Distributed and Parallel Databases (2023) 41:299–329
https://doi.org/10.1007/s10619-023-07425-1

Performance models of data parallel DAG workflows for large scale data analytics

Juwei Shi[1] · **Jiaheng Lu**[2]

Accepted: 19 April 2023 / Published online: 23 May 2023

Abstract

Directed Acyclic Graph (DAG) workflows are widely used for large-scale data analytics in cluster-based distributed computing systems. The performance model for a DAG on data-parallel frameworks (e.g., MapReduce) is a research challenge 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. Furthermore, to handle the skewness of various jobs, we refine the model with the order statistics theory to improve estimation accuracy. 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. For the refined skew-aware model, the average prediction error is under 3% when estimating the execution time of 51 hybrid analytics (HiBench) and query (TPC-H) DAG workflows.

Keywords Cost models · Data parallel processing · Query optimization

✉ Jiaheng Lu
jiaheng.lu@helsinki.fi

Juwei Shi
juwei.shi@gmail.com

[1] Microsoft STCA, Beijing, China

[2] University of Helsinki, Helsinki, Finland

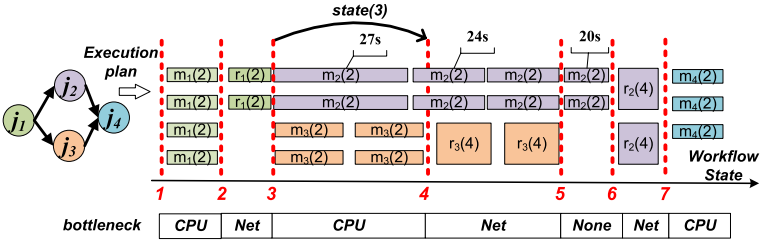


Fig. 1 The task execution plan of a DAG with four jobs. $m_i(k)$ denotes that a map task of job i requires k task slots (i.e., maximum number of Map or Reduce tasks can run simultaneously) for execution

1 Introduction

There is a trend towards unifying Big Data and AI on cluster-based distributed computing systems [1]. The data parallel paradigm makes it easy to automatically distribute the computation and manage task failures at a large scale. The data parallel analytics jobs are often represented by Directed Acyclic Graph (DAG) workflows [, , , , , 2–7]. 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 [8] jobs to be run in parallel, (2) the Spark [7] program and machine learning workloads are transformed to a DAG workflow for execution [3], and (3) the Tez [2] framework allows for a complex DAG of tasks for processing data.

The performance of DAGs are widely studied in the literature [, , , 5, 9–11]. 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 [, 9, 12], (2) capacity planning on the cloud [13], and (3) progress estimation [14]. However, existing cost models are limited to single jobs [, 9, 12], and it is still a challenge to build cost models for a DAG workflow of parallel jobs on cluster-based distributed computing such as MapReduce, Spark, and Tez.

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) [, 9, 12]. 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 [15] in MapReduce to illustrate the above challenge. As shown in Fig. 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 to 24 s 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 to 20 s 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 [9] and MRTuner [12] 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). Note that the cost model we proposed in this paper is a general model that can be extended to other data-parallel systems such as Spark and Tez. This is because the concept of the execution plan (e.g., stages, tasks, shuffle) is similar for these systems. MapReduce is an example to illustrate the implementation of the cost model. 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., 27 s, 24 s and 20 s for task m_2 among the stages in Fig. 1).

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

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 Fig. 1) is estimated by this state-based iterative approach.

To further improve the accuracy of the cost model, the skewness of tasks (i.e., the variation of task execution time caused by non-linear I/O complexity [16]) is addressed at both task-level and workflow-level. For the task-level model, the relation between the selectivity factor distribution and the execution time distribution is derived using the order statistic theory. Then, the impact of task execution time variation is estimated for the DAG workflow execution plan. Finally, the expected gain of our approach of handling the skewness is derived.

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.
- To further improve the accuracy of the model, we refine the granularity for both task-level and DAG level models to handle skewness. The time distribution of tasks in sub-stages is derived from the selectivity factor using the order statistic theory. Consequently, the expected gain of our approach to handle the skewness is obtained.
- 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. Furthermore, the overhead for calculating cost models is less than 1 s for all the DAG workflows. Both accuracy and overhead results indicate that our model is able to serve applications such as auto-tuning, capacity planning and progress estimation.

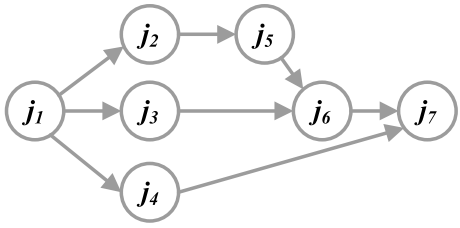


Fig. 2 Example of a DAG Workflow

The rest of the paper is organized as follows. We provide the background and problem description in Sect. 2. We conduct benchmarks to understand the system behavior for parallel jobs in Sect. 3. We propose the task-level execution model in Sect. 4. In Sect. 5, we present the holistic cost model for a DAG workflow. We improve the cost model in regard to skewness in Sect. 6. The evaluation results are presented in Sect. 7. We present the related work in Sect. 8 and conclude in Sect. 9.

2 Background and problems

In this section, we introduce the background, and formalize the cost estimation problem for a DAG workflow. The basic concept of Map-Reduce and execution stages can be found in [8]

2.1 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 [17] 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 [18]).

2.2 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 [19]; (2) SystemML [, 20, 21] compiles DML (Declarative Machine learning Language) to a DAG of hybrid MapReduce jobs and control programs; (3) Spark [7] 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).

2.3 Problem definition

In this paper, we focus on cost estimation for a DAG workflow in which jobs run in parallel. We formulate the problem of cost estimation for a DAG workflow as follows:

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

Details for G, D, P and C will be presented in Table 1. The main challenge to estimate the cost for a DAG workflow is the change of allocation of preemptable system resources among computational stages. However, the previous cost models for MapReduce [, 9, 12] do not consider the impact of preemptable resources, even for a single job. This is the main limitation for these models to be used for a DAG workflow with parallel jobs.

3 Benchmark analysis

We conduct a set of benchmarks to understand the system behavior for parallel jobs. The experimental setup about hardware and software is described in Sect. 7.1.1.

3.1 Task-level system behaviors

First, we study the impact of the degree of task parallelism on resource utilization for CPU, disk, and network. We use workloads including Word Count (WC), TeraSort (TS) and TeraSort with three replicas on HDFS (TS3R) since they are representative for CPU-bound, disk-bound, and network-bound workloads, respectively.

Figure 3 shows the median task execution time of each stage for WC, TS, and TS3R with varying the degree of parallelism. Note that only task execution time t is shown in the figure. The stage execution time should be $t \cdot \frac{N}{\Delta}$ where N is the number of tasks and Δ is the degree of parallelism. It means that the stage execution time may be reduced even if the task execution time is increased as Δ increases. For example, we assume the map stage has 2 tasks on a node. When the degree of parallelism is 1 and task execution time is 2, the map stage execution time should be 4. When the degree of parallelism is 2 and the task execution time is increased to 3 due to resource allocation changes, the map stage execution time is reduced to 3 due to the increased parallelism capacity.

