Performance Models of Data Parallel DAG Workflows for Large Scale Data Analytics

Juwei Shi
STCA
Microsoft Cooperation
juwei.shi@microsoft.com

Jiaheng Lu
Department of Computer Science
University of Helsinki
jiaheng.lu@helsinki.fi

***Abstract*—Directed Acyclic Graph (DAG) workflows are widely used for large-scale data analytics in cluster-based distributed computing systems. Building an accurate performance model for a DAG on data-parallel frameworks (e.g., MapReduce) is critical to implement autonomic self-management big data systems. An accurate performance model is challenging because the allocation of preemptable system resources among parallel jobs may dynamically vary during execution. This resource allocation variation during execution makes it difficult to accurately estimate the execution time. In this paper, we tackle this challenge by proposing a new cost model, called Bottleneck Oriented Estimation (BOE), to estimate the allocation of preemptable resources by identifying the bottleneck to accurately predict task execution time. For a DAG workflow, we propose a state-based approach to iteratively use the resource allocation property among stages to estimate the overall execution plan. Extensive experiments were performed to validate these cost models with HiBench and TPC-H workloads. The BOE model outperforms the state-of-the-art models by a factor of five for task execution time estimation.**

I. Introduction

There is a trend towards automatic configuring and managing thousands of database nodes to enable the self-management feature for big data systems. The big data analytics jobs are often represented by Directed Acyclic Graph (DAG) workflows [1], [2], [20], [26], [27], [41]. A DAG of computational stages are built for parallel execution. For example, (1) Hive Query Language (HQL) is translated to the execution plan of MapReduce [10] jobs to be run in parallel, (2) the Spark [41] program and machine learning workloads are transformed to a DAG workflow for execution [2], [27], and (3) the Tez [1] framework allows for a complex DAG of tasks for processing data. Performance model and optimization for DAG workflows is critical to implement self-management big data systems.

The performance of DAGs are widely studied in the literature [16], [25], [26], [28], [33]. While these works expose various aspects of the performance behavior of DAGs, a step forward is required to build a cost model that estimates DAG execution time for parallel jobs. Cost models are the fundamental building blocks for system management and optimization, for example, (1) job self-tuning [16], [31], (2) capacity planning on the cloud [17], and (3) progress estimation [28]. However, existing cost models are limited to single jobs [16], [23], [31], and it is still a challenge to build cost models for a DAG workflow of parallel jobs (i.e., many real computing workloads) on cluster-based distributed computing such as MapReduce, Spark, and Tez.

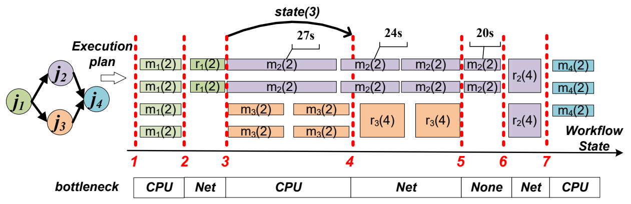


Fig. 1. The task execution plan of a DAG with four jobs. $m_i(k)$ denotes that a map task of job i requires k task slots (i.e., maximum number of Map or Reduce tasks can run simultaneously) for execution.

For data parallel computing frameworks, a precise yet useful cost model often measures the job execution time (i.e., beyond simple cost like I/O) [16], [31]. The main challenge in building an execution time based cost model for a DAG workflow is the inherent complexity of system resource allocation for heterogeneous tasks in each stage. This allocation may vary among computational stages. This is caused by two main factors that may vary among different stages: (1) the degree of parallelism (i.e., the number of simultaneously running tasks in the cluster) for parallel jobs, and (2) system resource bottleneck. Given the cluster computing resource, the degree of parallelism is determined by the resource requirement (i.e., CPU cores and memory) of running jobs. The resource requirement of tasks may be changed from one stage to the next due to computation stage changes, which may lead to the change of the degree of parallelism for each job. Then, the bottleneck resource may be changed from one stage to the next stage. It finally leads to the variation of the allocation for preemptable system resource and task execution time accordingly.

We use a DAG of web site analytics [8] in MapReduce to illustrate the above challenge. As shown in Figure 1, The DAG has four jobs to process the event log of page views to report the metrics. Job 1 pre-aggregates the duration of each visit to generate records that contain the page, visiting IP and duration on the page. Job 2 counts the number of views for each page (i.e., Word Count like job). Job 3 sorts the pages by the duration of each visit (i.e., Sort like job). Finally, job 4 generates a report for the pages of min, median and max duration on each page. The DAG workflow is divided into 7 stages (states) [1] based on the start and end of Map/Reduce stages. The task execution plan shows why the execution time estimation is challenging because of the parallel execution of job 2 and 3. In the 3rd state, the map task time for job 2 is 27 seconds, bounded by CPU. In the 4th state, the system bottleneck becomes network I/O due to the shuffle operation for job 3. The map task time for job 2 is reduced from 27 seconds to 24 seconds because its CPU resource allocation is increased. In the 5th state, there are only two map tasks in the cluster. The map task time for job 2 is further reduced from 24 seconds to 20 seconds due to the released CPU resources from job 3. In summary, the execution time of map tasks of job 2 varies between the 3rd and 5th state due to the variation of system bottlenecks (i.e., CPU-bound, network-bound and none). It indicates that the execution time of the same task may vary from one stage to the other due to the variation of system resource allocation. Unfortunately, the previous cost models such as Starfish [16] and MRTuner [31] are not able to capture the variation of resource allocation among stages because the degree of parallelism is assumed to be unchanged for a single job.

