1-1-2017

Data Placement And Task Mapping Optimization For Big Data Workflows In The Cloud

Mahdi Ebrahimi
Wayne State University,

DATA PLACEMENT AND TASK MAPPING OPTIMIZATION FOR BIG DATA WORKFLOWS IN THE CLOUD

by

MAHDI EBRAHIMI

DISSERTATION

Submitted to the Graduate School

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

2017

MAJOR: COMPUTER ENGINEERING

Approved By:

_____

Advisor Date

_____

_____

_____

_____

ACKNOWLEDGEMENTS

First and foremost, I would like to express my gratitude to Allah (God) for providing me the blessings to complete my Ph.D. studies. I would also like to express my deepest sense of gratitude to my advisors Dr. Shiyong Lu and Dr. Song Jiang. My sincere thanks goes to Dr. Lu for his patient supervision, generous encouragement and excellent advice throughout my years at Wayne State University. I have also benefited tremendously from his wisdom towards life in general. His guidance helped me in all the time of research and teaching. I could not have imagined having a better advisor and mentor for my Ph.D. studies.

Besides my advisors, I would like to thank the rest of my dissertation committee members: Dr. Alexander Kotov, Dr. Fengwei Zhang and Dr. Caisheng Wang, for providing their insightful comments, suggestions and encouragements.

I would also like to thank Dr. Farshad Fotouhi, Dr. Loren Schwiebert and Dr. Robert Reynolds for their encouragements and supports during my Ph.D. studies. I would also like to thank the Testing, Evaluation and Research Services Office at Wayne State University for providing me the Graduate Student Assistant position and finically supporting me to continue my education toward Ph.D. I would also like to thank my bright colleagues from the Big Data Research Laboratory. I would like to express my deep appreciation to Dr. Aravind Mohan for the strong collaboration and friendship that has resulted in several research publications. It is great privilege to work with Dr. Mohan, as he leaded the implementation of the DATAVIEW big data workflow management system.

Further, our collaboration has helped me to develop this dissertation work.

I am especially thankful to my lovely wife who has been incredibly supportive throughout my studies. My deep gratitude goes to my loving parents, my sister, my brothers and my entire family in Iran.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS ..... **II**

1 INTRODUCTION **1**

2 RELATED WORK **6**

2.1 Big Data 6

2.2 Data-centric Workflows vs. Business Workflows 8

2.3 Big Data Workflows 10

2.4 Data and Task Placement in Workflows 13

2.5 Workflow Scheduling 16

3 DATA PLACEMENT IN BIG DATA WORKFLOWS **19**

3.1 Introduction 19

3.2 Workflow Data Placement Model 20

3.3 Workflow Data Placement Algorithm-BDAP 31

3.4 Experiments and Discussion 35

3.4.1 Simulation Setting 36

3.4.2 Results 38

3.5 Data Placement Algorithm in DATAVIEW 41

4 TASK PLACEMENT IN BIG DATA WORKFLOWS **44**

4.1 Introduction 44

4.2 Workflow Task Placement Model 47

4.3 Workflow Task Placement Algorithm-TPS 53

4.4 Experiment and Case Study 55

4.4.1 Case Study 55

4.4.2 Implementation 58

4.4.3 Results 60

5 TASK SCHEDULING IN BIG DATA WORKFLOWS 62

5.1 Introduction 62

5.2 System Model 64

5.3 The BORRIS Algorithm 72

5.4 Experimental Results 78

5.4.1 Performance Evaluation 78

5.4.2 Results and Analysis 80

6 CONCLUSIONS AND FUTURE WORK 83

APPENDIX A: SUPPORT VECTOR MACHINE (SVM) P-WORKFLOW 88

Workflow Specification 88

Workflow Java Source Code 89

APPENDIX B: RANDOM FOREST P-WORKFLOW 92

Workflow Specification 92

Workflow Java Source Code 93

APPENDIX C: BAYESIAN NETWORK P-WORKFLOW 96

Workflow Specification..... 96

Workflow Java Source Code..... 97

REFERENCES..... 99

ABSTRACT..... 114

AUTOBIOGRAPHICAL STATEMENT..... 116

LIST OF TABLES

Table 3-1 Symbols and notations.....25

Table 3-2 Description of dataset and virtual machine of the experiment. 38

Table 3-3 Default setting for the BDAP algorithm.....38

Table 3-4 Some results of the BDAP running. 43

Table 4-1 Description of Task and virtual machine of the experiment. 57

Table 4-2 Default setting for the TPS algorithm. 57

Table 4-3 Some results of applying the TPS for the execution of workflow in Example1. 59

Table 4-4 OpenXC workflow of one car driver running in DATAVIEW.....60

Table 5-1 a) Cloud resource catoulge, b) Task computation cost c) Data communication cost, d) Initial budget allocation and e) Final budget allocation.....75

Table 5-2 Workload details for OpenXC workflow. 80

LIST OF FIGURES

Figure 2-1 Five V's of Big Data. 7

Figure 2-2 Architecture for DATAVIEW as a Big Data Workflow Management System. 11

Figure 3-1 A workflow with five tasks $t1, t2, t3, t4, t5$ and five datasets $d1, d2, d3, d4, d5$. The output of task ti is denoted by d'$_i$. The input datasets of task t1 are $d1, d3$, t2 are $d'1, d2, d3$, t3 are $d'1, d2, d4, t4$ are $d'2, d'3, d2, d4, d5$ and t5are $d'4, d5$. 22

Figure 3-2 A virtual machine configuration in the Cloud with three virtual machines for workflow of Example I. Datasets $d1, d3$ and tasks $t1, t2$ were placed and assigned in VM_1. Datasets $d2, d4$ and tasks $t3, t4$ were placed and assigned in VM_2. Similarly, dataset d5 and tasks t5 was placed and assigned in VM_3.....24

Figure 3-3 Flowchart of BDAP.....32

Figure 3-4 The structure of five realistic data-centric workflows [48].....37

Figure 3-5 Workflow Communication Cost by varying the number of datasets. 39

Figure 3-6 Workflow Communication Cost by varying the number of virtual machines……...……40

Figure 3-7 Workflow Communication Cost by varying the percentages of fixed-location datasets and for fixed number of workflow nodes, 1000, and datacenters, 50..41

Figure 4-1 a) Workflow of Example 1 b) Data placement c) Task placement. 46

Figure 4-2 OpenXC Workflow for Comparing Three Car Drivers. 58

Figure 4-3 Workflow Communication Cost (hours) by varying the number of workflow tasks……..61

Figure 4-4 Workflow Communication Cost (hours) by varying the number of virtual machines……...61

Figure 5-1 BORRIS flowchart. 72

Figure 5-2 Workflow example with seven tasks and ten data dependencies.....74

Figure 5-3 a) Resource utilization; b) Execution cost minimization. 81

1 INTRODUCTION

A workflow loosely use to define a sequence of connected either single-step or multiple-step tasks as long with their dependencies to model and computerize business processes. Workflow task is a description of the activity of an individual person or team within an organization or independently. Task dependency illustrates the flow of product or file that is transferred from one task to another task to complete the process. Although workflow technology rooted back to 1970s and mostly used for business processes, data-centric workflow proposed by Vouk et al., in 1996 [1] for solving scientific research problems by applying workflow techniques. Since then and with the advancement of IT technology and e-science, data-centric workflow turns into an essential technology for scientists and researchers to explore and test their hypothesis.

Data-centric workflows are formal description of scientific processes to represents and computerizes the scientific computational steps that scientists design to verify their scientific hypothesis [2, 3]. Data-centric workflows have been extensively employed in various data-intensive scientific areas such as bioinformatics, physics, astronomy, ecology and earthquake science [4]. They are usually modeled as directed acyclic graphs (DAGs) such that workflow tasks are represented by graph nodes and the data flow among tasks are represented by graph vertices. The direction of vertices shows the flow of the data among tasks. Scientists typically required modeling their hypothesis and analysis it with various collected data. They usually design a model of their initial hypothesis

and then try to refine it by repeatedly re-using the model against their collected data. Therefore, reproducibility is a key requirement of data-centric workflow management systems.

Due to applying data-centric workflows to formalize and structure the complex scientific research problems, they are potentially very large and comprise of thousands or hundreds of complex tasks and big datasets [5, 6]. They are naturally data-intensive application which the amounts of data used by the tasks are huge and moving the huge data among tasks incredibly increases the execution time of the data-centric workflows. Therefore, this type of applications can benefit from distributed high performance computing (HPC) infrastructures like cluster, grid or cloud computing.

The concept of Could Computing rooted back in 2007 [7] and has been studied as the next generation architecture of IT enterprise by providing cost-effective, scalable, on-demand and elastic provisioning distributed computing infrastructure over the web [8-10] and has been applied in many domains [11-14]. Executing data-centric workflows in the Cloud is a challenging problem as the data-centric workflow tasks and datasets are required to partition, distribute and assign to the execution sites (virtual machines). The advantages of using cloud computing for data-centric workflows are summarized as follows [15-17]: 1) providing large amount of storage space and computing resources; 2) improving resource utilization by allocation the resource accordingly with the number of workflow nodes at each stage; 3) providing a much larger room for

the trade-off between performance and cost.

Although data-centric workflows have been applied extensively to structure complex scientific data analysis processes, they fail to address the big data challenges as well as leverage the capability of dynamic resource provisioning in the Cloud. To address such limitations, the concept of big data workflows is proposed by our research group as the next generation of data-centric workflow technologies.

Besides theoretical, experimental and computational science, the data intensive computing is now viewed as the "fourth paradigm" in scientific research area [18]. According to Brewer's C.A.P. (Consistency, Availability and Fault tolerance) theorem [19, 20], a distributed system like Cloud Computing cannot satisfy *Consistency, High-Availability and Partition-tolerance* of dataset inside cloud datacenter simultaneously**.** So by having all the advantages and opportunities of cloud computing for executing data-centric workflows, several challenges raise such as managing required big dataset of workflows in a consistent and scalable way is a challenging problem [21].

Data management is typically more critical than the other resource management in Cloud Computing Infrastructure such that separate nodes allocate for just data storage [22]. As scientific applications, become more and more data intensive, managing data in large distributed systems like cloud computing needs to come up with an efficient data and task placement strategy. The Placement strategy needs to maximize data locality and minimize data

movement among virtual machines in the Cloud. Such that once the workflow tasks are partitioned and assigned to a virtual machine, its most required datasets are already stored at the same virtual machine.

In big data workflows, it is practically impossible to store all of the required dataset of tasks in one virtual machine due to the storage capacity limitation of virtual machines and the dataset movement is inevitable to execute big data workflows in the Cloud. Beside the storage limitation of individual virtual machine, there is a need to have multiple machines to enable parallel computing and exploit more computing power. In addition, it reduces the cost of the computation by using a network of commodity machines instead of a supercomputer.

In the topic of data management of workflows, the assumption is that it is often more efficient to migrate the computation job, workflow task, closer to where the data is located rather than moving the data to where the application is running. Therefore, the main goal of data and workflow task placement should be to minimize the total data movement as “moving computation to data is often cheaper than moving data to computation” [21, 23, 24].

As discussed above, task and data placement strategy plays a critical role in the successful execution of big data workflows. My dissertation goals are developing rebuts data and workflow task placement strategies for big data workflow running in the Cloud. This is required to come up with a strategy to find an optimal workflow execution plan. I have achieved the following

progresses toward my dissertation research goals:

- We formalized both data and workflow task placement problems in big data workflows.
- We proposed a new data placement strategy that considers both source input dataset and generated intermediate datasets obtained during workflow run.
- We proposed a task placement strategy that considers placement of workflow tasks before workflow execution. Our proposed workflow tasks placement into the available virtual machines is based on their required placed datasets.
- We proposed a workflow scheduling strategy that maps the workflow tasks into cloud virtual machines in design time. We considered one sub-problem of the general big data workflow scheduling problem, in which a deadline D is given for a workflow W, and the goal is to minimize the monetary cost of running W in the cloud while satisfying the given deadline.

The dissertation is organized as follows: Chapter 2 introduces the related work about data placement, task mapping optimization and workflow scheduling of big data workflows in the Cloud. Chapter 3 presents our work on data placement in big data workflows. Chapter 4 presents our work on task placement in big data workflows. Chapter 5 presents our work on workflow scheduling. Finally, Chapter 6 concludes the dissertation and presents the future works.

2 RELATED WORK

2.1 Big Data

The National Institute of Standards and Technology (NIST) [25], which is leading the development of a Big Data technology roadmap, has proposed the first version of definition of Big Data as follows [26]:

- "Big Data refers to digital data volume, velocity and/or variety, veracity that:
- Enable novel approaches to frontier questions previously inaccessible or impractical using current or conventional methods; and/or
- Exceed the capacity or capability of current or conventional methods and systems."

Although the above definition is not completed yet we can describe that Big Data loosely applies for complex and huge datasets which are difficult to be managed by using traditional data management tools such as Relational Database Management Systems (RDBMS). Big Data are naturally distributed and placed on different sources over the Internet. These data need to be collected, distributed and/or replicated therefore it requires extended and particular strategies and requirements[27].

Data-centric workflow typically models and analyzes complex scientific research experiments, which normally contain huge volume of datasets. Therefore, Big Data technologies are becoming a main focus in scientific

computing research. Big Data can be defined by 5 characteristics, called 5 V's, as illustrated in Figure 2-1[28].

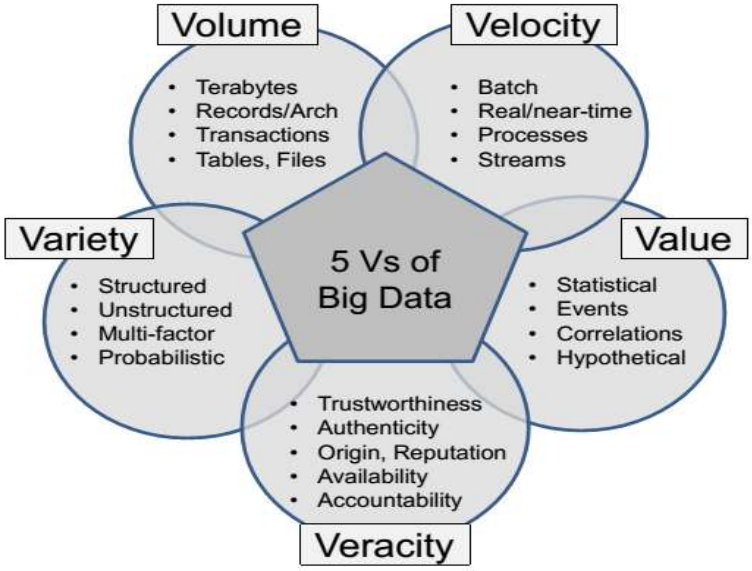


Figure 2-1 Five V's of Big Data.

The description of 5V's is as follows:

- **Volume:** The most important feature of Big Data is volume which is about large volume of datasets including terabytes records, transactions, tables or files. Based on IBM research [29] about 800,000 petabytes (PB) of data were stored in the year 2000 over the word and they expect this amount to reach around 35 zettabytes (ZB) by 2020. For example, Twitter generates more than 7 terabytes (TB) of data every day and Facebook 10 TB.

- **Value:** Value is about derived value from data. New advancements in IT technology bring the capability of collecting and accessing huge amounts of information and datasets not only by human beings, but also by computers and machines. So, getting meaningful values form collected big data is a main

concern for scientists.

