Scheduling Big Data Workflows in the Cloud under Deadline Constraints

Mahdi Ebrahimi
Dep. of Math. & Computer Science
Lawrence Technological University
Southfield, U.S.A.
mebrahimi@ltu.edu

Aravind Mohan
Dep. of Computer Science
Allegheny College
Meadville, U.S.A.
amohan@allegheny.edu

Shiyong Lu
Dep. of Computer Science
Wayne State University
Detroit, U.S.A.
shiyong@wayne.edu

***Abstract*—With the advent of cloud computing, an unbound number of compute resources can be leased from the cloud providers. In such an environment, the number of assigned resources to a workflow can be elastically scaled in and out on a demand basis using the added Quality of Service (QoS) constraints such as the budget and the deadline. The heterogeneous nature of the cloud resources makes the decision of selecting resource type for each workflow a challenging problem. Although there are several existing research studies that propose both static and dynamic scheduling algorithms for both homogeneous and heterogeneous cloud resource types, they do not take advantage of the data dependency information that is part of the workflow structure during the scheduling process. There is still room for improvement, since the scheduling problem is an NP-hard problem. In this paper we propose a new Big data wOrkflow scheduleR undeR deadlIne conStraint (BORRIS) that is used to minimize the execution cost of the workflow under a provided deadline constraint in a heterogeneous cloud computing environment. We have implemented the proposed algorithm in our big data workflow system called DATAVIEW and the experimental results show the competitive advantage of our approach.**

Keywords-big data workflows; big data; scheduling;BORRIS

I. INTRODUCTION

The current trend in the use of cloud-computing paradigms for big data querying and analytics has opened up a new set of challenges to the workflow-scheduling problem [7]. The cloud-computing environment provides an easily accessible and scalable framework that guides the process of leasing an unbounded set of resources with heterogeneous types. The workflow engine that is mainly responsible for the orchestration of the execution of the workflow, will now need to make more intelligent decisions about when and where to execute the tasks in a workflow. The existing big data workflow engine [13, 14], has a limitation on the assignment of resources to a workflow at design time based on the structure of the workflow. Due to the nature of big data processing in those workflows, the tasks are compute and data intensive, and hence there is a strong need for scheduling those tasks in different types of machines in the cloud by making the necessary decisions at run time.

The scheduling decision making process needs to be user interactive in order to emphasize on the usability of the system. Existing approaches such as [1, 6], do not consider any QoS constraints that relate to the update of user run time requirements. We take a different approach in this paper to schedule the workflow based on the user defined deadline constraints. We perform a single objective optimization task to minimize the execution cost of the workflow with an intuition that based on the provided deadline the cost can vary. It is based on the assumption that the provided deadline the cost can vary over time and that the workflow costs are smaller for large workflows than small ones.

In this paper, we propose a new Big data wOrkflow scheduleR undeR deadlIne conStraint (BORRIS) that is used to minimize the execution cost of the workflow under a provided deadline constraint in a heterogeneous cloud computing environment. We implemented the proposed algorithm in our big data workflow system called DATAVIEW and the experimental results show the competitive advantage of our approach.

The rest of the paper is organized as follows, first, in Section II we define and formalize our system model. In Section III we explain our workflow scheduling algorithm in detail. Then, in Section IV, the experimental results are shown and discussed. Section V presents the related work. Finally, the conclusions and future work are presented in Section VI.

II. SYSTEM MODEL

To execute a big data workflow in the cloud, we need to model the cloud first. A cloud computing environment is modeled as follows:

Definition 2.1 (Cloud Computing Environment *C*): A cloud computing environment is a 6-tuple C (R, R_T, R_C, F_B, F_R, R_S), where

- R is a set of resources. Each individual resource is denoted by R_i in the cloud computing environment.
- R_T is a set of resource types such as {"t2.nano", "t2.micro", "t2.small", "t2.medium, "t2.large", …}.
- R_T: $R \rightarrow Q^+$ is the resource usage time function. $R_T(R_i)$, $R_i \in R$ gives the time for the resource usage R_i in the cloud computing environment. The resource with the minimum R_T is called $R_{slowest}$ and the resource with the maximum R_T is called $R_{fastest}$.
- F_B: $R \times R \rightarrow Q^+$ is the data communication rate function. $F_B(R_{i1}, R_{i2})$, $R_{i1}, R_{i2} \in R$ gives the data communication rate between R_{i1} and R_{i2}. Q^+_0 is some pre-determined unit like bytes per second. This function is used to calculate the data movement time between two resources in the cloud.

- F_R: $R \rightarrow Q^+$ is the resource computing speed function. $F_R(R_i)$, $R_i \in R$ gives the speed for the computing resource R_i measured in some pre-determined unit like million instructions per machine cycles or million instructions per nanoseconds.
- F_S: $R_T \rightarrow R$ is the resource provisioning function. $F_S(R_t)$, $R_t \in R_T$ returns a resource instance of the resource type of R_t.

