Scheduling Big Data Workflows in the Cloud under Budget Constraints

Aravind Mohan, Mahdi Ebrahimi, Shiyong Lu, Alexander Kotov
Wayne State University
Detroit, MI, USA
{amohan, mebrahimi, shiyong, kotov}@wayne.edu

Abstract— **Big data is fast becoming a ubiquitous term in both academia and industry and there is a strong need for new data-centric workflow tools and techniques to process and analyze large-scale complex datasets that are growing exponentially. On the other hand, the unbound resource leasing capability foreseen in the cloud facilitates data scientists to wring actionable insights from the data in a time and cost efficient manner. In the data-centric workflow environment, scheduling data processing tasks onto appropriate resources are often driven by the constraints provided by the users. Enforcing a constraint while executing the workflow in the cloud adds a new optimization challenge on how to meet the objective while satisfying the given constraint. In this paper, we propose a new *Big dAta woRkflow schEduler uNder budgeT conStraint* known as *BARENTS* that supports high-performance workflow scheduling in a heterogeneous cloud computing environment with a single objective to minimize the workflow makespan under a provided budget constraint. Our case study and experiments show the competitive advantages of our proposed scheduler. The proposed *BARENTS* scheduler is implemented in a new release of *DATAVIEW*, one of the most usable big data workflow systems in the community.**

Keywords-Scheduler, BARENTS, Big Data

I. INTRODUCTION

Big data workflows [13] have recently emerged as a data-centric workflow approach to analyze the data that is ever increasing in scale, complexity, and rate of acquisition. Data scientists develop workflows by modeling their complex scientific applications as a set of data processing tasks with a set of data dependencies between the tasks. Although this approach facilitates the execution of tasks in a distributed cloud computing environment [17,18,19], it also adds a research challenge of how and where to schedule the tasks in a distributed and heterogeneous cloud environment in a usable manner [14]. The decision of how and where to schedule the tasks in a workflow is driven by the Quality of Service (QoS) requirements defined by data scientists such as the budget or the deadline for the workflow execution. The goal of the scheduling problem is to minimize the total cost for executing the workflow or makespan (i.e. total execution time of the workflow), while still meeting the QoS requirement defined by data scientists.

Cloud service providers such as Amazon EC2 offer a scalable infrastructure that allows an unlimited number of virtual machines that can be provisioned for an amount of time proportional to the cost for usage. There are different types of instances that range from less powerful to more powerful in terms of their CPU, memory, storage and networking capacity. The cost per machine usage is billed on an hourly rate with the cheapest resource offering the least performance to the most expensive resource offering the highest performance. The Amazon EC2 cloud service provider provides an easily accessible application programming interface through which the cloud resource manager of the big data workflow system can provision and deprovision resources. The primary responsibility of the workflow engine in a big data workflow system is to orchestrate the execution of the workflow by 1) identifying a type of cloud resource that is appropriate for each set of tasks in the workflow based on the QoS requirement; 2) provisioning and deprovisioning a set of resources that are of the identified resource types; 3) scheduling the tasks to the appropriate cloud resources and initiate the distributed execution process. The scheduling problem is non-trivial. In fact, it is a well known NP-complete problem [1].

In this paper, we propose a new *Big dAta woRkflow schEduler uNder budgeT conStraint* known as *BARENTS* that supports high-performance workflow scheduling in a heterogeneous cloud computing environment with a single objective to minimize the workflow makespan under a provided budget constraint. Our case study and experiments show the competitive advantages of our proposed scheduler. The proposed *BARENTS* scheduler is implemented in a new release of *DATAVIEW*, one of the most usable big data workflow systems in the community.

The rest of the paper is organized as follows. Section II provides a background on our system model. Section III introduces our BARENTS scheduling algorithm with an overview and presents the pseudo code with an example. Section IV presents our experimental results. Finally, Sections V and VI present related work and conclusions.

II. MODEL

A cloud computing environment provides a framework for enabling ubiquitous, on-demand access to a shared pool of resources of heterogeneous types which can be provisioned and deprovisioned with a minimal management effort. The computation tasks and their data counterpart are moved to the resources in the cloud, in order to perform the actual execution of the tasks. Every individual resource is

978-1-4673-9005-7/16/$31.00 ©2016 IEEE

associated with a predetermined cost, computing speed function and data communication rate function. More formally a cloud computing environment is defined as:

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_C: R→ Q+ is the resource usage cost function. $R_C(R_i)$, $R_i \in R$ gives the cost in some dollar amount for the resource usage R_i in the cloud computing environment. The resource with the minimum R_C is called $R_{cheapest}$ and the resource with the maximum R_C is called $R_{expensive}$.
- F_B: $R \times R \rightarrow Q^+_0$ 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.
- 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 the computerized modeling and automation of a process consisting of a set of computational tasks and their data interdependencies to process and analyze a large amount of data. The big data workflow consists of a set of interconnected tasks and their data counterparts. Because one or more of the input datasets connected to the tasks fall under the umbrella of big data, there is a need for a distributed computing environment such as the cloud to process the tasks in an efficient manner. More formally, a big data workflow is defined 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}> | 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. $D_{k1,k2}$ represents an amount of data required to be transferred after T_{k1} completes and before T_{k2} starts. D_k represents all the output datasets from task T_k.
- F_T: $T \rightarrow Q^+_0$ is the execution cost function. $F_T(T_k)$; $T_k \in T$ gives the execution cost of a task T_k, measured in some pre-determined unit like million instructions per machine cycles or million instructions per nanoseconds.
- F_D: $D \rightarrow Q^+_0$ is the data size function. $F_D(D_{k1,k2})$, $D_{k1,k2} \in D$ gives the size of a dataset $D_{k1,k2}$, measured in some predetermined unit like bits or bytes. □