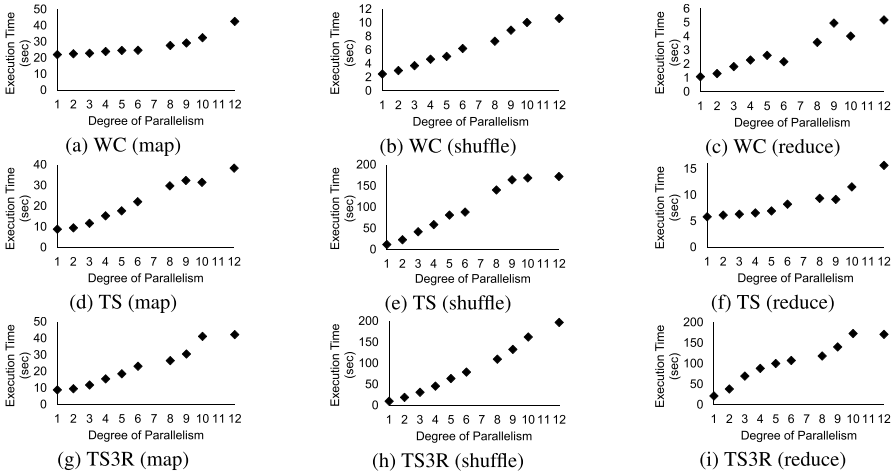


Fig. 3 The impact of degree of parallelism for a single job

Map As shown in Fig. 3a, the task execution time does not increase when the degree of parallelism is less than 6. This is because there are idle CPU cores as the number of simultaneously running tasks increases, and neither disk nor network is a bottleneck. When the degree of parallelism is higher than 6, we observe slow-down for map tasks since the CPU becomes a bottleneck. It means that the change of the degree of parallelism of a job may lead to the change of the system bottleneck, which could further affect the task execution time. In Fig. 3d and g, for the map stage of both TS and TS3R, the task execution time is proportional to the degree of parallelism. This is because the map stages of both jobs are disk-bound under all the parallelism configuration.

Note that multiple operations such as compression, serialization/de-serialization, GC, etc. are CPU intensive [22] and here we select the TeraSort configuration without compression to cover the disk-bound case in this experiment.

Shuffle For WC, TS, and TS3R, as shown in Fig. 3b, e and h, we observe that the median task execution time of the shuffle stage is proportional to the degree of parallelism. This is because the shuffle stage is network-bound for all the workloads. The increased degree of parallelism leads to contention in the saturated network resource which is a bottleneck for the shuffle stage.

Reduce As shown in Fig. 3c, f and i, for WC and TS3R, the median task time of the reduce stage is also proportional to the degree of parallelism. The reduce stage is network-bound because of HDFS write. For TS with only 1 output replica in HDFS, we observe that the task execution time is steady when the degree of parallelism is less than 6, which is similar to the map stage of WC as shown in Fig. 3a. It is also because the reduce stage is CPU-bound for TS, and there are idle CPU cores for use when the degree of parallelism is less than 6.

Implications for task-level cost model Through the set of single job experiments, we find the following insights to develop the cost model: (1) The task execution time during a DAG stage is determined by the bottleneck system resource

(e.g., CPU, disk or network) for that stage. (2) The bottleneck may be changed with varying the degree of parallelism. Consequently, given the degree of parallelism, the cost model should be able to identify the bottleneck resources and the allocation among tasks to accurately estimate the task-level execution time.

3.2 Workflow-level system behavior

We study the resource usage behavior for multiple jobs running simultaneously. In this set of experiments, we configure a fixed parallelism 8 and evaluate resource utilization behavior for various stages of jobs that run in parallel.

We first evaluate the execution plan for parallel WC and TS jobs. There are two overlapped stages: (1) WC map and TS map, and (2) WC map and TS reduce. We find that: (1) For the first overlapped stage (i.e., WC map and TS map), WC's median map task time is increased from 22.7 to 27.7 s in comparison to the single job benchmark result. This is because of the saturated CPU resource under the parallelism configuration. (2) For the second overlapped stage (i.e., WC map and TS reduce), WC's map task time is reduced to 24.3 s. This is because the network-bounded TS releases CPU to WC. For TS, its reduce task time (including both shuffle and reduce) is reduced by a factor of two. This is because only 50% of simultaneous tasks now compete for the bottleneck resource (i.e., network for the shuffle). (3) For each computational stage of which boundary is start/end of map/reduce stages, the resource allocation for tasks is steady. This is because the degree of parallelism for each job is fixed within a stage according to the task scheduling policy such as DRF [18].

Next, we evaluate the execution plan for parallel WC and TS3R jobs. We observe that the first stage is bounded by the CPU. The second stage is bounded by the network I/O. TS is 1.9× faster than its single job case for the reduce task time. This is because of the reduction in the degree of parallelism for TS reduce tasks that compete for the network bandwidth. Furthermore, we also observe that the resource allocation for tasks is steady during a stage. Figure 4 illustrates the correlation between resource utilization (on a slave node) and 4 workflow stages. Each stage is visually correlated with preemptable resources (CPU, network and disk) through the time-line.

Implications for workflow-level cost model Through this set of multiple job experiments, we find that the resource allocation for tasks from each job is steady during a stage of which boundary is start/end of map/reduce stages. The share rate of the bottleneck resource among jobs is determined by the resource usage characteristic during a stage. Therefore, this property could be used to iteratively divide the execution plan to multiple stages for holistic execution plan estimation.

4 Task-level model

In this section, we present a cost model, Bottleneck Oriented Estimation (BOE), for task-level execution time estimation. BOE models the system behavior presented in Sect. 3.1.

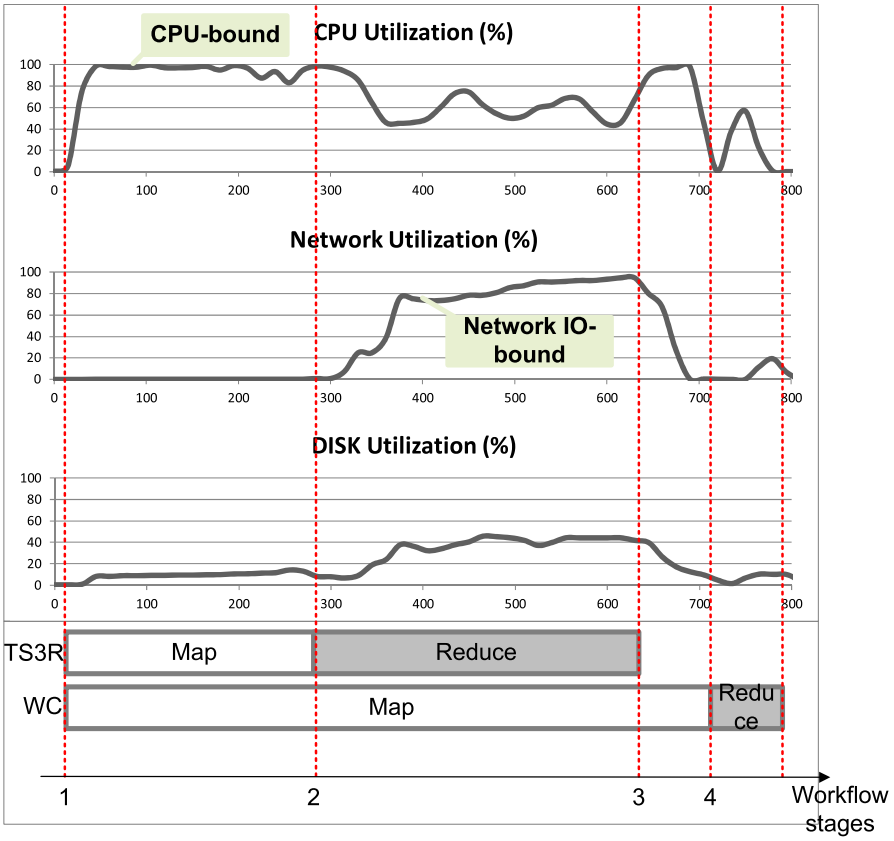


Fig. 4 The resource usage behavior of workflow stages

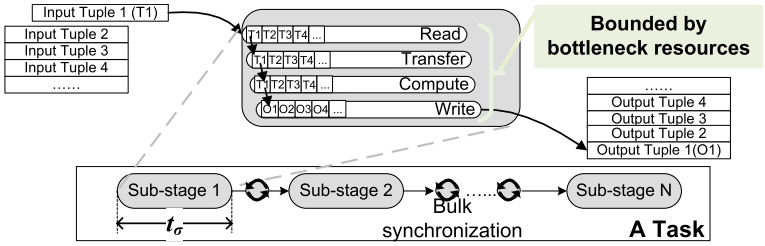


Fig. 5 The task execution model

4.1 BOE model

4.1.1 Task execution model

We first model the fundamental behavior for task execution on data parallel computing frameworks. As shown in Fig. 5, 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.