In this paper, we study the cost models for a DAG workflow on data parallel frameworks (i.e., MapReduce). We use MapReduce programming paradigm because it is a well-known framework in distributed computing, and the result is easy to be extended to other cluster-based distributed systems such as Spark and Tez, of which the key mechanisms for execution model, task distribution and fault-tolerance are similar. A starting point of our study is a thorough understanding of the system behavior for parallel jobs, by using a set of benchmarks. We have two findings to build cost models:

(1) The task execution time variation is caused by the change of system resource bottleneck among computation stages. The cost model for parallel jobs should be able to handle bottleneck resource estimation. (2) For each computation stage, the resource allocation for each running job is steady. This property can be used to estimate the DAG execution plan break-down in an iterative manner.

We propose the Bottleneck Oriented Estimation (BOE) model to estimate the execution time at the task level. The model estimates the bottleneck resource and its allocation among tasks by predicting the cost of each type of tuple level operations (i.e., read, transfer, compute and write). The pipelined and blocked operations are modeled separately. The effective time of the identified bottleneck resource is derived as the execution time for the pipelined operations. The BOE model identifies the bottleneck resource and accurately estimates the task execution time variation (e.g., 27s, 24s and 20s for task m_2 among the stages in Figure 1).

Next, we use a state-based approach to integrate the task-level BOE model in holistic estimation for the execution plan of a DAG workflow. For each stage of a DAG workflow, we estimate the degree of parallelism for each job using the properties of schedulers and estimate the task-level execution time for each job through the BOE model. Then, we iteratively estimate the task execution plans for parallel jobs on each stage. The workflow level execution time (e.g., the DAG execution time from stage 1 to stage 7 in Figure 1) is estimated by this state-based iterative approach.

[1]In this paper, the terms stage and state, DAG and workflow, are used interchangeably.

The key contributions of this paper are as follows.

- We study a set of workloads to thoroughly understand the system behavior for typical parallel DAG jobs. Our key insight includes (1) the key reason for task time variation during DAG execution is because of the change of the underlying resource bottlenecks, and (2) the resource allocation for parallel tasks is steady during each computation stage. This insight guides the design of the cost model.
- We propose a BOE model to estimate task-level execution time for parallel jobs. To the best of our knowledge, this is the first general cost model that addresses the problem of preemptable resources allocation for parallel jobs.
- We use a state-based approach to integrate the BOE model for a holistic execution plan estimation of a DAG workflow. This framework makes use of the property that the resource bottleneck is steady during a stage, and iteratively estimates the execution plan for parallel tasks from one stage to the next stage.
- We conduct extensive experiments with hybrid analytics benchmarks (HiBench) and query benchmarks (TPC-H) to validate the cost models. The result shows that the BOE model can correctly identify resource bottlenecks. As a comparison, the BOE model outperforms existing MapReduce models including Starfish and MRTuner by a factor of five for task-level estimation. For the state-based approach, the average prediction error is under 3% when predicting the execution time of 51 hybrid analytics and query DAG workflows.

The rest of the paper is organized as follows. We provide the background and problem description in Section II. We propose the task-level execution model in Section III. In Section IV, we present the holistic cost model for a DAG workflow. The evaluation results are presented in Section V. We present the related work in Section VI and conclude in Section VII.

II. Background and Problems

A. MapReduce

MapReduce is a data-parallel computing framework to execute user-defined map and reduce functions in parallel. A MapReduce *job* is divided into three *stages*: map, shuffle and reduce. Each *stage* has parallel *tasks* for execution.

Map: The map stage reads input tuples to execute the user-defined map function, and writes the results (e.g., `(k1, v1), (k1, v2)`) to the local disk for a shuffle. In the case when the intermediate map output cannot fit into the memory, the framework uses external merge & sort for generating the map output. Users can choose to compress map output to trade CPU overhead for disk and network I/O reduction.

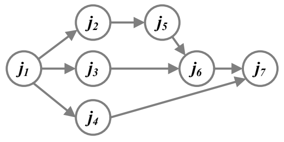


Fig. 2. Example of a DAG Workflow

Shuffle: The shuffle stage is responsible for copying the intermediate map output to the reduce side. The task may read data from the OS buffer caches during the shuffle stage when the intermediate data is just written by the previous stage. To reserve memory for the user-defined reduce function, the reduce input is materialized on the disk before each tuple is sent to the reduce function for processing.

Reduce: The reduce stage processes the value list for each key (e.g., `(k1, List(V1, V2))`), and writes the output to HDFS. By default, there are three replicas configured for the reduce output.

B. Resource Management and Job Scheduling

The job scheduler is responsible for assigning tasks based on the availability of system resources on nodes. To separate the task scheduling and resource management, Apache YARN [35] uses a resource manager to monitor and allocate resources for multiple jobs. The resource manager provides multi-dimensional fairness (e.g., Dominant Resource Fairness, DRF [14]).

In this paper, we follow the scheduling model in YARN for parallel tasks. For both single job and parallel job cases, the tasks in a computation stage are scheduled based on DRF.

C. DAG Workflow

Directed Acyclic Graph (DAG) based execution is popular for modern data analytics workloads. For example, (1) the physical execution plan for HiveQL is a DAG of MapReduce jobs [34]; (2) SystemML [3], [15] compiles DML (Declarative Machine learning Language) to a DAG of hybrid MapReduce jobs and control programs; (3) Spark [41] transforms the user-defined analytic program to a DAG workflow for parallel execution.

In this paper, we define the DAG workflow as follows.

Definition 1: A **DAG workflow** is composed of a set of jobs connected through a DAG relationship $G_F(J, E)$, where J is the set of jobs that compose the workflow, and the arc $(j_m, j_n) \in E$ indicates that the start of j_n depends on the completion of j_m.

Figure 2 presents an example of such a DAG workflow composed of 7 MapReduce jobs: (1) A job in the workflow is started if and only if all its parent jobs finish (e.g., j_6 has to wait for the completion of both j_3 and j_5), and (2) multiple jobs from the DAG can run simultaneously (e.g., j_2, j_3, and j_5 run in parallel).