A big data workflow graph is a weighted directed acyclic graph that includes a set of vertices a.k.a. tasks and a set of edges a.k.a. data dependencies, which represent the output datasets that originate from one task and passed as an input to another task in the workflow. We divide the workflow graph into several partitions a.k.a. levels. Each partition in the workflow graph has a set of vertices and a set of outgoing edges that represent the data passed as input to the next partition. The weight of the vertices is calculated by the average task computation cost function and the weight of the edges is calculated by the average data communication cost function. More formally, a big data workflow graph is defined as:

Definition 2.3 (Big Data Workflow Graph G): Given a workflow W in a cloud environment C, a big data workflow graph G, represents a weighted directed acyclic graph with 14-tuple G(T, D, R, F_c, $F_{\bar{c}}$, F_p, $F_{\bar{p}}$, 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 environment.
- F_c: $D \times R \times R \rightarrow Q^+_0$ 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 $D_{k1,k2}$ from resource R_{i1} to resource R_{i2}.
- $F_{\bar{c}}$: $D \rightarrow Q^+_0$ is the average data communication cost function. $F_{\bar{c}}(D_{k1,k2})$, $D_{k1,k2} \in D$ gives the average data communication cost of $D_{k1,k2}$ for all the 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 $D_{k1,k2}$ from resource R_{i1} to resource R_{i2}.

- $F_{\overline{m}}$: $D \rightarrow Q^{+}_{0}$ is the average data communication time function. $F_{\overline{c}}(k_1, k_2)$, $D_{k1,k2} \in D$ 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. $F_n(T_k, R_i)$, $T_k \in T$, $R_i \in R$ gives the computation time of T_k on resource R_i.
- $F_{\overline{n}}$: $T \rightarrow Q^{+}$ is the average task computation time function, $F_{\overline{n}}(T_k)$ gives the average computation time of task T_k.
- P: $N \rightarrow T$ is the partition task function, P[j] or P_j gives all the tasks of partition j. R_{Pj} represents the set of resources assigned to the tasks in 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 P_j. □

While executing a workflow on a set of resources in the cloud, there is a cost and time incurred during the execution process. Makespan is the total time taken to complete the workflow execution. Each resource in the cloud has a type associated with it. In addition, there is also a cost associated with each type of resource per unit time. The execution of the workflow on a cheapest resource take more time than executing the workflow on an expensive resource. Because there is a tradeoff between performance of the workflow execution and the cost associated with the workflow execution, we compute the cost associated with the workflow execution at different level of granularity such as the minimum, the average and the maximum completion cost. In order to achieve our objective of minimizing the makespan, we calculate the time taken to complete the execution of each partition in the workflow. More formally, we define the workflow execution environment as:

Definition 2.4 (Workflow Execution Environment W_E): Given a workflow W in a cloud environment C, a workflow execution environment, represents the cost and time incurred during the execution of the workflow with 5-tuple W_E (CC, $\overline{CC}$, CT, minCC, maxCC), where

- CC: Partition $\times$ R $\rightarrow Q^{+}_{0}$ 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 as:

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

- $\overline{CC}$: Partition $\rightarrow Q^{+}_{0}$ is the average workflow partition completion cost function. $\overline{CC}(P_j)$, $P_j \in$ Partition gives the sum of average task computation cost of all the tasks $T_k \in P_j$ and the average data communication for all the outgoing edges from all the tasks $T_k \in P_j$. We formally define as:

$$\overline{CC}(P_j) = \sum_{k=1}^{K} F_{\overline{p}}(T_k) + \sum_{k=1,k1=1, k \neq k1}^{K} F_{\overline{c}}(k, k1)$$

- CT: Partition $\rightarrow Q^{+}_{0}$ is the workflow partition completion time function. $CT(P_j)$, $P_j \in$ Partition gives the maximum of average task computation time of all the tasks $T_k \in P_j$ and the maximum of average data communication time of all the outgoing edges from all the tasks $T_k \in P_j$. We formally define as:

$$CT(P_j) = Max_{T_k \in Pj}\{F_{\overline{n}}(T_k)\} + Max_{T_k \in Pj}\{F_{\overline{m}}(D_{k,k1})\}$$

- minCC: Partition $\rightarrow Q^{+}_{0}$ is the minimum workflow partition completion cost function. $minCC(P_j)$, $P_j \in$ Partition gives the sum of minimum task computation cost of all the tasks $T_k \in P_j$ and the minimum data communication for all the outgoing edges from all the tasks $T_k \in P_j$. We formally define minCC as:

$$minCC(P_j) = \sum_{T_k \in P_j} CC(T_k, R_{cheapest})$$

- maxCC: Partition $\rightarrow Q^{+}_{0}$ is the maximum workflow partition completion cost function. maxCC (P_j), $P_j \in$ Partition gives the sum of maximum task computation cost of all the tasks $T_k \in P_j$ and the maximum data communication for all the outgoing edges from all the tasks $T_k \in P_j$. We formally define maxCC as:

$$maxCC(P_j) = \sum_{T_k \in P_j} CC(T_k, R_{expensive})$$ □

At workflow run time, a user driven budget is allocated as a dollar amount for the entire workflow. We compute the sub-budget as a dollar amount for each partition in the workflow based on the user provided budget. The sub-budget is computed based on the average completion cost for each partition. The partition with the high computation intensive tasks and high data intensive outgoing edges has more sub-budget than the partition with the low computation intensive tasks and the low data intensive outgoing edges. In addition to the sub-budget, there is also a threshold provided to each partition as a dollar amount. The threshold for a partition is computed based on the sub-budget allocated to the next subsequent partition and the completion cost incurred for executing the partition by using the most expensive resource provided by the maximum completion cost. The threshold is set to be always greater than or equal to zero. The partition resource type (PRT) function identifies the most expensive resource type for each partition in a workflow while still meeting the budget constraint. The goal of PRT function is to minimize the completion time of the tasks in each partition. We minimize the workflow makespan by applying PRT for each partition in the workflow.

