TPS: A Task Placement Strategy for Big Data Workflows

Mahdi Ebrahimi, Aravind Mohan, Shiyong Lu, Robert Reynolds
Wayne State University
Detroit, U.S.A.
{mebrahimi, amohan, shiyong, robert.reynolds}@wayne.edu

Abstract— **Workflow makespan is the total execution time for running a workflow in the Cloud. The workflow makespan significantly depends on how the workflow tasks and datasets are allocated and placed in a distributed computing environment such as Clouds. Incorporating data and task allocation strategies to minimize makespan delivers significant benefits to scientific users in receiving their results in time. The main goal of a task placement algorithm is to minimize the total amount of data movement between virtual machines during the execution of the workflows. In this paper, we do the following: 1) formalize the task placement problem in big data workflows; 2) propose a task placement strategy (TPS) that considers both initial input datasets and intermediate datasets to calculate the dependency between workflow tasks; and 3) perform extensive experiments in the distributed environment to demonstrate that the proposed strategy provides an effective task distribution and placement tool..**

Keywords- Big Data Workflow, Task Placement, Cloud Computing, Evolutionary Algorithms, Genetic Algorithms.

1. INTRODUCTION

Complex data-centric computations are generally modeled as workflows [5, 17]. Among other benefits, representing a complex application as a workflow simplifies design effort, enables the reuse of computational modules and allows their parallel and/or pipelined execution. The concept of workflow applications has been in use for quite some time in research in domains such as bioinformatics, physics, astronomy, and ecology [7, 10]. With the progress in computing, storage, networking, and sensing technologies and the ease of performing collaborative scientific research, it is feasible to create much more complex data-centric workflows that involve big datasets [3, 15, 19] and run them over distributed and heterogeneous computing environments such as Clouds [9]. We proposed big data workflow as the next generation of data-centric workflows to address the above challenges. Big data workflows involve big data sets and can be executed over the Cloud [17].

A data-centric workflow management system (WFMS) is a platform to support two key functions: 1) the design and specification of workflows; and 2) the configuration, execution and monitoring of workflow runs. Traditionally, these systems have used a directed acyclic graph (DAG) abstraction in order to model a workflow where each vertex of the graph represents a workflow task, and the directed edge between two vertices depicts dataflow between the corresponding tasks. A workflow task can be either a built-in task, a web service [21, 22] or comprised of heterogeneous components.

The size of scientific datasets are often in terabytes or even petabytes, and require dedicated virtual machines just for data storage purposes [6]. Since scientific applications have become more and more data intensive, it is even more critical to assign workflow tasks to the same virtual machines which are already hosted their required datasets as "moving computation to data is often cheaper than moving data to computation" [1]. For this, workflow management needs to utilize an effective data and task placement strategy in order to maximize data locality, and minimize data movement between virtual machines in the Cloud. In big data workflows, it is practically impossible to store all of the required datasets of tasks in one virtual machine due to the storage capacity limitation of virtual machines. Thus, data movement is necessary to execute workflows.

Big data workflows typically model and analyze complex scientific research experiments as a very large number of tasks along each with a huge volume of datasets. These large datasets are physically distributed, and placed on different data sources by scientist users over the Internet. Big datasets are difficult to manage by traditional data management tools and strategies. As a result, big data technology is becoming the main focus in scientific computing research. [2, 20].

In our previous work, we proposed BDAP as a big data placement strategy for data-centric workflows [16]. To continue our research, we propose TPS as a task placement strategy for big data workflows in this paper. Big data workflows consist of both data and tasks. BDAP places workflow *data* but TPS places workflow *tasks* into the available virtual machines in the Cloud. TPS is an evolutionary algorithm (EA) employing the Genetic Algorithm framework. [23]. It clusters the most interdependent workflow tasks ogether, and can possibly assignthem to the same virtual machine in the Cloud so as to minimize data movement between virtual machines. We explain our proposed task placement strategy in section 2 in detail.

In order to illustrate the idea let e us consider an example to show how a big data workflow can be executed in a cloud computing environment. Fig. 1.(a) illustrates a sample workflow with five tasks (t_k), five original datasets (dj) and five generated intermediate datasets (d'_k). Fig. 1.(b) shows an instance of its virtual machines configuration along with a data placement scheme. Any data placement strategy like BDAP can be used to assign datasets to the appropriate virtual

978-1-4799-9926-2/15/$31.00 ©2015 IEEE

machines. In this figure, datasets d_1 and d_3 were placed in virtual machine 1, VM_1. Similarly, d_2 and d_4 were placed in VM_2 and d_5 was placed in VM_3. Fig. 1.(c) represents a complete instance of a virtual machines configuration as well as data and workflow task placement. Tasks t_1 and t_2 were assigned to virtual machine VM_1. Tasks t_3 and t_4 were assigned to VM_2, and t_5 was assigned to VM_3. In this example, once the workflow is executed, dataset d_2 is required to move from VM_2 to VM_1 to in order complete the process of task t_2. However, the execution of t_3 only requires transferring the output of task t_1, d'_1, from VM_1 to VM_2 since all of its required source datasets, d_2 and d_4, are already placed in VM_2.

The rest of the paper is organized as follows, first, in Section 2 we define and formalize our system model. In Section 3 we explain our task placement strategy in detail. Then, in Section 4, the experimental results are shown and discussed. Section 5 presents the related work. Finally, conclusion and future work are presented in Section 6.