4.1.2 Resource usage model

Given the above task execution model, we use the resource usage model in [23] 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. It means that we assume that memory will not impact task-level execution time. Different memory configurations will lead to task success or failure due to out-of-memory issues. The impact of memory on job-level execution time is modeled by the degree of parallelism parameter. 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.

4.1.3 Bottleneck oriented estimation

Given the task execution model in Fig. 5, we estimate the execution time t_σ for a sub-stage of a task as follows,

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

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 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 $t_{read} = t_{transfer} = t_{compute} = t_{write}$. That is, $t_\sigma = \max\{t_{read}, t_{transfer}, t_{compute}, t_{write}\}$.

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

Table 1 Parameters for cost model

Symbol	Description
Job profiles	
t_{load}	Time to create, schedule and destroy a task
θ_{map}	CPU throughput for a map task per core
θ_{reduce}	CPU throughput for a reduce task per core
θ_{sort}	CPU throughput for the sort stage of a reduce task per core
α_m	Selectivity for materialized map output
α_r	Selectivity for materialized reduce output
α_s	The fraction of reduce input for the second pass of sort and merge
Data parameters	
B	Size of the input split for a task
System resource profiles	
θ_{dr}	Throughput of local disk write for a node
θ_{dw}	Throughput of local disk read for a node
θ_{dfsw}	Throughput of HDFS disk read for a node
θ_{dfsw}	Throughput of HDFS disk read for a node
θ_{net}	Network throughput for a node
N_{cores}	Number of CPU cores for a node
γ	Fraction of data read from OS buffer for a Map or Reduce task
Job Runtime Parameters	
Δ	Degree of Parallelism on a node
N_m	Number of map tasks
N_r	Number of reduce tasks

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

$$t_\sigma = \max\left\{ \frac{B}{\mu_{read}(\Delta)\cdot\theta_{read}}, \frac{B}{\mu_{transfer}(\Delta)\cdot\theta_{transfer}}, \frac{B}{\mu_{compute}(\Delta)\cdot\theta_{compute}}, \frac{s\cdot B}{\mu_{write}(\Delta)\cdot\theta_{write}} \right\} \quad (2)$$

where B is the size of input for the task.

4.2 Details of task estimation

To implement the BOE Model for MapReduce, we select the parameters of profiling-and-estimation framework in [, 9, 12] to build the cost model. The profiling-and-estimation framework collects the historical profiles in terms of job, data and system resources as the input for the cost model. In particular, the parameters are defined in Table 1. A task represents a map/reduce task unless otherwise specified. θ_{map} and θ_{reduce} are obtained when a CPU core is fully allocated to the task.

4.2.1 Single job case

For single jobs, the cost model of a map task is presented in Proposition 1.

Proposition 1 *Given the job profile* $(t_{load}, \theta_{map}, \alpha_m)$, *input split* B, *cluster resources* $(\theta_{dfsr}, \theta_{dw})$, *the degree of parallel* Δ, *as defined in Table 1, the execution time of a map task for single jobs is*

$$t_{map} = t_{load} + \max\left\{\frac{B}{\mu_{dfsr}(\Delta)\cdot\theta_{dfsr}}, \frac{B}{\mu_{cpu}(\Delta)\cdot\theta_{map}}\right\} + \frac{\alpha_m \cdot B}{\mu_{dw}(\Delta)\cdot\theta_{dw}} \tag{3}$$

where $\mu_{dfsr}(\Delta) = \mu_{dw}(\Delta) = \frac{1}{\Delta}$, *and* $\mu_{cpu} = \min\left\{\frac{N_{cores}}{\Delta}, 1\right\}$.

Analysis For the first sub-stage to read data from HDFS and compute in map function, the execution time is estimated based on the bottleneck resources (HDFS read or CPU). If the map task is HDFS-bound and there is only one job using the system resource, the HDFS read and disk write throughput allocated to each task is $\frac{\theta_{hdfs}}{\Delta}$ and $\frac{\theta_{dw}}{\Delta}$, respectively. Thus the execution time for a map task is $t_{load} + \frac{B\cdot\Delta}{\theta_{dfsr}} + \frac{\alpha_m\cdot B\cdot\Delta}{\theta_{dw}}$. Otherwise, if the map task is CPU-bound, the execution time for a map task is $t_{load} + \frac{B\cdot\max\{\Delta/N_{cores},1\}}{\theta_{map}} + \frac{\alpha_m\cdot B\cdot\Delta}{\theta_{dw}}$. It is noteworthy that the max operation automatically estimates the execution time based on the bottleneck resource during the execution. For the second sub-stage to write map output on local disks, there is only one resource (i.e., disk write) that is fully utilized. When map output spills to disk in case of insufficient memory, the selectivity factor α_m models the accurate I/O needed for this stage.

A reduce task has two sub-stages including shuffle and reduce. The execution time for a shuffle sub-stage is estimated as follows.

Proposition 2 *Given the job profile* (t_{load}, s_m), *input split* B, *cluster resources* $(\theta_{dr}, \theta_{dw})$, *job configuration* (Δ, N_m, N_r), *as defined in Table 1, the execution time for the shuffle sub-stage of a reduce task for single jobs is*

$$t_{shuffle} = t_{load} + \max\left\{\frac{\alpha_m \cdot B \cdot N_m}{\mu_{dw}(\Delta)\cdot\theta_{dw}\cdot N_r}, \frac{\alpha_m \cdot B \cdot N_m}{\mu_{net}(\Delta)\cdot\theta_{net}\cdot N_r}\right\} \tag{4}$$

where $\mu_{dw}(\Delta) = \mu_{net}(\Delta) = \frac{1}{\Delta}$.

Analysis The execution time is estimated based on the bottleneck (disk I/O or network transfer) resources. The input size of a reduce task is $\frac{B\cdot N_m\cdot\alpha_m}{N_r}$. If disk I/O is a bottleneck, the allocated disk resource for each task is $\frac{\theta_{dw}}{\Delta}$, and the execution time of the shuffle stage is $t_{load} + \frac{\alpha_m\cdot B\cdot N_m\cdot\Delta}{T_{dw}\cdot N_r}$. Otherwise, if the network is a bottleneck, the execution time of the shuffle stage is $t_{load} + \frac{\alpha_m\cdot B\cdot N_m\cdot\Delta}{\theta_{net}\cdot N_r}$. Here, either disk or network

throughput for each task is allocated as $\frac{1}{\Delta}$ of the total throughput if it is a bottleneck. The max operation automatically estimates the execution time based on the bottleneck resource of the shuffle stage.

The execution time for a reduce sub-stage is estimated as follows.

Proposition 3 *Given the job profile* $(\theta_{sort}, \theta_{reduce}, s_m, s_r)$, *input split* B, *cluster resources* $(\theta_{dr}, \theta_{dfsw})$, *job configuration* (Δ, N_m, N_r), *as defined in Table 1, the execution time for the reduce stage of a reduce task for single jobs is*

$$\begin{aligned} t_{reduce} = &\max\left\{\frac{\alpha_s \cdot \alpha_m \cdot B \cdot N_m}{\mu_{dw}(\Delta) \cdot \theta_{dw} \cdot N_r}, \frac{\alpha_s \cdot \alpha_m \cdot B \cdot N_m}{\mu_{cpu}(\Delta) \cdot \theta_{sort} \cdot N_r}\right\} \\ &+ \max\left\{\frac{\alpha_r \cdot B \cdot N_m}{\mu_{dfsw}(\Delta) \cdot \theta_{dfsw} \cdot N_r}, \frac{\alpha_m \cdot B \cdot N_m}{\mu_{cpu}(\Delta) \cdot \theta_{reduce} \cdot N_r}\right\} \end{aligned} \tag{5}$$

where $\mu_{dfsw}(\Delta) = \mu_{dw}(\Delta) = \frac{1}{\Delta}$, *and* $\mu_{cpu} = \min\{\frac{N_{cores}}{\Delta}, 1\}$.