Definition 2.5 (Workflow Partition Cost P_C): Given a workflow W in a cloud computing environment C and a budget B, a workflow partition cost represents the budget allocated to each partition of the workflow and the actual cost incurred at each partition of the workflow with 6-tuple P_C(SB, Threshold, PRT, ACC, Credit, Debit), where

- SB: Partition $\rightarrow Q^+$ is the sub-budget function. SB(P_j), $P_j \in$ Partition gives the sub-budget assigned to the partition P_j and can be calculated formally as follows:

$$SB(P_j) = [\overline{CC}(P_j) / \sum_{j1=1}^{J} \overline{CC}(P_{j1})] * B$$

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

$$Threshold(P_j) = Max\ \{0, SB(P_{j+1}) - maxCC(P_{j+1})\}$$

- PRT: Partition $\times$ R $\times$ SB $\times$ Threshold $\rightarrow$ RT is the partition resource type function that is used to identify the most expensive resource type for each partition. PRT is based on the criteria that the total completion cost for all the tasks in the partition is less than equal to the sum of the sub-budget and the threshold assigned to the partition.
- ACC: Partition $\rightarrow Q^+$ is the actual completion cost that is used to compute the total cost for completing all the tasks in a partition, that are assigned to the resources of a particular resource type. Supposedly, all the tasks in partition j are assigned to $R_T[j]$ then ACC can be formally calculated as:

$$ACC[P_j] = \sum CC(P_j, F_S(R_T[P_j]))$$

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

$$Credit[P_j] = Max\{0, SB(P_j) - ACC[P_j]\}$$

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

$$Debit[P_j] = Max\{0, ACC[P_j] - SB(P_j)\}$$ □

Our objective is to minimize workflow makespan while still satisfying the budget constraint. We formally define our objective function and constraints as follows:

Definition 2.6 (Workflow Makespan Minimization W_M): Given a workflow W in a cloud environment C, and budget B, a workflow makespan minimization represents the objective function to minimize makespan under the given budget constraint.

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

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

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

There can be cases as follows:

1) if $B < \sum_{j=1}^{J} minCC(P_j)$, then we cannot satisfy the budget constraint and hence we assign all the partitions to the cheapest resource.

2) if $B > \sum_{j=1}^{J} maxCC(P_j)$, then we can satisfy the budget constraint and hence we assign all the partitions to the most expensive resource.

3) if $\sum_{j=1}^{J} minCC(P_j) <= B <= \sum_{j=1}^{J} maxCC(P_j)$, then we use our strategy to find the optimal solution. □

III. THE BARENTS SCHEDULER

A. BARENTS Overview

Our BARENTS scheduler parses the specification of the workflow and generates a weighted directed acyclic graph with each node representing the tasks and each edge representing the data dependency between the tasks in the workflow. We estimate the number of instructions that exists in each task and the size of the data dependencies that exists between each task for all the workflows in our repository. In addition, we also estimate the number of instructions that can be executed per unit time and the size of the data that can be transferred per unit time for different cloud resource types. The estimates are adjusted automatically during every workflow run to achieve accuracy. The weight of the nodes and the edges in the graph are calculated by using the average computation cost and average data movement cost, respectively based on our estimation. We partition the workflow into different partitions in a topological manner. We validate that the tasks in each partition do not have any data dependency between them and at the same time there is at least one or more data dependency between the tasks in different partitions. After creating the partitions, we assign an initial sub-budget to each partition based on the user defined budget dollar amount provided for the workflow. Our Cloud Resource Manager (CRM) [13, 14] is used to manage the cloud resources by maintaining a catalogue that provides a list of resource types and their associated cost incurred for an hourly rate. We compute the minimum completion cost for each partition, which is the cost incurred for executing all the tasks in a partition. This cost includes both the computation and movement of all the data dependencies using the cheapest resource provided in the catalogue. We compute the maximum completion cost for each partition, which is the cost incurred for executing all the tasks in a partition. This cost includes both the computation and