2. Formalizing Workflow Task Placement

Big data workflows are typically executed in the cloud. To model the cloud computing environment, we consider a set of virtual machines in the cloud as the sites to execute the workflow. Each virtual machine can be provided by a different Cloud Computing Providers (CCP) such as Amazon EC2, Google App Engine, and Microsoft Azure. A Cloud computing environment is modeled as follows:

Definition 2.1 (Cloud Computing Environment C). A cloud computing environment, C, is a 4-tupl $C = (VM, CC, SC, DTR)$, where

- VM is a set of virtual machines in the cloud vm_i $(i = 1, 2, \dots, I)$.
- $CC: VM \rightarrow R^+$ is a computation capacity function. $CC(vm_i)$, $vm_i \in VM$ gives the maximum available computation capacity of virtual machine vm_i in the Cloud computing environment C. R^+ is the set of positive real numbers.
- $SC: VM \rightarrow R^+$ is a storage capacity function. $SC(vm_i)$, $vm_i \in VM$ gives the maximum available storage capacity of virtual machine vm_i in the Cloud computing environment C. It is measured in some pre-determined unit such as mega-bytes, giga-bytes or tera-bytes. R^+ is the set of positive real numbers.
- $DTR: VM \times VM \rightarrow Q_0^+$ is the data transfer rate function. $DTR(vm_{i1}, vm_{i2})$, $vm_{i1}, vm_{i2} \in VM$ gives the data transfer rate between two virtual machines vm_{i1} and vm_{i2}. It is measured in some pre-determined unit such as mega-bytes, giga-bytes per second. Q_0^+ is the set of positive rational numbers.

Please note that the above three attributes, CC, SC and DTR are not fixed or static for a virtual machine at all times. However, these are considered to be established by a priori negotiation and remain unchanged during the execution of a given individual workflow.

For solving complex scientific problems, scientists are

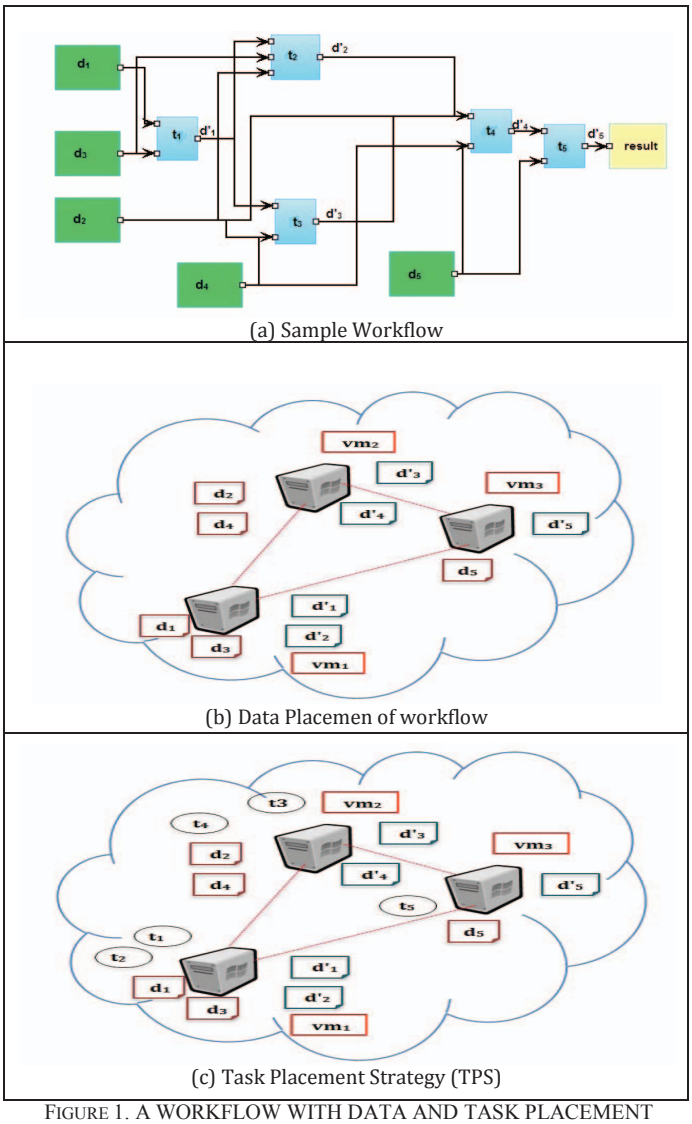

Figure 1. A workflow with data and task placement

able to create and run their own big data workflows simultaneously. Each individual workflow contains a set of tasks that consume various datasets, and may produce intermediate datasets as well. Those intermediate datasets will be sent to other tasks as their inputs by following the data flow logic. A big data workflow is formalized as follows:

Definition 2.2 (Big Data Workflow W). A big data workflow W can be modeled formally as a 6-tuple that consists of three sets and two functions as follows:

$$W = (T, D, D', S, TS, DS)$$

- T is the set of workflow tasks. Each individual task is denoted by t_k, $T = \{t_1, t_2, t_3, \dots, t_K\}$.
- D is the set of input datasets connected to workflow W. Each individual dataset is denoted by d_j, $D = \{d_1, d_2, \dots, d_J\}$.
- D' is the set of output datasets produced by workflow W. The total number of output datasets is equal to the total number of workflow tasks as each workflow task, t_k, generates one output dataset, d_k, which can flow to

the other tasks as the input dataset. Each individual output dataset is denoted by d'_k, $D' = \{d'_1, d'_2, \dots, d'_K\}$.

- $S: D \cup D' \rightarrow R^+$ is the dataset size function. $S(d_j)$, $d_j \in D \cup D'$ returns the size of an original or generated dataset d_j. The size of a dataset is defined in some pre-determined unit such as mega-bytes, giga-bytes or tera-bytes. R^+ is the set of positive real numbers.
- $TS: D \cup D' \rightarrow T$ is the dataset-task function. $TS(d_j)$, $d_j \in D \cup D'$ returns the set of workflow tasks that consume d_j as their input.
- $DS: T \rightarrow D \cup D'$ is the task-dataset function. $DS(t_k)$, $t_k \in T$ returns the set of datasets that are consumed by t_k as its input. The datasets can be either original or generated datasets.

To evaluate and compare TPS with other proposed algorithms, Workflow Communication Cost is defined as follows:

Definition 2.3 (Workflow Communication Cost, WCC).