A big data workflow is used to model a process as a set of computational tasks and their data dependencies to analyze big datasets. A big data workflow can be modeled as a directed acyclic graph (DAG) such that workflow tasks are denoted as graph nodes and dependencies between tasks are denoted as graph edges. Formally, we define a big data workflow as:

Definition 2.2 (Big Data Workflow W): A big data workflow can be formally defined as a 4-tuple $W = (T, D, F_T, F_D)$, where

- T is a set of tasks in the workflow W. Each individual task is denoted by T_k.
- $D = \{<T_{k1}, T_{k2}> \mid T_{k1}, T_{k2} \in T, k_1 \neq k_2; k_1, k_2 \leq |T|, T_{k2}$ consumes data $D_{k1,k2}$ produced by $T_{k1}\}$ is a set of data dependencies. $D_{k1,k2}$ denotes that an amount of data is required to be transferred after T_{k1} completes and before T_{k2} starts. D_k represents all the outgoing edges from task T_k.
- F_T: $T \rightarrow Q^+$ is the execution time function. $F_T(T_k)$; $T_k \in T$ gives the execution time of a task T_k, measured in some pre-determined unit like million instructions per machine cycles or million instructions per nanoseconds.
- F_D: $D \rightarrow Q^+$ is the data size function. $F_D(D_{k1,k2})$, $D_{k1,k2} \in D$ gives the size of a dataset $D_{ki,k2}$, measured in some predetermined unit like bits or bytes.

To schedule a big data workflow to a set of cloud resources, more measurements like number of instructions of tasks and data sizes are required. Therefore, we define big data workflow graph as a weighted directed acyclic graph that includes a set of tasks and their data dependencies. The weights of the tasks and data edges are based on the average task computation and average data communication time, respectively. In addition, the workflow can be partitioned into a set of partitions such that there is no data dependency between all the tasks of each partition. A big data workflow graph can be defined formally as:

Definition 2.3 (Big Data Workflow Graph G): Given a workflow W in a cloud computing environment C, a big data workflow graph G, represents a weighted directed acyclic graph with 14-tuple $G(T, D, R, Fc, F_{\bar{c}}, Fp, F_{\bar{p}}, F_m, F_{\bar{m}}, F_n, F_{\bar{n}}, P, TP, RT)$, where

- The vertices of the graph represent a set of tasks T.
- The edges of the graph represent a set of data dependencies D.
- R is a set of resources in the cloud computing environment.
- Fc: $D \times R \times R \rightarrow Q + 0$ is the data communication cost function; $F_c(D_{k1,k2}, R_{i1}, R_{i2})$, $D_{k1,k2} \in D$; $R_{i1}, R_{i2} \in R$ gives the data communication cost of $D_{k1,k2}$ while transferring from resource R_{i1} to resource R_{i2}.
- $F_{\bar{c}}$: $D \rightarrow Q + 0$ is the average data communication cost function. $F_{\bar{c}}(k_1, k_2)$, $D_{k1,k2} \in D$ gives the average data communication cost of $D_{k1,k2}$ in resources R, which is taken as the weight of edge in the graph G. The weight of the edge is 0 for the same resource.
- F_p: $T \times R \rightarrow Q^+$ is the task computation cost function. $F_p(T_k, R_i)$, $T_k \in T$, $R_i \in R$ gives the computation cost of T_k on resource R_i.
- $F_{\bar{p}}$: $T \rightarrow Q^+$ is the average task computation cost function, $F_{\bar{p}}(T_k)$ gives the average computation cost of task T_k, which is taken as the weight of vertex in the graph G.
- F_m: $D \times R \times R \rightarrow Q^+_0$ is the data communication time function; $F_m(D_{k1,k2}, R_{i1}, R_{i2})$, $D_{k1,k2} \in D$; $R_{i1}, R_{i2} \in R$ gives the data communication time of $D_{k1,k2}$ while transferring from resource R_{i1} to resource R_{i2}.
- $F_{\bar{m}}$: $D \rightarrow Q + 0$ is the average data communication time function. $F_{\bar{m}}(D_{k1,k2})$, $D_{k1,k2} \in D$ gives the average data communication time of $D_{k1,k2}$ for all the resources, R.
- F_n: $T \times R \rightarrow Q^+$ is the task computation time function. $F_p(T_k, R_i)$, $T_k \in T$, $R_i \in R$ gives the computation time of T_k on resource R_i.
- $F_{\bar{n}}$: $T \rightarrow Q +$ is the average task computation time function, $F_{\bar{n}}(T_k)$ gives the average computation time of task T_k, which is taken as the weight of vertex in the graph G.
- P: $N \rightarrow T$ is the partition task function, $P[j]$ or P_j gives all the tasks in partition j. R_{Pj} represents the set of resources of partition P_j.
- TP: $T \rightarrow N$ is the task partition function, $TP[T_k]$ or TP_{Tk} gives the partition number of task T_k.
- RT: $P \rightarrow R_T$ is the partition resource type function. $RT[P_j]$ gives the resource type that is assigned to partition j.

Workflow makespan is the total time needed to execute the whole workflow starting from the beginning task(s) to the end task(s) on the cloud. Our goal is to come up with an optimal workflow schedule such that the workflow execution cost is minimized while the workflow makespan meets the given deadline. To this end, we need to model workflow execution in order to be able to measure both workflow makespan and execution cost. As we divide the workflow into a set of partitions, the workflow makespan and cost will be the summation of the execution times and costs of all the partitions. We define the workflow execution environment as follows:

Definition 2.4 (Workflow Execution Environment WE_C):

Given a workflow W in a cloud computing environment C, a workflow execution, represents the execution time of the

workflow with 5-tuple WE_C (CT, $\overline{CT}$, $minCT$, $maxCT$, CC), where

- CT: $Partition \times R \rightarrow Q^+$ is the workflow partition completion time function. CT (P_j, R_i), $P_j \in Partition$ gives the maximum of task computation time of all the tasks $T_k \in P_j$ assigned to R_i as well as the maximum of data communication time of all the outgoing edges from all the tasks $T_k \in P_j$. We formally define CT as:

$$CT(P_j, R_i) = \text{Max}_{T_k \in Pj}\{F_n(T_k, R_i)\} + \text{Max}_{T_k \in Pj}\{F_m(D_{k,k1}, R_i, R_{i1})\}$$

