Modeling Interference for Apache Spark Jobs

Kewen Wang, Mohammad Maifi Hasan Khan, Nhan Nguyen and Swapna Gokhale
Department of Computer Science and Engineering
University of Connecticut
Email: kewen.wang@uconn.edu, maifi.khan@engr.uconn.edu,nhan.q.nguyen@uconn.edu,ssg@engr.uconn.edu

***Abstract*—To maximize resource utilization and system throughput, hardware resources are often shared across multiple Apache Spark jobs through virtualization techniques in cloud platforms. However, while the performance of these jobs running in virtualized environment can be negatively affected due to interference caused by resource contention, it is nontrivial to predict the effect of interference on job performance in such settings, which is critical for efficient scheduling of such jobs and performance troubleshooting. To address this challenge, in this paper, we develop analytical models to estimate the effect of interference among multiple Apache Spark jobs running concurrently on job execution time in virtualized cloud environment. We evaluated the accuracy of our models using four real-life applications (e.g., Page rank, K-means, Logistic regression, and Word count) on a 6 node cluster while running up to four jobs concurrently. Our experimental results show that the model can achieve high prediction accuracy, and ranges between 86% to 99% when the number of concurrent jobs are four and all start simultaneously, and ranges between 71% to 99% when the number of concurrent jobs are four and start at different times.**

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

I. Introduction

Among different cloud computing platforms, Apache Spark [1] is one of the recently popularized open-source platforms that is currently used by over 500 organizations, including companies such as Amazon, eBay and Baidu[1]. Apache Spark leverages the concept of resilient distributed datasets (RDDs) [2] and in-memory computation to enable fast processing of large volume of data, making it suitable for large-scale data analytic applications. However, while performance prediction in such systems is important to optimize resource allocation [3] [4] [5], it is nontrivial for Apache Spark jobs for several reasons as follows. First, the 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 difficult to predict job performance. Second and finally, with advancement in hardware technology, virtualization technique is increasingly being used to share resources among applications [6]. However, while virtualization isolates multiple applications running on separate virtual machines, the interference among these applications still affects the execution performance. Due to the aforementioned factors, modeling performance of multiple Apache Spark jobs running in a virtualized environment concurrently is extremely challenging. While our own prior effort looked into the problem of performance prediction for a single job running on Apache Spark platform [7], that approach does not address the challenge of performance modeling for multiple jobs running in parallel on the same cluster. To address this void, in this paper, we focus on modeling interference among multiple Apache Spark jobs, and predict the execution time of a job when interfered with other jobs. In contrast to hard to interpret machine learning approaches that are often used to predict system performance leveraging past system execution data, we apply analytical approach that can provide a better understanding regarding the observed behavior (e.g., execution slowdown), exposing the underlying interactions among multiple jobs [8] [9] [10] [11]. Specifically, we use jobs (implemented by us) to predict the slowdown ratio while running multiple jobs concurrently, and use the slowdown ratio to predict the execution time. As Apache spark jobs follow a multi-stage execution model (more details are discussed in Section III) and different stages have different characteristics (e.g., I/O intensive vs. CPU intensive), our framework develops interference models for each stage, and predict execution time for each stage separately. We evaluated our framework with four real-world applications, namely, Page Rank, K-means clustering algorithm, Logistic regression, and Word Count application. We vary the number of concurrent jobs up to 4 and predicted execution time for individual stages. While the prediction accuracy for individual stages vary, it ranges between 86% to 99% when the number of concurrent jobs are four and all start simultaneously, and ranges between 71% to 99% when the number of concurrent jobs are four and start at different times. The rest of the paper is organized as follows. Section II describes prior research that is related to our work. Section III explains the models that are used to predict job performance. Section IV presents the experimental results. Limitations of our current work is discussed in Section V. Finally, Section VI concludes the paper.

II. Related Work

With the proliferation of cloud computing platforms, significant volume of prior work looked into the problem of performance modeling in cloud settings and distributed systems in general [12], [13], [14], [15], [16], [17], [18], [19], [20], [21]. Among these, PREDIcT [12] looked into the problem of predicting runtime for network intensive iterative algorithms and focus on Hadoop MapReduce platform. Starfish [15]

[1] http://spark.apache.org/faq.html

leverages analytical approaches to predict job performance based on job simulation data. CloudScope [22] is one of the more recent efforts that employs a discrete-time Markov Chain model to predict the performance interference of co-resident applications by modeling an application as a sequence of job slices and estimating the probability of a job moving from one state to another considering different factors such as current workloads and slowdown. Matrix [23] utilizes machine learning methods to predict application performance on virtual machines by applying clustering methods to classify applications and predict the performance of new applications by comparing against the previously trained models. Interference modeling among multiple applications running on MapReduce framework is tried before as well for the purpose of efficient job scheduling [24]. However, this approach requires training using different combinations of applications, which can quickly become prohibitive. MIMP [25] presents a progress aware scheduler for Hadoop framework that applies regression model to train and predict task completion time based on past execution. HybridMR [26] presents another MapReduce scheduler for hybrid data center consisting of physical and virtual machines. This scheduler uses performance interference models to guide resource allocation, and applies linear and non-linear exponential regression model to capture CPU, I/O, and memory interference. While these prior efforts provide invaluable insight to the problem of performance modeling, however, most of them uses black-box approaches. Moreover, due to the stage execution model and in-memory computation feature of Apache Spark platform, it is non-trivial to apply these approaches as is for predicting effect of interference on job execution time. Hence, to complement prior efforts, we focus on developing data-driven analytical models for modeling interference among multiple Apache Spark jobs as follows.