D. Problem Definition

We formulate the problem of cost estimation for a DAG workflow as follows:

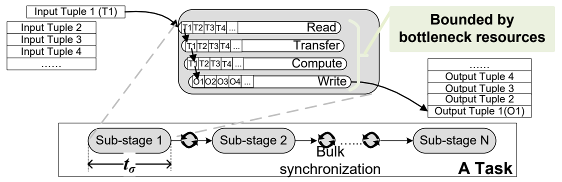


Fig. 3. The Task Execution Model

Problem 1: Given a DAG workflow $G(J, E)$ with job profile J and topology dependency E, the objective is to estimate the execution time $t(G, D, P, C)$ of G against data D with parameter sets P and cluster resources C.

III. Task-level Model

In this section, we present a cost model, Bottleneck Oriented Estimation (BOE), for task-level execution time estimation.

A. BOE Model

1) Task Execution Model: We first model the fundamental behavior for task execution on data parallel computing frameworks. As shown in Figure 3, we break down a task into multiple sub-stages. For each sub-stage, the task is executed in a pipelined fashion from one tuple to the next tuple, which consists of a *subset* of operations including reading, transferring, computing, and writing. There is bulk synchronization at the end of each sub-stage which blocks all the tuples to be processed by the next sub-stage. The task execution model is general for data parallel computing frameworks that follow the functional programming model (e.g., MapReduce, Spark, and Tez).

This execution model distinguishes pipelined and blocked processing in the tuple level, which formalizes task-level execution plans to predict the allocation of preemptable resources for parallel tasks.

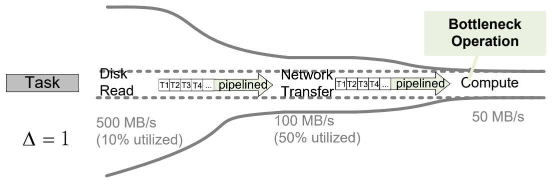
2) Resource Usage Model: Given the above task execution model, we use the resource usage model in [13] to make a *uniformity assumption* for resource usage behavior. For each sub-stage, since the subset of read, transfer, compute and write operations is executed in the pipeline from one tuple to the next tuple, the usage of preemptable resources is uniform during a sub-stage. We assume that disk and network are preemptable. CPU is preemptable when there is no free CPU core (e.g, the number of simultaneously running tasks is larger than that of CPU cores). Memory is not preemptable because it is managed by JVM.

This resource usage model follows the execution model to distinguish pipelined and blocked processing in the tuple level and provides the hint to estimate resource utilization (i.e., effective time) for a bottleneck resource.

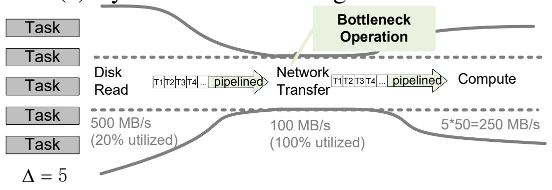
3) Bottleneck Oriented Estimation: Given the task execution model in Figure 3, we estimate the execution time t_σ for a sub-stage of a task as follows,

$$t_\sigma = \Lambda(t_{read}, t_{transfer}, t_{compute}, t_{write}) \qquad (1)$$

where $\Lambda(\cdot)$ estimates the non-overlapped time among $t_{\mathbb{X}}$ to process tuples in the pipeline, and $t_{\mathbb{X}}$ is the actual execution



(a) System Resource Usage with a Task



(b) System Resource Usage with 5 Parallel Tasks

Fig. 4. Example of BOE Model

time of the operation $\mathbb{X}$. Note that for specific sub-stages in MapReduce, we have a subset of operations according to the implementation. For example, there is only a disk write for map output sub-stage.

According to the resource usage model, the resources are uniform over the pipelined execution of tuples. Since the processing time for each tuple is very short, we omit the processing time for the first tuple and last tuple. Then, we have

$$t_{read} = t_{transfer} = t_{compute} = t_{write} \tag{2}$$

That is,

$$t_\sigma = \max\{t_{read}, t_{transfer}, t_{compute}, t_{write}\} \tag{3}$$

We assume that the resource throughput for $\mathbb{X}$ is $\theta_{\mathbb{X}}$. The resource usage for $\mathbb{X}$ is $\mu(\Delta)$ when X is fully utilized by tasks with Δ parallelism. Thus we have

$$t_{\mathbb{X}} = \frac{\mathbb{D}}{p_{\mathbb{X}} \cdot \mu(\Delta) \cdot \theta_{\mathbb{X}}} \tag{4}$$

where $p_{\mathbb{X}} \cdot \mu_{\mathbb{X}}(\Delta)$ is the actual resource usage for $\mathbb{X}$ when it is not a bottleneck. $\mathbb{D}$ is the size of data to process.

For the bottleneck resource $\mathbb{X}$, we have $p_{\mathbb{X}} = 1$ and $t_{\mathbb{X}} = \frac{\mathbb{D}}{\mu(\Delta)\cdot\theta_{\mathbb{X}}}$. Otherwise, $0 \le p_{\mathbb{X}} < 1$, and we have $t_{\mathbb{X}} < \frac{\mathbb{D}}{\mu(\Delta)\cdot\theta_{\mathbb{X}}}$.

If there is at least one bottleneck resource, we have

$$\begin{aligned} t_\sigma \quad = \quad & \max\{\frac{B}{\mu_{read}(\Delta)\cdot\theta_{read}}, \frac{B}{\mu_{transfer}(\Delta)\cdot\theta_{transfer}}, \\ & \frac{B}{\mu_{compute}(\Delta)\cdot\theta_{compute}}, \frac{s\cdot B}{\mu_{write}(\Delta)\cdot\theta_{write}}\} \end{aligned} \tag{5}$$

where B is the size of input for the task.

Example of the BOE Model: Figure 4 shows how the BOE model estimates task execution time. Suppose that there are 10 million records (with 100 bytes for each) to be processed on a node. For a task with one sub-stage, there are three pipelined operations including reading, network transferring and computing. The aggregated read throughput is 500 MB/s on the node. The network throughput on that node is 100 MB/s. For the task, the compute throughput using a CPU core is 50 MB/s.