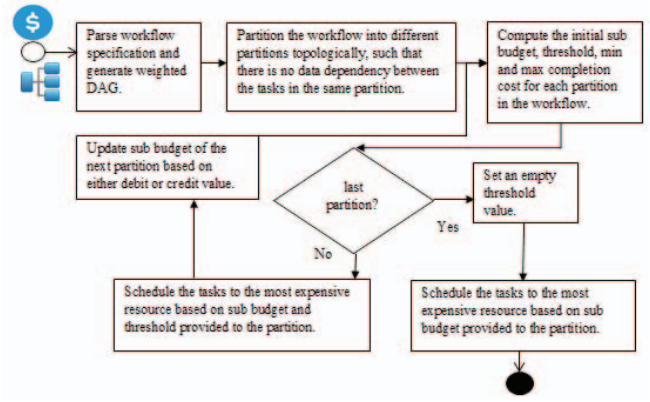

Figure 3.1 The BARENTS flowchart

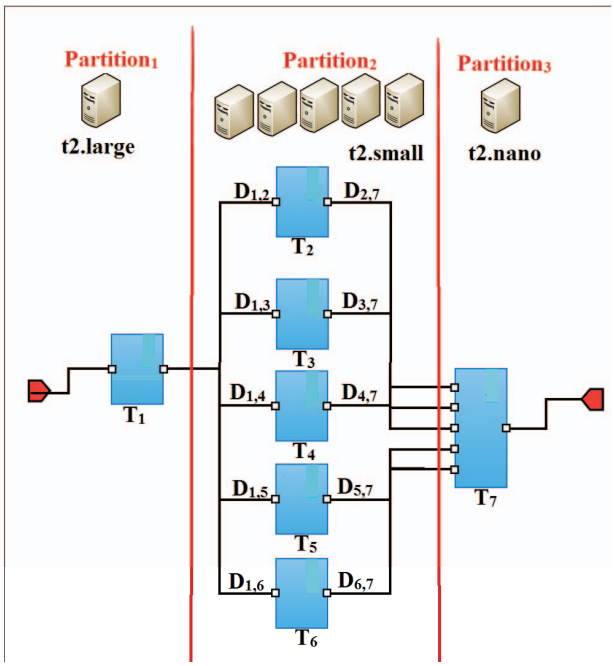

Figure 3.2 An example Workflow w

movement of all the data dependencies using the most expensive resource provided in the catalogue.

In addition, we compute a threshold value for each partition that is used as a triggering factor by exploiting the dependency between the partitions. The threshold value for each partition provides more processing power for executing the tasks in each partition by borrowing some budget from the next subsequent partition. By doing so, we are able to schedule the tasks in each partition to the most expensive resource listed in the catalogue within the range of the combined sub-budget and threshold value assigned to the partition and thereby minimizing the makespan of the partition. After identifying the appropriate resource type, each task in the partition is scheduled to execute in a different set of resources that are of the same resource type. The actual completion cost for each partition is calculated and we compute the credit or debit value based on the actual completion cost and the initial sub-budget provided to the

Table 3.1 a) Cloud resource catalogue, b) Task computation cost c) Data communication cost, d) Initial budget allocation and e) Final budget

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 (Millions)	$F_{\bar{P}}$ ($)
T_1	1	0.3698
T_2	2	0.7395
T_3	3	1.1093
T_4	4	1.4791
T_5	5	1.8488
T_6	6	2.2187
T_7	7	2.5885

(b)

Partition	Sub budget	minCC	maxCC	Threshold
P_1	0.3825	0.2263	0.7453	2.7878
P_2	7.1323	4.3445	13.9273	0.9686
P_3	2.4852	1.5167	4.8533	0

(d)

Data Dependency	Data Size (MB)	$F_{\bar{c}}$ ($)
$D_{1,2}$	100	0.0019
$D_{1,3}$	200	0.0038
$D_{1,4}$	300	0.0057
$D_{1,5}$	400	0.0076
$D_{1,6}$	500	0.0095
$D_{2,7}$	150	0.0027
$D_{3,7}$	250	0.0047
$D_{4,7}$	350	0.0104
$D_{5,7}$	450	0.0085
$D_{6,7}$	550	0.0104

(c)

Partition	ACC	Credit	Debit	Sub budget	ACT
P_1	0.7453	0	0.3628	0.3825	410
P_2	5.8080	0.9615	0.0	6.7694	4022
P_3	2.0224	0.8257	0.0	2.8481	4667

(e)

partition. We adjust the sub-budget of the next partition by using the credit or debit value that was calculated for the current partition. The recalculation and adjustment of the sub-budget is done for every partition in the workflow except for the last partition. Because the last partition does not have any subsequent partition to borrow the budget from, the threshold value, the credit value and the debit value is set to be 0 for the last partition. We present the flowchart of our BARENTS scheduler in Figure 3.1.

B. The BARENTS Algorithm

In Table 3.1 a, we show the resource catalogue maintained by the CRM. For example, we show the 5 resource types with the corresponding number of instructions (in million lines of code) executed per minute, the data movement size (in mega bytes) per minute and the cost in dollar amount associated with the provisioning of the resource per hour. In Figure 3.2, we show an example workflow that consists of seven tasks and ten data dependencies. First, the BARENTS scheduler parses the specification of the workflow and generates the weighted DAG shown in Table 3.1 b, c. Second, the workflow is partitioned in a topological manner with: $P_1=\{T_1, D_{1,2}, D_{1,3}, D_{1,4}, D_{1,5}, D_{1,6}\}$, $P_2=\{T_2\text{-}T_6, D_{2,7}, D_{3,7}, D_{4,7}, D_{5,7}, D_{6,7}\}$ and $P_3=\{T_7\}$. Next, as presented in Table 3.1 d, we calculate the initial sub-budget, the minimum completion cost, the maximum completion cost and the threshold value for each partition of the workflow. We compute the resource type for partition 1 by finding the most expensive resource, that is "*t2.large*", with a debit of $0.3628 and credit of $0.0. We update the sub-budget of partition 2. We compute the resource type for partition 2 by finding the most expensive resource, that is "*t2.small*", with a debit of 0.0$ and credit of $0.9615. We update the sub-budget of partition 3. We compute the resource type for partition 3 by finding the most expensive resource, which is "*t2.nano*", with a debit of $0.0 and credit of $0.8257. Thereby, we minimized the workflow makespan to 9099 minutes.