Analysis For the sort sub-stage, the execution time is estimated based on the bottleneck resources of sort compute or local disk write. α_s is the fraction of reduce input data to have the second pass of sort & merge. Thus the input size of the sort sub-stage is $\frac{\alpha_s \cdot \alpha_m \cdot B \cdot N_m}{N_r}$. For the reduce sub-stage, the execution time is estimated based on the bottleneck of HDFS writes or reduce function computation. If HDFS write is a bottleneck, the execution time of the reduce stage of the task is $\frac{\alpha_r \cdot B \cdot N_m \cdot \Delta}{\theta_{dfsw} \cdot N_r}$. Otherwise, if CPU time for the reduce function is a bottleneck, the execution time of the reduce stage is $\frac{\alpha_m \cdot B \cdot N_m \cdot \max\{\Delta / N_{cores}, 1\}}{\theta_{reduce} \cdot N_r}$. The HDFS write throughput for each task is allocated as $\frac{1}{\Delta}$ of the total throughput if it is a bottleneck.

In summary, Eqs. 3, 4, and 5 are materializations of the BOE model in Eq. 2 for MapReduce tasks.

4.2.2 Multiple job case

We extend the single job cost model to handle the parallel job case. For N jobs run in parallel, we denote the utilization of the bottleneck resource $\mathbb{X}$ for a task of job i as $\mu_{\mathbb{X}_i}$, and we have $\sum_{i=1}^{N} \mu_{\mathbb{X}_i} \cdot \Delta_i = 1$. Based on the *uniformity assumption* in Sect. 4.1.2, $\mu_{\mathbb{X}_i}$ represents the effective time for which the bottleneck resource is utilized for a task of the running job i. For system resources such as disk and network, $\mu_{\mathbb{X}_i}$ is determined by the scheduling policy of the underlying driver as well as the workload characteristic of the job (i.e., the proportion of computation and I/O) [24].

Given the estimation for the resource utilization $\mu_{\mathbb{X}_i}$, we extend the task time estimation model for single jobs to estimate the resource utilization of $\theta_{\mathbb{X}}$ for each task.

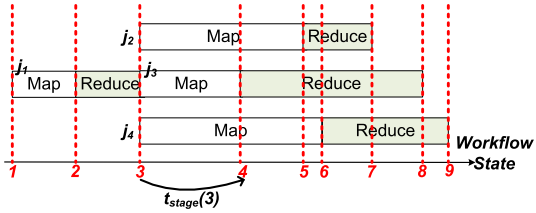


Fig. 6 Workflow state transition

The bottleneck could also be identified by using the max operator in the BOE model. For example, as shown in the Proposition 1, we have $\theta_{dfsr_i} = \mu_{dfsr_i} \cdot \theta_{dfsr}$ and $\theta_{dw_i} = \mu_{dw_i} \cdot \theta_{dw}$. If the disk is the bottleneck for job i, $\frac{B}{\mu_{dfsr_i} \cdot \theta_{dfsr}}$ will be larger than $\frac{B}{\mu_{cpu_i} \cdot \theta_{map_i}}$. Otherwise, the job i is CPU-bound. Note that the bottleneck of a DAG workflow may be different in various stages, and our model can automatically identify the bottleneck for each stage.

5 Workflow level model

In this section, we present the workflow level model to estimate the holistic execution plan of a DAG workflow. This model makes use of the observation on resource usage property in Sect. 3.2.

5.1 The state-based approach

The result in Sect. 3.2 indicates that 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.

5.1.1 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 Fig. 6, 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

$t_{dag} \leftarrow 0$
$s \leftarrow 0$
while G is not empty **do**
  Add new jobs to job queue $\mathcal{Q}$ from G
  job_end_flag=0
  while $job\_end\_flag = 0$ **do**
    Estimate Δ_i for each job $i \in \mathcal{Q}$ ▷ Section 5.2.1
    for each job $i \in \mathcal{Q}$ **do**
      Estimate $t_{task}(i, s)$ using BOE model ▷ Section 4.2
      Estimate the stage time $t_{stage}(i, s)$ ▷ Section 5.2.2
    end for
    $t_{stage}(s) = t_{stage}(k, s) = \min_{i=1}^{N_s}\{t_{stage}(i, s)\}$ ▷ stage of job k finishes
    for each job $i \in \mathcal{Q}$ **do**
      update the progress for job i ▷ Section 5.2.3
    end for
    if job k in reduce stage **then**
      job_end_flag=1
      $\mathcal{Q}.remove(k)$
    else
      update job k to the reduce stage
    end if
    $t_{dag} = t_{dag} + t_{stage}(s)$
    $s = s + 1$
  end while
end while
Return t_{dag}

5.1.2 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 Fig. 6, 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.

5.2 Details of DAG workflow estimation

5.2.1 Degree of parallelism for jobs

As shown in Fig. 6, the degree of parallelism Δ_i for each running job i is fixed during a stage. We assume that tasks in the job queue are scheduled using multi-dimensional fairness DRF [18]. For each job i in the job queue, the scheduler maximizes its degree of parallelism Δ_i while minimizing its allocation of dominate resources. For a stage in DAG, we assume there are N running jobs. Each pair of jobs i and j should satisfy the following condition.

$$\max\left(\frac{\mathbb{C}_i \cdot \Delta_i}{N_{cores}}, \frac{\mathbb{M}_i \cdot \Delta_i}{B_{mem}}\right) = \max\left(\frac{\mathbb{C}_j \cdot \Delta_j}{N_{cores}}, \frac{\mathbb{M}_j \cdot \Delta_j}{B_{mem}}\right) \tag{6}$$

where $\mathbb{C}_i$ and $\mathbb{M}_i$ are the required CPU and memory for tasks of job i, and N_{cores} and B_{mem} are the quanta for CPU cores and memory on the cluster. Based the resource capacity, we have $\sum_{i=1}^{N} \mathbb{C}_i \cdot \Delta_i \leq N_{cores}$ and $\sum_{i=1}^{N} \mathbb{M}_i \cdot \Delta_i \leq B_{mem}$. Therefore, Eq. 6 can be solved to obtain the degree of parallelism for each job.

Note that other schedulers such as the max-min fairness scheduler[2] can be supported by adjusting the resource allocation model in Eq. 6.

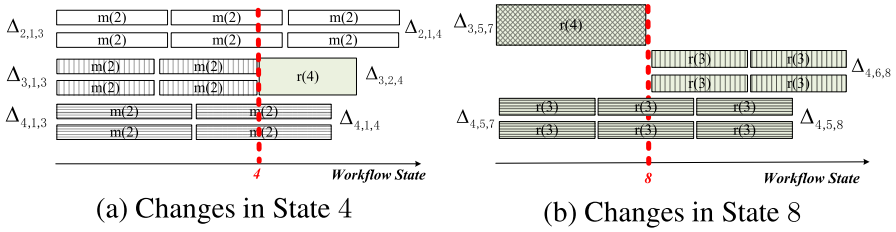
5.2.2 Stage duration estimation

Given the estimated degree of parallelism Δ_i for each job, we use the BOE model implementation in Sect. 4.2 to estimate the task execution time for each job. Note that the degree of parallelism for a job may change from one stage to the next stage. We propose the *task tracks* to keep track of the progress of tasks that have the same progress (i.e., start at the same time). We denote $\Delta_{i,\tau,s}$ as the number of tasks run in parallel for the τth task track of job i during the sth stage, and there are totally T_s task tracks for job i during the sth stage. Consequently, the degree of parallelism for job i during the sth stage is $\Delta_{i,s} = \sum_{\tau=1}^{T_s} \Delta_{i,\tau,s}$.

When we iterate the estimation from one stage to the next stage, a new task track may be created due to the increased degree of parallelism for that job. For the task track τ of job i, $\Delta_{i,\tau,s}$ is monotonically decreasing as s increases during the processing of a job. For a new stage, all the increased degree of parallelism should be assigned to the new task track so that all the tasks in the same task track have the same progress.

Figure 7 presents an example of the task track assignment for stage 4 and 8 of the DAG workflow in Fig. 2. As shown in Fig. 7a, during the stage 3, the task track assignment for job 2, 3 and 4 is the same (i.e., $\Delta_{2,1,3} = \Delta_{3,1,3} = \Delta_{4,1,3} = 2$). Once job 3 finishes its map stage and the workflow enters the stage 4, the task

[2] https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/FairScheduler.html.