III. Overview

In our model, each Apache Spark job consists of multiple execution stages where each stage implements distinct operations of an application program and are executed sequentially. Moreover, 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.

For example, if the input data size of the 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 the same as the input block number and the number of tasks in each stage is same within one Spark job, there will be 40 tasks in each stage. 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 multistage execution model, the main idea behind our work is as follows (Figure 1). First, for a given Apache Spark job, we predict the execution time for each stage leveraging the performance model developed based on performance of sample job with smaller input data set. Please note that this model is presented in our prior work [7], and assumes that there are no interference in the system from other jobs. Next, we estimate the slowdown ratio for a given number of jobs running concurrently by executing our simulation job, which is implemented by us (more details in Section IV) and is different than any of the four jobs that we used for evaluation. However, as the slowdown ratio due to interference among simulated jobs can be different compared to actual jobs, for a given job, we adjust the expected slowdown ratio by taking into account the actual job parameters such as input data size and disk I/O characteristics. Once we estimate the expected slowdown ratio, we estimate the execution time considering the interference. For completeness, we first briefly present the model that is used to predict execution time assuming no interference from our earlier work [7], and then present the model for predicting the slowdown ratio due to interference, which is the contribution of this work that allows us to predict execution time in the presence of interference among multiple jobs.

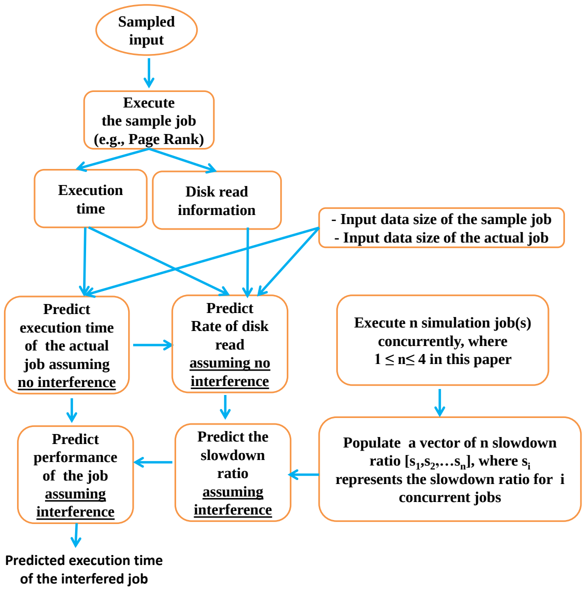
Sampled input
Execute the sample job (e.g., Page Rank)
Execution time
Disk read information
- Input data size of the sample job
- Input data size of the actual job
Predict execution time of the actual job assuming no interference
Predict Rate of disk read assuming no interference
Execute n simulation job(s) concurrently, where 1 ≤ n≤ 4 in this paper
Predict performance of the job assuming interference
Predict the slowdown ratio assuming interference
Populate a vector of n slowdown ratio [s1,s2,...sn], where si represents the slowdown ratio for i concurrent jobs
Predicted execution time of the interfered job

Fig. 1: Performance Prediction for Interfered Jobs

A. Model for Estimating Execution Time

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

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

$$Stage_i = \{Task_{i,j} \mid 0 \leq j \leq N\} \tag{2}$$

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:

$$JobTime = Startup + \sum_{s=1}^{M} StageTime_s + Cleanup \quad (3)$$

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:

$$P = \sum_{h=1}^{H} CoreNum_h \quad (4)$$

Here, $CoreNum_h$ is the number of CPU cores of working node h 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 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:

$$\begin{aligned} StageTime = Startup + \max_{c=1}^{P} \sum_{i=1}^{K_c} TaskTime_{c,i} \\ +Cleanup \end{aligned} \quad (5)$$

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:

$$\begin{aligned} TaskTime = DeserializationTime + RunTime \\ +SerializationTime \end{aligned} \quad (6)$$

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. 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 during the simulated run.

Next, to predict the execution time of the actual run based on the extracted performance metric from simulated run, 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 (4). 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 sampled data, we calculate the average execution time for a task for a given stage for a worker node h as follows.

$$\begin{aligned} TaskRunTime_{h,i} = DeserializeTime_{h,i} \\ +RunTime_{h,i} \\ +SerializationTime_{h,i} \end{aligned} \quad (7)$$

$$AvgTaskTime_h = \frac{1}{n_h} \sum_{i=1}^{n_h} TaskRunTime_{h,i} \quad (8)$$

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.

$$Ratio_h = \frac{\frac{1}{n_h - P_h} \sum_{i=P_h+1}^{n_h} TaskTime_{h,i}}{\frac{1}{P_h} \sum_{j=1}^{P_h} TaskTime_{h,j}} \quad (9)$$

