within a stage, the transformation operation that generates new RDD based on previous RDD is implemented in $ShuffleMapTask$ and the action operation that output the result data which is implemented in $ResultTask$. The I/O cost during the shuffle phase in these two types of tasks can be classified into two categories, namely, the shuffle read cost and the shuffle write cost. Shuffle write cost is the cost of writing the interim data to local disk buffer, and shuffle read cost refers to the network I/O cost for fetching the interim data from other worker nodes. As shuffle phase is the most I/O intensive phase where frequent data fetching and transmission occurs, in our model, for I/O cost measurement, we specifically focus on data transmission during the shuffle phase that involves data fetching from remote hosts and the interim data writing into the disk. The stage-by-stage I/O cost is calculated as follows:

$$StageIOWrite_i = \sum_{j=1}^{N} TaskIOWrite_{i,j} \tag{7a}$$

$$StageIORead_i = \sum_{j=1}^{N} TaskIORead_{i,j} \tag{7b}$$

Here N is the number of tasks in $Stage_i$.

D. Performance Prediction

Based on the above model, to predict job performance, the presented modeling framework first executes the program on a cluster using limited amount of sample input data and collect performance metrics such as run time, I/O cost, and memory cost during the simulated run. Next, the extracted performance metric from simulated run is used to predict the performance metric for the actual run. Specifically, to predict the execution time, we first calculate the number of tasks that will be executed in the actual job as follows: $N = InputSize/BlockSize$, where $InputSize$ is the size of the input data, and $Blocksize$ is the size of one data block in HDFS. The tasks within a stage are scheduled to run batch by batch, and the number of tasks in each batch P is computed as shown in equation (3). In one batch of tasks, while the tasks may start simultaneously, they may not finish at the same time due to various factors such as data skew problem, and differences in computing capability of different worker nodes. Hence, using simulation data, we calculate the average execution time for a task for a given stage for a worker node h as follows.

$$TaskRunTime_{h,i} = DeserializeTime_{h,i} \\ +RunTime_{h,i} \\ +SerializationTime_{h,i} \tag{8}$$

$$AvgTaskTime_h = \frac{1}{n_h} \sum_{i=1}^{n_h} TaskRunTime_{h,i} \tag{9}$$

Here n_h is the number of tasks running in host h in a particular stage of the sample job. Moreover, during our experiment, we observed that the average execution time of the first batch is significantly different compared to the subsequent batches within the same stage, which we capture as follows.

$$Ratio_h = \frac{\frac{1}{n_h - P_h} \sum_{i=P_h+1}^{n_h} TaskTime_{h,i}}{\frac{1}{P_h} \sum_{j=1}^{P_h} TaskTime_{h,j}} \tag{10}$$

Here n_h is the number of tasks running in host h, and P_h is the number of tasks in the first batch. Please note that, to trace two batches of tasks to calculate this ratio for every working node, $SampleSize$ needs to be doubled (discussed in Section III-E). As tasks execute on different hosts in parallel, to predict the execution time for a particular stage during actual execution, stage $Startup$ time and $Cleanup$ time are viewed as constants which are extracted from simulation logs, and stage execution time is estimated as follows:

$$EstStageTime = Startup + \max_{c=1}^{P} \sum_{i=1}^{K_c} AvgTaskTime_{c,i} + Cleanup \tag{11}$$

$$EstTaskTime_{c,i} = \begin{cases} AvgTaskTime_c, & i = 1 \\ AvgLaterTaskTime_c, & i > 1 \end{cases} \tag{12}$$

Here P is the number of total CPU cores calculated in equation (3), K_c is the number of sequential tasks running in CPU core c. $AvgTaskTime_c$ is the average time for tasks in the first batch for CPU core c of the corresponding host, and is calculated in equation (9). $AvgLaterTaskTime_c$ is the average time of the following batches of tasks, which could be calculated as $Ratio_h \times AvgTaskTime_h$. For predicting I/O cost, the average shuffle read and write costs of a typical task is computed and then used to compute the I/O cost for a specific stage j as follows:

$$EstStageIOWrite_j = \sum_{h=1}^{H} (N_h \times \frac{1}{n_h} \sum_{i=1}^{n_h} (TaskIOWrite_{h,i})) \tag{13}$$

$$EstStageIORead_j = \sum_{h=1}^{H} (N_h \times \frac{1}{n_h} \sum_{i=1}^{n_h} (TaskIORead_{h,i})) \tag{14}$$