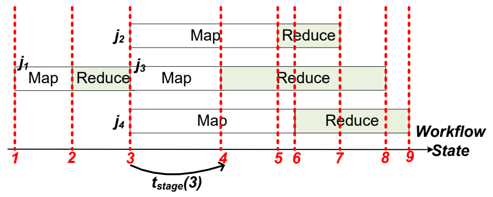


Fig. 5. Workflow State Transition

In Figure 4 (a), there is a task run on the node. According to the BOE model, the execution time of the task is $t_\sigma = \max\{\frac{10000\ MB}{500\ MB/s}, \frac{10000\ MB}{100\ MB/s}, \frac{10000\ MB}{50\ MB/s}\} = \max\{20s, 100s, 200s\} = 200s$. The disk utilization is $\frac{20}{200} = 10\%$, and the network utilization is $\frac{100}{200} = 50\%$. The task is bounded by a CPU core's processing bandwidth.

In Figure 4 (b), the degree of parallelism for tasks is increased to 5 on the same node. We assume that there are more than 5 cores on the node. The disk read throughput for each task is $\mu_{read}(5) \cdot \theta_{read} = \frac{1}{5} \cdot 500\ MB/s = 100\ MB/s$ when the disk read resource is fully utilized. The network throughput for each task is $\mu_{transfer}(5) \cdot \theta_{transfer} = \frac{1}{5} \cdot 100\ MB/s = 20\ MB/s$. According to the BOE model, the execution time of the sub-stage is $t_\sigma = \max\{\frac{10000\ MB}{100\ MB/s}, \frac{10000\ MB}{20\ MB/s}, \frac{10000\ MB}{50\ MB/s}\} = \max\{100s, 500s, 200s\} = 500s$. This means that the disk utilization is $\frac{100}{500} = 20\%$, and the network utilization is $\frac{500}{500} = 100\%$. That is, we have $p_{disk} = 20\%$ and $p_{transfer} = 100\%$. This is consistent with the resource throughput for disk read and network transfer (i.e., $\frac{20\ MB/s}{100\ MB/s} = 20\%$). The tasks are bounded by network bandwidth.

The example shows how the BOE model estimates the execution time for each task by identifying the bottleneck resources. The resource allocation is estimated with respect to the degree of parallelism.

IV. Workflow Level Model

In this section, we present the workflow level model to estimate the holistic execution plan of a DAG workflow.

A. The State-based Approach

The resource allocation for each job is steady during a stage that is divided by the map/reduce stages. We make use of this property to break down a DAG workflow into multiple stages and propose a state-based approach for a DAG workflow level estimation.

1) State division for a DAG workflow: We define the state (i.e., stage) s ($s = 1, 2, \cdots, S$) for a DAG workflow based the map or reduce stage transition of its jobs. As shown in Figure 5, the workflow state is transited from 3 to 4 when job j_3 is transformed from the map stage to the reduce stage. During the execution of a stage for a DAG workflow, the degree of parallelism Δ_i for the running job i will not change. Consequently, the allocation of shared bottleneck resources (i.e., disk, HDFS, and/or network) is fixed for each running job during a stage of a DAG workflow. This property provides the foundation to estimate the allocation of shared resources among running jobs of a DAG workflow.

Algorithm 1 State-based Cost Estimation for a DAG workflow

```
1: $t_{dag} \leftarrow 0$
2: $s \leftarrow 0$
3: while $G$ is not empty do
4:     Add new jobs to job queue $\mathcal{Q}$ from $G$
5:     $job\_end\_flag$=0
6:     while $job\_end\_flag = 0$ do
7:         Estimate $\Delta_i$ for each job $i \in \mathcal{Q}$
8:         for each job $i \in \mathcal{Q}$ do
9:             Estimate $t_{task}(i, s)$ using BOE model
10:            Estimate the stage time $t_{stage}(i, s)$
11:        $t_{stage}(s) = t_{stage}(k, s) = \min_{i=1}^{N_s}\{t_{stage}(i, s)\}$
12:        for each job $i \in \mathcal{Q}$ do
13:            update the progress for job $i$
14:        if job $k$ in reduce stage then
15:            $job\_end\_flag$=1
16:            $\mathcal{Q}.remove(k)$
17:        else
18:            update job $k$ to the reduce stage
19:        $t_{dag} = t_{dag} + t_{stage}(s)$
20:        $s = s + 1$
21: Return $t_{dag}$
```

2) Cost Estimation for a DAG workflow: Algorithm 1 presents the algorithm of the state-based approach to iteratively estimate the execution time for a DAG workflow. Given a DAG workflow, we estimate the state transition sequence $1 \rightarrow 2 \cdots \rightarrow S$ by iteratively estimating the duration for each workflow stage. For each iteration, we estimate the stage duration as follows: (1) estimate the degree of parallelism Δ_i for each running job i; (2) identify the bottleneck resource and task execution time for each running job using the BOE model; (3) estimate the rest of the execution time of the current stage for all the running jobs; (4) find the job with minimum stage duration time; (5) update the progress for other running jobs. Therefore, given a DAG workflow G, input data $\mathbb{D}$, cluster resources C, and historical profile P, we estimate the execution time for the DAG workflow by estimating the duration of stages: $t_{dag} = \sum_{s=1}^{S} t_{stage}(s)$.

As the example shown in Figure 5, when job j_1 completes (i.e., the DAG workflow enters the stage 3), we estimate the degree of parallelism Δ_3, Δ_4, and Δ_5, for j_2, j_3, and j_4, respectively. Next, we estimate the task execution time for each job, and estimate the state duration time $t_{stage}(3)$, and update the state and progress for each running job. Finally, we enter the state 4.