- $\overline{CT}$: $Partition \rightarrow Q^+$ is the average workflow completion time function. $\overline{CT}$ (P_j), $P_j \in Partition$ gives the max of average task computation time of all the tasks $T_k \in P_j$ and the average data communication time for all the outgoing edges from all the tasks $T_k \in P_j$. We formally define $\overline{CT}$ as:

$$\overline{CT}(P_j) = \sum_{k=1}^{K} F_{\bar{n}}(T_k) + \sum_{\substack{k=1,k1=1 \\ k \neq k1}}^{K} F_{\bar{m}}(D_{k,k1})$$

- $minCT$: $Partition \rightarrow Q^{+}{}_0$ is the minimum workflow partition completion time function. $minCT(P_j)$, $Pj \in Partition$ gives the minimum task computation time of all the tasks $T_k \in P_j$ and the minimum data communication time for all the outgoing edges from all the tasks $T_k \in P_j$. We formally define it as:

$$minCT(P_j) = \sum_{T_k \in P_j} CT(T_k, R_{fastest})$$

- $maxCT$: $Partition \rightarrow Q^{+}{}_0$ is the maximum partition completion time function. $maxCT(P_j)$, $P_j \in Partition$ gives the maximum task computation times of all the tasks $T_k \in P_j$ and the maximum data communication times for all the outgoing edges from all the tasks $T_k \in P_j$. We formally define $maxCT$ as:

$$maxCT(P_j) = \sum_{T_k \in P_j} CT(T_k, R_{slowest})$$

- CC: $Partition \times R \rightarrow Q^+$ is the workflow partition completion cost function. CC (P_j, R_i), $P_j \in Partition$ gives the sum of task computation cost of all the tasks $T_k \in P_j$ assigned to R_i as well as the data communication cost for all the outgoing edges from all the tasks $T_k \in P_j$. We formally define CC as:

$$CC(P_j, R_{i1}) = \sum_{k=1}^{K} F_p(T_k, R_{i1}) + \sum_{\substack{i1,i2=1 \\ i1 \neq i2}}^{I} \sum_{\substack{k=1,k1=1 \\ k \neq k1}}^{K} F_c(D_{k,k_1}, R_{i1}, R_{i2})$$

Our scheduling algorithm is a partition-based approach. First, the workflow is divided and partitioned into the several partitions. Then, the given deadline is partitioned into the sub-deadlines and assigned to each workflow partition. The sub-deadlines are calculated based on the execution time of each workflow partition. To calculate the sub-deadlines, we define workflow partition makespan as follows:

Definition 2.5 (Workflow Partition Makespan P_M): Given a workflow W in a cloud computing environment C and deadline D, a workflow partition makespan, represents the sub-deadline provided to each partition of the workflow with 6-tuple P_M(SD, $Threshold$, PRT, ACT, $Earliness$, $Lateness$), where

- SD: $Partition \rightarrow Q^+$ is the sub-deadline partition function. SD (P_j), P_j □ $Partition$ gives the sub-deadline assigned to the partition P_j. Supposedly CTM is the makespan of the critical path in the workflow, then SD can be calculated formally as follows:

$$SD[P_j] = (\text{Max}_{T_k \in \text{Pj}}\{\overline{CT}(P_J)\} / CTM) * D$$

- $Threshold$: $Partition \rightarrow Q+$ is the threshold partition function. $Threshold$ (P_j), $P_j \in Partition$ gives the threshold time assigned to the partition P_j. It can be calculated as follows:

$$Threshold(P_j) = \text{Max}\{0, SD(P_{j+1}) - minCT(P_{j+1})\}$$

- PRT: $Partition \times R \times SD \times Threshold \rightarrow RT$ is the partition resource type function. It is used to identify the slowest resource for executing the workflow tasks in a partition. The execution time of the resource type output of PRT is required to be less than the sum of sub-deadline and threshold allocated to the partition.

- ACT: $Partition \rightarrow Q^+$ is the actual completion time that is used to compute the total time for completing all the tasks in a partition. It can be formally defined as:

$$ACT(P_j) = \sum_{\substack{k=1 \\ T_k \in P_j}}^{K} CT(T_k, F_S(R_T[P_j]))$$

- Earliness: Partition $\rightarrow$ Q^+ is the earliness partition function. $Earliness$ (P_j), P_j □ $Partition$ gives the earliness time that the partition P_j can be executed. It is calculated as follows:

$$Earliness(P_j) = \text{Max}\{0, SD(P_j) - ACT(P_j)\}.$$

- $Lateness$: $Partition \rightarrow Q^+$ is the lateness partition function. $Lateness$ (P_j), $P_j \in Partition$ gives the lateness time that the partition P_j is executed. It is calculated as follows:

$$Lateness(P_j) = \text{Max}\{0, ACT(P_j) - SD(P_j)\}$$

The critical path in a workflow can be computed by the SCPOR algorithm [6]. Our goal is to minimize workflow execution cost while satisfying the deadline constraint. We formally define our objective function and the constraints as follows:

Definition 2.6 (Workflow Cost Minimization W_C): Given a workflow W in a cloud computing environment C, and deadline D, workflow execution cost is the objective function and can be defined as follows

$$W_C = \sum_{j=1}^{J} \sum_{i=1}^{I} CC(P_j, R_i) \times X_{ji}$$

where,

$$X_{ji} = \begin{cases} 1, & if\ partition\ P_j\ is\ assigned\ to\ resource\ R_i \\ 0, & otherwise \end{cases}$$

such that the following constraints are satisfied:

1) $\sum_{j=1}^{J} \sum_{i=1}^{I} CT(P_j, R_i) \times X_{ji} <= D$

2) $\sum_{j=1}^{J} X_{ji} = 1$ for all the tasks in partition j assigned to the resource $R_i \in R$.