(a) Changes in State 4 (b) Changes in State 8

Fig. 7 Example of Task Tracks Changes in States of DAG. The label in a task denotes the task type and quota of required memory. For example, $m(2)$ represents a map task requiring 2 GB memory for its container. We assume the total memory available for containers is 12 GB, and memory is the dominant resource for all the jobs

tracks of job 2 and job 4 remains the same (i.e., $\Delta_{2,1,4} = \Delta_{4,1,4} = 2$). Since the reduce task of job 3 requires more memory than its map task, the task track for job 3 is reduced to $\Delta_{3,2,4} = 1$.

As shown in Fig. 7b, once the DAG workflow is processed from the state 7 to state 8, there are only reduce tasks for job 4 on the cluster. The new task track $\Delta_{4,6,8}$ is assigned to job 4 due to the newly released containers from job 3. Thus the total task tracks for job 4 during the state 8 is $\Delta_{4,8} = \Delta_{4,5,8} + \Delta_{4,6,8} = 4$.

Given the task execution time $t_{task}(i, s)$ for each job i during the state s using the BOE model, and the above estimation for the degree of parallelism $\Delta_{i,\tau,s}$ for task tracks, we estimate the duration of a stage as follows.

Proposition 4 *Given the state s, the remaining execution time of job i for that stage is*

$$t_{stage}(i, s) = t_{task}(i, s) \cdot \left(1 - \min_{\tau=1}^{T_{s-1}} p(i, \tau, s-1) + \lceil \frac{N_{rm}(i, s-1)}{\Delta(i, s)} \rceil\right) \tag{7}$$

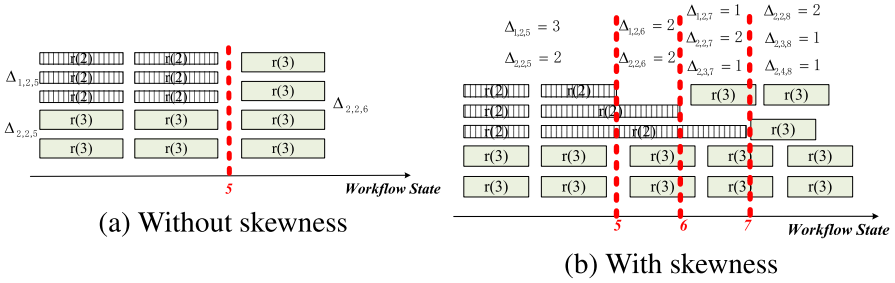
where $t(i, s)$ is the estimated task execution time using the BOE model, and $p(i, \tau, s)$ is the progress of the task track τ for job i at the end of the state s, and N_{rm} is the number of pending tasks for job i at the end of the state s.

The above stage estimation takes the progress of each task track into account. The duration of a stage is determined by the slowest task track.

5.2.3 Progress estimation

Based on the stage duration time, we update the progress for each running job from one stage to the next stage. For each task track τ of job i, the progress of jobs at the end of the state s is $p(i, \tau, s) = \frac{t_{stage}(s)}{t_{stage}(i,s)}$.

Furthermore, for job i, the number of pending tasks at the end of the state s is



(a) Without skewness

(b) With skewness

Fig. 8 The impact of skewness on task tracks in DAG

$$N_{rm}(i,s) = N_{rm}(i,s-1) - \sum_{\tau=1}^{T_s} \Delta_{i,\tau,s} \cdot \frac{t_{stage}(s)}{t_{task}(i,s)} \tag{8}$$

Note that we keep track of both the progress of task tracks and the number of pending tasks during iterations for the DAG workflow estimation.

5.2.4 Analysis

For a DAG workflow with n jobs, there is at most $2n$ sub-stages. For each stage, the time complexity to figure out the degree of parallelism is $O(1)$ based on the third constraint of the optimization problem in Sect. 5.2.1. The worse case time to estimate the execution time of a stage is $O(n)$ while all the jobs run in parallel. The time to update the progress in a stage is constant. Therefore, the time complexity to estimate the execution time for a DAG workflow is $O(n^2)$. For typical DAG workflows, the time to estimate the execution time of a DAG workflow in Algorithm 1 is often on a time scale of milliseconds.

6 Handling skewness

Through the detailed analysis for the execution plans of benchmarks in Sects. 3.1 and 3.2, we further find that another source of the loss of accuracy for the execution plan estimation is the skewness among tasks [, 16, 25]. For the task level execution, the main cause of skewness is the variation of output and computation overhead among tasks, which leads to skewness on task execution time. For the workflow level execution, the skewness affects the degree of parallelism of a job, which leads to additional sub-stages with the changed degree of parallelism for each job, and may delay the total execution time of a DAG.

Figure 8a shows the execution plan for a DAG without skewness. For this DAG, the reduce tasks of job 1 complete at the end of state 5, and there is no new job can be started based on the DAG dependency at the state 6. Thus more reduce tasks of job 2 starts accordingly to fill the newly available container slots.

Figure 8b shows the execution plan for a DAG with skewness. For this DAG, the execution time of the last three reduce tasks varies. This lead to no new tasks get

started at the state 6, and only 1 reduce task of job 2 get started at the state 7. Therefore, the degree of parallels are $\Delta_{1,5} = 3$ and $\Delta_{2,5} = 2$ for the state 5, and $\Delta_{1,6} = 2$ and $\Delta_{2,6} = 2$ for the state 6, and $\Delta_{1,7} = 1$ and $\Delta_{2,7} = 3$ for the state 7, and $\Delta_{2,8} = 4$ for the state 8. This is different from the degree of parallelism for the case without skewness in Fig. 8a.

6.1 Task-level model extensions

The skewness of MapReduce is caused by the non-uniform distribution of I/O overhead for output key-value tuples. We assume that the distribution of the I/O for output key-value tuples is the same for both profiling data and estimating data. In particular, the main source that leads to the skewness is the variation of the selectivity factor (i.e., α_m and α_r) among tasks. Take the map task for example, we assume that the user-defined logic output is $O(B^2)$ for an input tuple with size B. The first task has 3 records and each record is 2 bytes. The second task has 2 records and each record is 3 bytes. The input size for the two tasks is the same (i.e., 6 bytes). But the output for the first task is $3 * 2^2 = 12$ bytes and the output for the second task is $2 * 3^2 = 18$ bytes.

Note that the variation of task input size has been addressed by the BOE model in Sect. 4.2.

We assume that α_m, and α_r of all tasks from a job are independent normal random variables, denoted as $\bar{\alpha}_m$, and $\bar{\alpha}_r$, respectively.

During the profiling stage, we get the mean (μ) and variance (σ) of each random variable from MapReduce task counters. For the estimation stage, we use these statistics to generate the samples of $\bar{\alpha}_m$ and $\bar{\alpha}_r$ for each task by using the normal distribution $N(\mu, \sigma^2)$.

We formalize the distribution for the map stage of tasks considering skewness. According to the Proposition 1,

$$\bar{t}_{map} = a + b \cdot \bar{\alpha}_m \tag{9}$$

Then,

$$E(\bar{t}_{map}) = a + b \cdot E(\bar{\alpha}_m) \tag{10}$$

$$V(\bar{t}_{map}) = b^2 \cdot Var(\bar{\alpha}_m) \tag{11}$$

where a and b are constants.