1: **Algorithm 1** BARENTS Scheduler
2: **input**: workflow *w,* budget *B*
3: **output**: *d*, a map storing task-VM assignments.
4: parse w and generate a weighted DAG (w).
5: *tasksByPartition* ← partition workflow topologically.
6: TCC = $\sum_{j=1}^{J} \sum_{T_k \in P_j} \overline{CC}(T_k, R)$
7: **for each** partition $P_j \in$ *tasksByPartition*
8: *SB* [P_j] = $\sum_{T_k \in P_j} \overline{CC}(T_k, R)$ / TCC * B
9: **if** (P_j is first partition) **then**
10: *SB* [P_{j+1}] = $\sum_{T_k \in P_{j+1}} \overline{CC}(T_k, R)$ / TCC * B
11: **end if**
12: **if** (P_j is not last partition) **then**
13: *maxCC* [P_{j+1}] = $\sum_{T_k \in P_{j+1}} CC(T_k, R_{expensive})$
14: *Thres* [P_j] = Max {0, *SB* [P_{j+1}] – *maxCC* [P_{j+1}]}
15: RT [P_j] = PRT (P_j, R, *SB* [P_j] + *Thres* [P_j])
16: d ← d ∪ MAP (T_k, F_S(RT [P_j])) ∀ $T_k \in P_j$
17: ACC [P_j] = $CC(P_j, F_S(RT[P_j]))$
18: DEBIT [P_j] = Max{0, ACC [P_j] – SB [P_j]}
19: SB [P_{j+1}] = SB [P_{j+1}] – DEBIT [P_j]
20: CREDIT [P_j] = Max{0, SB[P_j] – ACC [P_j]}
21: SB [P_{j+1}] = SB [P_{j+1}] + CREDIT [P_j]
22: **else if** (P_j is last partition) **then**
23: SB [P_j] = B - $\sum_{j1=1}^{J-1}$ ACC [P_{j1}]
24: RT [P_j] = PRT (P, R, SB [P_j])
25: d ← d ∪ MAP (T_k, F_S(RT [P_j])) ∀ $T_k \in P_j$
26: **end if**
27: **end for**
28: **return d**
29: **end function**

We present the pseudo code of our BARENTS scheduler in Algorithm 1. The inputs of the algorithm are the specification of the workflow and a user defined budget dollar amount. The output of the algorithm is the map that consists of all the tasks and the corresponding resources where the tasks are scheduled to execute. In line 4, we parse the specification of the input workflow w and generate a weighted DAG. In line 5, we partition the workflow topologically and generate different partitions. In line 6, we calculate the total completion cost by adding the average completion cost of all the tasks in the workflow w. In lines 7-27, we loop through each partition in the workflow w to identify the appropriate resource for each task in the partition. In line 8, we calculate the sub-budget for the current partition. In lines 9-11, we calculate the sub-budget for the second partition. In lines 12-21, we validate if the current partition is not the last one in the workflow. In line 13, we calculate the maximum completion cost for the next subsequent partition. In line 14, we calculate the threshold for the current partition. In line 15, we calculate the most expensive resource type by calling the partition resource type (PRT) function. In line 16, we assign all the tasks in the partition to different resources of the resource type computed in line 15. The schedule is then added to the output schedule map. In line 17, we compute the actual completion cost for executing all the tasks using the resources assigned to them. In line 18, we calculate the debit value and in line 20 we calculate the credit value. In lines 19 and 21, we recalculate the sub-budget and make the necessary adjustment based on the credit or debit value. In lines 22-25, we validate whether the current partition is the last partition of the workflow. In line 23, we calculate the sub-budget of the last partition by subtracting the sum of the actual costs of all the previous partitions from the given budget B. In line 24, we calculate the most expensive resource type by calling the partition resource type (PRT) function. In line 25, we assign all the tasks in the last partition to different resources of the resource type computed in line 24. The schedule is then added to the output schedule map. Finally, in line 28, we output the final schedule that consists of all the tasks and the corresponding resources where the tasks are scheduled to execute and exit the code.

IV. Experimental Discussion

A. Performance Evaluation

In our DATAVIEW system, we implemented a big data workflow that is from automotive domain. We evaluated the BARENTS scheduler using the OpenXC Autoanalytics workflow [14]. OpenXC is an open source platform that produces 19 signal oriented data trace files from the vehicle automatically at periodic intervals. As mentioned in [13, 14], there is an exponential growth in the data size as the number of miles a vehicle is driven increases with the time for each driver. On average, every year the growth of the data exceeds 14 Eb. OpenXC data analysis is extremely useful for the automotive insurance companies to analyze the driving behavior of their customers by analyzing the large datasets generated from their registered vehicles. Since the analytics is performed using the cloud resources, there is a cost associated based on the computation and data intensity of the tasks in the analytics workflow. There is often a need to minimize the execution time that is taken to perform the analytics based on the budget provided for the analytics by the user. The BARENTS scheduler automatically learns the complexity of the tasks computation and the data transfer between the tasks from an initial estimate. During every workflow run, the accuracy of the complexity level estimates is improved by automatic adjustment of the actual execution measurements.

We performed our experiments in the Amazon EC2 cloud computing environment, which provides a framework to provision and deprovision virtual machines (instances) that are of heterogeneous instance types. The Amazon EC2 cloud environment offers a total of 39 different instance types that are of varied CPU, memory, storage and networking capacity. Each type of instances 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 instance types in the general purpose T2 category are listed as*: {"t2.nano", "t2.micro", "t2.small", "t2.medium", "t2.large"}*. The performance of the analytics using the resources of type "*t2.nano*" for a given workload is the slowest and the cheapest option in terms of cost. On the other hand, for the same workload the performance using the resources of type "*t2.large*" is the fastest and the most expensive option in terms of cost.