There are three cases to consider:

1) If $D < \sum_{j=1}^{J} minCT(P_j)$, then we can satisfy the deadline constraints and so a solution is to assign all the partition tasks to the slowest resource.

2) If $D > \sum_{j=1}^{J} maxCT(P_j)$, then we satisfy the deadline constraint by assigning all the partition tasks to the fastest resource as a solution.

3) If $\sum_{j=1}^{J} minCT(P_j) <= D <= \sum_{j=1}^{J} maxCT(P_j)$, then we use our strategy to find the optimal solution.

III. THE BORRIS ALGORITHM

The main steps of the BORRIS algorithm are shown in Fig.1 Workflow specification and deadline are the two required inputs for BORRIS. In the first step, BORRIS parses the given workflow specification and assigns a non-negative number (weight) to each workflow task and edge to generate a weighted DAG. We use the number of instructions in of tasks, and data movement size of the edges along with the cloud resource types information in order to generate their weights. The average computation times are calculated as the weights of tasks and the average data movement times are calculated as the weights of edges.

After generating the weighted DAG for the workflow, BORRIS partitions the workflow into several partitions such that there is no data dependency (edge) between the tasks inside each partition however, there is a possibility to have data dependencies between the partitions. In the next step, BORRIS distributes the given deadline and assigns initial sub-deadlines to all of the partitions. For the deadline distribution, BORRIS computes the maximum time needed to execute the workflow (i.e. workflow makespan) by calculating the makespan of the critical path. Then, it assigns the sub-deadlines to all of the partition based on the workflow makespan and average completion time of each partition.

In the next step, the maximum and minimum completion times for each partition are calculated. The maximum completion time is the completion time of the partition once all its tasks are assigned to the slowest cloud resource and the minimum completion time is the completion time once all its tasks are assigned to the slowest cloud resource.

In addition, BORRIS computes a threshold value for each partition by taking away some extra time from their subsequent partitions. The initial sub-deadline of each partitions is increased by the threshold and it provides more room to select a slower resource for the partition and therefore the execution cost of the partition is minimized.

For the next step, BORRIS goes through all the partitions sequentially and complete the schedule map by assigning all the partitions on to the most appropriate cloud resources.

After identifying the appropriate resource type for the partition, each task in the partition is scheduled to execute in a resource instance of the resource type in parallel. The actual completion time, the earliness and lateness values for each partition is calculated after partition execution. Then BORRIS adjusts the sub-deadline of the subsequent partition by using these earliness and lateness values. If the partition is the last partition, BORRIS does not need to calculate the earliness and lateness values as there is no subsequent partition that uses them.

For example, let us consider the workflow of Fig. 2 with 200 minutes as the deadline. This workflow consists of seven tasks as the vertices and ten data dependencies as the edges. The workflow is partitioned into three partitions as P_1= {T_1, $D_{1,2}$, $D_{1,3}$, $D_{1,4}$, $D_{1,5}$, $D_{1,6}$}, P_2= {T_2-T_6, $D_{2,7}$, $D_{3,7}$, $D_{4,7}$, $D_{5,7}$, $D_{6,7}$} and P_3={T_7}. Once the weighted DAG of the workflow (TABLE I. B, C) is computed, then the initial sub-deadline, maximum and minimum completion time as well as the threshold value of the three partitions are calculated and shown in TABLE I. D.

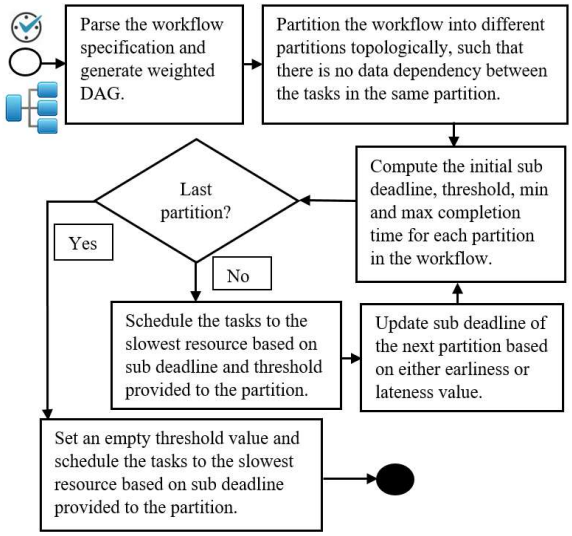

Fig. 1. The BORRIS Flowchart.

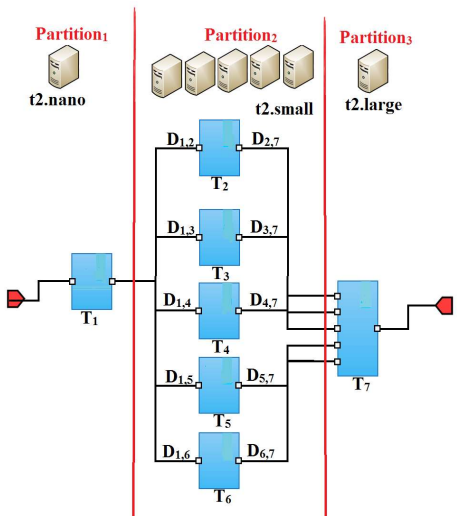

Fig. 2. Workflow example with seven tasks and ten data dependencies.