According to Proposition 2, the distribution for the shuffle stage of tasks is formalized in the same fashion.

$$E(\bar{t}_{shuffle}) = a' + b' \cdot E(\bar{\alpha}_m) \tag{12}$$

$$V(\bar{t}_{shuffle}) = b'^2 \cdot Var(\bar{\alpha}_m) \tag{13}$$

Finally, we model the distribution for the reduce stage of tasks. According to Proposition 3, the reduce task time is modeled as follows.

$$\bar{t}_{reduce} = a'' \cdot \bar{\alpha}_m + \max\{b'' \cdot \bar{\alpha}_m, c'' \cdot \bar{\alpha}_r\} \tag{14}$$

Since $\bar{\alpha}_m$ and $\bar{\alpha}_r$ are independent, the last order statistics of a normal random variable is approximated as follows based on Fisher–Tippett–Gnedenko theorem [26].

$$E(\bar{t}_{reduce}) = a'' \cdot E(\bar{\alpha}_m) + b'' \cdot E(\bar{\alpha}_m) + c'' \cdot E(\bar{\alpha}_r) \tag{15}$$

$$\begin{aligned} &+ \lambda \cdot \sqrt{b''^2 \cdot Var(\bar{\alpha}_m) + c''^2 \cdot Var(\bar{\alpha}_r)} \\ V(\bar{t}_{reduce}) &= a''^2 \cdot Var(\bar{\alpha}_m) + \xi \cdot (b''^2 \cdot Var(\bar{\alpha}_m) + c''^2 \cdot Var(\bar{\alpha}_r)) \end{aligned} \tag{16}$$

where λ and ξ are constants.

Algorithm 2 State-based Cost Estimation for a DAG workflow with Skewness

$t_{dag} \leftarrow 0$
$s \leftarrow 0$
while G is not empty **do**
  Add new job.stage to stage queue $\mathcal{Q}$ from G
  $stage\_end\_flag = 0$
  while $stage\_end\_flag = 0$ **do**
    Estimate Δ_i for each job.stage $i \in \mathcal{Q}$ ▷ Section 5.2.1
    for each job.stage $i \in \mathcal{Q}$ **do**
      Estimate the number of pending tasks N
      for j=1 to N **do**
        Estimate $t_{task}(i, s, j)$ using extended BOE model
      end for
      Estimate $t_{state}(i, s)$ using $t_{task}(i, s, j), j = 1 \ldots N$
    end for
    $t_{state}(s) = t_{state}(k, s) = \min_{i=1}^{N_s}\{t_{state}(i, s)\}$ ▷ state of job k finishes
    for each job $i \in \mathcal{Q}$ **do**
      update the progress for job i ▷ Section 5.2.3
    end for
    if job i has no tasks in Q **then**
      $stage\_end\_flag$=1
      if job k in reduce stage **then**
        $\mathcal{Q}.remove(k)$
      else
        update job k to the reduce stage
      end if
    end if
    $t_{dag} = t_{dag} + t_{state}(s)$
    $s = s + 1$
  end while
end while
Return t_{dag}

6.2 Workflow-level model extensions

To estimate the execution time for tasks with skewness, we reduce the size of each *task track* $\Delta_{i,\tau,s}$ to 1. It means that two newly available containers will be assigned to task track $\Delta_{i,\tau,s}$ and $\Delta_{i,\tau+1,s}$ separately for the ith job with $\tau - 1$ existing task tracks at the stage s.

The skewness affects the degree of parallelism (Δ) of a job, which leads to additional sub-stages. The state-based approach in Fig. 6 is extended to finer-grained to address these additional sub-stages caused by the skewness. For the map or reduce

stage of a job, if the number of remaining map tasks is not enough to occupy the currently allocated slot, the container slot will be released to other jobs. In such a case, when there is a container slot released from a job, we re-estimate the degree of parallelism (Δ_i) for each job i in the DAG. Then, we estimate the task execution time as well as progress for each job. Note that we estimate the number of pending tasks for the state since the pending tasks of the state may not equal to all the pending tasks of the job. This refinement of the state-based approach is shown in Algorithm 2. Here a state is a sub-stage division caused by the skewness of tasks.

6.2.1 Analysis

Complexity For a DAG workflow with n jobs, there is a maximum of $2n$ stages without considering the skewness. According to Algorithm 2, the prediction is extended to a per task manner for each stage. Suppose the number of tasks in a job is m, the time complexity to predict the overall execution time of a DAG is $O(m \cdot n^2)$. Since there are often a few jobs in a DAG workflow, the time complexity is approximate to $O(m)$.

Bounds We provide the bounds of the gain from the skewness based cost estimation by the following theorem.

Theorem 1 *Given the degree of parallel Δ and the number of waves N of a job in a stage, the lower and upper bound of the expected gain from the skewness based approach is*

$$0.23\sqrt{\frac{N}{\Delta}}\sigma\sqrt{\log\Delta} \leq E[t_{gain}] \leq \sqrt{\frac{2N}{\Delta}}\sigma\sqrt{log\Delta} \tag{17}$$

where σ is the variance derived from Eqs. 9–15. The number of waves of a job the number of parallel task batches in a stage. Take Fig. 8a for example, there are 2 waves for job 2 in stage 5.

Proof The execution time of the stage is bounded by the slowest task track. We firstly figure out the distribution of total execution time for tasks on the same track, and then derive the upper bound and lower bound of the slowest task track among all the tracks run in parallel.

The task-level model from Eq. 9 to Eq. 15 in Sect. 6.1 shows that the task execution time follows a normal distribution of which mean and variance can be expressed by the feature of selectivity factors (i.e., α_m and α_r). We denote that tasks of a job in a stage follow the normal distribution $N(\mu, \sigma^2)$. For each task track in a stage, there are N/Δ tasks run in series (i.e., N/Δ waves). Therefore, the distribution of the execution time of a task track follows the normal distribution $N(\frac{N}{\Delta} \cdot \mu, \frac{N}{\Delta} \cdot \sigma^2)$.

According to the analysis in [27], the expected maximum of N' samples from the normal distribution $N(\mu', \sigma'^2)$ is bounded by the following inequality.

$$0.23\sigma'\sqrt{\log N'} \leq E[Y - E[X]] \leq \sqrt{2}\sigma'\sqrt{\log N'} \tag{18}$$

where $Y = \max_{1 \leq i \leq N'} X_i$.

Since there are Δ *tasks run in parallel in the cluster during a stage, the lower bound and upper bound of the gain from the skewness based estimation in Algorithm* 2 *is obtained as follows.*

$$0.23\sqrt{\frac{N}{\Delta}}\sigma\sqrt{\log \Delta} \leq E[t_{gain}] \leq \sqrt{\frac{2N}{\Delta}}\sigma\sqrt{log\Delta} \tag{19}$$

Theorem 1 means that the expected gain from the skewness based approach can be expressed by the selectivity factors of a task. Since the statistics of selectivity factors can be collected from history job profiles, the bounds can be figured out before cost estimation.

7 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 [9] and MRTuner [12], 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.

Our main experimental findings and lessons learned are summarized as follows.

- We validate that the BOE model can identify system bottleneck with varying the degree of parallelism, hence effectively to estimate the task-level execution time, for both single job and parallel jobs. As a comparison, the BOE model outperforms the best case of the previous cost models including Starfish [9] and MRTuner [12] by a factor of five on average.
- We observe the high accuracy for the skew-aware state-based approach to estimate both the overall execution time and break-down time of each stage for a DAG workflow. The average prediction error is under 3% when estimating the execution time of 51 hybrid analytics (HiBench) and query (TPC-H) DAG workflows.
- The overhead of the computation of our cost models is very low (within 1 s for all the cases). This is sufficient for serving applications such as auto-tuning and online progress estimation.

Table 2 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

7.1 Experimental setup

7.1.1 Cluster configuration

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. Hadoop is deployed on Java 1.7.0. We use Hadoop version 2.8.4 to run MapReduce on YARN [17]. We use one node as the master for management and scheduling, and the other ten nodes as slaves for computation and storage. We use 2 disks to store intermediate data for MapReduce and for storing HDFS data as well. We configure HDFS with 128 MB block size and a default replication factor of 3. The default JVM heap size is set to 2 GB per task.

7.1.2 Workload description

We define a set of representative workloads for the experimental evaluation. As shown in Table 2, we use Word Count and TeraSort for micro-benchmarks. We use PageRank for graph analysis and Kmeans for machine learning, both from HiBench [28]. The query workload is selected from TPC-H.[3] **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. We set the `hive.exec.parallel` parameter to true to make independent jobs in a query DAG run in parallel.