We used two approaches to evaluate the strength of our BARENTS algorithm. The first one is the Workflow Responsive Resource Provisioning and Scheduling (WRPS) [8] algorithm that assigns sub deadline to each bag of tasks and schedules them on to a heterogeneous type of cloud resources. In WRPS, the authors model the problem as an unbounded knapsack minimization problem. The WRPS algorithm is the most noted recent work in the field of workflow scheduling. One limitation of WRPS is that the workflow schedule is generated under a simulated workflow execution environment. In addition, the workflow tasks are assumed to be homogeneous, whereas in reality tasks in a workflow are heterogeneous [13] with different types of tasks that consist of the component code such as the command line application, the web service based application, etc. Hence, WRPS does not consider any optimization strategies for heterogeneous tasks in a workflow. In contrast, using our approach, we model the execution time and execution cost of each heterogeneous task in a workflow by considering both the complexity of the computation in the task, as well as the outgoing data dependencies for each task which are combined for each partition in the workflow. However, we are still interested in comparing BARENTS to WRPS, when both are using homogeneous workflow tasks in each partition as a bag of tasks. In order to make our comparison realistic, we implemented a slight variation to the original WRPS algorithm by implementing the algorithm with an objective to minimize the makespan under a user driven budget constraint.

The second approach is a slight variation of our BARENTS algorithm called BARENTS*, in which we relaxed the dependencies between the partitions by setting the threshold, the credit and the debit to be 0. The WRPS algorithm provides an optimization to the bag of tasks by scheduling the tasks in a bag to different types of machines. In contrast to WRPS, BARENTS assigns all the tasks inside a given partition to the same type of machine. By performing this comparison, we are able to validate how the partition dependencies and run time sub-budget adjustment proposed in BARENTS is used as a distinguishing feature to outperform WRPS.

B. Results and Analysis

BARENTS was evaluated using 10 distinctive workflows developed in the OpenXC domain with different levels of complexity and with different dollar amounts provided as budget. In Table 4.1, we presented all the 10 workflows

Table 4.1. Workload details for OpenXC workflows

Workflow	No of drivers	Computation Size (MLOC)	Data Size (GB)	Budget ($)
w_1	5	1	1	45
w_2	10	2	2	60
w_3	15	3	3	70
w_4	20	4	4	80
w_5	25	5	5	85
w_6	30	6	6	95
w_7	35	7	7	110
w_8	40	8	8	115
w_9	45	9	9	130
w_{10}	50	10	10	150

with their complexity levels such as the computation and data intensity of all the tasks in each of the workflow and the user defined budget. We did the experiments by varying the types of machines (the K value) and presented the measurements from both the cost and makespan perspectives. In Figure 4.b, we show that BARENTS outperforms WRPS by roughly 6-10% margin as the complexity of the workflow increases from w3 to w10. For workflows between w1 and w3, which is of least complexity, WRPS outperforms BARENTS. The reason for this behavior is that WRPS schedules tasks in each bag to the resources of different machine types. The local optimization done at each partition outperforms the global optimization performed by BARENTS when the complexity level is low. We evaluated the behavior of all three approaches and have demonstrated the makespan minimization by varying the number of instance types for K = {5, 10, 15, 20, 25}. In order to vary the K values for same set of workflows with the same set of budgets, we created a bigger range with a larger difference between the instance types and then added new instance types within that range. Figure 4.a illustrates resource utilization in the cloud for different settings of K. The BARENTS algorithm outperforms WRPS because resources are utilized to the maximum extent for the tasks in each partition. Optimal resource utilization is achieved because BARENTS sets up partition dependency based on a system driven threshold and automatically adjusts the sub-budget at run time with a system driven credit or debit value, that is calculated from the actual completion cost of the previous partition. As K increases, there is a consistent improvement in makespan minimization and resource utilization.

V. RELATED WORK

While the cloud computing paradigm [17,18,19] provides a promising platform for running big data workflows, performance tuning of workflow execution in the cloud remains an important and challenging problem. One challenge is the selection of cloud resources. Given a workflow w, how many virtual machines are needed to run w in the cloud? What types of virtual machines are needed? As a cloud typically provides a set of heterogeneous virtual machine types that come with different configurations and prices, the selection of such cloud resources often need to