- **Velocity:** Velocity applies to the enormous volume of data that comes in/out with high speed from different sources. This type of generated data regularly need to be processed in real-time, or in batch or as a stream.
- **Variety:** Variety is about integrating different data formats and large number of diverse data sources. This is result in data collected as structured like relational table or unstructured like images and videos or semi-structured like html pages and text or mixed data. Data can origin from a different number of sources and/or devices such as online/offline social media, mobile, satellite, sensors, cameras, TV etc.
- **Veracity:** Veracity is related to data consistency/certainty and data trustworthiness that can be applied to various stages of data management like data searching stage, data collecting stage and data processing stage. This feature of Big Data guarantees the trustworthiness, authentication and protection of collected datasets against unauthorized data accesses and manipulations.

Our study is about the volume aspect of big data and how it operates to partition, distribute and place huge size of datasets including tables and files into cloud datacenters.

2.2 Data-centric Workflows vs. Business Workflows

Workflows have been intensively applied to business organizations to analyze and model their business processes from 1970s. After that, data-centric workflows were proposed to analyze and model scientific hypotheses and

improve scientific experiments to get scientific principles. Although business and data-centric workflows have the same origin to model and execute business/ scientific processes, they also have much dissimilarity. Their differences are in their requirements, characteristics, and life cycles. The differences can be categorized as follows:

- **Scientific goal vs. Business goal:** The goal of data-centric workflows is to increase the speed of unpredictable scientific discovery and therefore reduce human and computation costs. On the other hand, the goal of business workflows is increasing revenue and profit of the enterprise and therefore reducing human resources.
- **Dataflow oriented vs. Control flow oriented:** Data-centric workflows are naturally data-flow oriented and the control dependency of tasks is not a concern. But business workflows are often control-flow oriented and the control dependency and the coordination of tasks is the main concern.
- **Reproducible vs. Non-reproducible:** Data-centric workflows required to be reproducible as they can be used by scientists to test their scientific ideas. Therefore, it is critical that other scientists be able to reproduce the same workflow to verify the correctness of their hypothesis. But, business workflows do not need to be reproducible as they model well-formed business processes.
- **Mutable vs. Immutable:** Data-centric workflows naturally require to be modified frequently as they are used in trial manner by scientists. The reason is that the scientific ideas and hypothesis are changed frequently.

However, business workflows rarely need to be changed, since the business processes are consistent and well-defined in most cases.

2.3 Big Data Workflows

Big Data Workflows have been recently proposed as the next generation of data-centric workflows to address the challenges of big data analytics including volume, value, velocity, variety and veracity as well as execution challenges in the Cloud [30, 31].

Big Data workflows mainly can use the beneficiary of cloud computing and execute by different number of virtual machines in parallel and therefore big data can be horizontally scalable. It means we are able to add more virtual machines into the pool of resources once there is a need to have more resources to manage and analyze datasets. In the despite, data management is vertically scalable in big data workflows that mean adding more power and computation (CPU, RAM and …) to the server of executing workflows.

Horizontal-scaling is through partitioning and vertical-scaling is through multi-core support [32]. In term of data storage layer of big data workflows, horizontal-scaling is based on partitioning of the datasets such that each virtual machine hosts only portion of the datasets, in vertical-scaling the datasets resides on a single node (server) and scaling is achieved through multi-core i.e. spreading the load between the CPU and RAM resources of that machine. For example Apache Cassandra [33], MongoDB [34] apply horizontal scaling and MySQL-Amazon RDS (The cloud version of MySQL) applies vertical scaling

by switching from small to bigger virtual machines. In the remains of this section, we introduce our proposed concept, big data workflow, as the next generation of data-centric workflow.

Several data-centric workflow management systems have been proposed within using cloud computing environment; however, they are not generic and domain-independent. Some of them are developed in a specific domain like bioinformatics [35, 36] or astronomy [37], some are designed with applying different type of QoS constrains [38] and the others are a particular type of workflows like workflows with data parallelism [39]. Our research group has been proposing a generic and implementation-independency big data workflow called DATAVIEW as depicted in Figure 2-2 [30].

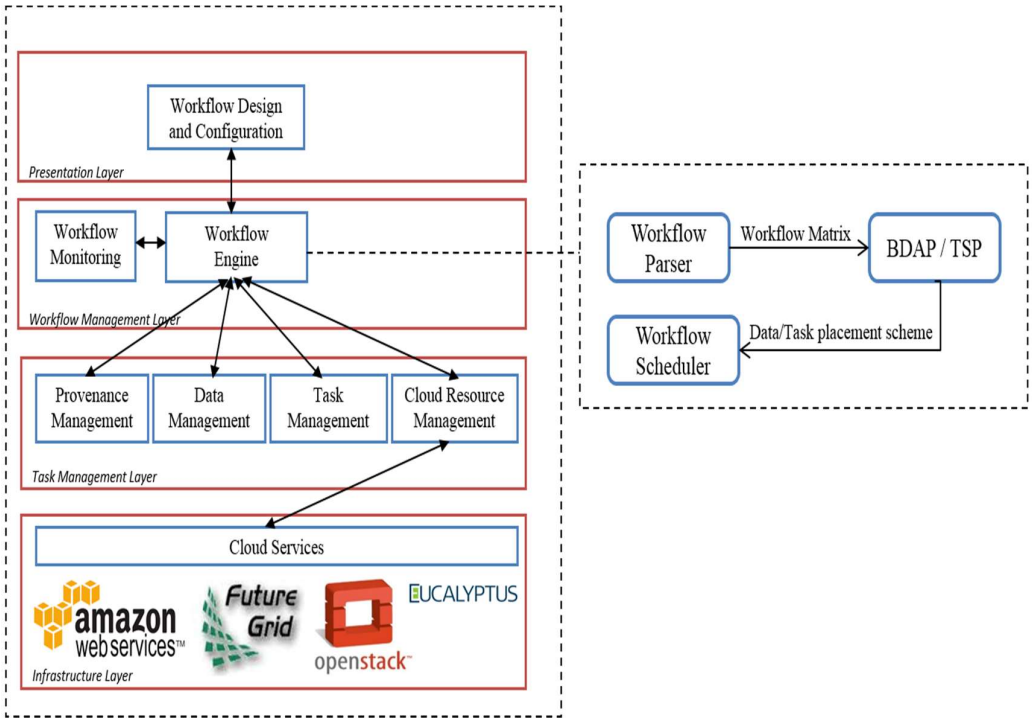


Figure 2-2 Architecture for DATAVIEW as a Big Data Workflow Management System.

Our proposed big data workflow, DATAVIEW, contains four layers as follows:

- Presentation Layer. This layer is the client-side of DATAVIEW and includes two major components:
  - Workflow Design and Configuration that provides graphical user interface (GUI) utility for the end users to design manipulate and save their workflows. In addition, workflow configuration offers capability to the scientists to specify the settings related to the Cloud as the execution environment. Cloud settings like selection the cloud providers (e.g., OpenStack Cloud Software [40], Amazon AWS EC2 [41] , FutureSystems [42]), specifying the number of virtual machines to execute the workflow.
- Workflow Management Layer. This layer contains two subsystems.
  - Workflow engine which is the core component of workflows. It executes the workflows and orchestrates the movement of the data flow between tasks within different virtual machines. Figure 2-2 shows its main components.
  - Workflow Monitoring. It is applied to keep track of each workflow entities like takes and data products within workflow execution.
- Task Management Layer. This layer is built on the top of the Infrastructure layer (Cloud Services layer) to collaborate and execute of workflow tasks in the Cloud. It contains four major components as follows:

- Task Management. It provides utilities to execute individual workflow tasks. Workflow tasks can be heterogeneous that means can be a built-in, web service, a script and so on.
- Data Product Management. It manages both source and generated intermediate data products.
- Provenance Management. Provenance is the history information about workflow data products in details within executing the workflow to allow reproducibility. Provenance management provides utilities to store browse and query workflow provenance.
- Cloud Resource Management. This component offers cloud resource allocation, provisioning, mapping, discovery, configuring, estimation and terminating.

- Infrastructure Layer. The Infrastructure Layer contains the underlying Infrastructure as a Service (IaaS) cloud platforms where workflows are submitted to execute. DATAVIEW applies the “all-in-the-cloud” approach [15] to run Big Data Workflow Management System.

2.4 Data and Task Placement in Workflows

Previous research studies for distributed computing environment have been mainly focused on the performance and optimization of job scheduling and task allocation. But due to the rapid increase in the size of available data over the internet and the emerging field of Big Data, data placement becomes a fundamental spot in the Cloud recently. Kosar et al., [6, 43] proposed a

framework for distributed computing systems which considered the data placement subsystem as an independent module along with computation subsystem. In their proposed model data placement jobs can be queued, scheduled, monitored, managed and even checkpointed. Kayyoor et al., [44] considered the data placement and replication problems together for the distributed environments. They claimed minimizing of query latencies is not a critical issue in many scenarios of analytical workloads and so they tried to minimize the average number of using computation nodes by grouping the most interdependent data together based on their occurrences of the common query accesses. Chervenak et al., [45] explored the advantages of separation of data placement as a service from workflow management systems. By applying an autonomous data placement service along with data replication service, they evaluate and display the benefits of pre-staging data compare to the data stage in and out strategies of Pegasus workflow management systems. However, none of the above studies decreasing the data movement among cloud virtual machines.

By advent of cloud computing, new data management systems are developed. For instances Google File System (GFS) [46] and Hadoop Distributed File System (HDFS) [47] are developed to provide of data access on remote servers by means of huge clusters of commodity hardware. GFS is developed by Google for its engine search but HDF is more general which has been used by many companies like Facebook and Amazon. The data placement in HDFS is straightforward as once it is pushed a file into HDFS, it splits the file

into one or more chunks and stores them in a set of distributed datanodes randomly. HDFS also applies replication technique to improve the performance. In addition, some of the workflow management systems have been extended to execute data-centric workflow in clouds. Pegasus [48-50] is designed to execute data-centric workflows on number of distributed resources such as local machines, clusters or cloud. Nimbus [51] is an integrated set of tools which allows scientific users to deploy a cluster into infrastructure clouds to execute their data-centric workflows. Eucalyptus [52] is an open source cloud management software to create on-demand, self-service private cloud resources.

In Catalyurek et al., [53] workflows were modeled by hypergraph concept and a hypergraph portioning technique, k-way partitioning, is applied to minimize the cutsize. In that way, they cluster the workflow tasks as well as their required data in the same execution site. One of the closet works to our data placement strategy is Yuan et al., [22] which they applied a greedy binary clustering to precluster datasets; then they greedily assigned the workflow tasks to an execution site that contains the most of the input datasets. At the end once an intermediate dataset was generated, they placed it to the execution site which has the most interdependent dataset. Although their approach placed the most interdependent dataset together and can reduce data movement, the algorithm is greedy and it clustered the data dependency matrix into two parts in each iteration and so their clustering technique was sensitive to the selection point in any iteration.

The other close work to our study is Er-Dun et al., [54] in which they applied genetic algorithm to find their data placement solution along with load balancing factor. Their approach reduced data movement however they did not consider data interdependency between datacenters and also they did not consider task assignment. In addition, they used mean measurement for the load balancing factor but harmonic mean is a more accurate measurement for the load balancing factor.

2.5 Workflow Scheduling

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 have been focusing on different aspects of the scheduling problem based on the various QoS requirements. One of the most recent work is [55] 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 [56-58] some other scheduling algorithms were proposed to minimize the execution cost with deadline constraints for the

Grid utility systems. In [59, 60] the authors considered both budget and makespan as the QoS constraints, but did not use an objective function to minimize them.

Lin et al. [61, 62] 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 [63-66], 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 [67-69], 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.

Workflow scheduling in cloud computing is known as NP-hard problem. The reason is that there is usually a large search space of solutions and it takes a long time to find an optimal solution. Therefore, there is no scheduling algorithm to produce optimal solution within polynomial time. In big data workflow domain, it is sometimes preferable to find a suboptimal solution, but in a short period of time. To achieve near optimal scheduling solutions within reasonable time, Metaheuristic-based approaches have been proposed [70-74]. Some of the popular meta-heuristic techniques are Genetic Algorithm (GA), Ant Colony Optimization (ACO), and Particle Swarm Optimization (PSO).

In our previous works [75-77], 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 one of our recent work [78], we propose a new big data workflow scheduler under budget constraint (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.

3 DATA PLACEMENT IN BIG DATA WORKFLOWS

3.1 Introduction

The makespan of a data-centric workflow [79-81] is the time elapsed between the start of the first task and the completion of the last task in the workflow, including the delivery of the final data product to the desired place. In big data workflows, makespan vary greatly depending on how the tasks and datasets are allocated in the distributed computing environment like Clouds. Incorporating a data and task allocation strategy to minimize the makespan in a big data workflow can deliver significant benefits to users in getting their results in time [76].

This dissertation provides a formal definition of the data movement minimization problem of big data workflows running in a distributed environment and proposes efficient data and workflow tasks placement strategies, BDAP and TPS.

Regarding to data placement in big data workflows, we propose BDAP, an evolutionary algorithm (EA) which is a generic population-based metaheuristic optimization strategy [82]. The main goal is to minimize the dataset movement between virtual machines during the execution of a workflow under the constraint of virtual machine storage capacity

Example 1. Let's consider an example to show how a workflow can be

executed in a cloud computing environment. Figure 3-1 illustrates a sample workflow with five tasks, five original datasets and five generated intermediate datasets [23]. Figure 3-2 shows an instance of its virtual machines configuration in the Cloud. In this example, tasks t_1 and t_2 as well as datasets, d_1 and d_3 were assigned to virtual machine 1, VM_1. Similarly, tasks t_3 and t_4 were assigned to VM_2 as well as datasets, d_2 and d_4. Once we execute the workflow, tasks t_2 needs transferring dataset d_2 from VM_2 to VM_1 to complete its process. However, there is no need to move any other original datasets from other virtual machine to VM_2 to run task t_3 because all its required original datasets, d_2 and d_4 are already placed in VM_2. Furthermore, t_3 only required transferring the output of task t_1, d_1 from VM_1 to VM_2 in the run-time stage.

Please note that the workflow scheduling [83-87] is out of the scope of this dissertation proposal. BDAP does not apply any specific strategy for the order (either sequential or parallel) execution of workflow tasks. BDAP can be used by any current workflow scheduling algorithms to improve the workflow throughput. In this dissertation, we simply execute workflow tasks in a sequential order to evaluate BDAP.

3.2 Workflow Data Placement Model

To model cloud computing environment, we consider I distributed virtual machines in the Cloud as the execution sites. Each virtual machine can be provided by different Cloud Computing Providers (CCP) such as Amazon EC2, Google App Engine [88], and Microsoft Azure [89]. Although CCPs normally

have their own data and computation placement strategy to store data and assign computation jobs to proper virtual machines, sometimes users (e.g., scientists) have concerns about their own datasets (e.g., data security or too large data or requirement for specific data processing utilities and equipment). Such users prefer to keep and store their data in a particular virtual machine and not allowed to move their data to the other virtual machines. This type of dataset is called fixed-location datasets.

For addressing these scientific user's concerns and managing fixed-location datasets, users need to have private execution sites or to be able to add their own local computation facilities as virtual machines. In that way, we need to apply a new data placement strategy to address the fixed-location datasets and minimize the total data movement across dedicated virtual machines in the Cloud.

To minimize data movement between virtual machines in the Cloud, we cluster the virtual machines such that the placed datasets have the highest data interdependency within each virtual machine as well as the lowest data interdependency between virtual machines. In the rest of this section, we model our data placement solution in detail. Table 3-1 summarizes all the used symbols and notations in this dissertation.