Here n_h is the number of tasks running in host h, and P_h is the number of tasks in the first batch. 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:

$$\begin{aligned} EstStageTime = Startup + \max_{c=1}^{P} \sum_{i=1}^{K_c} AvgTaskTime_{c,i} \\ +Cleanup \end{aligned} \quad (10)$$

$$EstTaskTime_{c,i} = \begin{cases} AvgTaskTime_c, & i = 1 \\ AvgLaterTaskTime_c, & i > 1 \end{cases} \quad (11)$$

Here P is the number of total CPU cores calculated in equation (4), K_c is the number of sequential tasks running in CPU core c. $AvgTaskTime_c$ is the average time of first batch tasks in CPU core c of the corresponding host, and is calculated in equation (8). $AvgLaterTaskTime_c$ is the average time of the following batches of tasks, which could be calculated as $Ratio_h \times AvgTaskTime_h$. While we can apply the prediction model presented in this section, which was developed in our previous paper [7], to estimate the execution time for a single job assuming no interference, we still need a way to predict the slowdown ratio when interfered with other jobs, which we address as follows.

B. Modeling Interference

As different stages of a job is expected to have different characteristics in terms of resource utilization (e.g., CPU, I/O, memory), different stages of multiple jobs running concurrently on a system is expected to result in different interference pattern, affecting the execution time differently. Based on this observation, we model the slowdown ratio due to interference among multiple jobs for each stage separately. Towards that, in our model, each stage is represented as a vector consisting of execution time, CPU usage, disk I/O rate, and network I/O rate as follows.

$$Res_i = (RunTime_i, CPU_i, DiskIO_i, NetIO_i) \quad (12)$$

Here $1 \leq i \leq M$, and M is the number of stages in a job. Next, the slowdown resulting from interference with other applications for a particular stage is represented as follows.

$$SlowdownRatio(Stage_{i,k}) = f(Res_{i,k}, Res_{Otherjobs}) \tag{13}$$

Here $1 \leq k \leq J$, $1 \leq i \leq M$, J is the number of jobs running in parallel, M is the number of stages in the Apache Spark job. $Res_{Otherjobs}$ represents the resources consumed by other jobs that are running concurrently with $Stage_{i,k}$. Simply put, this slowdown ratio is the ratio between execution time with interference over execution time without interference for a particular stage. Hence, once we can estimate the value of the slowdown ratio and the expected execution time when there is no interference, we can estimate the execution time if there are interference with other jobs. As the slowdown happens primarily due to contention for bottleneck resources in the system, to better understand the underlying reasons behind the slowdown, we ran a series of experiments and collected job event logs and resource consumption data, and then extracted the resource usage profile for each stage. Job event log is generated by Apache Spark platform, and resource consumption data is collected using system monitoring tool dstat [27]. Apache Spark log records the time line of different stages of a running job, which was used to determine the submission and completion time of different stages of a job. The resource usage for different stages of a job is represented as below:

$$CPU_i = (CPUusr_i, CPUsys_i, CPUidle_i, CPUwait_i) \tag{14}$$

$$DiskIO_i = (RateofDiskRead_i, RateofDiskWrite_i) \tag{15}$$

$$NetIO_i = (RateofNetReceive_i, RateofNetSend_i) \tag{16}$$

Here $1 \leq i \leq M$, and M is the number of stages in a Apache Spark job. As an Apache Spark job uses in-memory data processing to reduce execution time, in the first stage of a job, it reads the input data to memory, and then analyzes the in-memory data in the subsequent stages. Due to this characteristic, in the first stage, frequent I/O is expected, leading to longer I/O wait. Based on this observation, as bulk of the disk I/O happens in the first stage, in our model, we calculate the slowdown ratio for the first stage only, and assume that the slowdown ratio in cases where the first stage interferes with the following stages from another job is 1.0 (i.e., the slowdown due to interference is expected to be minimal). Please note that, while this assumption is not accurate for certain jobs and stages, the error introduced due to this assumption in prediction accuracy is not significant.

As most of the time spent in the first stage is due to reading data from disk to memory, we represent the relationship between the amount of data read in the first stage (e.g., size of input data), the rate of disk read, and the execution time of the first stage as follows:

$$RunTime_{Stage1} = c \times \frac{InputDataSize}{RateofDiskRead_{Stage1}} \tag{17}$$

Now, if we assume that we execute the same job twice, once with reduced input data set (i.e., sample job) and once with the complete input data set (i.e., complete job), from equation 17, we can have the following. Please note that the word **Int.** refers to **Interference** in the following equations.

$$\begin{gathered}\frac{InputDataSize_{Samplejob}}{InputDataSize_{Completejob}} = \\ \frac{RateofDiskRead_{Samplejob,Stage1}}{RateofDiskRead_{Completejob,Stage1}} \times \\ \frac{RunTime_{SampleJob,Stage1}}{RunTime_{CompletejobwithoutInt.,Stage1}}\end{gathered} \tag{18}$$

Based on equation 18, we can have the following equation for predicting the rate of disk read for a complete job.

$$\begin{gathered}PredictedRateofDiskRead_{CompleteJobwithoutInt.,Stage1} = \\ RateofDiskRead_{SampleJob,Stage1} \times \\ \frac{RunTime_{SampleJob,Stage1}}{RunTime_{CompleteJobwithoutInt.,Stage1}} \times \frac{InputSize_{CompleteJob}}{InputSize_{SampleJob}}\end{gathered}$$