TABLE I. A shows a list of cloud resources parameters including five resource types with their computation capacities and the associated costs.

In the first step, the resource type, "t2.nano" is computed for the first partition as it is the slowest resource that can meet the partition sub-deadline, 15. By assigning the first partition to "t2.nano" and calculating its actual completion time, earliness value is 0 and lateness value is 5. These earliness and lateness values are passed to the next partition to update the sub-deadline of the second partition. After this sub-deadline adjustment for partition 2, "t2.small" is selected as the slowest resource type for this partition. The earliness and lateness values of the second partition is calculated after execution the entire partition as earliness = 4 and lateness = 0. In the end, "t2.large" can be selected for the last partition as it is the slowest resource that meets the sub-deadline. Finally, the total completion cost of workflow execution which is minimized is $0.113. The earliness and lateness values are shown in TABLE I. E.

BORRIS assigns the workflow tasks onto the appropriate cloud resource such that it minimizes the workflow execution cost while meeting the deadline constraints. The BORRIS algorithm is presented as Algorithm 1. Workflow specification and deadline are the two required inputs. The output is a set of pairs (<task, resource>) for all the tasks which indicates the cloud resource instances for executing of all the workflow tasks. In the first step, BORRIS parses the given workflow in order to generate the weighted DAG (line 4). Then the workflow is partitioned into several partitions (line 5). In line 6, the critical path of the workflow is calculated. Then, the total completion time of the workflow is calculated based on the completion time of the tasks in the critical path (lines 7-10). To identify the appropriate recourse for each task the algorithm evaluates all the partitions sequentially (lines 11-33). In lines 12-13, an initial sub-deadline is assigned to the partition. In addition, the minimum and maximum completion times of the partition are calculated (lines 14-15). In lines 16-18, we calculate the sub-deadline for the second partition. If it is not the last partition (line 19), BORRIS then calculates the threshold (line 20) and the slowest resource type for all the tasks in the partition. It then adds this schedule to the output schedule map (lines 21-22). In line 23, BORRIS computes the maximum of actual completion time (ACT) of the partition tasks. In lines 24-27, BORRIS calculates the lateness and earliness values of the partition to update the sub-deadline of the next subsequent partition. In lines 28-33, if it is the last partition then BORRIS updates the sub-deadline of the last partition (line 29). It calculates the slowest resource type for it and assigns all of the tasks in the last partition to different resource instances of this resource type. In the end, the schedule of the last partition is added to the output schedule map (line 30). Finally, in line 33, BORRIS returns the complete schedule that consists of all the tasks and the corresponding resources as a set of pairs (<task, resource>).

TABLE I. A) CLOUD RESOURCE CATALOGUE, B) TASK COMPUTATION COST C) DATA COMMUNICATION COST, D) INITIAL BUDGET ALLOCATION AND E) FINAL BUDGET ALLOCATION.

Resource Type	Instructions per min	Data Movement (MB/Min)	Cost per hour
t2.nano	500	17	0.0064
t2.micro	1000	20	0.013
t2.small	1500	25	0.026
t2.meduim	2000	34	0.052
t2.large	2500	50	0.104

(a)

Task	# of ins (Million)	F_{ft} (min)
T_1	1	5.16
T_2	2	12.43
T_3	3	16.40
T_4	4	19.87
T_5	5	24.00
T_6	6	32.87
T_7	7	45.21

(b)

Partition	Sub Deadline (mins)	minCT	maxCT	Threshold
P_1	15	7	32	18
P_2	85	40	200	15
P_3	100	45	230	0

(d)

Data Dependency	Data Size (MB)	F_{ft} (min)
$D_{1,2}$	100	2.85
$D_{1,3}$	200	5.70
$D_{1,4}$	300	8.55
$D_{1,5}$	400	11.40
$D_{1,6}$	500	14.25
$D_{2,7}$	150	40.50
$D_{3,7}$	250	7.50
$D_{4,7}$	350	15.60
$D_{5,7}$	450	12.70
$D_{6,7}$	550	15.60

(c)

Partition	ACT	Earliness	Lateness	Sub Deadline	ACC
P_1	20	0	5	15	0.035
P_2	76	4	0	80	0.033
P_3	103	2	0	105	0.045

(e)

1: **Algorithm 1** BORRIS Scheduler
2: **input**: workflow *w,* deadline *D*
3: **output**: *d*, a map storing task-VM assignments.
4: parse *w* and generate a weighted DAG (*w*).
5: *tasksByPartition* ← partition workflow.
6: *CTL* ← get all critical tasks in the workflow w
7: *CTM* = 0 // Critical Task Makespan
8: **for each** $crt_i \in CTL$
9: $CTM = CTM + \overline{CT}(crt_i)$
10: **end for**
11: **for each** Partition $P_j \in tasksByPartition$
12: $PMax = Max_{T_k \in P_j}\{\overline{CT}(T_k)\}$
13: $SD[P_j] = (PMax / CTM) * D$
14: $minCT\ [P_j] = \sum_{T_k \in P_j} CC(T_k, R_{fastest})$
15: $maxCT\ [P_j] = \sum_{T_k \in P_j} CC(T_k, R_{slowest})$
16: **if** (P_j is first partition) **then**
17: $SD[P_{j+1}] = (\mathrm{Max}_{T_k \in P_{j+1}}\{\overline{CT}(T_k)\} / CTM) * D$
18: **end if**
19: **if** (P_j is not last partition) **then**
20: $Thres\ [P_j] = \mathrm{Max}\{0, SD\ [P_{j+1}] - minCT\ [P_{j+1}]\}$
21: $RT\ [P_j] = PRT\ (P_j, R, SD\ [P_j] + Thres\ [P_j])$
22: $d \leftarrow d \cup MAP\ (T_k, F_S(RT\ [P_j])) \forall\ T_k \in P_j$
23: $ACT\ [P_j] = CT(P_j, F_S(RT\ [P_j]))$
24: $Lateness\ [P_j] = \mathrm{Max}\ \{0, ACT\ [P_j] - SD[P_j]\}$
25: $SD[P_{j+1}] = SD[P_{j+1}] - Lateness\ [P_j]$
26: $Earliness\ [P_j] = \mathrm{Max}\ \{0, SD[P_j] - ACT\ [P_j]\}$
27: $SD[P_{j+1}] = SD[P_{j+1}] + Earliness\ [P_j]$
28: **else if** (P_j is last Partition) **then**
29: $SD\ [P_j] = D - \sum_{j1=1}^{j-1}(ACT[P_{j1}] + Lateness[P_{j1}])$
30: $RT\ [P_j] = PRT\ (P_j, R, SD\ [P_j])$
31: $d \leftarrow d \cup MAP\ (T_k, F_S(RT[P_j])) \forall\ T_k \in P_j$
32: **end if**
33: **end for**
34: **return d**
35: **end function**