Big data workflows are executed in Clouds as the execution environment. A Cloud computing environment is modeled as follows:

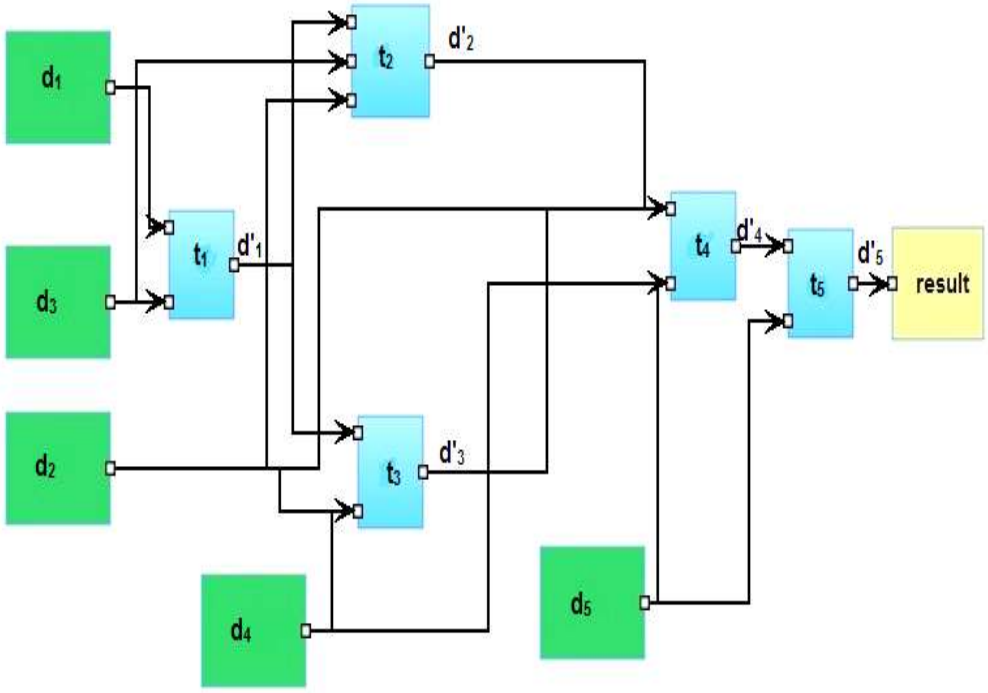


Figure 3-1 A workflow with five tasks $\{t_1, t_2, t_3, t_4, t_5\}$ and five datasets $\{d_1, d_2, d_3, d_4, d_5\}$. The output of task ti is denoted by d'$_i$. The input datasets of task t_1 are $\{d_1, d_3\}$, t_2 are $\{d'_1, d_2, d_3\}$, t_3 are $\{d'_1, d_2, d_4\}$, t_4 are $\{d'_2, d'_3, d_2, d_4, d_5\}$ and $t_5$are $\{d'_4, d_5\}$.

Definition 3.1 (Cloud Computing Environment C). A Cloud computing environment C is a 3-tuple $C = (VM, SC, DTR)$, where

- VM is a set of virtual machine in the Cloud vm_i $(i = 1, 2, \dots, I)$
- $SC: VM \rightarrow R^+$ 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 number.
- $DTR: VM \times VM \rightarrow Q_0^+$ 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 number.

For solving the complex scientific problems, scientists are able to create and run their own workflows simultaneously. Each individual workflow contains a set of tasks that consume various datasets and may produce intermediate datasets as well. Those produced 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 3.2 (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 for 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 for 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 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 number.

- 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 BDAP with the others proposed algorithms Workflow Communication Cost is defined as follows [61, 62]:

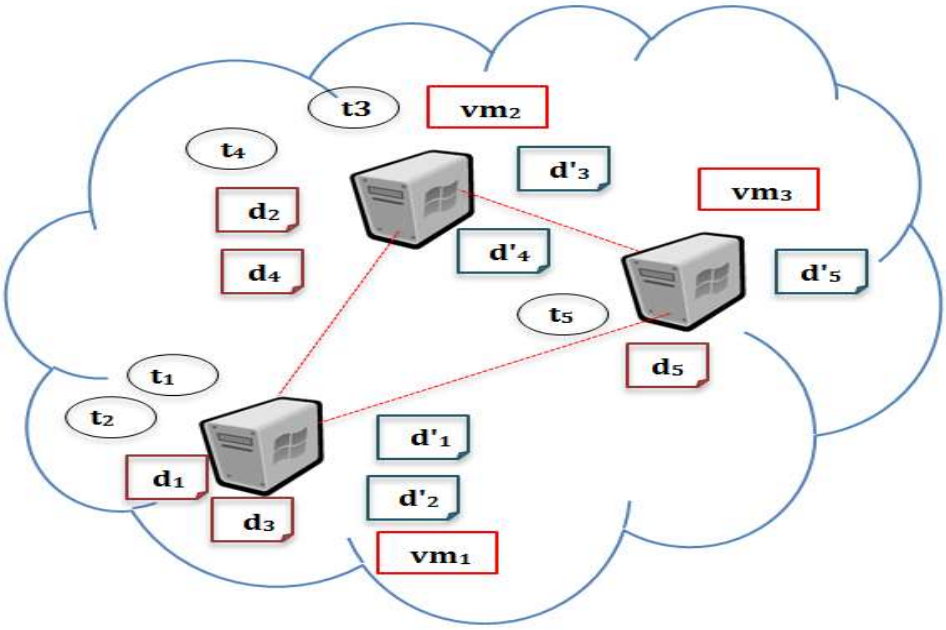


Figure 3-2 A virtual machine configuration in the Cloud with three virtual machines for workflow of Example I. Datasets $\{d_1, d_3\}$ and tasks $\{t_1, t_2\}$ were placed and assigned in VM_1. Datasets $\{d_2, d_4\}$ and tasks $\{t_3, t_4\}$ were placed and assigned in VM_2. Similarly, dataset $\{d_5\}$ and tasks $\{t_5\}$ was placed and assigned in VM_3.

Table 3-1 Symbols and notations.

Notations	Description
$\boldsymbol{VM}$	The set of virtual machines
$\boldsymbol{vm_i}$	The i^{th} virtual machine in VM
$\boldsymbol{SC(vm_i)}$	The storage capacity of virtual machine vm_i
$\boldsymbol{D}$	The set of datasets
$\boldsymbol{D_{fixed}}$	The set of fixed datasets, $D_{fixed} \subseteq D$
$\boldsymbol{d_j}$	The jth dataset in D
$\boldsymbol{S(d_j)}$	The size of dataset d_j
$\boldsymbol{T}$	The set of Tasks
$\boldsymbol{DS(t_k)}$	The set of datasets as the input of task t_k
$\boldsymbol{TS(d_j)}$	The set of tasks which get dataset d_j as the input
$\boldsymbol{DTR(vm_{i1}, vm_{i2})}$	The data transfer rate between two virtual machines, vm_{i1} and vm_{i2}
$\boldsymbol{dp(d_{j1}, d_{j2})}$	The data interdependency between datasets d_{j1} and d_{j2}
$\boldsymbol{tp(t_{k1}, t_{k2})}$	The task interdependency between tasks t_{k1} and t_{k2}
$\boldsymbol{DP}$	The data interdependency matrix of D
$\boldsymbol{TP}$	The task interdependency matrix of D
$\boldsymbol{\Psi}$	The J-element vector of datasets placement scheme which J is the number of workflow datasets.
$\boldsymbol{\Psi(d_j)}$	The virtual machine to which dataset d_j is assigned in the placement scheme Ψ
$\boldsymbol{\Phi}$	The K-element vector of tasks placement scheme which K is the number of workflow tasks.
$\boldsymbol{\Phi(t_k)}$	The virtual machine to which task t_k is assigned in the placement scheme
$\boldsymbol{P}$	The set of data placement schemes
$\boldsymbol{Q}$	The set of task placement schemes

Definition 3.3 (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

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

Given a workflow W and Cloud C, workflow communication cost is equal to the total data movement cost for executing workflow W in C is defines 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 gives the total data movement within executing the whole workflow in the Cloud C. In the remainder of this section, we define and model the problem and our solution. Our solution is based on the clustering technique. The three main concepts in clustering are objects which need to be clustered, clusters and a separation measure to compute the similarity among the objects [90].

In this dissertation, datasets and workflow tasks are considered as the objects and virtual machines in the Cloud are considered as the clusters. The most important concept is defining a good separation measurement to cluster the most similar objects together to meet the objective goal.

The goal of our proposed data is minimizing data movement among virtual machines. Therefore, we consider data interdependency as the separation

measurement. For this, two datasets are interdependent and should be collocated in the same virtual machine if they are simultaneously needed as inputs by many tasks. The definition for the interdependency of a pair of datasets is as follows:

Definition 3.4 (Data Interdependency). We consider the number of common tasks that take a pair of datasets as input to define the data interdependency of the datasets. Data interdependency value is divided by the total number of workflow tasks in order to be normalized in the range of [0 1]. Formally, given two datasets d_{j1} and d_{j2}, the data interdependency is calculated by:

$$dp(d_{j1}, d_{j2}) = \frac{\left|TS(d_{j1}) \cap TS(d_{j2})\right|}{|T|} \qquad (3)$$

For instance, if the set of tasks that consume d_1 is $TS(d_1) = \{t_1, t_2\}$ and d_2 is $TS(d_2) = \{t_2, t_3, t_4\}$ then the data interdependency between d_1 and d_2 is $dp(d_1, d_2) = \frac{|TS(d_1) \cap TS(d_2)|}{|T|} = \frac{|\{t_1, t_2\} \cap \{t_2, t_3, t_4\}|}{|\{t_1, t_2, t_3, t_4, t_5\}|} = \frac{1}{5} = 0.20.$

In this way, two datasets are interdependent once they have at least one common task consuming both of them. Two datasets have a higher interdependency when they are used by more common tasks and the greater the number of common tasks is, the higher is the data interdependency of datasets.

To maximize data locality, it is necessary to pre-cluster the datasets initially. In the first step, we calculate the data interdependency of all the workflow datasets and generate the data interdependency matrix (DM). In the interdependency matrix, rows and columns are the workflow datasets and the

value of interdependency matrix is the data interdependency between two datasets. For instance, data interdependency matrix of workflow in Example1 is as follows:

$$
\begin{array}{c}
\ddots \\
d_1 \\
d_2 \\
d_3 \\
d_4 \\
d_5 \\
d'_1 \\
d'_2 \\
d'_3 \\
d'_4
\end{array}
\begin{array}{c}
\begin{array}{ccccccccc} d_1 & d_2 & d_3 & d_4 & d_5 & d'_1 & d'_2 & d'_3 & d'_4 \end{array} \\
\begin{pmatrix}
0.4 & 0.2 & 0.4 & 0.0 & 0.0 & 0.2 & 0.0 & 0.0 & 0.0 \\
0.2 & 0.6 & 0.2 & 0.4 & 0.2 & 0.4 & 0.2 & 0.2 & 0.0 \\
0.4 & 0.2 & 0.4 & 0.0 & 0.0 & 0.2 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.4 & 0.0 & 0.4 & 0.2 & 0.2 & 0.2 & 0.2 & 0.0 \\
0.0 & 0.2 & 0.0 & 0.2 & 0.4 & 0.0 & 0.4 & 0.2 & 0.2 \\
0.2 & 0.4 & 0.2 & 0.6 & 0.4 & 0.4 & 0.8 & 0.6 & 0.4 \\
0.2 & 0.2 & 0.4 & 0.2 & 0.2 & 0.2 & 0.4 & 0.2 & 0.2 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.2 & 0.0 & 0.2 & 0.0 & 0.2
\end{pmatrix}
\end{array}
$$

BDAP partitions and distributes the original datasets into 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. Data placement scheme is defined to represent the place of each workflow dataset in a virtual machine. A data placement scheme is defined formally as follows:

Definition 3.5 (Data Placement Scheme Ψ). Suppose there are I virtual machines and J datasets, a data placement scheme is represented by a J-element vector Ψ such that $\Psi(d_j)$ indicates the virtual machine to which d_j is placed. For example the data placement scheme of Example I is $\Psi = (1, 2, 1, 2, 3, 1, 1, 2, 2)$ and it means datasets d_1, d_3, d_1^* and d_2^* are placed in virtual machine vm_1 ($\Psi(d_1) = \Psi(d_3) = \Psi(d_1') = \Psi(d_2') = vm_1$), datasets d_2, d_4, d_3' and d_4' in virtual machine vm_2 ($\Psi(d_2) = \Psi(d_4) = \Psi(d_3') = \Psi(d_4') = vm_2$) and the

dataset d_5 in virtual machine vm_3 ($\Psi(d_5) = vm_3$).

Definition 3.6 (Fixed-Location Datasets D_{fixed}). Given the set of datasets, fixed-location datasets $D_{fixed} \subseteq D$ is a subset of D such that they have pre-determined allocations and cannot be moved. Formally suppose $D_{fixed} = \{d_{j1}, d_{j2}, \dots, d_{jm}\} \subseteq D$ then

$$\Psi(d_{j1}) = vm_{i1}, \Psi(d_{j2}) = vm_{i2}, \dots and\ \Psi(d_{jm}) = vm_{im}\ \{vm_{i1}, vm_{i2}, \dots, vm_{im}\} \subseteq VM$$

the other datasets, $D - D_{fixed}$, are called flexible.

We consider all the workflow tasks are flexible and there are no fixed tasks because moving computation task to datasets is often cheaper than moving datasets to computation task nodes. To define a good measurement to compare separation between virtual machines, data interdependency within and between virtual machines are defined as follows:

Definition 3.7 (Within-VirtualMachine Data Interdependency VMD_W).

$$VMD_W(\Psi) = \sum_{i=1}^{I} \sum_{\substack{\Psi(d_{j1}) = vm_i \\ \Psi(d_{j2}) = vm_i}} dp(d_{j1}, d_{j2}) \quad (4)$$

Where $dp(d_{j1}, d_{j2})$ is the data interdependency between task $d_{j1}$and d_{j2} , I is the maximum number of virtual machines in the Cloud.

Definition 3.8 (Between-VirtualMachine Data Interdependency VMD_B).

$$VMD_B(\Psi) = \sum_{\substack{i_1 \neq i_2 \\ (i_1,i_2)\in(I,I)}}^{(I,I)} \sum_{\substack{\Psi(d_{j1})= vm_{i_1} \\ \Psi(d_{j2})= vm_{i_2}}} dp(d_{j1}, d_{j2}) \quad (5)$$

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

Definition 3.9 (Data Interdependency Greedy DG). The DG heuristic biases BDAP to select the data placement scheme with higher data interdependency. It is defined as:

$$DG(\Psi) = \frac{VMD_W(\Psi) + 1}{VMD_B(\Psi) + 1} \quad (6)$$

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

In our system model, we consider two types of system constraints in terms of data which are defined as follows:

Definition 3.10 (Data Placement Scheme Legality Constraints). Two types of illegal data placement schemes are considered in BDAP:

- Virtual machine storage capacity constraint: The total amount of

placed datasets into a virtual machine should be less than the available storage capacity of the virtual machine as it is impossible to fit all those datasets into the same virtual machine.

- Non-replication constraint: Once a dataset is placed into a specific virtual machine, it is not allowed to place it into another virtual machine as data and task replication is not in the scope of this version of BDAP.