Please note that, in the above equation, we can estimate the value of $RunTime_{CompleteJobwithoutInt.,Stage1}$ using the model described in Section III-A from our earlier work [7]. Once we predict the rate of disk read for a complete job with no interference, next, we need to model the relation between the rate of disk read and the slowdown ratio when there is interference. For that, first, we run a simulation program (written by us as described in Section IV) to collect the runtime information with and without interference and calculate the parameter β_n as follows.

$$\begin{gathered}\beta_n = \frac{1}{RateofDiskRead_{SimulationRunwithoutInt.,Stage1}} \times \\ (\frac{RunTime_{SimulationRunwithInt.fornjobs,Stage1}}{RunTime_{SimulationRunwithoutInt.,Stage1}} - \\ \left\lfloor \frac{RunTime_{SimulationRunwithInt.fornjobs,Stage1}}{RunTime_{SimulationrunwithoutInt.,Stage1}} \right\rfloor)\end{gathered} \tag{19}$$

In this paper, we assume that there can be at most 4 concurrent jobs in a system, and varied n between 2 to 4 to calculate β_2, β_3, and β_4. Please note that running the simulation job and calculating β_n takes only few minutes and need to be done only once, making this approach quite scalable. Next, we use β_n to estimate the slowdown ratio when there are n concurrent jobs in the system as follows.

$$\begin{gathered}SlowdownRatio(Stage_{(1,k)}) = \\ \frac{RunTime_{CompleteJobwithInt.,Stage1}}{RunTime_{CompleteJobwithoutInt.,Stage1}} = \\ \beta_n \times PredictedRateofDiskRead_{CompleteJobwithoutInt.,Stage1} + \\ \left\lfloor \frac{RunTime_{SimulationrunwithInt.,Stage1}}{RunTime_{SimulationRunwithoutInt.,Stage1}} \right\rfloor\end{gathered} \tag{20}$$

C. The Cascading Effect

Given the above formulation, if we assume that all the jobs are of same type and start at the same time, modeling interference is straightforward as they all have the same execution behavior in each stage. However, for interference among different types of jobs possibly starting at different times, this is a bit more complicated due to possible cascading effect. For example, the slowdown of stage 1 of job A may push this stage to interfere with stage 2 of job B. Hence, a dynamic interference estimation algorithm is designed to solve this problem. The main idea behind the algorithm is as follows. First, the algorithm uses the execution time line of each job as input, and calculates the slowdown ratio for each stage of different jobs within the same time slot, and generates the execution time line of each job under interference condition. Based on that, the algorithm identifies the job that will finish its first stage the earliest, removes that job from the list, and recalculates the effect of interference for the remaining jobs for the remainder of the execution time. The algorithm applies this repeatedly until the list becomes empty. This dynamic interference estimation algorithm is described in Algorithm 1.

IV. Evaluation

To evaluate the accuracy of our modeling framework, we used a cluster of 6 nodes and used Xen hypervisor [28] to create up to four virtual machines on each physical machine. Each virtual machine is configured with 4GB of memory and 1 CPU core. For the deployed Apache Spark platform, one machine serves as the master node, and the remaining five machines serve as working nodes. In six physical machines, we create multiple clusters leveraging virtual machines to execute multiple Apache Spark jobs in parallel. In our evaluation, for prediction, first, we need to estimate the parameter β_n in equation (19). Towards that, we implemented our own Apache Spark job and executed that on our cluster to obtain the execution time and resource consumption information. This simulation job consists of three stages executing distinct(), groupByKey(), and count() operation respectively. Distinct() implements a mapping function and parses the input data, groupByKey() processes the output of distinct() operation, and count() is a CPU intensive operation performing data summarization. This simulation job is executed with 2.5 GB of sample data where the first stage implementing the Distinct() operation involves significant I/O compared to the following two stages. To measure β_n, we executed n (n=1,2,3,4) instances of this simulation job in parallel. As shown in Figure 2, the effect of interference is significant for the first stage but minimal for the subsequent stages. Once we estimate the value of β_n, subsequently, we used our formulation to predict the execution time for each stage for each job separately in different execution scenarios and add up the prediction error for each stage to calculate the total prediction accuracy R as below.

$$R = |1 - \sum_{i=1}^{M} \frac{|PredictedTime_i - MeasuredTime_i|}{\sum_{j=1}^{M} Time_j}| \quad (21)$$