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

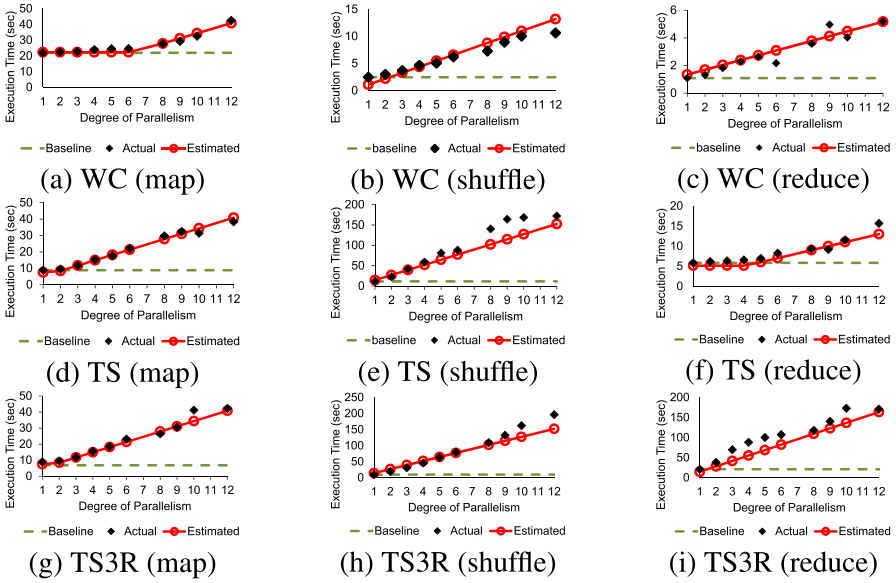


Fig. 9 The task-level effect of BOE model for a single job

7.2 BOE model

We evaluate the effect of the task-level BOE model. We use the best cases of Starfish [9] and MRTuner [12] 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.

7.2.1 Single job

Figure 9a–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 6.6×, 4.3×, and 4.1× 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. The max operation in Proposition 1 can automatically identify this change of the bottleneck, and hence we have the turning point when the degree of parallelism is 6 for the BOE model. For the shuffle stage, the WC job is network-bound under all the configuration. Our BOE cost model can identify this bottleneck by using the max operation in Proposition 2. For the reduce stage of the WC job, the bottleneck is network due to the HDFS write. The execution time for reduce

Table 3 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	–	–

tasks is proportional to the degree of parallelism according to the cost estimation in Proposition 3.

Figure 9d–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 the baseline by a factor of 4.3×, 10.6×, and 1.9× 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 in Proposition 3.

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

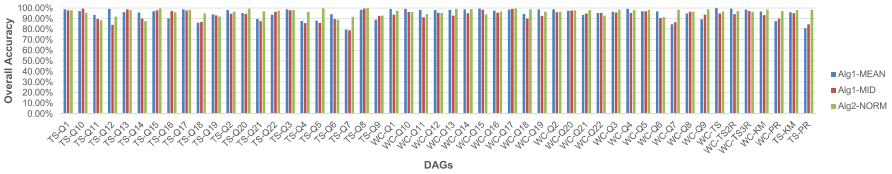
7.2.2 Multiple jobs

We evaluate the task level BOE model for parallel jobs. The DAG workflow has two parallel jobs. Table 3 presents the accuracy of the task level model for parallel jobs including WC, TS, and TS3R, running simultaneously.

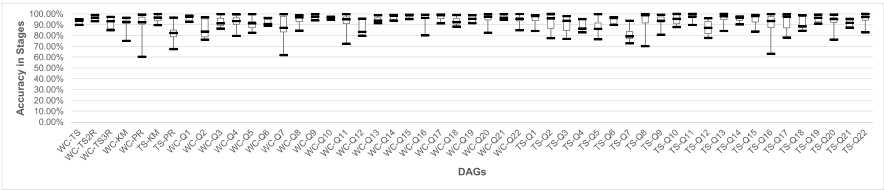
For WC and TS run in parallel, the average accuracy is 99.7% and 88.7% for states 1 and 2, which consist of parallel jobs. For state 1, the BOE model identified the CPU bottleneck. When the workflow enters state 2, the BOE model identifies bottlenecks for the TS reduce stage, which is network-bound for 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 Sect. 7.2.1.

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.

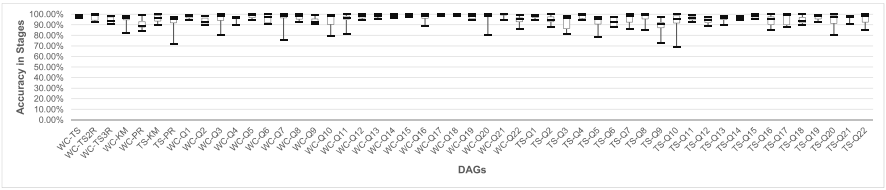
Consequently, the BOE model can identify the bottleneck resource and its allocation for each job, and hence to estimate the execution time for each task.



(a) Estimation Accuracy for DAGs



(b) Estimation Accuracy for Stage Break-downs using Algorithm 1



(c) Estimation Accuracy for Stage Break-downs using Algorithm 2

Fig. 10 Evaluation of the state-based approach for hybrid query and computing DAGs

7.3 State-based approach

We evaluate the effectiveness of the state-based estimation framework as well as the skewness related improvements 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. To evaluate the effectiveness of the skewness related improvements, we compare the estimation result by using three profiling approaches including mean (Algorithm 1), median (Algorithm 1) and normal distributions (Algorithm 2). 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 break-down 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 in Fig. 10a 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. Furthermore, when taking the skewness into account, the minimal accuracy of

DAG estimation for all the workloads is improved to 91.2% for all the workloads by using Algorithm 2. Take TS-PageRank(PR) for example, the accuracy of estimation is improved by 20.1% when the skewness related optimization is enabled. This is because PageRank has significant skewness. Furthermore, all of the stage break-downs are correctly identified by using the state-based approach.

The rest of the columns in Fig. 10a 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. Fig. 10a presents the overall DAG estimation accuracy for hybrid TS+TPC-H workloads. 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. Algorithm 2 achieves 99.83% of accuracy in DAG estimation for WC+Q21.

For the impact of skewness, there is a significant improvement for some DAG workflows like TS+Q7, and almost no improvement for other DAG workflows like TS+Q17. The improvement depends on the skewness of tasks. The different degrees of skewness (e.g., standard deviation) for statistics are collected in profiles. The query with higher degree of skewness will have much more improvement since the skewness related profiling captured the characteristic. Note that the skewness profiling and estimation is transparent to users.

Stage break-down results Next, we break down to the execution of each stage of DAGs. Figure 10b and c show the statistic of accuracy which is broken down to each stage, for Algorithm 1 (using median) and Algorithm 2, respectively. Boxes depict the 25th, 50th, and 75th percentiles, and whiskers depict min and max. The average accuracy of the 75th percentiles is 98.18% and 96.6%, for Algorithms 1 and 2, respectively. It indicates that the model can also be used to predict detailed progress for the DAG. Furthermore, the result shows the accuracy is not sensitive to the number of stages.

We observe that the overall estimation of DAG workflows is slightly better than the break-down of cost estimation for each stage. We find that there are two sources for the loss of accuracy: (1) task scheduling order (e.g., WC uses all the slot when the Q3 query is in compilation at the beginning of WC+Q3 DAG) and (2) estimated bound of each stage. The impact of scheduling on the estimation accuracy is reduced as more rounds of tasks are executed. The accuracy of the late stages is better than that of the early stages. Thus we observe that the overall estimation for DAG workflows is better than break-down for stages.

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 s 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.

8 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 to tune MapReduce configurations [, , 9, 12, 29]. 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 [31]. 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 [32] 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 [33] 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.

The preliminary version of the BOE model was proposed in our previous workshop paper [34]. This new paper includes more than 30% new materials, including that (1) we addressed the skewness issue in the workflow level model; (2) we added the analysis of expected gain of the model, and (3) we added the evaluation of the skewness impact.

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 [35]. There do exist interesting works on the resource usage model for parallel queries such as join [23], 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 [36]. 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.

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 [5]. It uses the What-if Engine building block of Starfish for the cost estimation [9]. 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 [14], 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 [11]. However, this work focuses on DAG-level, and it does not address the task-level cost models. The work in [10] 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 [37] and the BSP model [38] 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 [39] 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.