V. Evaluation

In this section, we conduct a set of experiments to evaluate the proposed cost models using a variety of representative DAG workflows. First, we evaluate the effectiveness of the BOE model in comparison with existing models including Starfish [16] and MRTuner [31], for both single job and multiple jobs. Second, we evaluate the state-based approach for the execution plan estimation of DAG workflows. Finally, we evaluate the latency overhead for the estimation, which validates the cost model application scenarios like DAG workflow auto-tuning.

A. Experimental Setup

The Hadoop clusters are deployed on identical hardware, with a total of eleven servers. Each node has 6 physical CPU cores at 2.4 GHz, 2 disk drives at 7.2k RPM with 500 GB each, and 32 GB of physical memory. Nodes are connected using a 1 Gbps Ethernet switch.

TABLE I
Overview of Workloads for Evaluation

		C	R	Bottleneck
Micro Single-Job	Word Count (WC)	Y	3	CPU
	TeraSort (TSC)	Y	1	CPU
	TeraSort (TS)	N	1	CPU, Disk
	TeraSort (TS3R)	N	3	CPU, Network
Micro Multi-Jobs	WC+TS	N	3, 1	CPU
	WC+TS3R	N	3, 3	CPU, Network
Hybrid	WC+TPC-H(Q1-Q22)	Y	3	Hybrid
	TS+TPC-H(Q1-Q22)	Y	3	Hybrid
	WC+KMeans	Y	3	Hybrid
	WC+PageRank	Y	3	Hybrid
	TS+KMeans	Y	3	Hybrid
	TS+PageRank	Y	3	Hybrid

We define a set of representative workloads for the experimental evaluation. As shown in Table I, we use Word Count and TeraSort for micro-benchmarks. We use PageRank for graph analysis and Kmeans for machine learning, both from HiBench [19]. The query workload is selected from TPC-H[2]. **C** represents the compression is enabled or not. **R** denotes the number of replicas. The hybrid workload means to run two jobs/queries in parallel. For WC and TS, we use 100 GB input. We use the `huge` data set for Kmeans and PageRank in HiBench. For TPC-H, we generate 80 GB input for 8 input tables.

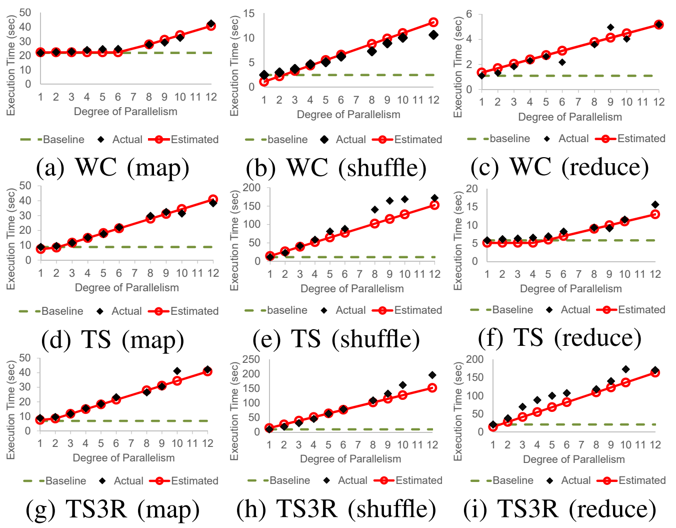
B. BOE Model

We evaluate the effect of the task-level BOE model. We use the best cases of Starfish [16] and MRTuner [31] as the baseline. That is the ground truth execution time when the degree of parallelism is equal to that in the profiling stage. We use the median execution time of tasks as the ground truth in all the evaluations.

1) Single Job: Figure 6 (a), (b), and (c) present the WC evaluation result for the map, shuffle, and reduce stages, respectively. The average accuracy for the execution time estimation is 95.2%, 82.3%, and 85.1% for the BOE model. When the degree of parallelism is 12, the BOE model outperforms the baseline by a factor of 6.6x, 4.3x, and 4.1x for the map, shuffle and reduce stages, respectively. For the map stage, there are enough idle CPU cores when the degree of parallelism is less than 6. When the degree of parallelism is higher than 6, the job becomes CPU-bound due to the saturated computing resource.

Figure 6 (d), (e), and (f) present the TS evaluation result for the map, shuffle, and reduce stages, respectively. The average accuracy for the execution time estimation is 94.0%, 81.5%, and 86.8% for the BOE model. For the case in which the degree of parallelism is 12, the BOE model outperforms

[2]https://github.com/rxin/TPC-H-Hive



(a) WC (map) (b) WC (shuffle) (c) WC (reduce)
(d) TS (map) (e) TS (shuffle) (f) TS (reduce)
(g) TS3R (map) (h) TS3R (shuffle) (i) TS3R (reduce)

Fig. 6. The Task-level Effect of BOE Model for a Single Job

the baseline by a factor of 4.3x, 10.6x, and 1.9x for the map, shuffle and reduce stages, respectively. The bottleneck is steady with varying the degree of parallelism for both the map and the reduce stage. The map stage is disk-bound, and the shuffle stage is network-bound. For the reduce stage, the job is CPU-bound for the low degree of parallelism, and it becomes disk-bound for the high degree of parallelism. Our BOE cost model can identify the change of bottleneck by using the $\max$ operation.

Consequently, for single jobs, the BOE model accurately estimates the execution time with respect to the degree of parallelism, by identifying the bottleneck for each stage.

TABLE II
TASK LEVEL EFFECT FOR PARALLEL JOBS

DAG	Job	s1	s2	s3	s4
WC+TS	WC	99.5%	84.9%	88.6%	70.5%
	TS	99.9%	92.4%	-	-
WC+TS3R	WC	99.9%	92.7%	97.9%	71.7%
	TS3R	99.9%	99.9%	-	-

2) Multiple Jobs: We evaluate the task level BOE model for parallel jobs. The DAG workflow has two parallel jobs. Table II presents the accuracy of the task level model for parallel jobs including WC, TS, and TS3R, running simultaneously.