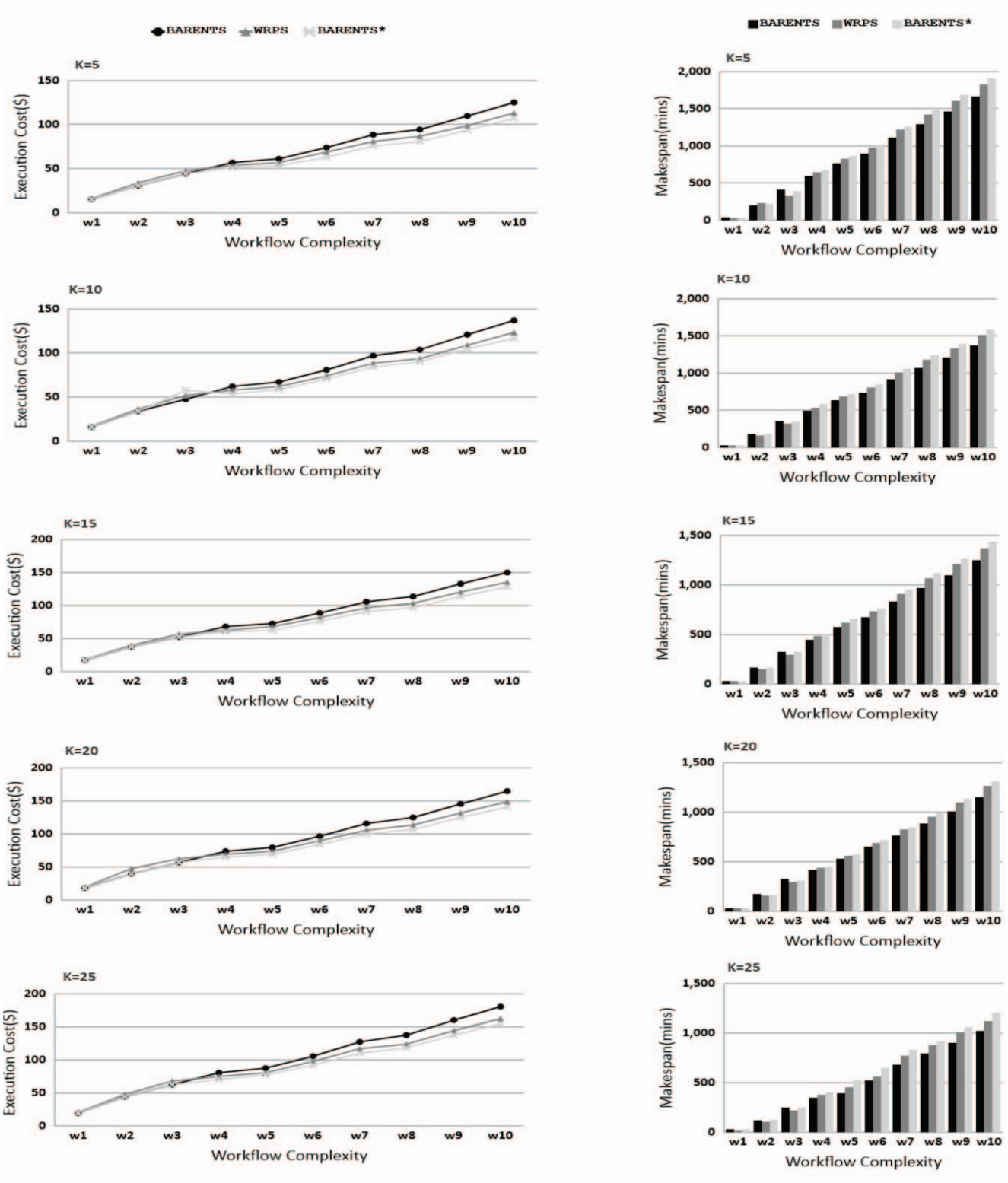

Figure 4. a) Resource utilization; b) Makespan minimization

consider the characteristics of input datasets and workflows, and the QoS parameters provided by a user such as budget and deadline. Over the past decade, there have been several workflow scheduling algorithms [1,8,20,21,22,23,24,25] proposed that play a crucial role in running workflows efficiently on the cloud. The scheduling algorithms are

widely categorized into static and dynamic algorithms. The existing static algorithms [1,2,3,4] do not consider the runtime estimation and hence are not very efficient in a heterogeneous cloud computing environment, where there is a cost and time for the computation performed in different resource types and a cost and time for the data movement from one resource to another. On the other hand, many existing dynamic algorithms [5,6,7,8,10,11,12] are capable of adapting to the unexpected delays that occur while executing the workflow in the cloud. The existing scheduling algorithms are either user driven with the QoS constraints set by the user or system driven with no constraints. There are two types of QoS constraints primarily proposed so far in the existing algorithms: 1) budget constraint [20,21,22,23,24,25], 2) deadline constraint [1,8,22,25]. Budget-constrained workflow scheduling algorithms aim to minimize the total execution time of a workflow while meeting a user specified budget constraint. Deadline-constrained workflow scheduling algorithms aim to minimize the total monetary cost of running a workflow while meeting a user specified deadline. There are some algorithms that belong to both categories [22, 25] as they aim to satisfy both the budget and deadline constraints, and hence are used for each category by relaxing one of the constraints. There are some algorithms that are solely based on system driven optimization such as [7], which does not consider any constraints. The goal of those algorithms is to generate the schedule with a single objective, which is to minimize the makespan.

There have been several existing works in dynamic workflow scheduling algorithms. Malawski et al. [5], propose Dynamic Provisioning Dynamic Scheduling (DPDS) algorithm. The authors propose to schedule the workflow ensembles on the cloud by maximizing the execution of the total number of user-prioritized workflows under a user provided QoS constraints such as budget and deadline. Zhou et al. [6] develop a probabilistic scheduling framework called Dyna that minimizes the execution cost under deadline constraint by considering the dynamic nature of the cloud computing such as performance and amazon spot instances pricing. Lin et al. [7] propose the SCPOR scheduling algorithm to dynamically schedule a workflow in heterogeneous cloud environment. The SCPOR algorithm prepares the workflow schedule with the goal of minimizing the makespan by dynamically provisioning and deprovisioning resources of several types with no constraints. Rodriguez et al. [8] propose an adaptive, resource provisioning and scheduling algorithm called WRPS to generate the workflow schedule in a heterogeneous cloud environment. The algorithm has a single objective to minimize the execution cost of the workflow under a user provided deadline constraint by modeling the problem as an unbounded knapsack minimization problem and is based on dynamic programming. The major limitation of this approach is that each partition is considered as an independent bag of tasks and hence only local optimization is performed in each partition of the workflow.