If dataset d_j is required to transfer from virtual machine vm_{i1} to vm_{i2} then the data movement cost of d_j is defined as

$$DMC(d_j, vm_{i1}, vm_{i2}) = \begin{cases} 0, & if\ i_1 = i_2 \\ \dfrac{S(d_j)}{DTR(vm_{i1}, vm_{i2})}, & if\ i_1 \neq i_2 \end{cases} \quad (1)$$

where DTR is the data transfer rate function of the Cloud, and S function returns data size.

Given a workflow W and Cloud computing C, the Workflow Communication Cost (WCC) is equal to the total data movement cost for executing the complete workflow W in C. It is defined as follows:

$$WCC(W, C) = \sum_{k=1}^{K} \sum_{\substack{d_j \in DS(t_k) \\ t_k \in W}} DMC(d_j, vm_{i1}, vm_{i2}) \quad (2)$$

WCC returns the total data movement during workflow execution of W in the Cloud C. The three main concepts in clustering are those objects that need to be clustered, the clusters, and a separation measure to compute the similarity among the objects in a cluster. In this work, tasks are considered as the objects and virtual machines in the Cloud are considered as the clusters. Therefore we need to come up with a good separation measurement to cluster the most similar objects (tasks) together in order to meet the objective goal (minimizing the total data movement).

We consider task interdependency as a vehicle to assess the separation virtual machines. Two tasks are interdependent and should be co-allocated in the same virtual machine if they simultaneously need many of the same datasets as their inputs. The definition for the interdependency of a pair of tasks is as follows: **Definition 2.4 (Task Interdependency)**. We consider the size of the common datasets that a pair of tasks needs as input to be a measure of the interdependency of the tasks. Task interdependency value is divided by the total size of the workflow datasets in order to be normalized in the range of [0 1]. Formally, given two tasks t_{k1} and t_{k2}, the task interdependency is calculated by:

$$tp(t_{k1}, t_{k2}) = \frac{S(\, DS(t_{k1}) \cap DS(t_{k2})\,)}{S(D)} \quad (3)$$

where S(D) is the sum of the sizes of the datasets in D.

For instance, if size of datasets is $S(d_1) = 10MB$, $S(d_2) = 35MB$, $S(d_3) = 110MB$, $S(d_4) = 60MB$ and $S(d_5) = 55MB$, then the set of tasks that consume d_1 is $DS(t_1) = \{d_1, d_3\}$ and d_2 is $DS(t_2) = \{d_1, d_2, d_3\}$ and the task interdependency between t_1 and t_2 is

$$tp(t_1, t_2) = \frac{S(DS(t_1) \cap DS(t_2))}{S(D)} = \frac{S(\{d_1, d_3\} \cap \{d_1, d_2, d_3\})}{S(\{d_1, d_2, d_3, d_4, d_5\})}$$
$$= \frac{S(d_1) + S(d_3)}{S(d_1) + S(d_2) + S(d_3) + S(d_4) + S(d_5)} = 0.44.$$

In this way, two tasks are interdependent once they have at least one common dataset as input for both of them. Two tasks have a higher interdependency when they consume larger sizes of common datasets. In order to o maximize data locality, it is necessary to pre-cluster the workflow tasks initially. In the first step, we calculate the task interdependency of all the workflow tasks, and generate the task interdependency matrix (TM). In the interdependency matrix, rows and columns are the workflow tasks, and the value of a cell in the interdependency matrix is the task interdependency between two tasks. For instance, task interdependency matrix of workflow in Example 1 is as follows:

$$TM = \begin{array}{c} \\ t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \end{array} \begin{array}{c} \begin{array}{ccccc} t_1 & t_2 & t_3 & t_4 & t_5 \end{array} \\ \begin{pmatrix} 0.44 & 0.44 & 0.00 & 0.00 & 0.00 \\ 0.44 & 0.54 & 0.13 & 0.13 & 0.00 \\ 0.00 & 0.13 & 0.22 & 0.35 & 0.00 \\ 0.00 & 0.13 & 0.35 & 0.55 & 0.20 \\ 0.00 & 0.00 & 0.00 & 0.20 & 0.20 \end{pmatrix} \end{array}$$

TPS partitions and distributes the original datasets onto all appropriate virtual machines in the Cloud. Then, the related tasks will be assigned to the corresponding virtual machine so that their required datasets are stored there. In this way, the total amount of data movement between virtual machines is decreased, and the overall workflow execution time will be reduced. This task placement scheme is used to create a mapping of each workflow task onto corresponding virtual machine. A task placement scheme is defined formally as follows:

Definition 2.5 (Task Placement Scheme Ψ). Suppose there are *I* virtual machines and K tasks, a task placement scheme is represented by a K-element vector Ψ such that $\Psi(t_k)$ indicates the virtual machine to which t_k is placed. For example if the task placement scheme is $\Psi = (1, 2, 1, 2, 3)$ it means tasks t_1 and t_3 are placed in virtual machine vm_1 ($\Psi(t_1), \Psi(t_3) = vm_1$), tasks t_2 and t_4 in virtual machine vm_2 ($\Psi(t_2), \Psi(t_4) = vm_2$), and the task t_5 in virtual machine vm_3 ($\Psi(t_5) = vm_3$). We consider all the workflow tasks to be flexible and can be executed by any of the virtual machines. To define a good

metric to compare the separation between virtual machines, the task interdependency within and between virtual machines are defined as follows:

Definition 2.6 (Within-VirtualMachine Task Interdependency $\mathbf{VMT_W}$).

$$VMT_W(\Psi) = \sum_{i=1}^{I} \sum_{\substack{\Psi(t_{k1})=vm_i \\ \Psi(t_{k2})=vm_i}} tp(t_{k1}, t_{k2}) \quad (4)$$

where $tp(t_{k1}, t_{k2})$ is the task interdependency between task $t_{k1}$and t_{k2} , I is the maximum number of virtual machines in the Cloud.