IV. EXPERIMENTAL RESULTS

A. Performance Evaluation

In order to evaluate the performance of BORRIS, we developed a big data workflow for the automotive domain

DATAVIEW platform [13]. This workflow is an auto analytics workflow based on the OpenXC datasets [17]. OpenXC is an open source platform that can capture automatically various instruments of a car and sensors readings in real time. The OpenXC dataset is growing exponentially in terms of data size with an average growth in excess of 14 Eb [13]. OpenXC data analysis is very useful for different stock holders like automotive insurance companies to analyze how their customers drive by capturing the large OpenXC datasets received from their registered vehicles. As the OpenXC datasets are large, it is beneficial to analyze the data using cloud distributed computing resources. As a result, there is a need to minimize the execution cost for performing the analytics. BORRIS automatically learns the complexity of the tasks computation and the data transfer between the tasks from an initial estimate and it can be more accurate after each workflow run.

Here we used Amazon EC2 cloud computing environment to perform our experiments. Amazon EC2 provides a framework that can provision and deprovision a variety of heterogeneous virtual machines (instances) with different compute, memory, storage and network capabilities. Each type of instance consists of an hourly cost for resource utilization and the execution time is based on the complexity level of the analytics workload. For example, the general purpose instance types are listed as:

{"t2.nano", "t2.micro","t2.small","t2.medium","t2.large"}. The performance of the data analytics for a given workloads cheapest option and resources of type "t2.large" is the fastest and the most expensive option in terms of cost.

We compared the BORRIS algorithm with two more approaches. The first one is the Workflow Responsive resource Provisioning and Scheduling (WRPS) algorithm [7]. The WRPS algorithm is the most recent work in the field of workflow scheduling. WRPS computes a set of bags of tasks (BoT) such that the tasks inside of each BoT are independent and can be executed in parallel. Then, it assigns a sub-deadline to each bag of tasks based on the given deadline and then schedules them onto heterogeneous types of cloud resources with the goal of workflow execution cost minimization and deadline constraints. The cost optimization problem is modeled as an unbounded knapsack minimization problem in that work.

In WRPS, the authors assumed the tasks inside each BoT are homogeneous. We do not have this limitation and the tasks inside each level can be heterogeneous. However, in order to compare our strategy to WRPS we developed our OpenXC workflow such that the tasks of each level are homogenous.

One of our main contributions is the application of a sub-deadline adjustment technique that updates the assign sub-deadline of the levels after completing each level. To demonstrate this technique, we then relaxed BORRIS (called BORRIS*) by setting the threshold, the earliness and the lateness to be zero. The WRPS algorithm provides an optimization to the BoT by scheduling the tasks in a bag to different types of machines but it does not update the sub-deadlines of the other BoT based on the executed BoT. In our strategy, BORRIS assigns all the tasks inside a given level to the same type of machines. However, it has the capabilities of adjusting the sub-deadlines of the remaining levels after execution of the current level.

B. *Results and Analysis*

BORRIS was evaluated against the other two approaches using 10 distinctive workflows that were developed in the OpenXC domain with different levels of complexity and with different provided deadlines. In TABLE II, we presented all the 10 workflows with their complexity levels like the computation and data intensity of all the tasks in each of the workflow and the user defined deadline. We did the experiments by varying the types of machines and presented the results for both makespan and cost parameters.

In Fig. 3. b, we show that BORRIS outperforms WRPS by roughly 4-11% margin as the complexity of the workflow increases from w3 to w10. For the workflows between *w1* and *w3*, which is of least complexity, WRPS outperforms BORRIS because WRPS assigns tasks in each bag onto resources of different types. The local optimization done at each level outperforms the global optimization performed by BORRIS when the complexity level is low.

We evaluated the results of all three approaches and have demonstrated the cost minimization by varying the instance types from $K = \{5, 10, 15, 20, 25\}$. Please notice in these experiments we provided sufficient deadlines to execute each workflow as there are some cases that the provided deadlines are not enough to complete the workflow. In Fig. 3. a, we show the resource utilization in the cloud for various K values. The BORRIS algorithm outperforms WRPS because the resource is utilized to the maximum extent for the tasks in each level since we setup the level dependencies through a system driven threshold value and automatically update the sub-deadline with a system driven earliness or lateness value at run time. The earliness and lateness are calculated after the actual execution time of the previous level. By increasing the number of resource types (K) we can observe BORRIS has better performance compared to the other algorithms.