Definition 3.11 (Data Placement Solution). The data placement solution for big data workflow, W, to execute in a cloud computing environment, C, is to select a data placement scheme $\Psi \in P$ to minimize the workflow communication cost (WCC) under the virtual machine storage capacity and non-replication constraints. In the next section, we explain our data placement strategy, BDAP, in detail.

3.3 Workflow Data Placement Algorithm-BDAP

The main goal of BDAP is to minimize workflow communication cost by minimizing the data movement between virtual machines in the Cloud within running a workflow. The main steps of BDAP which applies in design-time are depicted in Figure 3-3. BDAP starts with calculating the data interdependency matrix. Then, it generates a set of legal data placement schemes randomly and calculates their heuristic values. In the following, for each data placement scheme, BDAP applies three main operators, Selection, Crossover, and Mutation sequentially to generate possibly better schemes with higher heuristic values. At the end, the best observed data placement scheme is recorded in Ψ_{best} and will

be returned as the output of BDAP.

Selection, crossover and mutation operators are defined as follows:

Definition 3.11 (Selection SE). Selection is the process of choosing two schemes for recombination and generation two new schemes. There are many methods to perform selection. We use the Roulette Wheel Selection techniques for BDAP.

In this selection operator, the probability to choose a certain scheme is proportional to its heuristic value.

Definition 3.12 (Crossover CO). This operator combines two selected schemes to reproduce two new schemes. The idea is that the new generated schemes may be better and have higher heuristic value if they take the best characteristics from their parent schemes. For instance, suppose $\Psi_{l1} < 1, 2, 1, 2, 3 >, \Psi_{l2} < 2, 2, 1, 3, 1 >$ and the selected row number to crossover is 3 then $\Psi'_{l1} < 1, 2, 1, 3, 1 >$ and $\Psi'_{l2} < 2, 2, 1, 2, 3 >$.

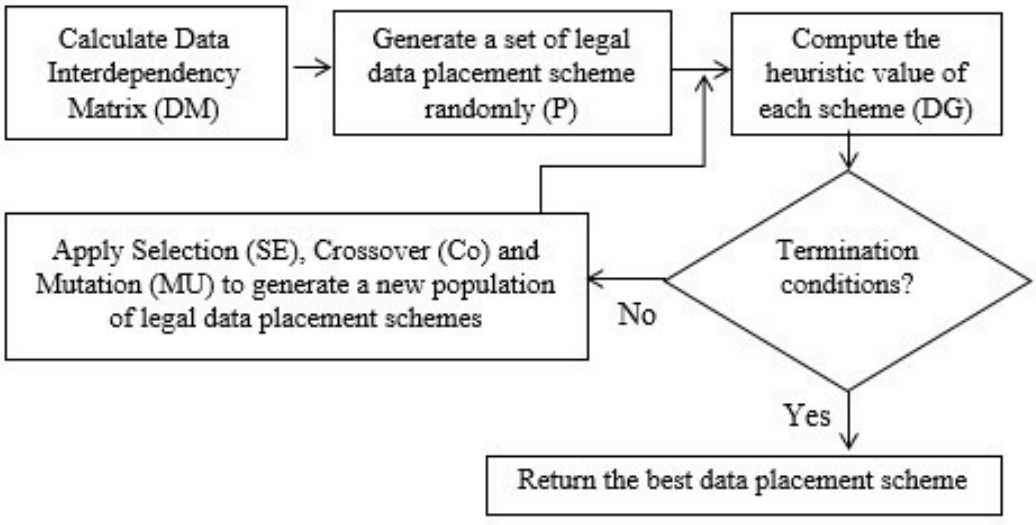


Figure 3-3 Flowchart of BDAP.

Definition 3.13 (Mutation MU). After crossover, BDAP applies mutation operator to an individual scheme to generate a new version of it such that a virtual machine position in the scheme have been randomly changed. Mutation prevents BDAP to be trapped in a local maximum heuristic value. For example, suppose $\Psi_1 < 1, 2, 1, 2, 3 >$ and the select row number is 4 and generated randomized number for position 4 is 3 then $\Psi'_1 < 1, 2, 1, 3, 3 >$.

For applying data placement strategy and analyzing the data interdependency, the whole workflow must be designed. It means all tasks and datasets of the big data workflow must be specified. The BDAP algorithm is outlined in Algorithm 1.

In the first step, BDAP generates popsize number of feasible and valid data placement schemes randomly with the locations for fixed-location datasets fixed. It also calculates the heuristic value of each individual scheme (lines1-5). The position numbers of the fixed-location datasets in the generated data placement scheme is fixed and will not change through the whole algorithm.

In the next steps, BDAP applies three main operators to generate new schemes with a hopefully higher heuristic values until it reaches the max number of iterations. First, it selects ne = popsize × elitism_rate number of scheme 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 schemes to guarantee the convergence of BDAP. We apply the fitness proportionate selection, roulette wheel selection, for this step. The idea behind the roulette

Algorithm 1. Big Data Placement (BDAP).

Input:

D: set of workflow datasets,
DP: data interdependency matrix,
popsize: size of population,
er: rate of elitism,
cr: rate of crossover,
mr: rate of mutation,
num_iteration: number of iterations,

Output:

The best data placement scheme, Ψ_{best}

1. **Begin**
2. **for** *i = 1* **to** *popsize* **do**
3. $\Psi \leftarrow$ Generate a legal data placement scheme randomly;
4. $Pop \leftarrow < \Psi,\ DG(\Psi) >$;
5. **end for**
6. *idx = 0;*
7. **while** (*idx* ≤ *num_iteration)* **do**
8. *ne = popsize* × *er; //* number of elitism
9. $Pop_E \leftarrow$ The best *ne* data placement schemes in *Pop*;
10. *nc = popsize * cr; //* number of crossover
11. **for** *i =1* **to** *nc* **do**
12. randomly select two data placement scheme Ψ_A and Ψ_B from *Pop;*
13. generate Ψ_C and Ψ_D by one-point crossover for flexible datasets of Ψ_A and Ψ_B*;*
14. $Pop_C \leftarrow < \Psi_C, DG(\Psi_C) >$;
15. $Pop_C \leftarrow < \Psi_D, DG(\Psi_D) >$;
16. **end for**
17. *nm= popsize* × *mr;//* number of mutation
18. **for** *i =1* **to** *nm* **do**
19. select a data placement scheme Ψ_j from Pop_C*;*
20. $\Psi_j' \leftarrow$ mutate randomly a flexible virtual machine position number in Ψ_j *;*
21. **if** Ψ_j' is illegal
22. update Ψ_j' with a data placement scheme by repairing Ψ_j';
23. **end if**
24. $\Psi_j \leftarrow \Psi_j'$*;*
25. **end for**
26. *Pop* ← Pop_E *and* Pop_C*;*
27. *idx = idx +1;*
28. **end while**
29. return the best data placement scheme Ψ_{best}*;*
30. **End**

wheel selection technique is that each scheme is given a chance to select in proportion to its heuristic value. Then, it applies the crossover function and computes the heuristic value of the new generated schemes (lines 11-16). In the last step, BDAP applies the mutation operator for a randomly selected scheme along with computing its heuristic value (lines 17-25). In the crossover and mutation phases, BDAP does not change the number of virtual machine position for the fixed-location datasets and applied those functions only on flexible datasets.

The idea behind the roulette wheel selection technique is that each scheme is given a chance to select in proportion to its heuristic value. Then, it applies the crossover function and computes the heuristic value of the new generated schemes (lines 11-16). In the last step, BDAP applies the mutation operator for a randomly selected scheme along with computing its heuristic value (lines 17-25).

In the crossover and mutation phases, BDAP does not change the number of virtual machine position for the fixed-location datasets and applied those functions only on flexible datasets. These three operators apply to the schemes till it reaches a certain number of iterations, a parameter defined by the user at the beginning of the algorithm. In the last step, the best data placement scheme Ψ_{best} is returned as the output of BDAP.

3.4 Experiments and Discussion

In this section, we present and discuss the simulation results and compares

BDAP with the most competitive and Random approaches.

3.4.1 Simulation Setting

To evaluate performance of our proposed data placement approach, BDAP, we compare it with Yuan's work and random strategies. Yuan's work is the one of the most competitive algorithms in this field. It is a K-means based clustering algorithm which applies a heuristic binary clustering algorithm to precluster datasets into their appropriate virtual machines. Then, it greedily assigns the workflow tasks to each virtual machine such that it stores the most of its input dataset. Once an intermediate dataset is generated, it places it to the virtual machine that has the most interdependent datasets with the newly generated dataset.

We simulate a cloud computing environment on the Wayne State University's high performance Grid Computing. We use eight grid computation nodes along with total storage capacity of 100 GB and compared the three algorithms by simulating a variety of real and synthetic workflows. We test BDAP using five synthetic workflow applications based on real data-centric workflows [11]: Montage [91, 92], CyberShake [93-96], Epigenomics [97-99], LIGO [26, 100, 101] and SIPHT [102, 103] (Figure 3-4). These workflow applications are developed through the Pegasus workflow management system for different research domains like bioinformatics and astronomy. We select the large-size of each workflow with about 1000 number of tasks and assume each task can be executed on every virtual machine. For our experiments, we run 100

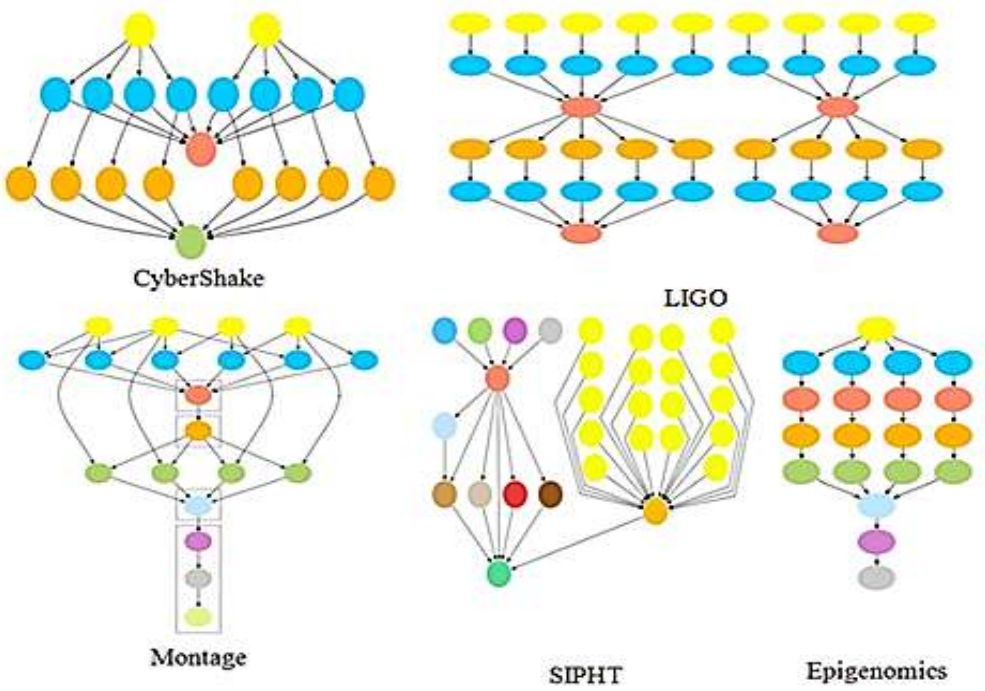


Figure 3-4 The structure of five realistic data-centric workflows [48].

times each of the selected workflows along with assigning five different numbers of datasets to their tasks randomly. The numbers of datasets are 5, 10, 25, 50 and 100, and dataset sizes are uniformly distributed in the range of [1TB 100TB]. In addition, we consider five size numbers of virtual machines, 5, 10, 15, 20 and 25 in the range of [200TB 1PB] of storage capacity (as shown in Table 3-2).Virtual machines storage capacities are selected in a uniformly distributed manner too. We demonstrate the performance of our proposed data placement algorithm and Yuan and Random approaches in terms of the average of the workflow communication cost (WCC) defined in the previous section. In our experiments, we assume that the data transmission rates among all virtual machines are fixed. Virtual machines storage capacities are selected in a uniformly distributed

Table 3-2 Description of dataset and virtual machine of the experiment.

# of datasets	[5,10,25,50,100]
Dataset size	1TB – 100TB
# of virtual machines	[5,10,15,20,25]
Virtual machines storage capacity	200TB – 1PB

manner too. We demonstrate the performance of our proposed data placement algorithm and Yuan and Random approaches in terms of the average of the workflow communication cost (WCC) defined in the previous section. In our experiments, we assume that the data transmission rates among all virtual machines are fixed. Table 3-3 shows the value of parameters using in BDAP. We do our experiments for two different scenarios, one scenario with considering 20% of fixed-location datasets and the other one without considering fixed-location datasets and consider the average of it.

Table 3-3 Default setting for the BDAP algorithm.

Population size	100
Initial population	Randomly generation
Maximum generation	100
Crossover probability	0.8-0.9
Mutation probability	0.3-0.5
Maximum iteration	1000

3.4.2 Results

Figure 3-5 shows the Workflow Communication Cost (WCC), in terms of hour by varying the number of datasets and fixing the number of virtual machines. WCC is increased by increasing the number of datasets in all three

strategies. However, it can be seen clearly that our strategy reduces WCC compared to the other strategies. This results in greater improvement margin with more number of datasets.

In the next step (Figure 3-6), we calculate WCC by varying the numbers of virtual machines and fixing the number of datasets. Although WCC is increased by increasing the number of virtual machines, the increasing rate of our strategy is slower than the others. This results in greater improvement margin with more number of virtual machines.

We demonstrate performance of BDAP in terms of workflow communication cost by varying the number of datasets and virtual machines for

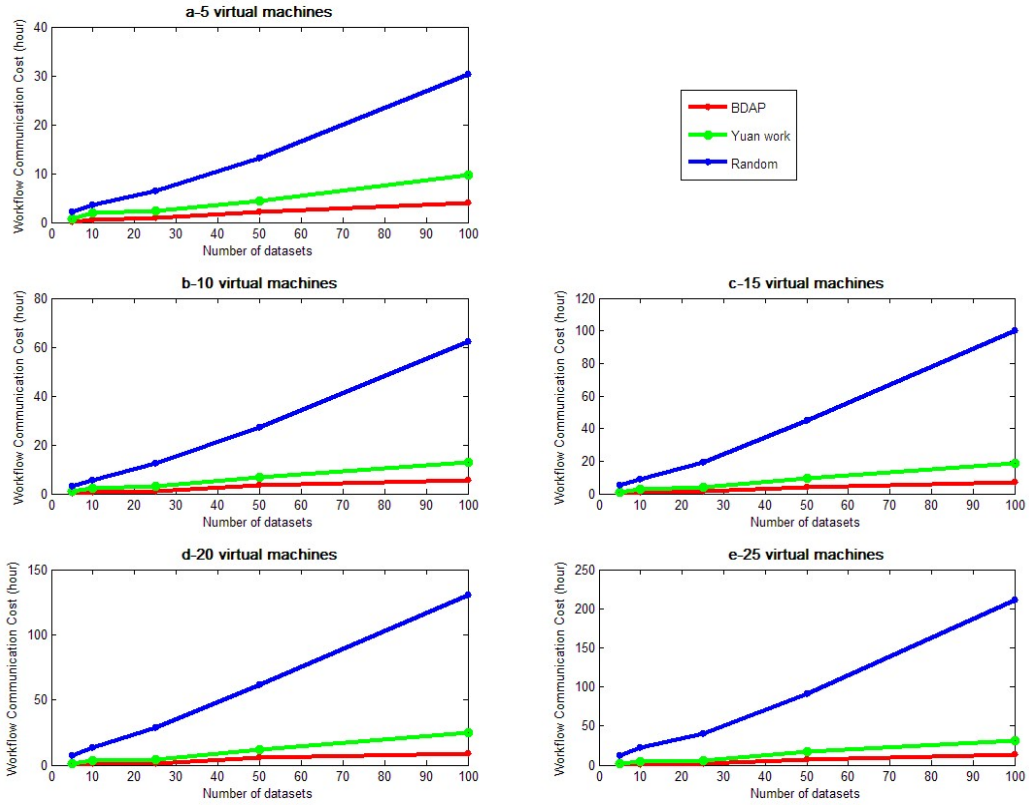


Figure 3-5 Workflow Communication Cost by varying the number of datasets.

five different types of workflows. We compare the BDAP strategy with Yuan as well as with random strategies. The result shows that BDAP manages to decrease effectively workflow communication cost more than Yuan and Random approaches. To see the impact of the total number of fixed-location datasets, we compare the three approaches for fixed number of datasets and datacenters and varying the percentages of fixed-location datasets in Figure 3-7 by having more fixed-location datasets, WCC is increased in BDAP and Yuan algorithms and there is almost no change for Random strategy. The reason is that the BDAP and Yuan algorithms are not allowed to change the location of the fixed-location datasets and the impact of these algorithms are on flexible datasets.