Definition 2.7 (Between-VirtualMachine Task Interdependency $\mathbf{VMT_B}$).

$$VMT_B(\Psi) = \sum_{\substack{i_1 \neq i_2 \\ (i_1,i_2)\in(I,I)}}^{(I,I)} \sum_{\substack{\Psi(t_{k1})=vm_{i_1} \\ \Psi(t_{k2})=vm_{i_2}}} tp(t_{k1}, t_{k2}) \quad (5)$$

To achieve the task placement goal, TPS uses heuristic information for its search direction of finding the best task placement scheme. Heuristic information should consider both inter and intra virtual machine interdependency. The heuristic is defined in TPS as follows:

Definition 2.8 (Task Interdependency Greedy TG). The TG heuristic biases TPS to select the task placement scheme with higher task interdependency. It is defined as:

$$TG(\Psi) = \frac{VMT_W(\Psi) + 1}{VMT_B(\Psi) + 1} \quad (6)$$

In this formula, the numerator measures the Within-VirtualMachine Task Interdependency, and the denominator measures the Between-VirtualMachine Task Interdependency. The bias 1 is set to avoid divided-by-zero in the case that the task interdependency between virtual machines is zero. A good task placement scheme has a higher TG. Therefore the output of TPS is a task placement scheme with the highest TG.

In this paper we do not consider task replication. It means once a task is mapped into a specific virtual machine, it cannot be placed placed into another virtual machine. .

Definition 2.9 (Task Placement Solution). The task placement solution for big data workflow, W, to execute in a Cloud computing environment, C, is to select a task placement scheme $\Psi \in P$ that minimizes the workflow communication cost (WCC) under the virtual machine storage capacity and non-replication constraint. In the next section, we explain our task placement strategy, TPS, in detail.

3. Proposed Task Placement Algorithm

The main steps of TPS are depicted in Fig. 2. In the first phase, TPS applies a metaheuristic optimization Genetic Algorithm to place the workflow tasks. The main goal is to minimize workflow communication cost by minimizing the data movement between virtual machines in the cloud while running a workflow. TPS starts with calculating the task interdependency matrix. Then, it generates a set of task placement schemes randomly, and calculates their heuristic values. In the following, for each task placement scheme, TPS applies three main genetic operators: Selection, Crossover, and Mutation. They are applied sequentially to generate possibly better schemes with higher heuristic values. At the end of the algorithm, the best task placement scheme is recorded in Ψ_{best} and will be returned as the output of TPS.

In order to apply the task placement strategy and to analyze the task interdependency, the entire workflow has to be designed. It means that all tasks and datasets of the big data workflow have to be specified. The TPS algorithm is outlined in Algorithm 1.

In the first step, TPS generates *popsize* number of feasible and valid task placement schemes randomly as well as calculating the heuristic value of each individual scheme (lines1-5). In the next step, TPS applies the three main operators in order to generate new schemes with a hopefully higher performance function values until it reaches the max number of iterations. First, it selects ne = popsize × elitism_rate number of schemes with the highest heuristic value and saves them in the Pop_E (lines 9-10). These high-value schemes will transfer directly to the next generation of task schemes to guarantee the convergence of TPS. We apply the fitness proportional selection, roulette wheel selection, for this step. The idea behind the roulette wheel selection technique is that each scheme is given a chance to select in proportion to its performance function value. Then, it applies the crossover function and, computes the heuristic value of the new generated schemes (lines 11-16). In the last step, TPS applies the mutation operator for a randomly selected scheme along with computing its heuristic value (lines 17-25). The idea behind the roulette wheel selection technique is that each scheme is given a chance to be selected as a parent in proportion to its performance value. These three operators apply to the task schemes until it reaches user-defined TG threshold which is minimum acceptable TG defined by the user at the beginning of the algorithm. In the last step, the best task placement scheme Ψ_{best} is returned as the output of TPS.

In the next section, we present and discuss the experiments results and compare TPS with k-means clustering and Random approaches. The k-means clustering is based on Yuan's work [14] which is the one of the most competitive algorithms in this field. It was originally proposed for data clustering and we revised it to use for workflow task clustering. It applies a matrix based k-means clustering strategy to precluster tasks into k virtual machines by using data interdependency and BEA (Bond Energy Algorithm).

4. Experiment and Case Study

A. Case Study

To evaluate performance of our proposed task placement approach (TPS) we compare it with k-means clustering and

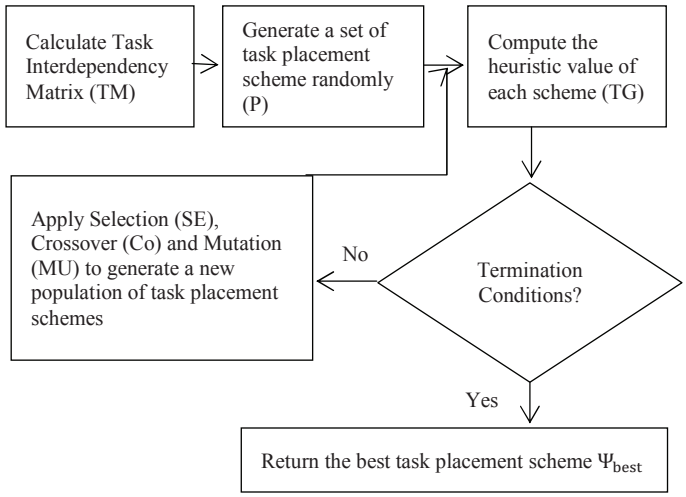

FIGURE 2. FLOWCHART OF TPS.

Random strategy. We developed a real Cloud-based workflow for OpenXC dataset to compare any number of car drivers with each other.