Here H is the number of worker nodes and N_h is the number of total tasks on host h during real execution. n_h is the number of tasks running on host h during simulation at stage j. Finally, the average memory footprint for each stage is estimated as follows:

$$EstRddMem = \sum_{h=1}^{H} (\frac{N_h}{n_h} \sum_{i=1}^{n_h} TaskRddMem_{h,i}) \tag{15}$$

E. Simulation Methodology

For simulation, we tried two alternative setup as follows. In the first setup, to make sure that all worker nodes in the cluster is used during simulation, we extract sufficient amount of sample input data so that each CPU core gets to process at least one block of input data. Hence, given that one block of HDFS data is configured to be equal to the size $BlockSize$, the minimum value of $SampleSize$ can be calculated as $BlockSize \times P$, where P is the number of tasks that can run in parallel (P is calculated in equation (3)). However, as the prediction mechanism needs to extract $2 \times P$ blocks of sample data from the original input to simulate on the whole cluster, for clusters with a large number of worker nodes, total CPU core number P may be very huge, resulting in a big sample job and long simulation time. In order to reduce the simulation time, we tried another alternative where the sample job is executed in a smaller cluster which has fewer number of CPU cores p. As a result, only $2 \times p$ blocks of sample data is needed. In our simulation, we ran simulation on a smaller cluster that includes one node of each type (as shown in Figure 2(b)). Basically, all computing nodes in a cluster can be classified into D groups, where each group has Num_g computing nodes and each computing node in a group has the same computing capability (e.g., CPU speed, RAM). Next, one node is selected from each group, and total D working nodes are chosen to construct this new cluster. In such a setting, the size of sample data is reduced to $D / \sum_{g=1}^{D} Num_g$ times of the original input data, reducing the simulation time significantly. Finally, to reduce the impact due to data skew, our sampling technique divides the input data into multiple sections, and extracts data from each section with equal probability.

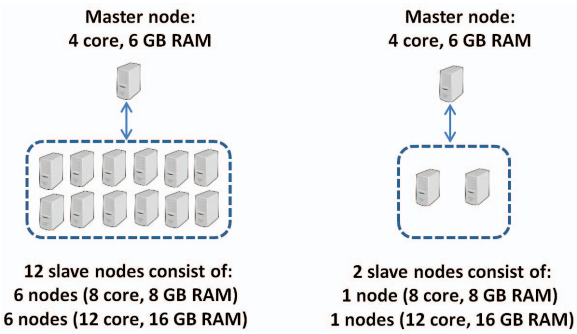
Master node:
4 core, 6 GB RAM
12 slave nodes consist of:
6 nodes (8 core, 8 GB RAM)
6 nodes (12 core, 16 GB RAM)
Master node:
4 core, 6 GB RAM
2 slave nodes consist of:
1 node (8 core, 8 GB RAM)
1 nodes (12 core, 16 GB RAM)

Figure 2: Simulation Setup

IV. Experimental Evaluation

To evaluate our model, experiments are performed on a cluster of 13 nodes, where one node serves as the master node, and the other 12 nodes serve as worker nodes. The master node has 4 CPU cores, and 6 GB of memory. Among the 12 worker nodes, 6 nodes have the same configuration: 8 CPU cores and 8 GB of memory, and the other 6 nodes have the same configuration: 12 CPU cores and 16 GB of memory. For evaluation, we ran Apache Spark using its standalone cluster mode on top of Hadoop Distributed File System (HDFS) with default 64 MB block setting. For data collection, we leveraged spark event logs that are generated by the Apache Spark platform to record execution profiles and performance metrics that are directly obtained from the Spark event listeners in the Apache Spark program, and are saved in JSON [17] format. By analyzing this log file, $StageTime$ in equation (2) and $TaskTime$ in equation (4) can be easily calculated. Job $Startup$ time and job $Cleanup$ time, and stage $Startup$ time and stage $Cleanup$