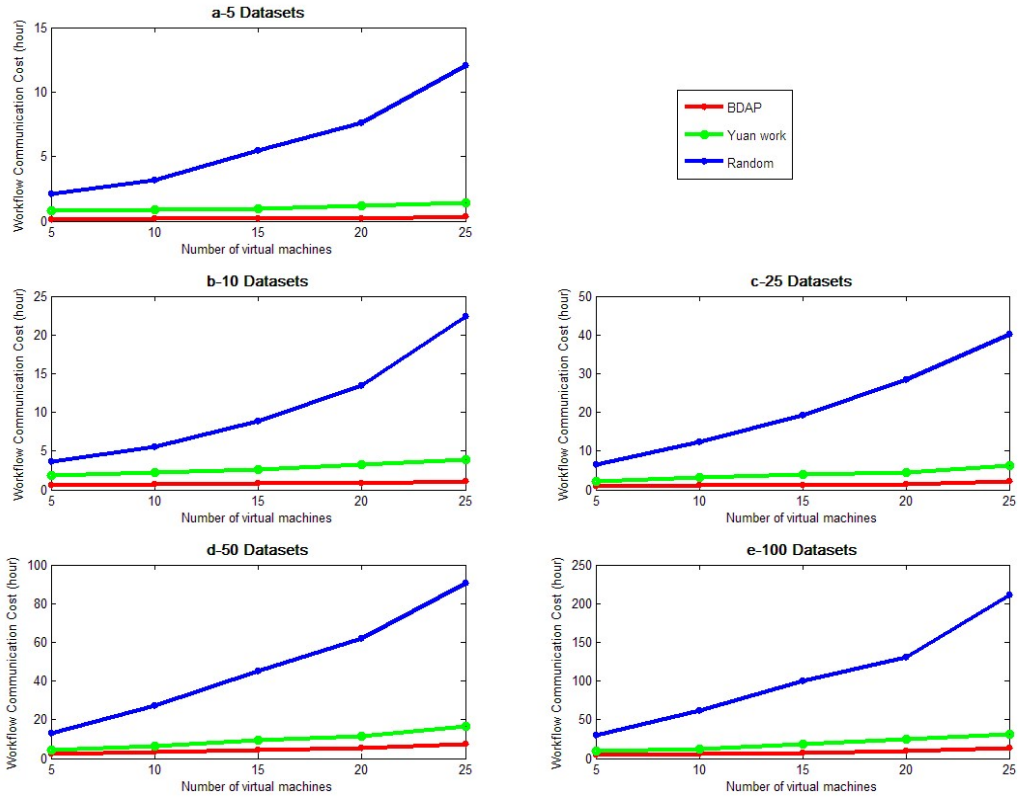


Figure 3-6 Workflow Communication Cost by varying the number of virtual machines.

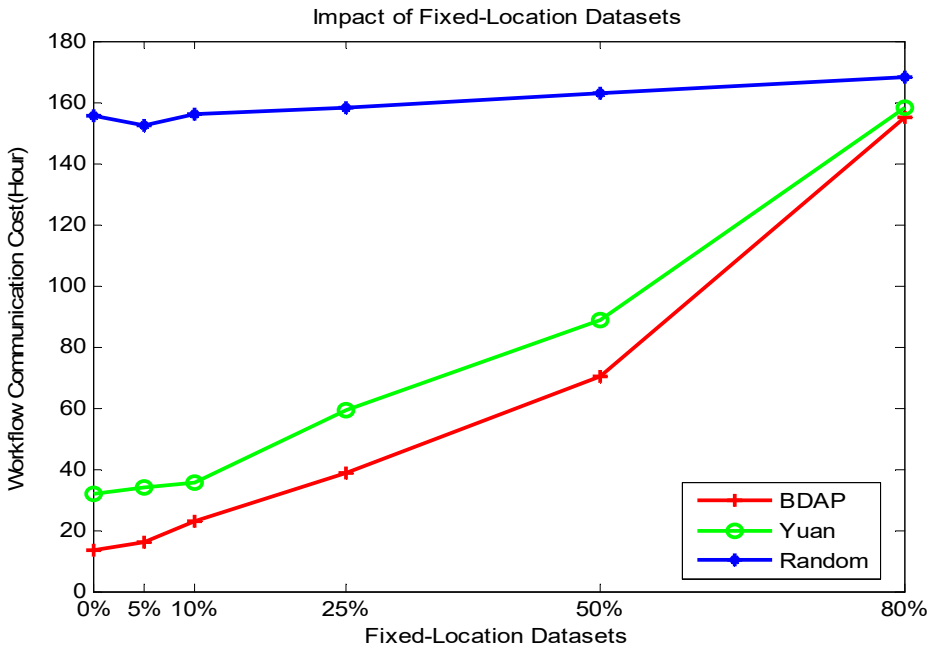


Figure 3-7 Workflow Communication Cost by varying the percentages of fixed-location datasets and for fixed number of workflow nodes, 1000, and datacenters, 50.

3.5 Data Placement Algorithm in DATAVIEW

In our DATAVIEW system [104, 105], we integrated the big data workflow engine subsystem with FutureSystems academic research cloud provider to automatically provision virtual machines to execute data-centric workflows in the Cloud. We implemented bash scripts to automatically provision VMs by first creating new VM images in the FutureSystems framework through configuring both hardware and software stack. Workflows 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.

In design time, we created the sophisticated XML parser to parse the workflow specification, which is stored in the XML format. The XML parser

extracted all workflow tasks, a set of input data products and a set of output datasets that will be generated at run time. The XML parser generated output (dp_{ID}, $task_{ID}$) key/value pairs that contain mapping details to map datasets to corresponding workflow tasks. BDAP algorithm validate the (key, value) pairs to identify the optimal mapping of datasets and workflow tasks to the corresponding virtual machines.

In running time, DATAVIEW provisioned a set of virtual machines in FutureSystems and deployed datasets to the corresponding virtual machines based on the output of BDAP. In our DATAVIEW system, we used files as a dataset type and used SCP command to move actual files from our DATAVIEW system to the provisioned virtual machines. In the next step, we assigned all the workflow tasks to the provisioned virtual machines. After assigning workflow tasks and datasets, the workflow was executed and intermediate datasets were moved to the corresponding virtual machines. Data flow between each workflow task was implemented by the SCP command. The final dataset was moved from its virtual machine to the DATAVIEW system and the results were published to the user. In this way, all low-level details were hidden from the data scientists and only the intermediate and final data products generated by the workflow were visible to data scientists. Table 3-4 shows some of the result of applying BDAP for the execution of workflow in Example 1.

Table 3-4 Some results of BDAP running.

The best data placement scheme in the :	
First population <$d_{\#}$, $vm_{\#}$>	Last (10th) population <$d_{\#}$, $vm_{\#}$>
<d_1,vm_1><d_2,vm_2><d_3,vm_3><d_4,vm_3><d_5,vm_1><d'_1,vm_2><d'_2,vm_1><d'_3,vm_1><d'_4,vm_3> DG = 0.1671 and WCC = 0.0097 hr	<d_1,vm_3><d_2,vm_1><d_3,vm_3><d_4,vm_1><d_5,vm_2><d'_1,vm_1><d'_2,vm_2><d'_3,vm_1><d'_4,vm_2> DG = 3.4032 and WCC = 0.0041 hr
<d_1,vm_2><d_2,vm_1><d_3,vm_3><d_4,vm_1><d_5,vm_3><d'_1,vm_3><d'_2,vm_1><d'_3,vm_2><d'_4,vm_3> DG = 0.2513 and WCC = 0.0083 hr	<d_1,vm_1><d_2,vm_2><d_3,vm_1><d_4,vm_2><d_5,vm_3><d'_1,vm_2><d'_2,vm_1><d'_3,vm3><d'_4,vm2> DG = 3.4678 and WCC = 0.0033 hr
<d_1,vm_2><d_2,vm_2><d_3,vm_1><d_4,vm_3><d_5,vm_3><d'_1,vm_3><d'_2,vm_3><d'_3,vm_1><d'_4,vm_3> DG = 0.3165 and WCC = 0.0081 hr	<d_1,vm_2><d_2,vm_2><d_3,vm_1><d_4,vm_2><d_5,vm_3><d'_1,vm_2><d'_2,vm_3><d'_3,vm_3><d'_4,vm_1> DG = 3.3692 and WCC = 0.0042 hr

4 TASK PLACEMENT IN BIG DATA WORKFLOWS

4.1 Introduction

Decomposing a complex application as a workflow simplifies design effort, enables reuse of computational modules and allows their parallel and/or pipelined execution. With the progress in computing, storage, networking, and sensing technologies and the ease of performing collaborative scientific research, it is feasible to conceive much more complex data-centric workflows that involve big data sets and run over distributed and heterogeneous computing environments.

Workflow management system is a platform to support two key functions: 1) design and specification of workflows, and 2) configuration, execution and monitoring of workflow runs. Examples of notable data-centric workflow management system include Taverna [106], Kepler [107], Vistrails [108], Pegasus, Swift [109] and VIEW [110]. Traditionally, these systems have used a directed acyclic graph (DAG) abstraction to model a workflow where each vertex of the graph represents a workflow task, and the directed edges between two vertices depicts dataflow between the corresponding tasks.

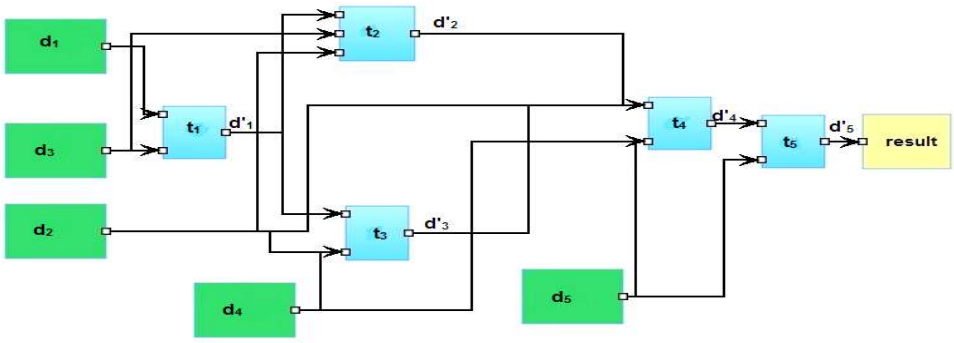
Since scientific applications become more and more data intensive, it is more critical to assigned workflow tasks to the same virtual machines which are already hosted their required datasets to maximize data locality and minimize

data migrations between virtual machines in the Cloud. Practically, it is impossible to store all the required datasets of workflow tasks in one virtual machine due to the storage capacity limitation of virtual machines and so data movement is necessary to execute data-centric workflows. The main goal of task and data placement is to minimize the total data movement between virtual machines.

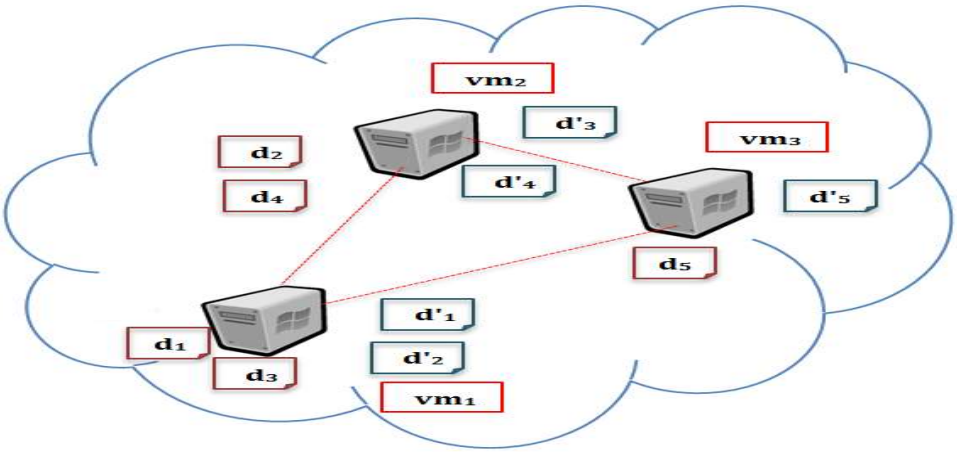
In this chapter of dissertation, we propose task placement strategy (TPS), an evolutionary algorithm (EA) which is a genetic-based task placement in big data workflows such that the data movement between virtual machines during the execution of a workflow gets minimized. Let's consider the example 1 in chapter 3. Figure 4-1.a) illustrates the workflow with five tasks, five original datasets and five generated intermediate datasets. Figure 4-1.b) shows an instance of its virtual machines configuration in the Cloud. In this example, datasets, d_1 and d_3 were assigned to virtual machine 1, VM_1. Similarly, datasets d_2 and d_4 were assigned to VM_2. Figure 4-1.c) shows an instance of the virtual machines configuration in the Cloud with assigning tasks t_1 and t_2 as well as datasets, d_1 and d_3 virtual machine 1, VM_1. Tasks t_3 and t_4 were assigned to VM_2 and task t_5 and dataset d_5 were assigned to VM_3.

To come up with a task placement of big data workflows, our proposed strategy, TPS, clusters the most interdependent workflow tasks together and assign them possibly in the same virtual machine in the Cloud.

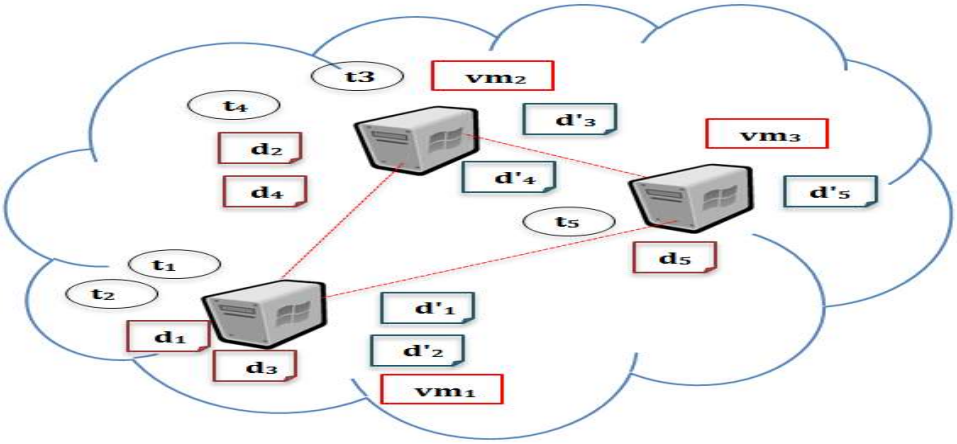
A random set of task placement schemes are generated in the first step. In



a. Sample Workflow.



b. Big Data Placement (BDAP).



c. Task Placement Strategy (TPS).