V. RELATED WORK

Big data workflows are resource-intensive applications as they naturally consist of a large number of tasks and produce massive datasets. The efficient workflow scheduling strategies can have significant impact on workflow performance. There has been extensive research on the workflow scheduling problem in the distributed computing community. These studies

TABLE II. WORKLOAD DETAILS FOR OPENXC WORKFLOW.

Workflow	No of drivers	Computation Size (MLOC)	Data Size (GB)	Deadline (Hours)
w_1	5	1	1	4
w_2	10	2	2	7
w_3	15	3	3	9
w_4	20	4	4	10
w_5	25	5	5	11
w_6	30	6	6	13
w_7	35	7	7	15
w_8	40	8	8	17
w_9	45	9	9	19
w_{10}	50	10	10	21

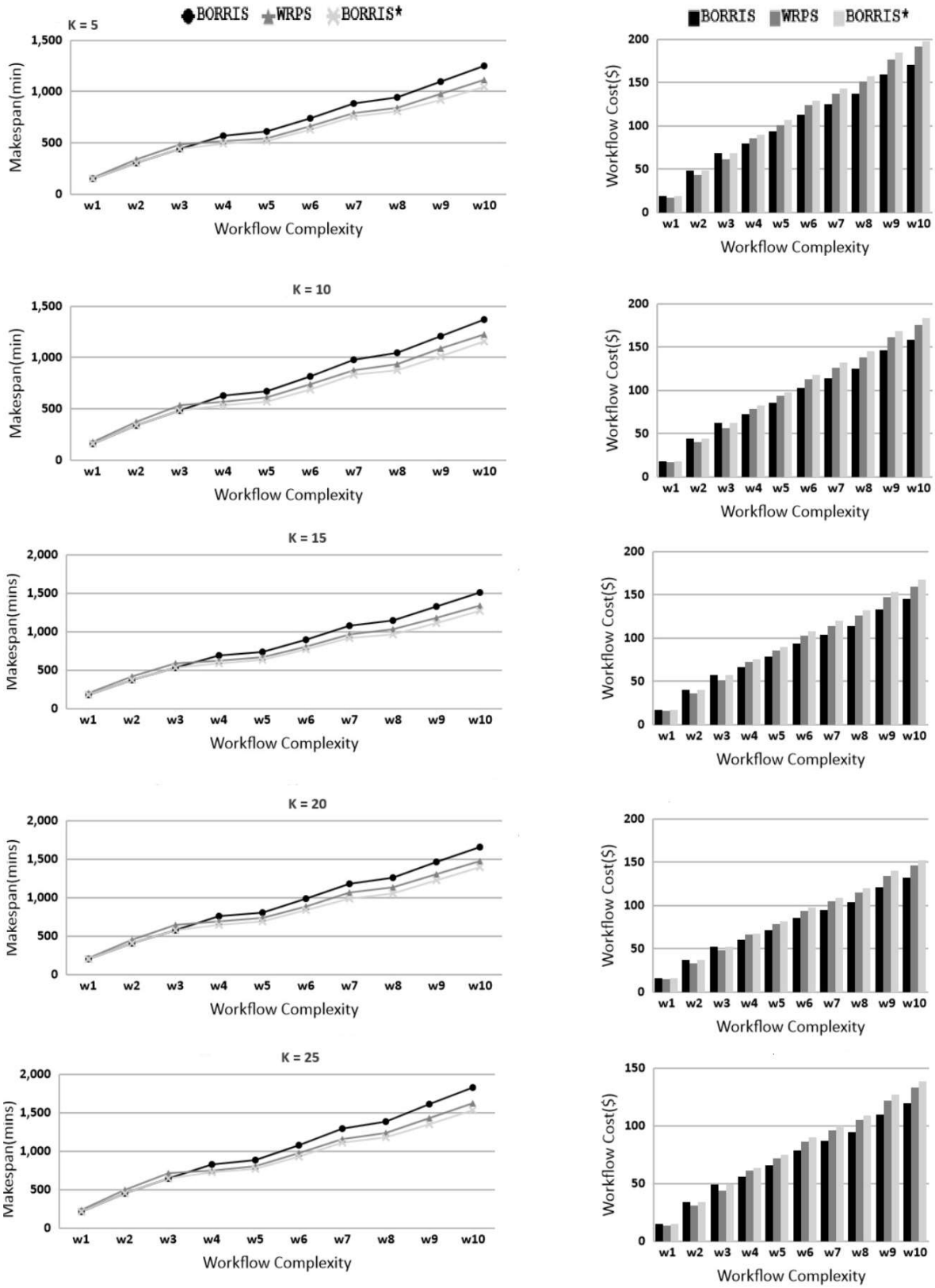

Fig. 3. a) Resource utilization, b) Execution cost minimization.

problem based on the various QoS requirements. One of the most recent work is [7] in which the authors proposed a workflow scheduler that minimizes the execution cost while meeting a specified deadline. In their approach, they apply unbounded knapsack problem (UKP) to find an optimal schedule for bags of homogenous tasks. Although they are able to schedule a workflow into different cloud resources types efficiently they did not consider heterogeneous tasks. In addition, they did not use any run time sub-deadline adjustments. In [2, 3, 4] some other scheduling algorithms were proposed to minimize the execution cost with deadline constraints for the Grid utility systems. In [5, 8], the authors considered both budget and makespan as the QoS constraints, but did not use an objective function to minimize them.