Here M is the number of stages in a job, $PredictedTime_i$ is the predicted execution time for $stage_i$, and $MeasuredTime_i$ is the actual execution time of $stage_i$. Different evaluation scenarios are presented below.

```
Input: List $JobProfiles$ listing Execution Information
       without Interference
Output: List $JobTime$ listing Execution Time with
        Interference
Function PredictJobExecution
    Initialize List $Phases$, List $JobTime$;
    for all $job \in JobProfiles$ do
        $Phases$.add($job$.getStage(0)); //first stage
    end
    while $Phases.size > 0$ do
        Initialize MinTime $\leftarrow$ MaxValue;
        for all $phase \in Phases$ do
            r $\leftarrow$ $phase$.calculateSlowdownRatio($Phases$);
            phaseTime $\leftarrow$ $phase$.getStageTime() $\times$ r;
            $phase$.setPhaseTime(phaseTime);
            $phase$.setSlowdownRatio(r);
            if $phaseTime < MinTime$ then
                MinTime $\leftarrow$ phaseTime;
            end
        end
        for all $phase \in Phases$ do
            if $phase.getPhaseTime = MinTime$ then
                StageTimeInterfere $\leftarrow$ MinTime +
                $phase$.getPartialTime();
                $JobTimes$.add($phase$
                ,StageTimeInterfere);
                $Phases$.remove($phase$);
                if $JobProfiles.hasNextStage(phase)$
                then
                    nextStage $\leftarrow$
                    $JobProfiles$.nextStage($phase$);
                    $Phases$.add(nextStage);
                end
            else
                $phase$.setStageTime($phase$.getStageTime()
                $- \frac{MinTime}{phase.getSlowdownRatio()}$);
                $phase$.setPartialTime($phase$.getPartialTime
                + MinTime);
            end
        end
    end
end
```

Algorithm 1: Interference Estimation Algorithm

A. Interference Among Multiple Jobs of the Same Type Starting Simultaneously

In this part of the evaluation, we present the accuracy of prediction while modeling the effect of interference among multiple jobs of the same type (e.g., interference between n instances of job x). Towards that, we choose four Apache Spark jobs: PageRank, K-Means, Logistic Regression and WordCount. WordCount job is a non-iterative job while the

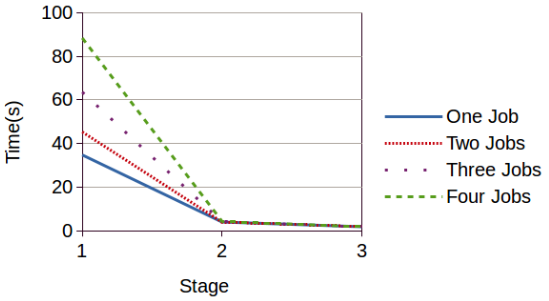

Fig. 2: Execution Time for Different Number of Simulation Jobs

TABLE I: Prediction Accuracy for Interference Among Same Jobs

JobName	Job Number	First Stage	Whole Job
PR	2	0.97	0.80
	3	0.96	0.85
	4	0.92	0.82
KM	2	0.75	0.70
	3	0.71	0.68
	4	0.98	0.92
LR	2	0.74	0.78
	3	0.79	0.81
	4	0.97	0.97
WC	2	0.87	0.86
	3	0.96	0.94
	4	0.95	0.94

remaining three are iterative jobs. For PageRank, we use the LiveJournal network dataset from SNAP [29], which is processed through mapping each node id into longer string to form a 20 GB input data set. K-Means and Logistic Regression applications use 20 GB of numerical Color-Magnitude Diagram data of galaxy from Sloan Digital Sky Survey (SDSS) [30]. WordCount application uses 20 GB of Wikipedia dump data. For prediction, we first executed the sample job (e.g., Page rank) with 2.5 GB of input data to collect the job execution profile, which is then used to predict the execution time assuming no interference. Finally, we used our framework to adjust the prediction assuming interference. The prediction accuracy is summarized in Table I. In the table, PR, KM, LR, and WC refers to PageRank, K-Means, Logistic Regression, and WordCount application respectively. Column Job number (e.g., 2, 3, 4) indicates the number of jobs that were executed in parallel. For instance, a value of 2 indicates that two instances of the same job were executed in parallel. As can be seen, prediction accuracy is highest for Logistic regression application (97%) and lowest for K-means (68%).The predicted execution time and the actual execution time when we executed four instances of the same job in parallel are shown in Figure 3, Figure 4, Figure 5, and Figure 6 for PageRank, K-Means, Logistic Regression, and WordCount respectively.