Figure 4-1 a) Workflow of Example 1 b) Data placement c) Task placement.

the next step, TPS computes and compares the generated schemes by applying a defined heuristic function and return the best scheme. The heuristic function is based on the task interdependency within and between the virtual machines in the Cloud. The best scheme is the one which maximizes the task interdependency within each virtual machine and minimizes the task interdependency between virtual machines.

4.2 Workflow Task Placement Model

To minimize data movement between virtual machines in the Cloud, we cluster the virtual machines such that the placed tasks have the highest task interdependency within each virtual machine as well as the lowest task interdependency between virtual machines. In the rest of this section, we model our task placement solution in detail.

To model TPS we customized the definitions of chapter 3 and add the new required sets or functions. A cloud computing environment is modeled as follows:

Definition 4.2.1 (Cloud Computing Environment C). A cloud computing environment C is a 4-tuple $C = (VM, CC, SC, DTR)$, where

- VM is a set of virtual machine 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. It is measured in some pre-determined unit such that 1000 cycle in millisecond. R^+ is the set of

positive real number.

- $\mathrm{SC}: \mathrm{VM} \rightarrow \mathrm{R}^{+}$ is a storage capacity function. $\mathrm{SC}(\mathrm{vm_i})$, $\mathrm{vm_i} \in \mathrm{VM}$ gives the maximum available storage capacity of virtual machine $\mathrm{vm_i}$ 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 number.

- $\mathrm{DTR}: \mathrm{VM} \times \mathrm{VM} \rightarrow \mathrm{Q}_0^{+}$ is the data transfer rate function. $\mathrm{DTR}(\mathrm{vm_{i1}}, \mathrm{vm_{i2}})$, $\mathrm{vm_{i1}}$, $\mathrm{vm_{i2}} \in \mathrm{VM}$ gives the data transfer rate between two virtual machines $\mathrm{vm_{i1}}$ and $\mathrm{vm_{i2}}$. It is measured in some pre-determined unit such as mega-bytes, giga-bytes per second. Q_0^{+} is the set of positive rational number.

The above three attributes, CC, SC and DTR are not fixed or static for a virtual machine at all times. These are considered to be established by a priori negotiation and remain unchanged during the execution of an individual workflow. Big data workflow is formalized as the previous chapter by adding one more function as follows:

Definition 4.2.2 (Big Data Workflow W). Big data workflow W can be modeled formally as a 6-tuple that consists of three sets and two functions as follows:

- $\mathrm{W} = (T, D, D', S, TS, DS)$

- T is the set of workflow tasks. Each individual task is denoted by $\mathrm{t_k}$, $\mathrm{T} = \{\mathrm{t_1}, \mathrm{t_2}, \mathrm{t_3}, \ldots, \mathrm{t_K}\}$.

- D is the set of input datasets for 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 for 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 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 number.

- $TS: D \cup D' \rightarrow T$ 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 the others proposed algorithms Workflow Communication Cost is applied as defined in the previous chapter.

We consider task interdependency as the separation measurement. Two tasks are interdependent and should be collocated in the same virtual machine if they simultaneously need many datasets as their inputs. The definition for the

interdependency of a pair of tasks is as follows:

Definition 4.2.3 (Task Interdependency tp). We consider the size of common datasets that a pair of tasks gets them as input to define the task interdependency of the tasks. Task interdependency value is divided by the total size of 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: Which S(D) is the sum of the sizes of datasets in D. 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 more size of common datasets and the greater the size of common datasets is, the higher is the task interdependency of tasks.

For instance, if size of datasets is $S(d_1) = 10\text{MB}$, $S(d_2) = 35\text{MB}$, $S(d_3) = 110\text{MB}$, $S(d_4) = 60\text{MB}$ and $S(d_5) = 55\text{MB}$, 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

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

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

$$= \frac{S(d_1) + S(d_3)}{S(d_1) + S(d_2) + S(d_3) + S(d_4) + S(d_5)} = 0.44.$$

Task interdependency matrix (TM) is defied similar to data interdependency matrix. In the interdependency matrix, rows and columns are

the workflow tasks and the value of 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 into 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. Task placement scheme is defined to represent the place of each workflow dataset in a virtual machine. A task placement scheme is defined formally as follows:

Definition 4.2.4 (Task Placement Scheme Φ). Suppose there are I virtual machines and K tasks, a task placement scheme is represented by a K-element vector Φ such that $\Phi(\mathrm{t_k})$ indicates the virtual machine to which $\mathrm{t_k}$ is placed. For example if the task placement scheme is $\Phi = (1, 2, 1, 2, 3)$ it means tasks $\mathrm{t_1}$and $\mathrm{t_3}$ are placed in virtual machine $\mathrm{vm_1}$ ($\Phi(\mathrm{t_1}) = \Phi(\mathrm{t_3}) = \mathrm{vm_1}$), tasks $\mathrm{t_2}$and $\mathrm{t_4}$ in virtual machine $\mathrm{vm_2}$ ($\Phi(\mathrm{t_2}) = \Phi(\mathrm{t_4}) = \mathrm{vm_2}$) and the tasks $\mathrm{t_5}$ in virtual machine $\mathrm{vm_3}$ ($\Phi(\mathrm{t_5}) = \mathrm{vm_3}$).

We consider all the workflow tasks are flexible and there are no fixed

tasks because moving computation task to datasets is often cheaper than moving datasets to computation task nodes. To define a good measurement to compare separation between virtual machines, task interdependency within and between virtual machines are defined as follows:

Definition 4.2.5 (Within-VirtualMachine Task Interdependency $\mathbf{VMT_W}$).

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

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

Definition 4.2.6 (Between-VirtualMachine Task Interdependency $\mathbf{VMT_B}$).

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

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 within and between virtual machine interdependency. The heuristic is defined in TPS as follows:

Definition 4.2.7 (Task Interdependency Greedy TG). The TG heuristic biases TPS to select the task placement scheme with higher task interdependency. It is defined as:

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

In this formula, the numerator measures 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 get 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 our system model, we consider a system constraint in terms of task which is defined as follows:

Non-replication constraint: Once a task is placed into a specific virtual machine, it is not allowed to place it into another virtual machine as task replication is not in the scope of this version of TPS.

Definition 4.2.8 (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 $\Phi \in Q$ to minimize the workflow communication cost (WCC) under the virtual machine storage capacity and non-replication constraints. In the next section, we explain our task placement strategy, TPS, in detail.

4.3 Workflow Task Placement Algorithm-TPS

Like BDAP, 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

Algorithm 2. Task Placement (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,

num_iteration: number of iterations,

Output: The best task placement scheme, Φ_{best}

1. **Begin**
2. **for** *i = 1* **to** *popsize* **do**
3. $\Phi \leftarrow$ Generate a task placement scheme randomly;
4. $Pop \leftarrow < \Phi,\ TG(\Phi) >$;
5. **end for**
6. *idx = 0;*
7. **while** (*idx* ≤ *num_iteration)* **do**
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 Φ_A and Φ_B from *Pop;*
13. generate Φ_C and Φ_D by one-point crossover for tasks of Φ_A and Φ_B*;*
14. $Pop_C \leftarrow < \Phi_C,\ TG(\Phi_C) >$*;*
15. $Pop_C \leftarrow < \Phi_D,\ TG(\Phi_D) >$*;*
16. **end for**
17. *nm= popsize* × *mr;//* number of mutation
18. **for** *i =1* **to** *nm* **do**
19. select a task placement scheme Φ_j from Pop_C*;*
20. $\Phi_j' \leftarrow$ mutate randomly a virtual machine position number in Φ_j *;*
21. $\Phi_j \leftarrow \Phi_j'$*;*
22. **end for**
23. *Pop* $\leftarrow Pop_E$ *and* Pop_C*;*
24. *idx = idx +1;*
25. **end while**
26. return the best task placement scheme Φ_{best}*;*
37. **End**

three main operators, Selection, Crossover, and Mutation sequentially to generate possibly better schemes with higher heuristic values. At the end of the algorithm, the best observed task placement scheme is recorded in Φ_{best} and will be returned as the output of TPS. Selection, crossover and mutation operators are defined in chapter 3. Algorithm 2 represents TPS.

4.4 Experiment and Case Study

4.4.1 Case Study

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

Figure 4-2 shows the OpenXC workflow for comparing driving behavior for three 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 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 4-1). 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 4-2 outperform the other task placement schemes.

Table 4-1 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 virtual machines	5MB per second

Table 4-2 Default setting for the 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

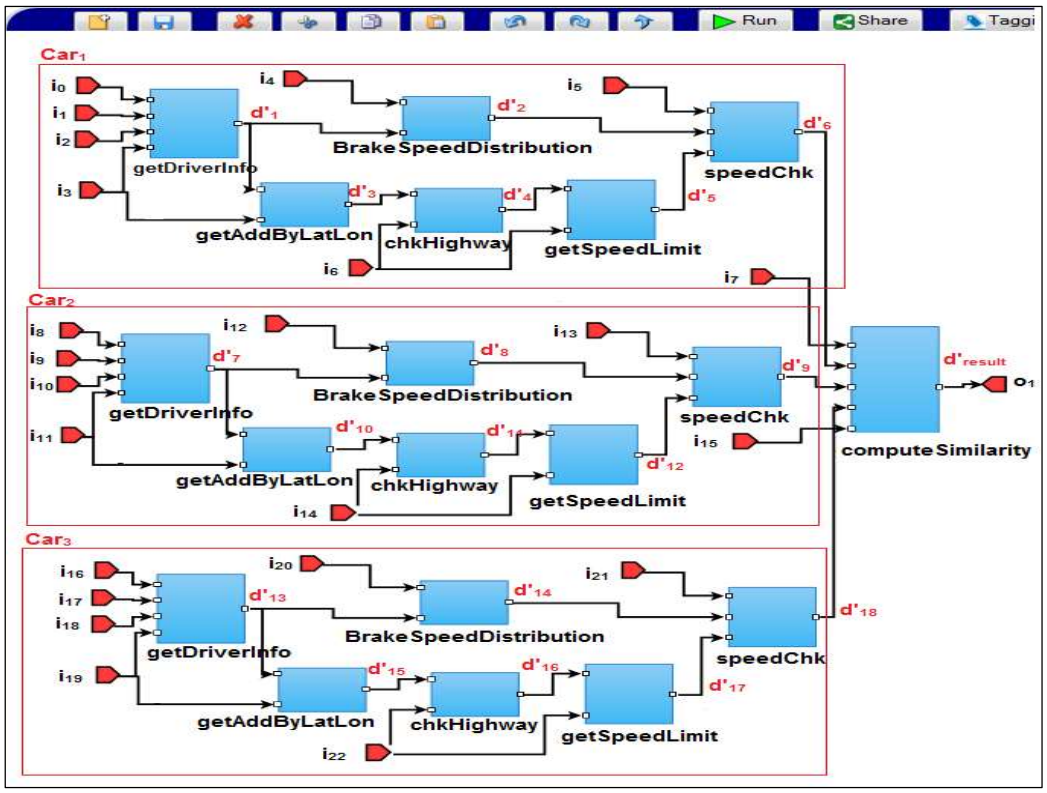


Figure 4-2 OpenXC Workflow for Comparing Three Car Drivers.

4.4.2 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 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 4-3 shows the description of running the OpenXC workflow of Figure 4-2. Table 4-4 shows some of the result of applying TPS for the execution of workflow in Example 1.

Table 4-3 Some results of applying TPS for the execution of workflow in Example1.

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 4-4 OpenXC workflow of one car driver running in DATAVIEW.

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

4.4.3 Results

Figure 4-3 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 (Figure 4-4). 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

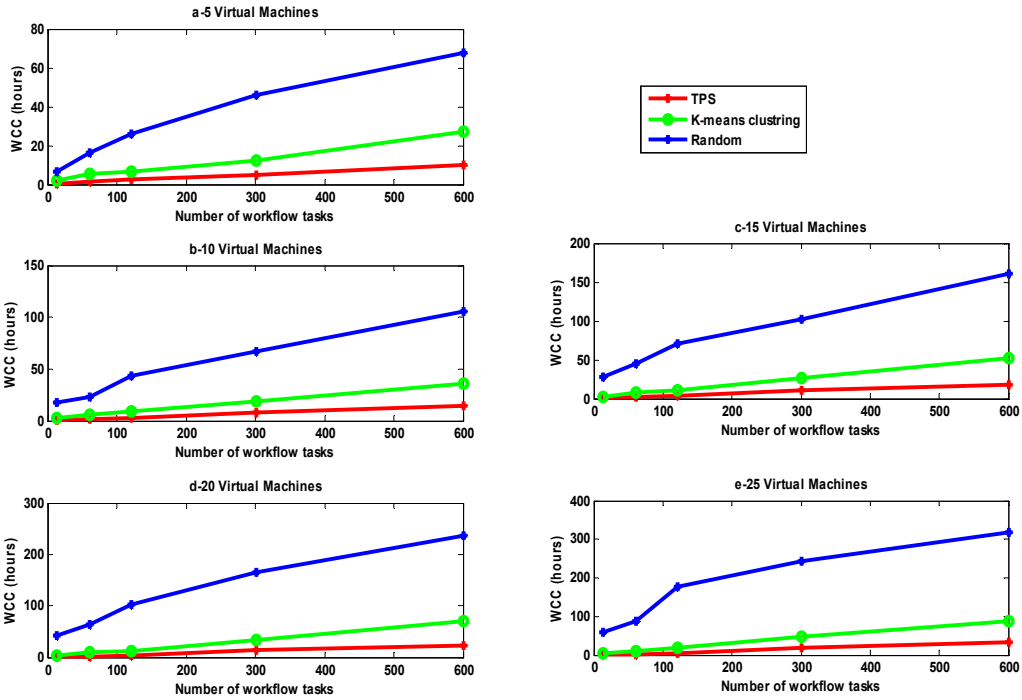


Figure 4-3 Workflow Communication Cost (hours) by varying the number of workflow tasks.