For WC and TS run in parallel, the average accuracy is 99.7% and 88.7% for states 1 and 2, which consist of parallel jobs. For state 1, the BOE model identified the CPU bottleneck. When the workflow enters state 2, the BOE model identifies bottlenecks for the TS reduce stage, which is network-bound for the shuffle and disk-bound for HDFS write (with 1 replica). For states 3 and 4, we skip detailed evaluation since it is covered by single job models in Section V-B1.

For WC and TS3R run in parallel, the average accuracy is 99.9% and 96.3% for states 1 and 2, which consists of parallel jobs. For state 1, the behavior is the same as the previous DAG (WC+TS). For state 2, the reduce stage of TS is network-bounded due to HDFS write (with 3 replicas). For the shuffle stage, the execution time for TS is reduced by a factor of 2 in comparison with the single job case. This is because the number of parallel tasks to use the bottleneck resource (i.e., network) is reduced by a factor of 2 for the state 2.

Consequently, for parallel jobs, the BOE model can identify the bottleneck resource and its allocation for each job, and hence to estimate the execution time for each task.

C. State-based Approach

We evaluate the effectiveness of the state-based estimation framework for DAG workflow cost estimation. To eliminate the error of task-level models, we use task execution time profiles with the identical degree of parallelism for each stage. For the TPC-H workload, we also count the compilation time for each query in estimation. We run both micro-benchmarks (WC or TS) and query/analytics DAGs (TPC-H or HiBench) in parallel to cover real workloads. Besides the end-to-end execution time for DAG workflows, we present the average accuracy of the estimated execution time for each stage (denoted as Stage Break-downs). This metric provides a breakdown for the estimation and evaluates the accuracy of the state-based approach for each stage.

Overall DAG Results: First, we present the accuracy for DAG execution time. The average accuracy of 51 workflows is 93.50%, 95.00%, 96.38% for median, mean and normal distribution respectively.

The last 7 columns of the third row group in Table III present the result for the state-based approach using analytics DAG workflows. Overall, the minimal accuracy of end-to-end execution time estimation is more than 81.13% for all the workflows by Algorithm 1.

The first 2 row groups and the first 10 columns of the third row group in Table III present the overall DAG estimation accuracy for hybrid HiBench and TPC-H workload. Overall, the estimation accuracy on average of 22 WC+TPC-H workflows is 94.62%, 96.58%, 97.42% for the median, mean and normal distribution, respectively. The prediction accuracy on average of these 22 workflows is 92.94%, 93.67%, and 96.21% for the median, mean and normal distribution, respectively. The result indicates that our state-based approach can handle various workloads from short to long. Some of the queries have many jobs. For example, Q21 has 9 MapReduce jobs, which leads to 18 stages when it is run in parallel with the WC job.

Execution time: Finally, we evaluate the execution time of the state-based approach for each DAG workflow used in the above evaluation. The result indicates that the overhead for computing the cost models is less than 1 second for all the DAG workflows. This means that the cost model is suitable to be used in runtime optimizations such as query re-writing and self-tuning for DAG workflows.

VI. RELATED WORK

MapReduce Cost Models: The cost models for MapReduce are studied since the bottleneck for data parallel computing framework are different compared to traditional database systems. The cost models for single MapReduce jobs are used

TABLE III
ESTIMATION ACCURACY FOR DAGS

	TS-Q1	TS-Q2	TS-Q3	TS-Q4	TS-Q5	TS-Q6	TS-Q7	TS-Q8	TS-Q9	TS-Q10	TS-Q11	TS-Q12	TS-Q13	TS-Q14	TS-Q15	TS-Q16	TS-Q17
Alg1-Mean	0.9876	0.9819	0.9887	0.8796	0.8805	0.9423	0.7976	0.9836	0.8905	0.9721	0.9356	0.9938	0.9634	0.959	0.9699	0.9045	0.9881
Alg1-Mid	0.9791	0.9475	0.9785	0.8589	0.8602	0.8972	0.7897	0.9954	0.9276	0.9958	0.8971	0.8423	0.9886	0.902	0.9781	0.9718	0.976
Alg2-Normal	0.9784	0.9647	0.9825	0.9627	0.9968	0.8912	0.9163	0.9986	0.9293	0.9563	0.8866	0.9202	0.9819	0.8758	0.9975	0.9612	0.9841
	TS-Q18	TS-Q19	TS-Q20	TS-Q21	TS-Q22	WC-Q1	WC-Q10	WC-Q11	WC-Q12	WC-Q13	WC-Q14	WC-Q15	WC-Q16	WC-Q17	WC-Q18	WC-Q19	WC-Q2
Alg1-Mean	0.8618	0.9392	0.9531	0.8968	0.9375	0.9927	0.9922	0.9833	0.9821	0.9837	0.9871	0.9948	0.9766	0.9854	0.9445	0.9889	0.9878
Alg1-Mid	0.8695	0.9306	0.9452	0.8756	0.9654	0.9386	0.9637	0.9129	0.9553	0.9297	0.9522	0.9834	0.9558	0.9936	0.9015	0.9265	0.9608
Alg2-Normal	0.9506	0.9194	0.9959	0.9706	0.9758	0.974	0.9640	0.9432	0.9548	0.9923	0.9899	0.9413	0.9672	0.9963	0.9885	0.9646	0.9651
	WC-Q20	WC-Q21	WC-Q22	WC-Q3	WC-Q4	WC-Q5	WC-Q6	WC-Q7	WC-Q8	WC-Q9	WC-TS	WC-TS2R	WC-TS3R	WC-KM	WC-PR	TS-KM	TS-PR
Alg1-Mean	0.9734	0.9347	0.9538	0.9661	0.9926	0.969	0.9689	0.8464	0.9486	0.8956	0.9994	0.9949	0.9862	0.9687	0.8762	0.9597	0.8113
Alg1-Mid	0.9772	0.9505	0.9549	0.9581	0.9544	0.9699	0.9062	0.8672	0.9651	0.939	0.9495	0.9418	0.9714	0.9331	0.9023	0.9507	0.8463
Alg2-Normal	0.9801	0.9819	0.9281	0.9857	0.9803	0.9831	0.9155	0.9827	0.9643	0.9878	0.9682	0.9723	0.9652	0.9862	0.9725	0.981	0.9839