Furthermore, the WRPS algorithm fails to consider the dependencies that exist between the partitions in a workflow, which is a salient feature of the data centric workflows [13, 14]. Another limitation of the WRPS algorithm is that the approach only works for a homogeneous set of tasks in a bag of tasks. However, in reality the data centric workflows consists of heterogeneous tasks and the scheduling optimization is required to consider it [9]. Our BARENTS scheduler is different from the WRPS scheduling algorithm, because we consider the dependencies that exist between the partitions in a workflow through a system generated threshold, credit and debit values. Besides creating initial budget allocation, our approach also dynamically creates the sub-budget adjustment through the runtime estimation. Other dynamic scheduling algorithms [10,11,12] have also been proposed. However, these algorithms have only been validated in a simulated environment and not in a real big data workflow system.

VI. Conclusions and Future Work

To schedule big data workflows in the cloud computing environment, we formalize a model of the cloud computing environment and a workflow graph model for the environment. Based on the models, we propose a new Big dAta woRkflow schEduler uNder budgeT conStraint known as BARENTS that supports high-performance workflow scheduling in a heterogeneous cloud computing environment with a single objective to minimize the workflow makespan under a provided budget constraint. Our case study and experiments not only show the competitive advantages of our proposed scheduler, but also enables resources to scale elastically during workflow execution. The proposed BARENTS scheduler is implemented in a new release of DATAVIEW, one of the most usable big data workflow systems in the community.

We envision future work to proceed along the following two directions. First, addressing the challenge of dynamic scheduling of workflow tasks. Where and when should a workflow task be executed? When should a new virtual machine provisioned and deprovisioned? Such decision-making is harder even for a pipeline workflow of n tasks and m virtual machines (VM), since the number of task-to-VM assignments is exponential. Second, addressing the challenge of optimal data placement in the cloud and leverage such a data placement mechanism by integrating it with the scheduler in a holistic manner.

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] S. Abrishami, et al., "Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds." Future Generation Computer Systems (FGCS), vol. 29, no. 1, pp. 158-169, 2013.

[2] Z. Wu, et al., "A revised discrete particle swarm optimization for cloud workflow scheduling." in Proc. Of the International Conference Computational Intelligence and Security (CIS), pp. 184-188, 2010.

[3] S. Yassa, et al., "Multi objective approach for energy-aware workflow scheduling in cloud computing environments." The Scientific World Journal, 2013.

[4] R. Calheiros, et al., "Meeting deadlines of scientific workflows in public clouds with tasks replication." IEEE Transaction Parallel and Distributed Systems, vol. 25, no. 7, pp. 1787–1796, 2014.

[5] M. Malawski, et al., "Cost and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds." in Proc. International Conference on High Performance Computing, Networking, Storage and Analysis, 2012.

[6] A. C. Zhou, et al., "Monetary cost optimizations for hosting workflow-as-a-service in IaaS clouds." IEEE Transactions on Cloud Computing., vol. 4, no. 1, pp. 34-48, 2015.

[7] C. Lin, et al., "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.

[8] M.A. Rodriguez, et al., "A responsive Knapsack-based algorithm for resource provisioning and scheduling of scientific workflows in clouds." in 44th International Conference on Parallel Processing, ICPP, pp.839-848, 2015.

[9] A. Mohan, et al., "Addressing the Shimming Problem in Big Data Scientific Workflows." in Proc. of the 2014 IEEE International Conference on Services Computing (SCC'14), pp. 347-354, 2014.

[10] M. Xu, et al., "A multiple QoS constrained scheduling strategy of multiple workflows for cloud computing." in Proc. International Symposium on Parallel and Distributed Processing with Applications (ISPA), pp. 629-634, 2009.

[11] T. T. Huu et al., "Virtual resources allocation for workflow-based applications distribution on a cloud infrastructure." in Proc. International Conference Cluster, Cloud Grid Computing (CCGrid), pp. 612-617, 2010.

[12] D. de Oliveira, et al., "A provenance-based adaptive scheduling heuristic for parallel scientific workflows in clouds." Journal of Grid Computing, vol.10, no. 3, pp. 521–552, 2012.

[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'14), 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 (BigData 2016), pp. 52-59, 2016.

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

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

[17] P. Mell, et al., "The NIST definition of cloud computing." Communications of the ACM, vol. 53, no. 6, p. 50, 2010.

[18] A. Lenk, et al., "What are you paying for? performance benchmarking for infrastructure-as-a-service offerings." Cloud Computing (CLOUD), 2011 IEEE International Conference on. IEEE, pp. 484-491, 2011.

[19] A. Lenk, et al., "What's inside the Cloud? An architectural map of the Cloud landscape." in Proc. of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing, pp. 23-31, 2009.

[20] H. Arabnejad, et al., "A budget constrained scheduling algorithm for workflow applications." Journal of Grid Computing, vol. 12, no. 4, pp. 665-679, 2014.

[21] R. Sakellariou, et al., "Scheduling workflows with budget constraints." in Integrated research in GRID computing, pp. 189-202. Springer US, 2007.

[22] W. Zheng, et al., "Budget-deadline constrained workflow planning for admission control." Journal of Grid Computing, vol. 11, no. 4, pp. 633-651, 2013.

[23] A. Hamid, et al., "Low-time complexity budget-deadline constrained workflow scheduling on heterogeneous resources." Future Generation Computer Systems (FGCS), vol. 55, pp. 29-40, 2016.

[24] R. Prodan, et al., "Bi-criteria scheduling of scientific grid workflows." IEEE Transactions on Automation Science and Engineering, vol. 7, no. 2, pp. 364-376, 2010.

[25] J. Yu, et al., "Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms." Scientific Programming, vol. 14, no. 3-4, pp. 217-230, 2006.