In DATAVIEW [17], we developed an OpenXC workflow, that consists of six individual workflow tasks (Fig. 3). For each individual car driver we calculated her driving brehavior. This workflow has two main stages, in the first stage it computes how unsafe the driver is based on the braking ability and in the second stage it evaluates the vehicle speed of the driver in order to to compute the risk level of the driver.

Description of the workflow tasks are as follows:

Task 1 – getDriverInfo: This workflow task gets the OpenXC raw data set as well as car driver id, and returns the signal details for that particular car driver.

Task 2 - BrakeSpeedDistribution: This step is used to compute how unsafe the driver is, based on her braking ability. For every pair of brake pressed (true and false value), the workflow will output the total time driven without pressing brake and the top 5 vehicle speed.

Task 3 – getAddByLatLon: In this step, the address where the signal is captured is calculated by using the Google API and Latitude and Longitude signal.

Task 4 – chkHighway: This task is used to compute decide if the car is on highway or not by using a google places API. It is based on the address where the signal is captured.

Task 5 – getSpeedLimit: This task is used to get the speed limit posted on the road. This workflow will automatically set the speed to 65 if it is highway. If not highway it will set the speed limit to 45.

Task 6 –speedCheck: This task is to compare the top 5 actual vehicle speed with the speed limit posted on the road in order to compute the total number of times the driver exceeded the speed limit.

Task 7 –compareDriver: This task is used to compare different drivers based on their speed distribution and braking ability.

Algorithm 1. Task Placement Strategy (TPS)

Input:

T: set of workflow tasks,
TP: task interdependency matrix,
popsize: size of population ,
er: rate of elitism,
cr: rate of crossover,
mr: rate of mutation,
ε: TG threshold,

Output:

The best task placement scheme, Ψ_{best}

1. **Begin**
2. **for** *i = 1* **to** *popsize* **do**
3. $\Psi \leftarrow$ Generate a task placement scheme randomly;
4. $Pop \leftarrow < \Psi, TG(\Psi) >$;
5. **end for**
6. **Do**
7. *oldTG = The highest TG in Pop;*
8. *ne = popsize* × *er; //* number of elitism
9. $Pop_E \leftarrow$ The best *ne* task placement schemes in *Pop*;
10. *nc = popsize* * *cr; //* number of crossover
11. **for** *i =1* **to** *nc* **do**
12. randomly select two task placement scheme Ψ_A and Ψ_B from *Pop;*
13. generate Ψ_C and Ψ_D by one-point crossover for tasks of Ψ_A and Ψ_B;
14. $Pop_C \leftarrow < \Psi_C, TG(\Psi_C) >$;
15. $Pop_C \leftarrow < \Psi_D, TG(\Psi_D) >$;
16. **end for**
17. *nm= popsize* × *mr;//* number of mutation
18. **for** *i =1* **to** *nm* **do**
19. select a task placement scheme Ψ_j from Pop_C;
20. $\Psi'_j \leftarrow$ mutate randomly a virtual machine position numbe in Ψ_j ;
21. $\Psi_j \leftarrow \Psi'_j$;
22. **end for**
23. $Pop \leftarrow Pop_E$ and Pop_C;
24. *newTG = The highest TG in Pop;*
25. **while** *(|newTG - oldTG| / oldTG* ≤ ε *)*
26. return the best task placement scheme Ψ_{best};
27. **end**

Fig. 3 shows the OpenXC workflow for comparing driving behavior for two car drivers. There are 6 individual tasks and 13 datasets (both original and intermediate datasets) for each car driver. To create a workflow with the large number of tasks and data products, we repeat the above workflow with a different number of car drivers under the assumption that each

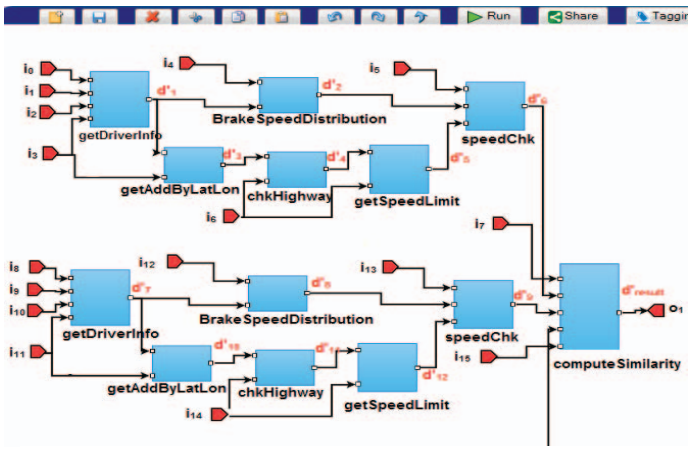

FIGURE 3. OPENXC WORKFLOW FOR COMPARING TWO CAR DRIVERS.

task can be executed on different virtual machines. For our experiments, we consider 2, 10, 20, 50 and 100 car drivers with a total number of tasks, [13, 61, 121, 301, 601]. In our experimental setting, we used virtual machines in the range of 5-25 with a range of 5GB-20GB of storage capacity (as shown in Table 1). The input OpenXC datasets are synthetic datasets built from the data recorded by real car drivers [18]. We demonstrate the performance of our proposed task placement algorithm by comparing it with k-means clustering, and a randomly generated task placement approaches with the average of the workflow communication cost defined in section 3. Based on our experiments, we observe that our results shown in Table 2 outperform the other task placement schemes.

B. Implementation

In our DATAVIEW system, we integrated the big data workflow engine subsystem with FutureSystems academic cloud provider in order to automatically provision virtual machines to execute big data workflows in the Cloud. We implemented bash scripts to automatically provision virtual machines by first creating a new image and configure both the hardware and software settings.