time is calculated from log data as well. From task level log, task time is calculated using equation (5). Moreover, since I/O cost details are provided for each task, $TaskIOWrite$ and $TaskIORead$ is calculated using the shuffle write and read metrics to construct the I/O profile. The initial RDD is created in one of the first few stages, and each RDD block is stored in memory while the corresponding memory usage is recorded in the tasks sections of logs. Based on this information, memory consumption profile is calculated using equation (6). Example applications we use to verify performance of our prediction mechanism include one non-iterative text processing algorithm: WordCount; two iterative machine learning algorithms: Logistic Regression and K-Means clustering; and one graph algorithm: PageRank. As input data, the WordCount application uses 75 GB Wikipedia dump. Logistic Regression and K-Means application use 50 GB of numerical Color-Magnitude Diagram data of galaxy from Sloan Digital Sky Survey (SDSS) [18]. For PageRank, we use the LiveJournal network dataset from SNAP [19], which is processed through mapping each node id into longer string to form 25 GB of data as the input for this algorithm. To measure the effect of simulation setup on prediction accuracy, we used two simulation setup as follows. In the first simulation setup, we used the whole cluster to simulate the execution. In this case, the number of CPU core P is 120 and we have two cases to consider. In the first case, the sample data size is set at 7.5 GB for P cores (based on 64 MB of block size) to simulate one task per core during the simulation. In this case, we assume that each task within a stage requires similar execution time. In the second case, the sample data size is set at 15 GB for P cores ($2 \times P$ based on 64 MB of block size) to simulate two tasks per core during the simulation. Simulating two tasks per core allows us to calculate the ratio based on the discrepancy in execution time for the first task compared to the subsequent tasks within a single stage (10). In the second simulation setup, we used two worker nodes of different computing capability from 12 worker nodes and kept the master node to construct a cluster of 3 nodes (as shown in Figure 2). In this case, the number of CPU core p is 20 and we again have two cases to consider. In the first case, the sample data size is set at 1.25 GB for p cores (based on 64 MB of block size) to simulate one task per core during the simulation. In this case, we assume that each task within a stage requires similar execution time. In the second case, the sample data size is set at 2.5 GB for p cores ($2 \times p$ based on 64 MB of block size) to simulate two tasks per core during the simulation. Simulating two tasks per core allows us to calculate the ratio based on the discrepancy in execution time for the first task compared to the subsequent tasks within a single stage (equation 10).

To evaluate the accuracy of our prediction model, we calculate the prediction accuracy for each stage and sum them as the total accuracy R as follows:

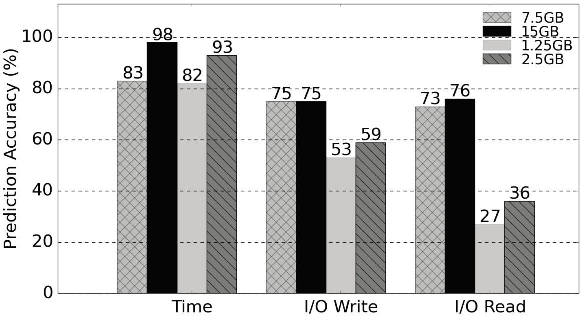

Figure 3: Prediction Accuracy for WordCount

$$R = |1 - \sum_{i=1}^{M} \frac{|PredictCost_i - Cost_i|}{\sum_{j=1}^{M} Cost_j}| \qquad (16)$$

Here M is the number of stages for a particular job, $PredictCost_i$ is the predicted performance cost for stage i, and $Cost_i$ is the actual cost. Equation (16) can be used to calculate the prediction accuracy, and M is set to 1 when computing memory cost prediction accuracy. Our experimental result is presented below.

A. WordCount

In the WordCount example, input data size is 75 GB. For WordCount application, there are two stages and no memory cost for the initial RDD creation. There is only I/O write cost in the first stage, and I/O read cost for the second stage. Figure 3 shows the accuracy for time and I/O cost prediction for different simulation setup. As can be seen in the figure, full cluster simulation achieves greater prediction accuracy. Also, as can be seen in Figure 3 and Figure 4, simulating two tasks per core and considering the discrepancy in execution time for the first task compared to the subsequent tasks within a single stage (equation 10) improves the prediction accuracy significantly. For I/O cost prediction, the full scale simulation achieves much better prediction accuracy compared to the limited-scale simulation (Figure 5 and Figure 6). This may be due to the fact that the limited-scale simulation does not capture the frequent network I/O that may happen in a large-scale setup.