Data skew The skewness is a well known issue in MapReduce framework [, 16, 25]. In contrast to our work for cost estimation under skew tasks, these works address the skewness issue by extending the MapReduce framework to repartition skew data for load balancing. The performance model of data skew effects is studied for parallel database systems [40]. However, these models are for relational queries such as parallel joins, hence they can not be used for MapReduce directly.

9 Conclusion

In this paper, we proposed the BOE model to predict the allocation of preemptable system resources for task-level execution time estimation. The state-based approach was proposed for workflow-level execution plan estimation. We refined the granularity for cost models to tackle the challenge of skewness to further improve the estimation accuracy. 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 apply our cost models in automatic tuning for DAG workflows.

Funding Open Access funding provided by University of Helsinki including Helsinki University Central Hospital.

References

1. Project hydrogen: unifying state-of-the-art ai and big data in apache spark. https://databricks.com/session/databricks-keynote-2
2. Apache tez. https://tez.apache.org/
3. Assefi, M., Behravesh, E., Liu, G., Tafti, A.P.: Big data machine learning using apache spark mllib. In: Proceedings of the IEEE International Conference on Big Data (Big Data), pp. 3492–3498. IEEE (2017)
4. Isard, M., Budiu, M., Yu, Y., Birrell, A., Fetterly, D.: Dryad: distributed data-parallel programs from sequential building blocks. ACM SIGOPS Oper. Syst. Rev. **41**, 59–72 (2007)
5. Lim, H., Herodotou, H., Babu, S.: Stubby: a transformation-based optimizer for mapreduce workflows. VLDB **5**(11), 1196–1207 (2012)
6. Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., et al.: Mllib: machine learning in apache spark. J. Mach. Learn. Res. **17**(1), 1235–1241 (2016)
7. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M.J., Shenker, S., Stoica, I.: Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: NSDI (2012)
8. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. CACM **51**(1), 107–113 (2008)
9. Herodotou, H., Babu, S.: Profiling, what-if analysis, and cost-based optimization of mapreduce programs. VLDB **4**(11), 1111–1122 (2011)
10. Li, T., Tang, J., Xu, J.: Performance modeling and predictive scheduling for distributed stream data processing. IEEE Trans Big Data **2**(4), 353–364 (2016)
11. Taufer, M., Rosenberg, A.L.: Scheduling dag-based workflows on single cloud instances: high-performance and cost effectiveness with a static scheduler. Int. J. High Perform. Comput. Appl. **31**(1), 19–31 (2017)
12. Shi, J., Zou, J., Lu, J., Cao, Z., Li, S., Wang, C.: MRTuner: a toolkit to enable holistic optimization for mapreduce jobs. VLDB **7**(13), 1319–1330 (2014)
13. Herodotou, H., Dong, F., Babu, S.: No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics. In: SoCC, p. 18 (2011)
14. Morton, K., Balazinska,, M. and Grossman, D.: ParaTimer: a Progress Indicator for MapReduce DAGs. In: SIGMOD, pp. 507–518 (2010)
15. Clifton, B.: Advanced web metrics with google analytics (2012)
16. Kwon, Y., Balazinska, M., Howe, B., Rolia, J.: Skewtune: mitigating skew in mapreduce applications. In: SIGMOD, pp. 25–36 (2012)
17. Vavilapalli, V.K., Murthy, A.C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., Saha, B., Curino, C., O'Malley, O., Radia, S., Reed, B., Baldeschwieler, E.: Apache Hadoop YARN: yet another resource negotiator. In: SoCC, pp. 5:1–5:16 (2013)
18. Ghodsi, A., Zaharia, M., Hindman, B., Konwinski, A., Shenker, S., Stoica, I.: Dominant resource fairness: fair allocation of multiple resource types. In: NDSI, pp. 323–336 (2011)
19. Thusoo, A., Sarma,, J.S., Jain, N., Shao, Z., Chakka, P., Zhang, N., Antony, S., Liu, H., Murthy, R.: Hive-a petabyte scale data warehouse using hadoop. In: ICDE, pp. 996–1005 (2010)
20. Boehm, M., Tatikonda, S., Reinwald, B., Sen, P., Tian, Y., Burdick, D.R., Vaithyanathan, S.: Hybrid parallelization strategies for large-scale machine learning in SystemML. VLDB **7**(7), 553–564 (2014)

21. Ghoting, A., Krishnamurthy, R., Pednault, E., Reinwald, B., Sindhwani, V., Tatikonda, S., Tian, Y., Vaithyanathan, S.: SystemML: declarative machine learning on MapReduce. In: ICDE, pp. 231–242 (2011)
22. Shi, J., Qiu, Y., Minhas, U.F., Jiao, L., Wang, C., Reinwald, B., Özcan, F.: Clash of the titans: MapReduce vs. Spark for large scale data analytics. VLDB **8**(13), 2110–2121 (2015)
23. Ganguly, S., Hasan, W., Krishnamurthy, R.: Query optimization for parallel execution. In: SIGMOD, pp. 9–18 (1992)
24. Saltzer, J.H., Kaashoek, M.F.: Principles of Computer System Design: An Introduction. Morgan Kaufmann, Burlington (2009)
25. Gufler, B., Augsten, N., Reiser, A., Kemper, A.: Handing data skew in mapreduce. ICCCSS **146**, 574–583 (2011)
26. David, H.A., Nagaraja, H.N.: Order statistics. Wiley Online Library (1970)
27. Kamath, G.: Bounds on the expectation of the maximum of samples from a gaussian. http://www.gautamkamath.com/writings/gaussian max. pdf (2015)
28. Huang, S., Huang, J., Dai, J., Xie, T., Huang, B.: The hibench benchmark suite: characterization of the mapreduce-based data analysis. In: ICDEW, pp. 41–51 (2010)
29. Khan, M., Jin, Y., Li, M., Xiang, Y., Jiang, C.: Hadoop performance modeling for job estimation and resource provisioning. IEEE Trans. Parallel Distrib. Syst. **27**(2), 441–454 (2016)
30. Shen, C., Tong, W., Hwang, J.-N., Gao, Q.: Performance modeling of big data applications in the cloud centers. J. Supercomput. **73**(5), 2258–2283 (2017)
31. Huang, B., Boehm, M., Tian, Y., Reinwald, B., Tatikonda, S., Reiss, F.R.: Resource elasticity for large-scale machine learning. In: SIGMOD, pp. 137–152 (2015)
32. Venkataraman, S., Yang, Z., Franklin, M.J., Recht, B., Stoica, I.: Ernest: efficient performance prediction for large-scale advanced analytics. In: NSDI, pp. 363–378 (2016)
33. Singhal, R., Singh, P.: Performance assurance model for applications on spark platform. In: Proceedings of the Technology Conference on Performance Evaluation and Benchmarking, pp. 131–146. Springer (2017)
34. Shi, J., Lu, J.: Performance models of data parallel DAG workflows for large scale data analytics. In: 2021 IEEE 37th International Conference on Data Engineering Workshops (ICDEW). IEEE (2021)
35. Chaudhuri, S.: An overview of query optimization in relational systems. In: PODS pp. 34–43 (1998)
36. Li, J., Naughton, J., Nehme, R.V.: Resource bricolage for parallel database systems. VLDB **8**(1), 25–36 (2014)
37. Culler, D., Karp, R., Patterson, D., Sahay, A., Schauser, K.E., Santos, E., Subramonian, R., Von Eicken, T.: LogP: towards a realistic model of parallel computation, vol. 28. ACM (1993)
38. Cheatham, T., Fahmy, A., Stefanescu, D., Valiant, L.: Bulk synchronous parallel computing paradigm for transportable software. In: TEPDS, pp. 61–76 (1996)
39. Ousterhout, K., Rasti, R., Ratnasamy, S., Shenker, S., Chun, B.-G., ICSI, V.: Making sense of performance in data analytics frameworks. In: NSDI, vol. 15, pp. 293–307 (2015)
40. Walton, C.B., Dale, A.G., Jenevein, R.M.: A taxonomy and performance model of data skew effects in parallel joins. VLDB **91**, 537–548 (1991)