Lin et al. [1, 6] proposed an elastic scheduling algorithm to schedule the workflow dynamically in the cloud with the goal of makespan minimization. However, they do not consider any QoS constraints. In list-based workflow scheduling algorithms [9, 10], the workflow tasks are ranked and sorted based on their start times and execution times and then the tasks are executed sequentially. In clustering-based approaches [11, 12], tasks are first clustered in terms of maximum execution time or size of data movement. Then assign them on to possibly the same resource to minimize the data movement based upon these clusters.

In our previous works [15, 16], we proposed data and task placement strategies for optimal workflow data and task placement in the cloud by considering the data and task interdependencies to cluster the most dependent data and tasks together. These clusters were used to assign onto the same resource in order to minimize time taken for data movement. The limitation of our previous strategies is that we did not consider any QoS constraints. In [18], we proposed a novel scheduling algorithm for executing big data workflows in the Cloud. The goal of that work was to minimize the workflow makespan for a user-specified budget. However, in this paper, we propose a new workflow scheduling algorithm with the goal of minimizing the workflow execution cost while meeting the specified deadline.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we propose a novel Big data wOrkflow scheduleR undeR deadlIne conStraint called BORRIS algorithm. Our goal was to minimize the workflow execution cost while meeting the user-specified deadlines. BORRIS is a partition-based scheduling algorithm that supports high-performance workflow scheduling in a heterogeneous cloud computing environment. We compared our strategy with the relaxed version of our approach and one of the most noted works in this area, we expressed these three strategies in our big data workflow system, DATAVIEW, in order to do a comprehensive comparison. The results of the comparison illustrate the performance advantages of the approach. In the future, we will compare our strategy with more existing workflow scheduling algorithms. In addition, we plan to improve the performance of our strategy by ranking the partitions and assigning them to various resource types.

ACKNOWLEDGMENT

This work is supported by U.S. National Science Foundation under ACI-1443069 and is based upon work supported in part by the National Science Foundation under Grant No. 0910812.

REFERENCES

[1] Cui Lin and Shiyong Lu, “Scheduling Scientific Workflows Elastically for Cloud Computing,” in Proc. of the IEEE International Conference on Cloud Computing (CLOUD), pp: 746-747, 2011.

[2] J. Yu, et al., “Cost-Based scheduling of scientific workflow application on utility grids,” in Proc. of the First International Conference on e-Science and Grid Computing, pp: 8-pp, 2005.

[3] S. Abrishami, et al., “Cost-driven scheduling of grid workflows using partial critical paths,” in Proc. of the 11th IEEE/ACM International Conference on Grid Computing, pp: 1400-1414 , 2010.

[4] M. Wieczorek, et al., “Towards a general model of the multi-criteria workflow scheduling on the grid,” in Proc. of the Future Generation Computer Systems, vol. 25, no. 3, pp: 237–256, 2009.

[5] M. Malawski, et al., “Cost and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds,” in Proc. of the International Conference on High Performance Computing, Networking, Storage and Analysis, p: 1-11, 2012.

[6] C. Lin and S. Lu, “SCPOR: An elastic workflow scheduling algorithm for services computing,” in Proc. of the International Conference on Service Oriented Computing and Applications (SOCA), pp: 1-8, 2011.

[7] M.A. Rodriguez, R. Buyya, “A responsive Knapsack-based algorithm for resource provisioning and scheduling of scientific workflows in clouds,” in Proc. of the 44th International Conference on Parallel Processing, ICPP, pp: 839-848, 2015.

[8] J. Yu , R. Buyya, “Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms,” in Proc. of the Scientific Programming, v.14 n.3,4, pp: 217-230, 2006.

[9] H. Arabnejad and J. G. Barbosa, "List Scheduling Algorithm for Heterogeneous Systems by an Optimistic Cost Table," in Proc. of the IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 3, pp: 682-694, 2014.

[10] H. Topcuoglu, et al., "Performance-effective and low-complexity task scheduling for heterogeneous computing," in Proc. of the IEEE Transactions on Parallel and Distributed Systems, vol. 13, no. 3, pp: 260-274, 2002.

[11] K. Bochenina, et al. "A clustering-based approach to static scheduling of multiple workflows with soft deadlines in heterogeneous distributed systems," in Proc. of the Procedia Computer Science 51, pp: 2827-2831, 2015.

[12] Deldari, Arash, et al. "A Clustering Approach to Scientific Workflow Scheduling on the Cloud with Deadline and Cost Constraints," in Proc. of the Amirkabir International Journal of Modeling, Identification, Simulation & Control 46.1, pp:19-29, 2014.

[13] A. Kashlev, et al., "A System Architecture for Running Big Data Workflows in the Cloud," in Proc. of the 2014 IEEE International Conference on Services Computing (SCC), pp: 51-58, 2014.

[14] A. Mohan, et al., "A NoSQL Data Model For Scalable Big Data Workflow Execution," in Proc. of the International IEEE Congress on Big Data (BigDataCongress), pp:52-59, 2016.

[15] M. Ebrahimi, et al., "TPS: A task placement strategy for big data workflows," in Proc. of the International IEEE Conference on Big Data, 2015, IEEE, pp: 523-530, 2015.

[16] M. Ebrahimi, et al., "BDAP: A Big Data Placement Strategy for Cloud-Based Scientific Workflows," in Proc. of the Big Data Computing Service and Applications (BigDataService), 2015 IEEE First International Conference on, IEEE, pp: 105-114, 2015.

[17] The OpenXC Platform, http://openxcplatform.com

[18] A. Mohan, et al., "Scheduling big data workflows in the cloud under budget constraints," in Proc. of the IEEE International Conference on Big Data (Big Data), IEEE, pp: 2775-2784, 2016.