B. Logistic Regression

Logistic Regression is an iterative algorithm with 10 stages, where there is no shuffle I/O cost. For Logistic Regression, the input data size is 50 GB. Figure 7 shows the prediction accuracy for execution time and memory usage for the whole job whereas Figure 8 shows the actual and predicted execution time per stage. As logistic regression is a computing-intensive job, this minimizes the effect of I/O and leads to better prediction accuracy. For memory cost prediction, all

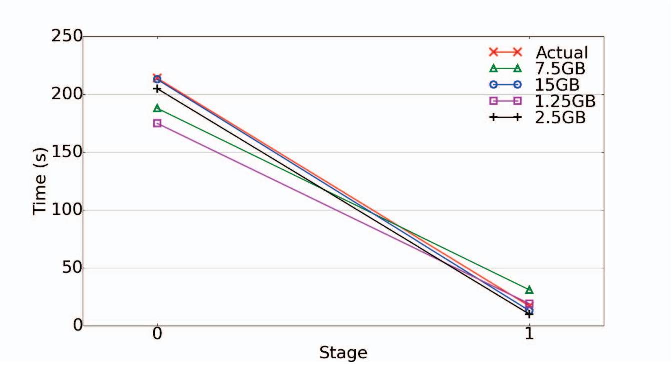

Figure 4: Time Prediction for WordCount

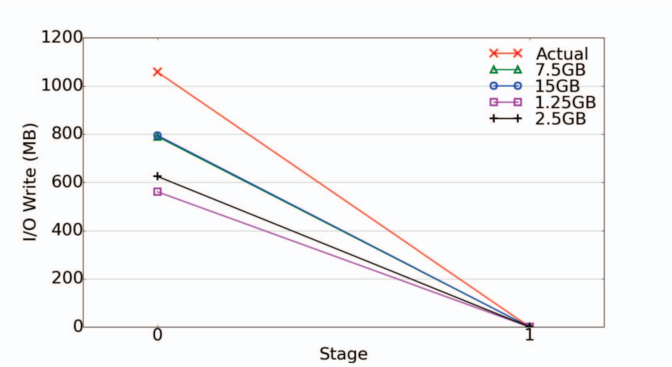

Figure 5: I/O Write Prediction for WordCount

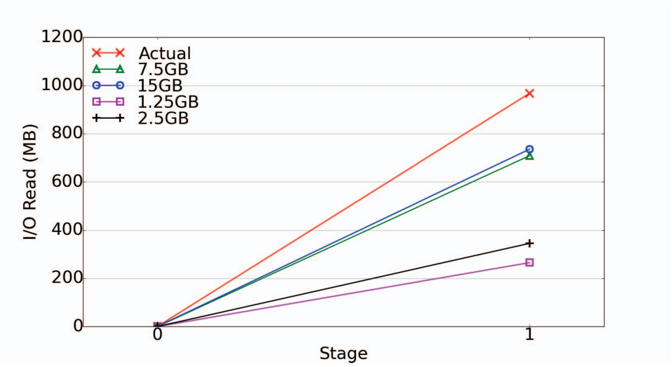

Figure 6: I/O Read Prediction for WordCount

four simulation setup achieved 100% accuracy and correctly predicted the value of 11.5 GB needed to create the initial RDD (Figure 7).

C. K-Means

We use the same data set as input for K-Means clustering algorithm that was used for testing Logistic Regression algorithm. K-Means is an iterative algorithm with 22 stages. In the third stage of the job, the I/O cost involves shuffle write, and the I/O cost involves shuffle read in the fourth stage. Later stages follow the same pattern. Figure 9 shows the prediction accuracy for execution time, memory usage, and I/O cost. Figure 10 shows the actual execution time along with predicted value for different stages. As the volume of data read and written was small (only few megabytes), the prediction error for I/O cost was high (Figure 11 and Figure 12). However, as the time cost for I/O operations is small and has minimal effect on total execution time, the model still achieved high prediction accuracy for execution time. For memory cost prediction, the model had 100% accuracy, correctly predicting the value of 11 GB (Figure 9).