Workflow execution is transparent to our data scientists. They can just create and run any arbitrary workflow and the system deploys a set of virtual machines, datasets and moves workflow tasks to the corresponding virtual machine. At design time, our TPS algorithm parses the specification of the workflow and identifies an optimal mapping of the workflow tasks to the corresponding virtual machines. At run time, the DATAVIEW system moves the workflow task to the corresponding virtual machines based on the mapping generated by TPS. Finally the workflow is executed in a distributed manner as a means to improve the performance of the TPS. Please note that workflow scheduling is out of the scope of this study. The order of workflow tasks execution (sequential, pipeline or parallel) is not specified by TPS. TPS can be invoked by any workflow scheduler to obtain an optimal placement and therefore minimize workflow makespan. For our experiments in three approaches, we ran workflow tasks sequentially from entry task till the exit/final task.

Table 3 shows the description of running the OpenXC workflow of Fig. 3 and Table 4 shows some of the result of applying TPS for the execution of workflow in Example 1.

TABLE 1. DESCRIPTION OF TASK AND VIRTUAL MACHINE OF THE EXPERIMENT.

Overall task and virtual machine	
# of tasks	[13, 61, 121, 301, 601]
# of virtual machines	[5, 10, 15, 20, 25]
virtual machines computing capacity	5GB – 20GB
data transfer rate between virtal machines	5MB per second

TABLE 2. DEFAULT SETTING FOR TPS ALGORITHM.

Overall dataset and virtual machine	
Maximum population size	100
Initial population	Randomly generation
Maximum generation	100
Crossover probability	0.8-0.9
Mutation probability	0.3-0.5
TG threshold	0.01-0.1

C. Results

Fig. 4 shows the Workflow Communication Cost (WCC), in terms of hour by varying the number of tasks and fixing the number of virtual machines. Our experiments show that WCC cost increases with a large number of tasks and data products in both algorithms. However, it can be seen clearly that our strategy reduces WCC compared to the k-means clustering and Random algorithms. In the next step, we calculate WCC by varying the number of virtual machines and fixing the number of tasks (Fig. 5). Although WCC is increased by increasing the number of virtual machines, the increasing rate of our strategy is slower than the k-means clustering and Random strategies. In addition, it shows at some point, provisioning new Cloud resources like virtual machines does not affect the workflow performance as we may have many idle virtual machines.

5. RELATED WORK

Both data and workflow task placement becomes a fundamental research task in the cloud due to the rapid increase of accessible large datasets over the Internet and the

TABLE 3. OPENXC WORKFLOW OF ONE CAR DERIVER RUNNING IN DATAVIEW

Task Name	Execution Time (hrs)	Input Data Size(GB)	Output Data Size(GB)	VM#
getDriverInfo	0.353	10.00	1.00	vm_1
BreakSpeedDistribution	1.514	1.00	0.098	vm_2
getAddbyLatLon	0.513	1.00	0.017	vm_1
chkHighway	0.012	0.017	0.019	vm_3
getSpeedLimit	0.025	0.019	0.024	vm_1
speedChk	0.001	0.122	0.098	vm_2
computeSimilarity	1.260	0.290	0.001	vm_1

TABLE 4. SOME RESULT OF APPLYING TPS FOR THE EXECUTION OF WORKFLOW IN EXAMPLE 1.

The best task placement scheme in the	
First population <$t_{\#}$, $vm_{\#}$>	**Last (10^{th}) population <$t_{\#}$, $vm_{\#}$>**
<t_1,vm_1><t_2,vm_2><t_3,vm_3><t_4,vm_3><t_5,vm_1> TG = 0.11 and WCC = 0.0101 hr	<t_1,vm_1><t_2,vm_1><t_3,vm_2><t_4,vm_1><t_5,vm_3> TG = 2.10 and WCC = 0.0061 hr
<t_1,vm_2><t_2,vm_1><t_3,vm_3><t_4,vm_1><t_5,vm_3> TG = 0.09 and WCC = 0.0176 hr	<t_1,vm_2><t_2,vm_2><t_3,vm_1><t_4,vm_1><t_5,vm_3> TG = 2.48 and WCC = 0.0043 hr