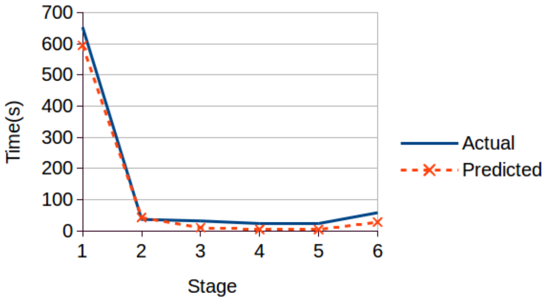

Fig. 3: Execution Time Prediction for Four Interfered PageRank Jobs

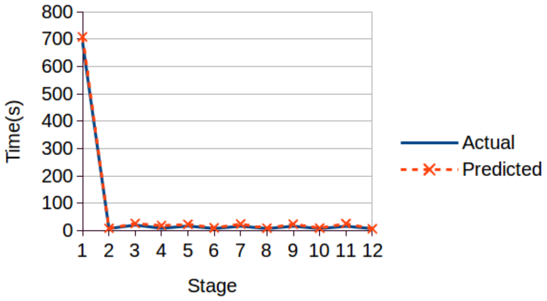

Fig. 4: Execution Time Prediction for Four Interfered K-Means Jobs

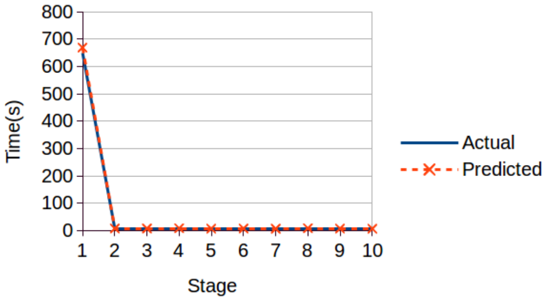

Fig. 5: Execution Time Prediction for Four Interfered Logistic Regression Jobs

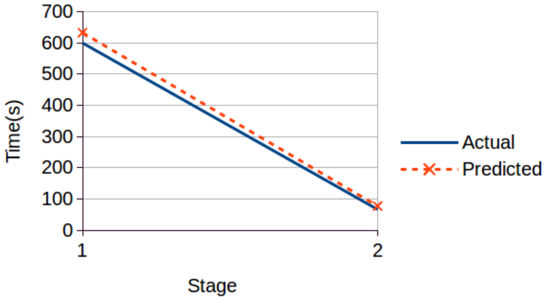

Fig. 6: Execution Time Prediction for Four Interfered WordCount Jobs

B. Interference Among Multiple Jobs of Different Types Starting Simultaneously

In this section, we present the accuracy of prediction while modeling the interference among n different jobs concurrently,

TABLE II: Prediction Accuracy for Two Different Jobs

JobName	Interfered Job	First Stage	Whole Job
PR	KM	0.91	0.79
	LR	0.93	0.81
	WC	0.99	0.85
KM	PR	0.89	0.80
	LR	0.80	0.73
	WC	0.75	0.69
LR	PR	0.97	0.97
	KM	0.73	0.77
	WC	0.70	0.75
WC	PR	0.96	0.87
	KM	0.93	0.84
	LR	0.96	0.88

TABLE III: Prediction Accuracy for Three Different Jobs

JobName	Interfered Jobs	First Stage	Whole Job
PR	KM, LR	0.96	0.87
	KM, WC	0.99	0.90
	LR, WC	0.99	0.90
KM	PR, LR	0.84	0.79
	PR, WC	0.92	0.87
	LR, WC	0.83	0.80
LR	PR, KM	0.84	0.85
	PR, WC	0.87	0.88
	KM, WC	0.83	0.84
WC	PR, LR	0.93	0.87
	PR, KM	0.93	0.87
	KM, LR	0.94	0.89

where n was varied between 2 to 4. For example, when n = 2, we execute two different jobs concurrently. The prediction accuracy while running two different jobs in parallel is summarized in Table II. As shown in the table, there are a total of 6 combinations to consider. As can be seen, prediction accuracy ranges between 97% and 69% for the whole job, and between 99% and 70% for the first stage, which incurs the bulk of the execution time. For n=3, we execute three different jobs concurrently. The prediction accuracy while running three different jobs in parallel is summarized in Table III. As shown in the table, there are a total of 4 combinations to consider. As can be seen, prediction accuracy ranges between 90% and 79% for the whole job, and between 99% and 83% for the first stage. Finally, for n=4, we execute four different jobs concurrently. The prediction accuracy while running four different jobs in parallel is summarized in Table IV. As shown in the table, there are a total of 1 combination to consider. As can be seen, prediction accuracy ranges between 99% and 86% for the whole job, and between 99% and 92% for the first stage. The predicted execution time and the actual execution time when we executed four different jobs in parallel are shown in Figure 7, Figure 8, Figure 9, and Figure 10 for PageRank, K-Means, Logistic Regression, and WordCount respectively.

TABLE IV: Prediction Accuracy for Four Different Jobs

JobName	Interfered Jobs	First Stage	Whole Job
PR	KM, LR, WC	0.92	0.86
KM	PR, LR, WC	0.99	0.95
LR	PR, KM, WC	0.99	0.99
WC	PR, KM, LR	0.95	0.90

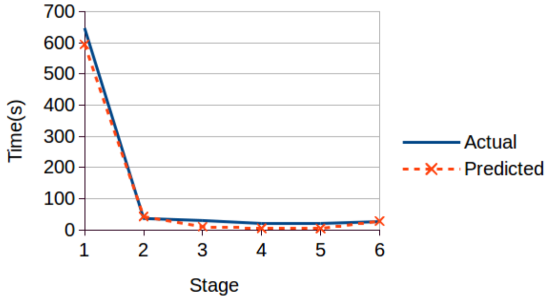

Fig. 7: Execution Time Prediction for PageRank Job Interfered with Other Three Jobs

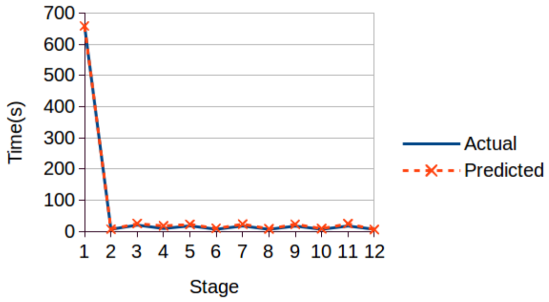

Fig. 8: Execution Time Prediction for K-Means Job Interfered with Other Three Jobs