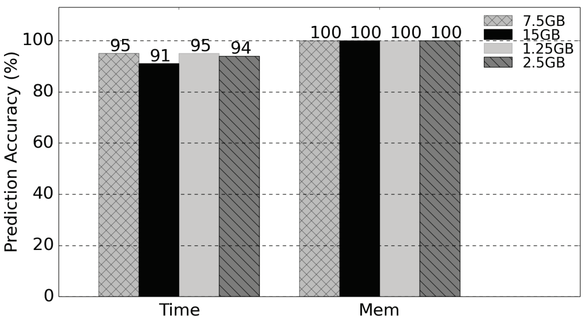

Figure 7: Prediction Accuracy for Logistic Regression

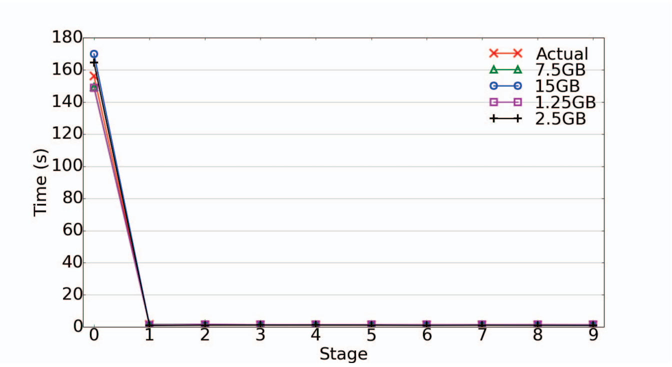

Figure 8: Time Prediction for Logistic Regression

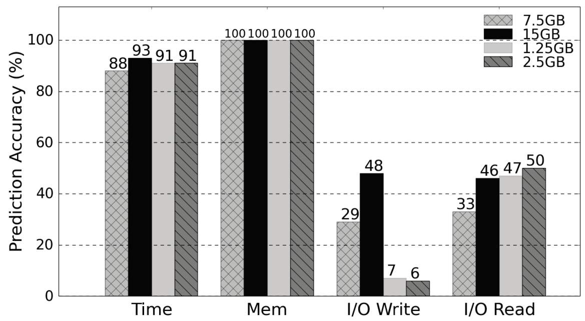

Figure 9: Prediction Accuracy for K-Means

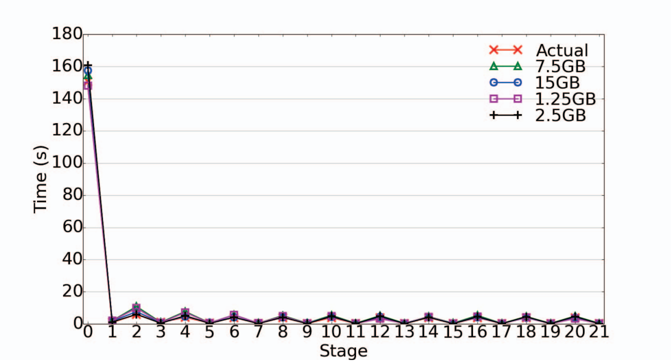

Figure 10: Time Prediction for K-Means

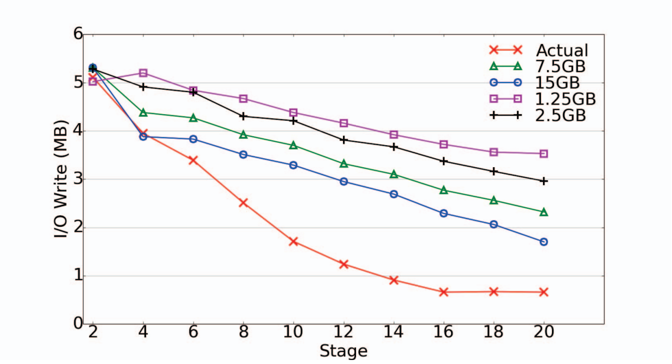

Figure 11: I/O Write Prediction for K-Means

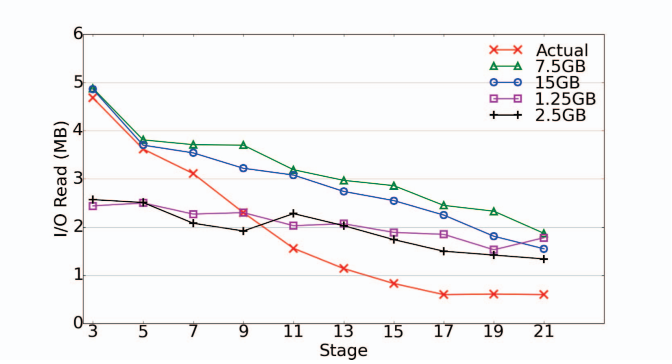

Figure 12: I/O Read Prediction for K-Means

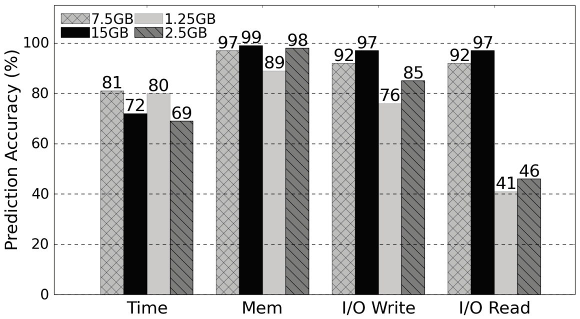

Figure 13: Prediction Accuracy for PageRank

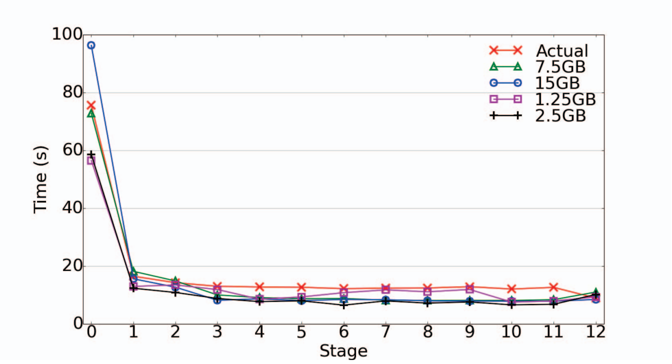

Figure 14: Time Prediction for PageRank

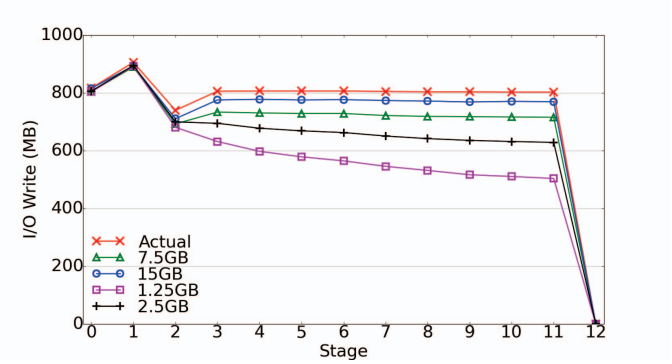

Figure 15: I/O Write Prediction for PageRank

D. PageRank

PageRank is an iterative algorithm with 13 stages where there are I/O read and write costs for each stage. In our evaluation, we used 25 GB of input data. Figure 13 shows the prediction accuracy. For time prediction, the accuracy is above 80% for simulating one task per core, but drops to around 70% for simulating two tasks per core (Figure 13 and Figure 14). This result may be due to the fluctuation in execution time across tasks within a stage. For I/O write cost prediction, the accuracy is above 90% for large-scale simulation (Figure 13 and Figure 15). For memory cost prediction, the accuracy is close to 97% for large-scale simulation (Figure 13). However, the I/O read prediction is below 50% based on small scale simulation (Figure 13 and Figure 16), which may be due to the inability to capture network activity in sufficient details in a small scale simulation.