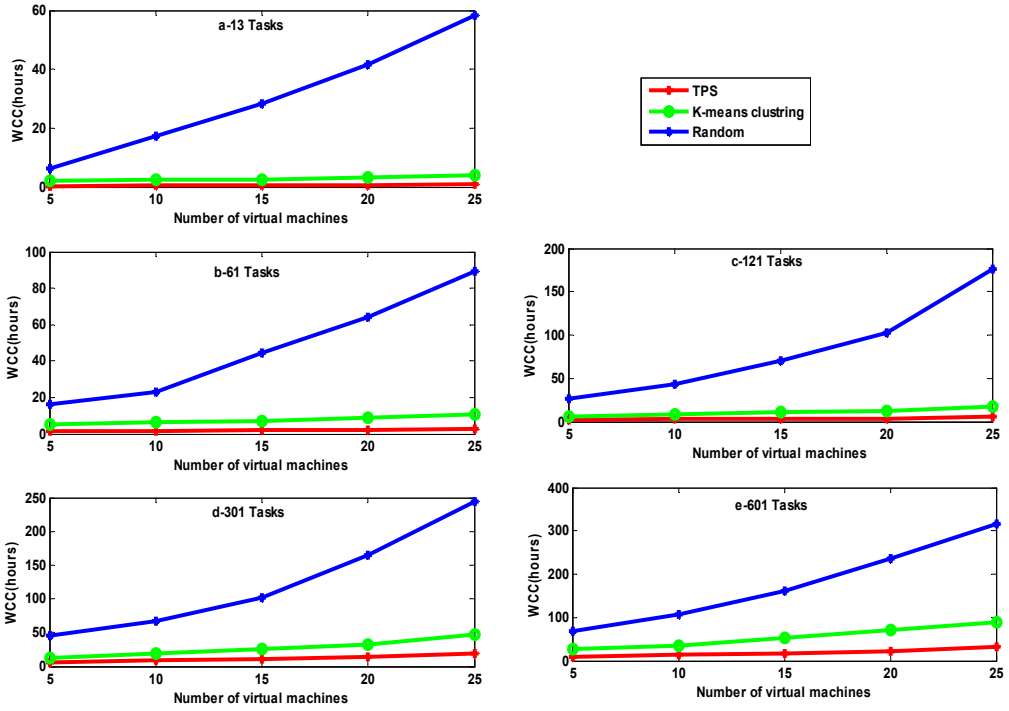


Figure 4-4 Workflow Communication Cost (hours) by varying the number of virtual machines.

5 TASK SCHEDULING IN BIG DATA WORKFLOWS

5.1 Introduction

Workflow scheduling has remained a critical component of modern data-centric workflow management systems. Cloud computing, which provides practically unlimited computing and storage resources, has created a new generation of data-centric workflows, called big data workflows, and the need for new workflow scheduling algorithms that consider the characteristics of cloud computing, such as heterogeneous virtual machines, the elastic resource provisioning model, and the pay-as-you-go pricing model, as well as the time and monetary cost of transfer of large amount of data. In this study, we consider one sub-problem of the general big data workflow scheduling problem, in which a deadline D is given for a workflow W, and the goal is to minimize the monetary cost of running W in the cloud while satisfying the given deadline.

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 [55]. 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 [27, 30], 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 [61, 62], do not consider any QoS constraints that relate to the update of user run time requirements. We took a different approach to schedule the workflow based on the user defined deadline constraints. We performed 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. We proposed 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.

5.2 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 type such as {"t2.nano", "t2.micro", "t2.small", "t2.medium, "t2.large", …}.
- R_C: $R \rightarrow Q^+$ is the resource usage time function. $R_C(R_i)$, $R_i \in R$ gives the time for the resource usage R_i in the cloud computing environment. The resource with the minimum R_C is called $R_{slowest}$ and the resource with the maximum R_C 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 can be defined formally as:

Definition 2.2 (Big Data Workflow W): A big data workflow can be formally defined as a 4-tuple W = (T, D, FT, FD), 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($T, D, R, F_c, F_{\bar{c}}, F_p, F_{\bar{p}}, F_m, F_{\bar{m}}, F_n, F_{\bar{n}}, P, TL, RT$), 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.
- F_c: $D \times R \times R \rightarrow Q^+_0$ is the data communication cost function; $D_{k1,k2} \in D$; $R_{i1}, R_{i2} \in R$ 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. $F_{\bar{c}}$ (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 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; $D_{k1,k2} \in D$; $R_{i1}, R_{i2} \in R$ gives the data communication time of $D_{k1,k2}$ from resource R_{i1}

to resource R_{i2}. $F_{\bar{m}}$: $D \rightarrow Q^{+}_{0}$ is the average data communication time function. $F_{\bar{c}}$ (k_1, k_2), $D_{k1,k2} \in D$ gives the average data communication time 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 same resource.

- *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, $F_{\bar{p}}$ (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 of partition j. R_{Pj} represents the set of resources of partition P_j.
- *TL:* $T \rightarrow N$ is the task partition function, TL [T_k] or TL_{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. Our goal is to come up with an optimal workflow schedule such that the workflow execution cost is minimized and the workflow makespan meets the given deadline. To this end, we need to model workflow cost in order to be able to measure both workflow makespan and excetion cost. As we partition the workflow into a set of partitions, the workflow makespan will be the summation of the execution times of all

partitions. We define the workflow makespan as follows:

Definition 2.4 (Workflow Makespan E_C): Given a workflow W in a cloud computing environment C, a workflow execution makespan, represents the execution time of the workflow with 5-tuple E_C(CT, $\overline{CT}$, minCT, maxCT, CC), where

- CT: Partition × R → Q^+ is the workflow partition completion time function. CT(P_j), P_j ∈ Partition gives the maximum of task computation time of all the tasks T_k ∈ P_j and the maximum of average data communication time of all the outgoing edges from all the tasks T_k ∈P_j. We formally define CT as:
- CT (P_j, R_i) = $Max_{T_k \in Pj}\{F_n\ (T_k, R_i)\} + Max_{T_k \in Pj}\{F_m(D_{k,k1}, R_i, R_{i1})\}$
- $\overline{CT}$: Partition → Q^+ is the workflow average completion time function. $\overline{CT}$ (P_j), P_j ∈ Partition gives the max of average task computation time of all the tasks T_k ∈ P_j and the average data communication time for all the outgoing edges from all the tasks T_k ∈ P_j. We formally define $\overline{CT}$ as:

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

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

$$\text{minCT}\ (P_j) = \sum_{T_k \in P_j} CC\left(T_k, R_{fastest}\right)$$

- 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:

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

- 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:

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

The critical path in the workflow can be computed by the SCPOR algorithm [62]. We define 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, PRT, ACT, Earliness, Lateness, Threshold,), where

- SD: Partition $\rightarrow Q^{+}$ is the sub-deadline partition function. SD(P_j), $P_j \in$ 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:

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

- PRT: Partition × R × SD × Threshold → RT is the resource partition identifier that is used to identify the slowest resource for executing the tasks in a partition while still managing to meet the deadline of executing the tasks in the partition to the sum of sub-deadline and threshold allocated to the partition.

- ACT: Partition → 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 → Q^+ is the earliness partition function. Earliness (P_j), $P_j \in$ Partition gives the earliness time of the partition P_j. It can be calculated as follows:

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

- Lateness: Partition → Q^+ is the lateness partition function. Lateness(P_j), $P_j \in$ Partition gives the lateness time of the partition P_j. It can be calculated as follows:

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

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

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

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): Given a workflow W in a cloud computing environment C, and deadline D, makespan of workflow is the objective function and can be defined as follows

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

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

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.

5.3 The BORRIS Algorithm

The main steps of the BORRIS algorithm are shown in Figure 5-1. 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

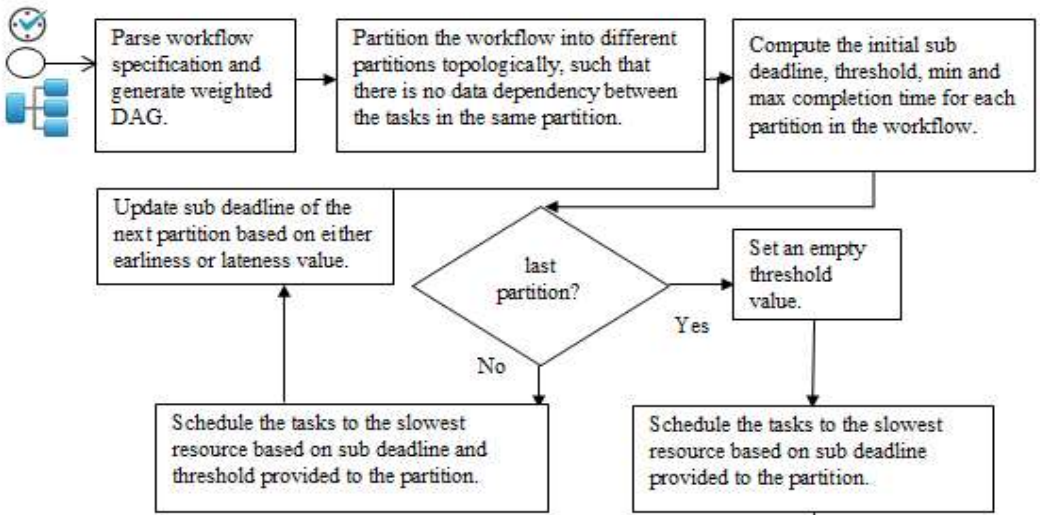


Figure 5-1 BORRIS flowchart.

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 Figure 5-2 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 5-1.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 5-1.d.

Table 5-1.a shows a list of cloud resources parameters including five resource types with their computation capacities and the associated costs.

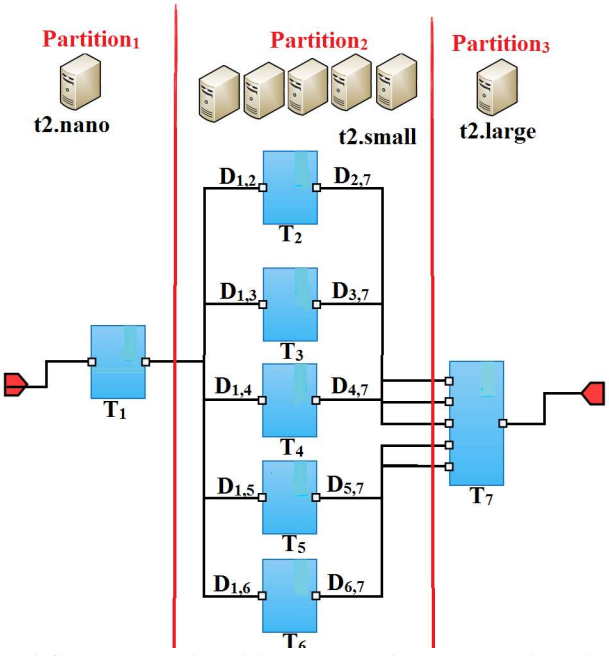


Figure 5-2 Workflow example with seven tasks and ten data dependencies.

Table 5-1 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)

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 secondpartition 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 5-1.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 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. 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 tasks the algorithm evaluates all the partitions sequentially (lines 11-31). 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). If it is not the last partition (line 16), BORRIS then calculates the threshold (line 17) and the slowest resource type for all the tasks in the partition. It then adds this schedule to the output schedule map (lines 18-19). In line 20, BORRIS computes the maximum of actual completion time (ACT) of the partition tasks. In lines 21-24, BORRIS calculates the lateness and earliness values of the partition to update the sub-deadline of the next subsequent partition. In lines 25-30, if it is the last partition then BORRIS updates the sub-deadline of the last partition (line 26). 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 27). Finally, in line 30,

BORRIS returns the complete schedule that consists of all the tasks and the corresponding resources as a set of pairs (<task, resource>).

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: $minCT\ [P_j] \leftarrow \sum_{T_k \in P_j} CC(T_k, R_{fastest})$
15: $maxCT\ [P_j] \leftarrow \sum_{T_k \in P_j} CC(T_k, R_{slowest})$
16: **if** (P_j is not last Partition) **then**
17: $Thres\ [P_j]$ = Max{0, $SD\ [P_{j+1}] - minCT\ [P_{j+1}]$}
18: RT $[P_j]$ ←PRT (P_j, R, SD $[P_j]$ + Thres $[P_j]$)
19: d ← d ∪ MAP (T_k, F_S(RT $[P_j]$)) ∀ $T_k \in P_j$
20: ACT $[P_j]$ = CT(P_j, F_S(RT $[P_j]$))
21: Lateness $[P_j]$ = Max {0, ACT $[P_j]$ - SD$[P_j]$}
22: SD$[P_{j+1}]$ = SD$[P_{j+1}]$ – Lateness $[P_j]$
23: Earliness $[P_j]$ = Max {0, SD$[P_j]$ – ACT $[P_j]$}
24: SD$[P_{j+1}]$ = SD$[P_{j+1}]$ + Earliness $[P_j]$
25: **else if** (P_j is last Partition) **then**
26: SD $[P_j]$ = D - $\sum_{j1=1}^{J-1}$(ACT$[P_{j1}]$ + Lateness$[P_{j1}]$)
27: RT $[P_j]$ ← PRT (P_j, R, SD $[P_j]$)
28: d ← d ∪ MAP (T_k, F_S(RT$[P_j]$)) ∀ $T_k \in P_j$
30: **end if**
31: **end for**
32: **return d**
33: **end function**

5.4 Experimental Results

5.4.1 Performance Evaluation

In order to evaluate the performance of BORRIS, we developed a big data workflow for the automotive domain DATAVIEW platform. This workflow is an auto analytics workflow based on the OpenXC datasets. 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 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 [55]. The WRPS algorithm is the most recent work in the field of workflow scheduling. WRPS computes a set of bag 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.

5.4.2 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 5-2, 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 Figure 5-3.b, we show that BORRIS outperforms WRPS by roughly 4-11% margin as the complexity of the workflow increases from w3 to w10.

Table 5-2 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

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.

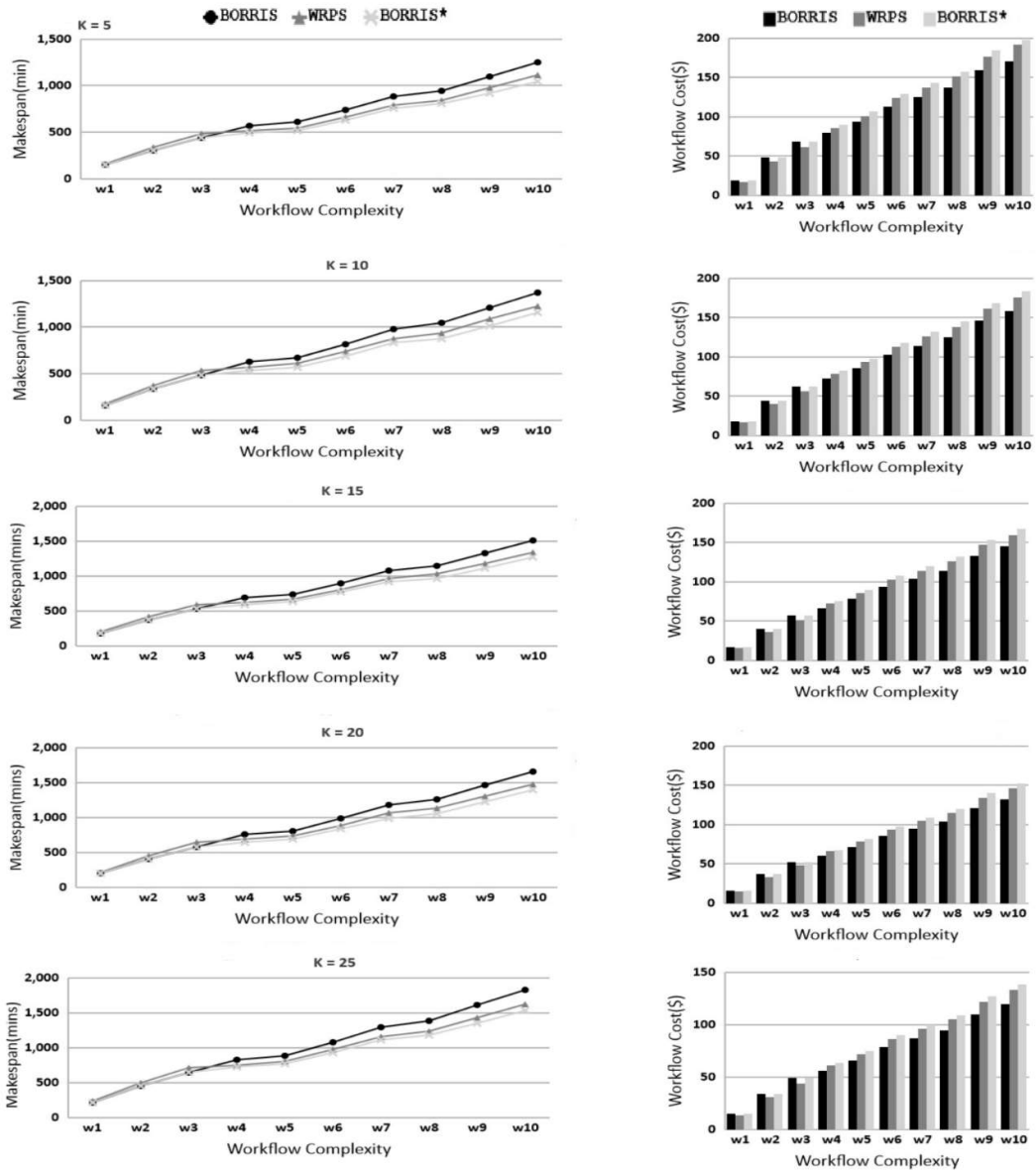


Figure 5-3 a) Resource utilization; b) Execution cost minimization.