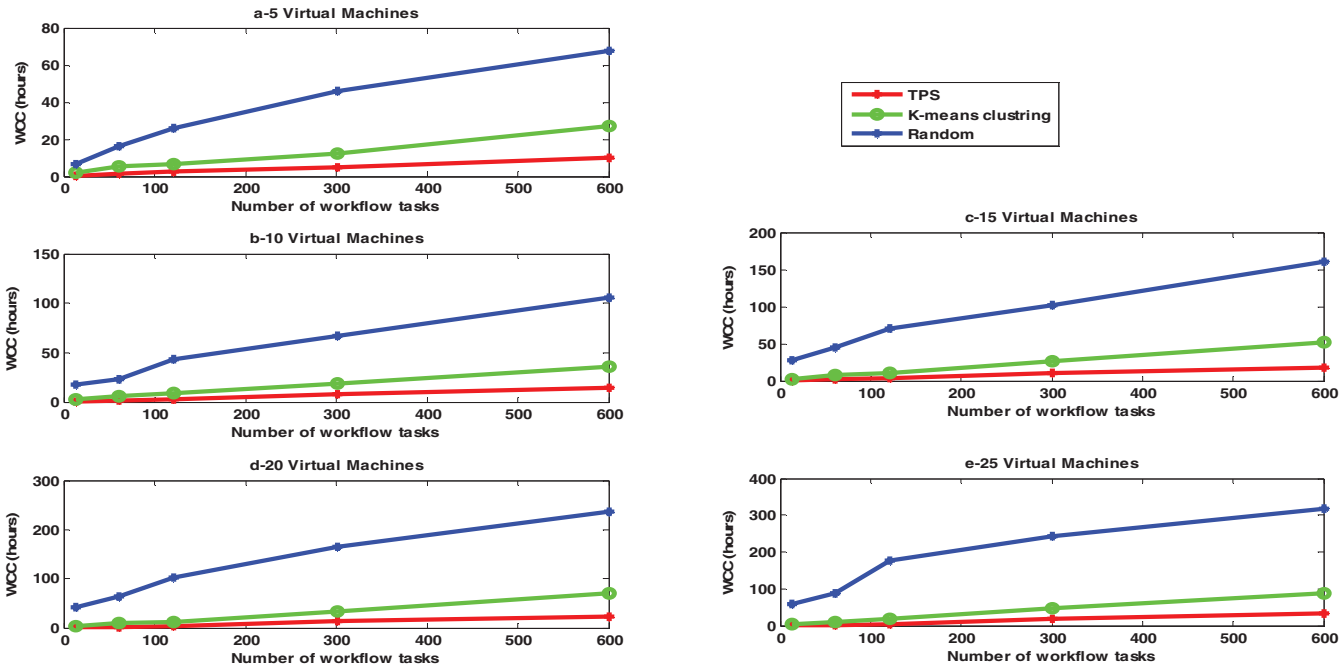

FIGURE 4. WORKFLOW COMMUNICATION COST (HOURS) BY VARYING THE NUMBER OF WORKFLOW TASKS.

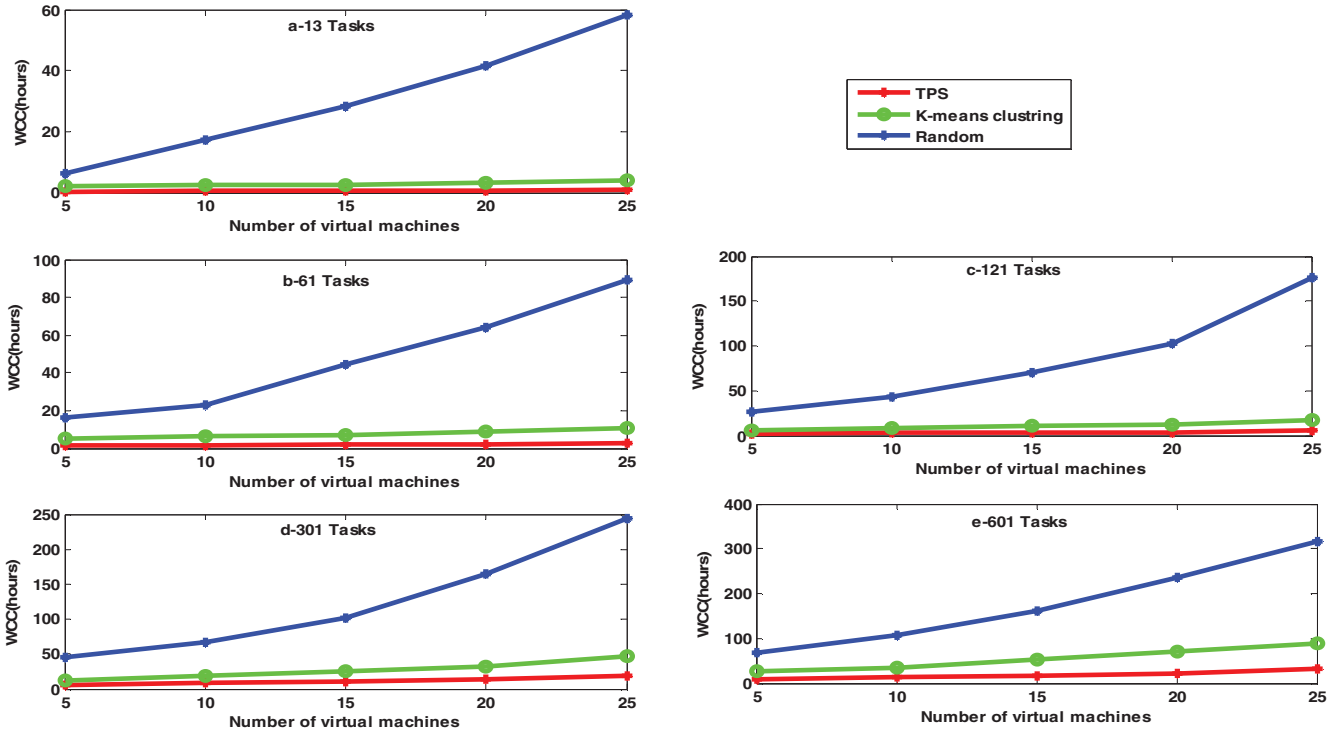

FIGURE 5. WORKFLOW COMMUNICATION COST (HOURS) BY VARYING THE NUMBER OF VIRTUAL MACHINES.

emerging field of big data. Previous research studies for distributed computing environment have been mainly focused on the performance modeling and optimization of job scheduling and task placement. Kosar et al., [12] proposed a framework for distributed computing systems, which considered both the data placement and computation modules as two separate subsystems. In this framework, data placement module was proposed as an independent job that can be scheduled and managed like the computation jobs. Kayyoor et al., [11] considered data replication along with data placement for the distributed environments. Instead of minimizing of query latencies, their goal was to minimize the average number of dedicated computation nodes. For this, they grouped the most similar data together based on their occurrences in common query accesses.

By applying task placement and data replication services, Chervenak et al., [13] evaluated and displayed the benefits of

pre-staging data compare to the data stage processing of the Pegasus.

In Catalyurek et al., [4] workflows were modeled by the hypergraph concept and a hypergraph partitioning technique, k-way partitioning, is applied to minimize the cutsize. In that way, they clustered the workflow tasks as well as their required data in the same execution site. Yuan et al., [14] applied a greedy binary clustering technique to pre-cluster datasets at design-time based on the workflow task dependencies. Er-Dun et al., [8] proposed a Genetic Algorithm for data placement that considered a load balancing factor. Their approach reduced data movement, but they did not consider task interdependency factor and cluster the data based on the data similarity.

In our previous work, we proposed big data placements strategy (BDAP) in order to place the most interdependent datasets in the same virtual machine in the Cloud by considering their common workflow tasks. BDAP minimized the total amount of data movement between virtual machines during the execution of the workflows. TPS is about task placement and can be applied independently or in conjecture with BDAP to place both data and workflow takes together.