V. Conclusion

This paper presents a performance prediction framework for jobs running on Apache Spark platform. We establish models for predicting job performance by simulating the execution of actual job in a limited scale on real cluster.

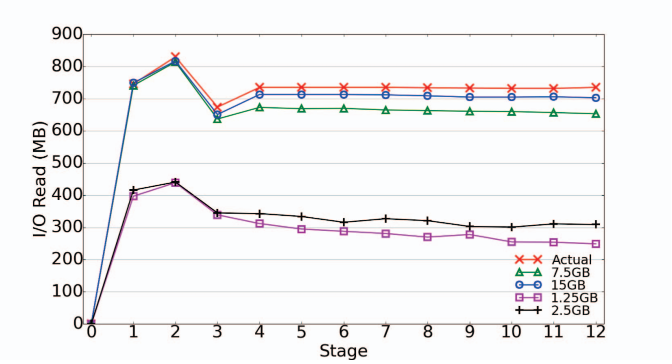

Figure 16: I/O Read Prediction for PageRank

The prediction accuracy is evaluated for iterative and non-iterative algorithms. While the prediction accuracy is found to be high for execution time and memory, the I/O cost prediction varied for different applications. We strongly believe that our proposed approach can be used to predict job execution time with high-accuracy in real-life and will lead towards better resource allocation framework.

Acknowledgment

This work is supported by the AFOSR under Grant No. FA 9550-15-1-0184. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agency.

References

[1] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, 2010.

[2] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 2–2.

[3] S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimization of resource provisioning cost in cloud computing," *Services Computing, IEEE Transactions on*, vol. 5, no. 2, pp. 164–177, 2012.

[4] P. Patel, D. Bansal, L. Yuan, A. Murthy, A. Greenberg, D. A. Maltz, R. Kern, H. Kumar, M. Zikos, H. Wu *et al.*, "Ananta: Cloud scale load balancing," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 207–218.

[5] Y. Tan, H. Nguyen, Z. Shen, X. Gu, C. Venkatramani, and D. Rajan, "Prepare: Predictive performance anomaly prevention for virtualized cloud systems," in *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*. IEEE, 2012, pp. 285–294.

[6] Apache hadoop. [Online]. Available: https://hadoop.apache.org/

[7] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[8] A. Ganapathi, Y. Chen, A. Fox, R. Katz, and D. Patterson, "Statistics-driven workload modeling for the cloud," in *Data Engineering Workshops (ICDEW)*. IEEE, 2010, pp. 87–92.

[9] A. D. Popescu, A. Balmin, V. Ercegovac, and A. Ailamaki, "Predict: towards predicting the runtime of large scale iterative analytics," *Proceedings of the VLDB Endowment*, vol. 6, no. 14, pp. 1678–1689, 2013.

[10] B. Mozafari, C. Curino, A. Jindal, and S. Madden, "Performance and resource modeling in highly-concurrent oltp workloads," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM, 2013, pp. 301–312.

[11] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu, "Starfish: A self-tuning system for big data analytics." in *CIDR*, vol. 11, 2011, pp. 261–272.

[12] E. Barbierato, M. Gribaudo, and M. Iacono, "Performance evaluation of nosql big-data applications using multi-formalism models," *Future Generation Computer Systems*, vol. 37, pp. 345–353, 2014.

[13] Z. Zhang, L. Cherkasova, and B. T. Loo, "Performance modeling of mapreduce jobs in heterogeneous cloud environments," in *Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing*. IEEE Computer Society, 2013, pp. 839–846.

[14] B. Mozafari, C. Curino, and S. Madden, "Dbseer: Resource and performance prediction for building a next generation database cloud." in *CIDR*, 2013.

[15] D. Didona, F. Quaglia, P. Romano, and E. Torre, "Enhancing performance prediction robustness by combining analytical modeling and machine learning," in *Proceedings of the International Conference on Performance Engineering (ICPE). ACM*, 2015.

[16] Apache hive. [Online]. Available: https://hive.apache.org/

[17] Json. [Online]. Available: http://json.org/

[18] Sloan digital sky survey. [Online]. Available: http://www.sdss.org/

[19] Stanford snap. [Online]. Available: http://snap.stanford.edu/