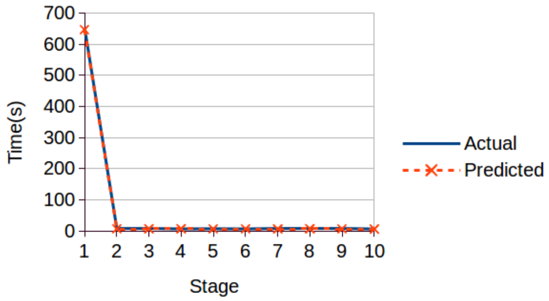

Fig. 9: Execution Time Prediction for Logistic Regression Job Interfered with Other Three Jobs

C. Interference Among Multiple Jobs Starting at Different Times

Finally, to test the prediction accuracy of our model where different jobs may arrive and start at different times, we use the four Apache Spark jobs and input data set as before, and start them randomly at different times. To ensure that each job will interfere with at least one other job while executing, we set the starting time for each job as $startingTime \in [minstagetime/10, minstagetime/2]$, where $minstagetime$ represents the smallest execution time

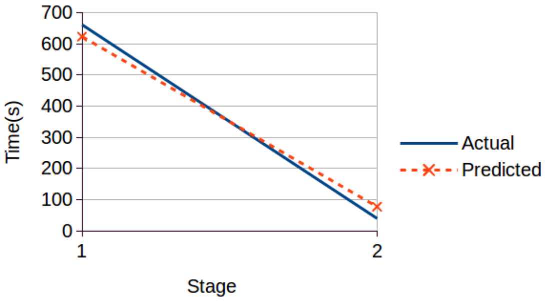

Fig. 10: Execution Time Prediction for WordCount Job Interfered with Other Three Jobs

TABLE V: Prediction Accuracy for Interference Among Different Jobs Starting at Different Times

Run	JobName	Starting Time(s)	First Stage	Whole Job
Scenario - I	PR	0	0.91	0.81
	KM	38	0.99	0.94
	LR	26	0.94	0.94
	WC	78	0.83	0.82
Scenario - II	PR	91	0.90	0.82
	KM	0	0.79	0.77
	LR	48	0.87	0.88
	WC	53	0.99	0.93
Scenario - III	PR	20	0.99	0.90
	KM	87	0.98	0.91
	LR	0	0.84	0.85
	WC	48	0.98	0.91
Scenario - IV	PR	77	0.93	0.85
	KM	25	0.72	0.71
	LR	86	0.99	0.99
	WC	0	0.99	0.93

for the first stage among all the jobs. In our case, $minstagetime = 190sec$, causing the starting time for different jobs to be between 19 sec and 95 sec.

Given the above range, for evaluation, we randomly pick one job and start at time 0, and then set the starting time for the remaining three jobs between 19 sec and 95 sec randomly. We consider four scenarios where the starting job is different in each scenario. The prediction accuracy for the whole job and the while running four different jobs in parallel starting at different times is summarized in Table V. As shown in the table, in our evaluation, prediction accuracy ranges between 99% and 71% for the whole job, and between 99% and 72% for the first stage. The predicted execution time and the actual execution time for PageRank, K-Means, Logistic Regression, and WordCount under Scenario - I are shown in Figure 11, Figure 12, Figure 13, and Figure 14 respectively.