6. CONCLUSIONS AND FUTURE WORK

We proposed TPS, a task placement strategy for big data workflows. TPS minimized the total amount of data movement between virtual machines during the execution of the workflows. Our extensive experiments and comparisons indicated that TPS outperformed the k-means clustering and Random algorithms so as to minimize data movement.

Big data workflows consume and produce huge datasets. Applying task/data replication can reduce data movement as well. So in future work, we plan to improve TPS by applying task/data replication techniques. In addition, we considered task placement for executing of an individual workflow. However, in real world, multiple workflows can be executed concurrently. Therefore, we plan to extend the TPS strategy in order to achieve task placement for the execution of multiple workflows simultaneously. For the other future work, we will enhance the performance of both BDAP and TPS strategies by using Cultural Algorithm (CA).

ACKNOWLEDGMENT

This work is supported by National Science Foundation, under grants NSF ACI-1443069. This material is based upon work supported in part by the National Science Foundation under Grant No. 0910812.

REFERENCES

[1] D. Agrawal, A. E. Abbadi, S. Antony, and S. Das, "Data management challenges in cloud computing infrastructures." In Databases in Networked Information Systems, pp. 1-10, 2010.

[2] J. Wang, D. Crawl, I. Altintas, W. Li, "Big Data Applications Using Workflows for Data Parallel Computing," Journal of Computing in Science and Engineering, vol. 16, no. 4, pp. 11-21, 2014.

[3] E. Bertino, "Big Data-Opportunities and Challenges." In IEEE 37th Annual Computer Software and Applications Conference. 2013.

[4] U. V. Catalyurek, K. Kaya and B. Ucar, "Integrated Data Placement and Task Assignment for Scientific Workflows in clouds," In Proceedings of the fourth international workshop on Data-intensive distributed computing, pp. 45-54, 2011.

[5] J.D. Montes, M. Zou, R. Singh, S. Tao, and M. Parashar, "Data-driven workflows in multi-cloud marketplaces." In Cloud Computing, 2014 IEEE 7th International Conference on, pp. 68-175. IEEE, 2014.

[6] E. Deelman and A. Chervenak, "Data Management Challenges of Data-Intensive Scientific Workflows," In Cluster Computing and the Grid. CCGRID'08. 8th IEEE International Symposium on, pp. 687-692, 2008.

[7] Y. Zhao, I. Raicu, X. Fei and S. Lu, "Opportunities and Challenges in Running Scientific Workflows on the cloud," In Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), International Conference on, pp. 455-462, 2011.

[8] Z. Er-Dun, Q. Yong-Qiang, X. Xing-Xing and C. Yi, "A Data Placement Strategy Based on Genetic Algorithm for Scientific Workflows," In Computational Intelligence and Security (CIS), Eighth International Conference on, pp. 146-149, 2012.

[9] I. Foster, Y. Zhao, I. Raicu and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared," In Grid Computing Environments Workshop, 2008. GCE'08, pp. 1-10, 2008.

[10] G. Juve and E. Deelman, "Scientific workflows and clouds," Journal of Crossroads, vol. 16, no. 3, pp. 14-18, 2010.

[11] A. K. Kayyoor, A. Deshpande and S. Khuller, "Data Placement and Replica Selection for Improving Co-location in Distributed Environments," arXiv preprint arXiv:1302.4168, 2013.

[12] T. Kosar and M. Livny, "A Framework for Reliable and Efficient Data Placement in Distributed Computing Systems," Journal of Parallel and Distributed Computing (JPDC), Vol.65, no. 10, pp. 1146-1157, 2005.

[13] A. Chervenak, E. Deelman, M. Livny, M. Su, R. Schuler, S. Bharathi, G. Mehta, and K. Vahi, "Data Placement for Scientific Applications in Distributed Environments," In Proceedings of the 8th IEEE/ACM International Conference on Grid Computing, pp. 267-274, 2007.

[14] D. Yuan, Y. Yang, X. Liu and J. Chen, "A data placement strategy in scientific cloud workflows," Future Generation Computing Systems 26, no. 8, pp. 1200-1214, 2010.

[15] D. Yuri, P. Membrey, P. Grosso and C. Laat, "Addressing Big Data Issues in Scientific Data Infrastructure," In Collaboration Technologies and Systems (CTS), 2013 International Conference on, pp. 48-55, 2013.

[16] Mahdi Ebrahimi, Aravind Mohan, Andrey Kashlev, and Shiyong Lu, "BDAP: A Big Data Placement Strategy for Cloud-Based Scientific Workflows", in Proceedings of the First IEEE International Conference on Big Data Computing Services and Applications, pp.105-114, 2015.

[17] Andrey Kashlev and Shiyong Lu, "A System Architecture for Running Big Data Workflows in the Cloud", in Proceedings of the IEEE International Conference on Services Computing (SCC), pp.51-58, 2014.

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

[19] H. Wenwey, Y. C. Huang, S. C. Hsu, and C. Pu. "Real-time collaborative planning with big data: Technical challenges and in-place computing." In Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom), 9th International Conference Conference on, pp. 96-104, 2013.

[20] M. B. Blake, D. J. Cummings, A. Bansal, and S. K. Bansal, "Workflow composition of service level agreements for web services." Decision support systems 53, no. 1, pp. 234-244, 2012.

[21] Z. Shen, and J. Su, "On completeness of web service compositions." In Web Services. IEEE International Conference on, pp. 800-807, 2007.

[22] M. Huhns, and M. P. Singh., "Service-oriented computing: Key concepts and principles." Internet Computing, IEEE 9, no. 1, pp. 75-81, 2005.

[23] C. K. Chang, M. J. Christensen, and T. Zhang, "Genetic algorithms for project management." Annals of Software Engineering 11, no. 1, pp. 107-139,2001.