to tune MapReduce configurations [11], [16], [21], [22], [31], [37], [38]. These works proposed the general cost-based estimation framework for MapReduce. The authors use queuing theory to predict key performance indicators (e.g., task waiting time and blocking probability) of MapReduce jobs in [30]. However, these cost models are for single MapReduce jobs and do not consider the resource allocation variance with respect to the degree of parallelism. Thus these cost models have the limitation in terms of resource estimation for parallel MapReduce jobs, which is the main focus of this paper. An analytical cost model is used as the fundamental building block to optimize resource configuration for SystemML programs [18]. In contrast to our work, this cost model is specifically for SystemML resource configuration and does not consider the general problem for resource contention among MapReduce tasks. Ernest [36] is a performance prediction model that collects as few training points as required by using a statistical technique (i.e., optimal experiment design). Like Starfish and MRTuner, Ernest also focuses on single jobs rather than DAGs with parallel jobs. The machine learning based prediction model is proposed in [32] to estimate job execution time for Spark. However, the identified features do not consider the impact of parallelism on system bottleneck. Thus the model does not fit for the multiple job scenario.

Query Optimizers: Prior to MapReduce, cost models are widely used in query optimizers in relational database systems. The cost estimation for relational queries is widely studied for a parallel database [6]. There do exist interesting works on the resource usage model for parallel queries such as join [13], which take the impact of resource contention into account for the cost estimation. However, the analytical model for MapReduce is different due to different task execution frameworks. Resource Bricolage is proposed for parallel query optimization in a heterogeneous cluster [24]. This approach quantifies the performance differences among machines with various resources by profiling workloads. Our problem differs from theirs as we aim to model the resource usage for parallel MapReduce tasks rather than parallel queries. A MapReduce cost model is proposed in [40] to estimate I/O and CPU costs. Since the model is specially designed for query optimizers, accurate running time is not estimated. The cost model in [39] is designed for multi-query optimization. However, it only models the disk and network I/O costs since these are the bottleneck in its problem.

DAG Workflow: DAG workflow is a natural representation for high level query in data parallel frameworks. Stubby is a transformation-based Optimizer for MapReduce Workflows [26]. It uses the What-if Engine building block of Starfish for the cost estimation [16]. However, the resource statistics are assumed to be the same between the profiling and estimation stages, and hence it does not address the preemptable resource issue for parallel jobs. ParaTimer is a Progress Indicator for MapReduce DAGs [28], which estimates the critical path for parallel jobs of a DAG workflow. However, ParaTimer does not consider resource contention among parallel tasks. The authors experimentally demonstrate the impact of the degree of parallelism on the execution time of DAGs in [33]. However, this work focuses on DAG-level, and it does not address the task-level cost models. The work in [25] estimates the execution time of DAGs in tuple-level for distributed streams. However, this work uses regression algorithms for the prediction, and the accuracy relies on the quality of the sample space.

Distributed and Parallel Computing: There are previous works to estimate the execution time for distributed and parallel computing frameworks. Bandwidth-latency models such as LogP model [9] and the BSP model [7] models are proposed to estimate latency and throughput for parallel computing systems. These models are not suitable for the MapReduce framework because MapReduce does not rely on a messaging-based asynchronous communication system. The work [29] measures the job completion time for a best-case scenario without blocking on network or disk use, by using finer-grained instrumentation to Spark compute thread. They find that the upper bound on the improvement from optimizing disk and network performance is limited. This work is cross-validation of the idea that tasks are pipelined executed using multiple resources, and it focuses on execution analysis rather than the cost models to estimate the task execution time based on data, system and job profiles. Jockey [12] uses a simulation-based approach to predict job completion time for SCOPE [5]. While the prediction framework is similar to that in this paper, it does not take the skewness into account. Task completion time estimation is proposed in [4] for scheduling. However, the prediction is coarse-grained without estimating the accurate running time.

VII. CONCLUSION

In this paper, we proposed the BOE model to predict the allocation of preemptable system resources for task-level execution time estimation. Based on the insights of resource allocation among stages, the state-based iterative approach was proposed for workflow-level execution plan estimation.

Our experimental evaluation showed that the BOE model can automatically identify the bottleneck resource for each stage. We performed comprehensive experiments to show that our new cost model outperforms existing models by a factor of five for task execution time estimation. For the skew-aware state-based approach to estimate the execution time of a DAG workflow, the average prediction error is under 3%.

As the follow-up research, we will study the impact of skewness in cost estimation and apply our cost models in automatic tuning for DAG workflows.

Acknowledgment

The authors would like to thank Xiaohua Cai from RUC (now at Tencent) for setting up the cluster and conducting preliminary benchmarks.

References

[1] Apache tez. https://tez.apache.org/.

[2] M. Assefi, E. Behravesh, G. Liu, and A. P. Tafti. Big data machine learning using apache spark mllib. In *Big Data (Big Data), 2017 IEEE International Conference on*, pages 3492–3498. IEEE, 2017.

[3] M. Boehm, S. Tatikonda, B. Reinwald, P. Sen, Y. Tian, D. R. Burdick, and S. Vaithyanathan. Hybrid parallelization strategies for large-scale machine learning in SystemML. *VLDB*, 7(7):553–564, 2014.

[4] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, and L. Zhou. Apollo: scalable and coordinated scheduling for cloud-scale computing. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pages 285–300, 2014.

[5] R. Chaiken, B. Jenkins, P.-Å. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. Scope: easy and efficient parallel processing of massive data sets. *Proceedings of the VLDB Endowment*, 1(2):1265–1276, 2008.