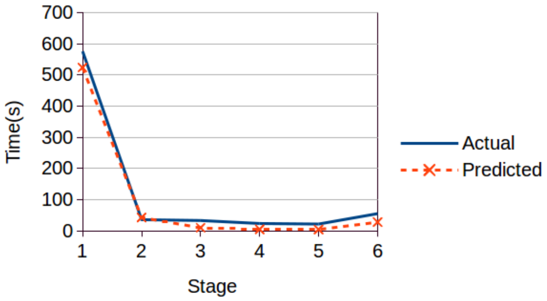

Fig. 11: Execution Time Prediction for PageRank Job in Scenario - I

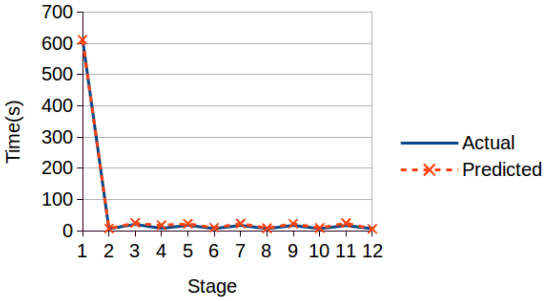

Fig. 12: Execution Time Prediction for K-Means Job in Scenario - I

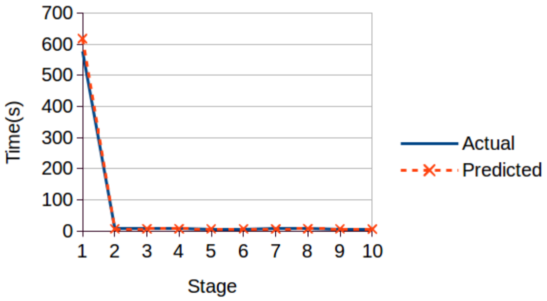

Fig. 13: Execution Time Prediction for Logistic Regression Job in Scenario - I

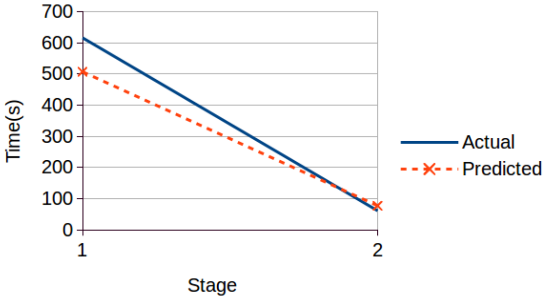

Fig. 14: Execution Time Prediction for WordCount Job in Scenario - I

V. Discussion

While our model can predict performance degradation due to interference among multiple Apache Spark jobs with high accuracy, we do acknowledge several limitations of our current work as follows. First, our model assumes that the stages within a job are executed sequentially and does not consider the possibility of parallel execution, which will require extending our models. Second, the model was evaluated on 6 node cluster with 4 concurrent jobs, which is smaller compared to real-life clusters. Furthermore, the value β_n depends on the number of concurrent jobs and needs to be calculated (once) as the number of concurrent jobs grows in a system. However, we strongly believe that our modeling framework will work well once the parameters are estimated for different cluster size and number of concurrent jobs in a system, which we plan to investigate in our future work.

VI. Conclusion

In this paper, to predict the execution time of Apache Spark job interfered with other jobs, we develop an interference model. This model combines the execution information and resource consumption profile for each stage of Apache Spark jobs to calculate the slowdown ratio resulting from the interference, and then predict the execution time when interfered with other jobs. The model is evaluated using four real-life applications (e.g., Page rank , K-means, Logistic regression, Word count) on a 6 node cluster while running up to four jobs concurrently. Experimental results demonstrate that our model can achieve high prediction accuracy. We strongly believe that the presented model can be leveraged to design efficient job schedulers and resource allocation algorithms for Apache Spark platform, thereby improving the system utilization significantly.

VII. Acknowledgement

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] M. Khan, Y. Jin, M. Li, Y. Xiang, and C. Jiang, "Hadoop performance modeling for job estimation and resource provisioning," 2015.

[4] D. Cheng, J. Rao, C. Jiang, and X. Zhou, "Resource and deadline-aware job scheduling in dynamic hadoop clusters," in *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*. IEEE, 2015, pp. 956–965.

[5] C. Delimitrou and C. Kozyrakis, "Quasar: resource-efficient and qos-aware cluster management," *ACM SIGPLAN Notices*, vol. 49, no. 4, pp. 127–144, 2014.

[6] B. P. Sharma, T. Wood, and C. R. Das, "Hybridmr: A hierarchical mapreduce scheduler for hybrid data centers," in *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*. IEEE, 2013, pp. 102–111.

[7] K. Wang and M. M. H. Khan, "Performance prediction for apache spark platform," in *High Performance Computing and Communications (HPCC), 2015 IEEE 17th International Conference on*. IEEE, 2015, pp. 166–173.

[8] K. Ousterhout, R. Rasti, S. Ratnasamy, S. Shenker, and B.-G. Chun, "Making sense of performance in data analytics frameworks," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 2015, pp. 293–307.

[9] Q. Noorshams, A. Busch, A. Rentschler, D. Bruhn, S. Kounev, P. Tuma, and R. Reussner, "Automated modeling of i/o performance and interference effects in virtualized storage systems," in *Distributed Computing Systems Workshops (ICDCSW), 2014 IEEE 34th International Conference on*. IEEE, 2014, pp. 88–93.

[10] Q. Zhu and T. Tung, "A performance interference model for managing consolidated workloads in qos-aware clouds," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. IEEE, 2012.

[11] C. A. Lai, Q. Wang, J. Kimball, J. Li, J. Park, and C. Pu, "Io performance interference among consolidated n-tier applications: Sharing is better than isolation for disks," in *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*. IEEE, 2014, pp. 24–31.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

[22] X. Chen, L. Rupprecht, R. Osman, P. Pietzuch, F. Franciosi, and W. Knottenbelt, "Cloudscope: Diagnosing and managing performance interference in multi-tenant clouds," in *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2015 IEEE 23rd International Symposium on*. IEEE, 2015, pp. 164–173.

[23] R. C. Chiang, J. Hwang, H. H. Huang, and T. Wood, "Matrix: Achieving predictable virtual machine performance in the clouds," in *11th International Conference on Autonomic Computing (ICAC 14)*, 2014.

[24] X. Bu, J. Rao, and C.-z. Xu, "Interference and locality-aware task scheduling for mapreduce applications in virtual clusters," in *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*. ACM, 2013, pp. 227–238.

[25] W. Zhang, S. Rajasekaran, T. Wood, and M. Zhu, "Mimp: deadline and interference aware scheduling of hadoop virtual machines," in *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*. IEEE, 2014, pp. 394–403.

[26] B. P. Sharma, T. Wood, and C. R. Das, "Hybridmr: A hierarchical mapreduce scheduler for hybrid data centers," in *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*. IEEE, 2013, pp. 102–111.

[27] Dstat: Versatile resource statistics tool. [Online]. Available: http://dag.wiee.rs/home-made/dstat/

[28] Xen project. [Online]. Available: http://www.xenproject.org/

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

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