[6] S. Chaudhuri. An overview of query optimization in relational systems. In *PODS*, pages 34–43, 1998.

[7] T. Cheatham, A. Fahmy, D. Stefanescu, and L. Valiant. Bulk synchronous parallel computing paradigm for transportable software. In *TEPDS*, pages 61–76. 1996.

[8] B. Clifton. Advanced web metrics with google analytics. 2012.

[9] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. Von Eicken. *LogP: Towards a realistic model of parallel computation*, volume 28. ACM, 1993.

[10] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *CACM*, 51(1):107–113, 2008.

[11] M. Ead, H. Herodotou, A. Aboulnaga, and S. Babu. Pstorm: Profile storage and matching for feedback-based tuning of mapreduce jobs. pages 1–12, 2014.

[12] A. D. Ferguson, P. Bodik, S. Kandula, E. Boutin, and R. Fonseca. Jockey: guaranteed job latency in data parallel clusters. In *Proceedings of the 7th ACM european conference on Computer Systems*, pages 99–112. ACM, 2012.

[13] S. Ganguly, W. Hasan, and R. Krishnamurthy. Query optimization for parallel execution. In *SIGMOD*, pages 9–18, 1992.

[14] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *NDSI*, pages 323–336, 2011.

[15] A. Ghoting, R. Krishnamurthy, E. Pednault, B. Reinwald, V. Sindhwani, S. Tatikonda, Y. Tian, and S. Vaithyanathan. SystemML: Declarative machine learning on MapReduce. In *ICDE*, pages 231–242, 2011.

[16] H. Herodotou and S. Babu. Profiling, what-if analysis, and cost-based optimization of mapreduce programs. *VLDB*, 4(11):1111–1122, 2011.

[17] H. Herodotou, F. Dong, and S. Babu. No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics. In *SoCC*, page 18, 2011.

[18] B. Huang, M. Boehm, Y. Tian, B. Reinwald, S. Tatikonda, and F. R. Reiss. Resource elasticity for large-scale machine learning. In *SIGMOD*, pages 137–152, 2015.

[19] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang. The hibench benchmark suite: Characterization of the mapreduce-based data analysis. In *ICDEW*, pages 41–51, 2010.

[20] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 59–72, 2007.

[21] M. Khan, Y. Jin, M. Li, Y. Xiang, and C. Jiang. Hadoop performance modeling for job estimation and resource provisioning. *IEEE Transactions on Parallel and Distributed Systems*, 27(2):441–454, 2016.

[22] N. Khoussainova, M. Balazinska, and D. Suciu. Perfxplain: debugging mapreduce job performance. *VLDB*, 5(7):598–609, 2012.

[23] B. Li, E. Mazur, Y. Diao, A. McGregor, and P. Shenoy. Scalla: a platform for scalable one-pass analytics using mapreduce. *TODS*, 37(4):27, 2012.

[24] J. Li, J. Naughton, and R. V. Nehme. Resource bricolage for parallel database systems. *VLDB*, 8(1):25–36, 2014.

[25] T. Li, J. Tang, and J. Xu. Performance modeling and predictive scheduling for distributed stream data processing. *IEEE Transactions on Big Data*, 2(4):353–364, 2016.

[26] H. Lim, H. Herodotou, and S. Babu. Stubby: A transformation-based optimizer for mapreduce workflows. *VLDB*, 5(11):1196–1207, 2012.

[27] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, et al. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17(1):1235–1241, 2016.

[28] K. Morton, M. Balazinska, and D. Grossman. ParaTimer: a Progress Indicator for MapReduce DAGs. In *SIGMOD*, pages 507–518, 2010.

[29] K. Ousterhout, R. Rasti, S. Ratnasamy, S. Shenker, B.-G. Chun, and V. ICSI. Making sense of performance in data analytics frameworks. In *NSDI*, volume 15, pages 293–307, 2015.

[30] C. Shen, W. Tong, J.-N. Hwang, and Q. Gao. Performance modeling of big data applications in the cloud centers. *The Journal of Supercomputing*, 73(5):2258–2283, 2017.

[31] J. Shi, J. Zou, J. Lu, Z. Cao, S. Li, and C. Wang. MRTuner: A toolkit to enable holistic optimization for mapreduce jobs. *VLDB*, 7(13):1319–1330, 2014.

[32] R. Singhal and P. Singh. Performance assurance model for applications on spark platform. In *Technology Conference on Performance Evaluation and Benchmarking*, pages 131–146. Springer, 2017.

[33] M. Taufer and A. L. Rosenberg. Scheduling dag-based workflows on single cloud instances: High-performance and cost effectiveness with a static scheduler. *The International Journal of High Performance Computing Applications*, 31(1):19–31, 2017.

[34] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy. Hive-a petabyte scale data warehouse using hadoop. In *ICDE*, pages 996–1005, 2010.

[35] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler. Apache Hadoop YARN: Yet another resource negotiator. In *SoCC*, pages 5:1–5:16, 2013.

[36] S. Venkataraman, Z. Yang, M. J. Franklin, B. Recht, and I. Stoica. Ernest: Efficient performance prediction for large-scale advanced analytics. In *NSDI*, pages 363–378, 2016.

[37] A. Verma, L. Cherkasova, and R. H. Campbell. Aria: automatic resource inference and allocation for mapreduce environments. In *Proceedings of the 8th ACM international conference on Autonomic computing*, pages 235–244. ACM, 2011.

[38] A. Verma, L. Cherkasova, and R. H. Campbell. Resource provisioning framework for mapreduce jobs with performance goals. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 165–186. Springer, 2011.

[39] G. Wang and C.-Y. Chan. Multi-query optimization in mapreduce framework. *Proceedings of the VLDB Endowment*, 7(3):145–156, 2013.

[40] S. Wu, F. Li, S. Mehrotra, and B. C. Ooi. Query optimization for massively parallel data processing. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 12. ACM, 2011.

[41] 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 *NSDI*, 2012.