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 this 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 Figure 5-3.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.

6 CONCLUSIONS AND FUTURE WORK

In big data workflows that involve big datasets, either as inputs or intermediate outputs, the workflow makespan can vary greatly depending on how the tasks and datasets are allocated in the distributed computing environment like the Cloud. Therefore, our research focus on the data and task placement and schedule for big data workflows that are execution in the heterogeneous and distributed execution environment like Cloud. Our main contributions in this dissertation are summarized as follows:

We formalized the data placement problem in big data workflows by defining data interdependency concept for clustering the most interdependent data products together and place them possibly to the same virtual machine. The goal is to minimize data movement among virtual machines during workflow execution. Therefore, we considered data interdependency as the separation measurement to maximize the data locality. For this, two datasets are interdependent and should be collocated in the same virtual machine if they are simultaneously needed as inputs by many tasks. We considered the number of common tasks that take a pair of datasets as input to define the data interdependency of the datasets. To define a good measurement to compare separation between virtual machines, data interdependency within and between virtual machines were defined. At the end, a data dependency greedy was defined based on the data interdependency within and between virtual machines.

We proposed BDAP, an evolutionary algorithm (EA) which is a generic

population-based metaheuristic optimization strategy for data placement in big data workflows. The main goal was to minimize the dataset movement between virtual machines during the execution of a workflow under the constraint of virtual machine storage capacity. In BDAP, a random set of data placement schemes were generated in the first step. In the next step, BDAP computed and compared the generated schemes by applying a defined heuristic function and returned the best scheme. The heuristic function was based on the data interdependency within and between the virtual machines in the Cloud. The best scheme was the one which maximized the data interdependency within each virtual machine and minimized the data interdependency between virtual machines.

We formalized the task placement problem in big data workflows. We defined Task Interdependency concept for clustering the most interdependent workflow tasks and place them possibly to the same virtual machine. Therefore, we considered task interdependency as the separation measurement to maximize the data locality. For this, two tasks were interdependent and should be collocated in the same virtual machine if they were simultaneously consumed many datasets. We considered the size of the total number of common input datasets to define the task interdependency of the tasks. Similarly, task interdependency within and between virtual machines were defined to define a good measurement to compare separation between virtual machines as the clusters. At the end, a task dependency greedy was defined based on the task

interdependency within and between virtual machines.

We proposed a generic population-based metaheuristic optimization strategy for task placement in big data workflows (TPS). The main goal was to minimize the dataset movement between virtual machines during the execution of a workflow under the constraint of virtual machine computing capacity. In TPS, a random set of task placement schemes were generated in the first step. Then, TPS computed and compared the generated schemes by applying a defined heuristic function and returned the best scheme. The heuristic function was based on the task interdependency within and between the virtual machines in the Cloud. The best scheme was the one which maximized the task interdependency within each virtual machine and minimized the task interdependency between virtual machines.

We considerd one sub-problem of the general big data workflow scheduling problem, in which a deadline D is given for a workflow W, and the goal is to minimize the monetary cost of running W in the cloud while satisfying the given deadline.

I plan several improvements and extensions of my work in the future. In the following, I briefly describe some of the problems I am particularly interested in contributing to work on fundable and applicable problems in the big data area by focusing on designing scalable big data applications and algorithms to support big data computing and analytics. Some of the future works are as follows:

Data Placement with Replica for Big Data Workflows in the Clouds.

The growth of big data in volume, variety and velocity is faster than Moore's law while the demand for more complex data analytics is increasing. We proposed BDAP, a data placement strategy for Cloud-based scientific workflows. Big data workflows consume and produce huge datasets. Applying data replication can reduce data movement as well. So, in future work, I plan to improve BDAP by applying data replication techniques. In addition, we considered data placement for executing of a single workflow. However, in real world, multiple workflows can be executed concurrently. Therefore, I plan to extend BDAP strategy to achieve data placement for the execution of multiple workflows simultaneously. I am interested in addressing these new works by using metaheuristic optimization approaches like Cultural Algorithms. For my experimentation and testing, I plan to use DATAVIEW with the new cloud testbeds like Chameleon provided by NSF as well as Amazon EC2 cloud. Moreover, I plan to use cloud data storages like Dropbox [111], Google Drive [112], Microsoft OneDrive [113].

Task Placement with Replication for Big Data Workflows in the Cloud. We proposed TPS, a task placement strategy for big data workflows. Big data workflows consume and produce huge datasets. Applying task/data replication can reduce data movement. So, in future work, I 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, I plan to extend the

TPS strategy in order to achieve task placement for the execution of multiple workflows simultaneously. For the other future work, I will enhance the performance of both BDAP and TPS strategies by using Cultural Algorithm (CA). One of the evolutionary computation systems that simulates the cultural evolution is Cultural Algorithm proposed by Dr. Reynolds [114-118]. Due to its nature, culture can be seen and understood as a complex adaptive system. In a complex system, such as culture, different heterogeneous agents are working together and interacting with the environment. This interaction of intelligent agents can result in a higher-level behavior used to solve different problems. Culture as a source of knowledge can significantly affect the behavior of individuals within a population. Cultural Algorithm has two major components: the Population Space and the Belief Space [119, 120]. In addition to those two components, there is a communication protocol that allows the Belief space, and Population space to interact with each other, and exchange the knowledge.

APPENDIX A: SUPPORT VECTOR MACHINE (SVM) P-WORKFLOW

Workflow Specification

```
<workflowSpec>
<workflow name="svm" root="true">
 <workflowInterface>
  <workflowDescription>simple svm workflow</workflowDescription>
  <inputPorts>
       <inputPort>
        <portID>i1</portID>
        <portName>a</portName>
        <portType>File</portType>
        <portDescription>port i1 description</portDescription>
       </inputPort>
       <inputPort>
        <portID>i2</portID>
        <portName>b</portName>
        <portType>File</portType>
        <portDescription>port i2 description</portDescription>
       </inputPort>
  </inputPorts>
  <outputPorts>
       <outputPort>
        <portID>o1</portID>
        <portName>c</portName>
        <portType>File</portType>
        <portDescription>port o1 description</portDescription>
```

```
        </outputPort>
    </outputPorts>
  </workflowInterface>
  <workflowBody mode="builtin">
    <builtin>svm</builtin>
  </workflowBody>
</workflow>
</workflowSpec>
```

Workflow Java Source Code

```
package datamining;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

import weka.classifiers.Classifier;
import weka.core.Instance;
import weka.core.Instances;
import weka.classifiers.functions.SMO;

public class SVM {
        public static BufferedReader readDataFile(String filename) {
                BufferedReader inputReader = null;
                try {
                        inputReader = new BufferedReader(new
FileReader(filename));
```

```
            } catch (FileNotFoundException ex) {
                System.err.println("File not found: " + filename);
            }
            return inputReader;
        }

        public static void writeDataFile(String fileName, String content) {
            try {
                File file = new File(fileName);
                // if file doesnt exists, then create it
                if (!file.exists()) {
                    file.createNewFile();
                }
                // true = append file
                FileWriter fileWritter = new FileWriter(file, true);
                BufferedWriter bufferWritter = new
BufferedWriter(fileWritter);
                bufferWritter.write(content);
                bufferWritter.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        public static void svmImplementor(String trainingData, String testData,
                String outputData) throws Exception {
            BufferedReader datafile = readDataFile(trainingData);
            BufferedReader testfile = readDataFile(testData);
            Instances data = new Instances(datafile);
            Instances test = new Instances(testfile);
```

```
                System.out.println("#read file success...");
                data.setClassIndex(data.numAttributes() - 1);
                test.setClassIndex(test.numAttributes() - 1);
                Classifier smo = new SMO();
                smo.buildClassifier(data);
                System.out.println("#classifier build success...");
                System.out.println(test.numInstances());
                for (int i = 0; i < test.numInstances(); i++) {
                        Instance testDataItem = test.instance(i);
                        double testDataItemsClass =
smo.classifyInstance(testDataItem);
                        System.out.println("#instance classified success...");
                        String content = "Data item: " + i + ", belong to class " +
                                        testDataItemsClass + "\r\n";
                        System.out.println(content);
                        // Write this to output file...
                        writeDataFile(outputData, content);
                }
        }
```

APPENDIX B: RANDOM FOREST P-WORKFLOW

Workflow Specification

```
<workflowSpec>
<workflow name="rf" root="true">
 <workflowInterface>
  <workflowDescription>simple rf workflow</workflowDescription>
  <inputPorts>
       <inputPort>
        <portID>i1</portID>
        <portName>a</portName>
        <portType>File</portType>
        <portDescription>port i1 description</portDescription>
       </inputPort>
       <inputPort>
        <portID>i2</portID>
        <portName>b</portName>
        <portType>File</portType>
        <portDescription>port i2 description</portDescription>
       </inputPort>
       <inputPort>
        <portID>i3</portID>
        <portName>a</portName>
        <portType>Integer</portType>
        <portDescription>port i3 description</portDescription>
       </inputPort>
  </inputPorts>
```

```
  <outputPorts>
        <outputPort>
          <portID>o1</portID>
          <portName>c</portName>
          <portType>File</portType>
          <portDescription>port o1 description</portDescription>
        </outputPort>
  </outputPorts>
 </workflowInterface>
 <workflowBody mode="builtin">
  <builtin>rf</builtin>
 </workflowBody>
</workflow>
</workflowSpec>
```

Workflow Java Source Code

```
package datamining;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import weka.classifiers.Classifier;
import weka.core.Instance;
import weka.core.Instances;
import weka.classifiers.functions.SMO;
import weka.classifiers.trees.RandomForest;
public class RF {
```

```
public static void rfImplementor(String trainingData, String testData,
                int numoftree, String outputData) throws Exception {
    BufferedReader datafile = new BufferedReader(new
FileReader(trainingData));
    BufferedReader testfile = new BufferedReader(new FileReader(testData));
            Instances data = new Instances(datafile);
            Instances test = new Instances(testfile);
            System.out.println("#read file success...");
            data.setClassIndex(data.numAttributes() - 1);
            test.setClassIndex(test.numAttributes() - 1);
            RandomForest rf = new RandomForest();
            rf.setNumTrees(numoftree);
            rf.setDebug(true);
            rf.buildClassifier(data);
            System.out.println("#classifier build success...");
            System.out.println(test.numInstances());
            String content = "";
            File file = new File(outputData);
            BufferedWriter bw = new BufferedWriter(new FileWriter(file,
true));
            for (int i = 0; i < test.numInstances(); i++) {
                Instance testDataItem = test.instance(i);
                double testDataItemsClass =
rf.classifyInstance(testDataItem);
                System.out.println("#instance classified success...");
                content = "Data item " + i + ", belong to class " +
testDataItemsClass;
                System.out.println(content);
                bw.write(content);
```

```
                bw.newLine();
                bw.flush();
            }
            if (bw != null) { bw.close();
            }
        }
}
```

APPENDIX C: BAYESIAN NETWORK P-WORKFLOW

Workflow Specification

```
<workflowSpec>
<workflow name="bayesnet" root="true">
 <workflowInterface>
  <workflowDescription>simple BayesNet workflow</workflowDescription>
  <inputPorts>
       <inputPort>
        <portID>i1</portID>
        <portName>a</portName>
        <portType>File</portType>
        <portDescription>port i1 description</portDescription>
       </inputPort>
       <inputPort>
        <portID>i2</portID>
        <portName>b</portName>
        <portType>File</portType>
        <portDescription>port i2 description</portDescription>
       </inputPort>
  </inputPorts>
  <outputPorts>
       <outputPort>
        <portID>o1</portID>
        <portName>c</portName>
        <portType>File</portType>
        <portDescription>port o1 description</portDescription>
```

```
        </outputPort>
    </outputPorts>
  </workflowInterface>
  <workflowBody mode="builtin">
    <builtin>bayesnet</builtin>
  </workflowBody>
</workflow>
</workflowSpec>
```

Workflow Java Source Code

```
package datamining;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import weka.classifiers.Classifier;
import weka.core.Instance;
import weka.core.Instances;
public class BayesNet {
        public static void bnImplementor(String trainingData, String testData,
String outputData) throws Exception {
      BufferedReader datafile = new BufferedReader(new
FileReader(trainingData));
      BufferedReader testfile = new BufferedReader(new FileReader(testData));
                Instances data = new Instances(datafile);
                Instances test = new Instances(testfile);
                System.out.println("#read file success...");
                data.setClassIndex(data.numAttributes() - 1);
                test.setClassIndex(test.numAttributes() - 1);
```

```
            // create a Bayes Network classifier
            weka.classifiers.bayes.BayesNet bn = new
weka.classifiers.bayes.BayesNet();
            bn.buildClassifier(data);

            System.out.println("#classifier build success...");
            System.out.println("number of instances: " + test.numInstances());
            String content = "";
            // Write this to output file...
            File file = new File(outputData);
            BufferedWriter bw = new BufferedWriter(new FileWriter(file,
true));
            System.out.println("#instance classified success...");
            for (int i = 0; i < test.numInstances(); i++) {
                  Instance testDataItem = test.instance(i);
                  double testDataItemsClass = ((Classifier)
bn).classifyInstance(testDataItem);
                  content = "Instance " + i + ", belong to class " +
testDataItemsClass;
                  System.out.println(content);
                  bw.write(content);
                  bw.newLine();
                  bw.flush();
            }
            if (bw != null) {
                  bw.close();
            }
      }
```

REFERENCES

1. Vouk, M.A. and M.P. Singh, *Quality of service and scientific workflows.* Quality of Numerical Software, 1996. 76: p. 77-89.

2. A. Chebotko, C.L., X. Fei, Z. Lai, S. Lu, J. Hua, and F. Fotouhi. *VIEW: a visual scientific workflow management system*. in *Proceedings of the First IEEE International Workshop on Scientific Workflows (SWF)*. 2007. Salt Lake City, UT, USA.

3. Ogasawara, E., et al., *An algebraic approach for data-centric scientific workflows.* Proc. of VLDB Endowment, 2011. 4(12): p. 1328-1339.

4. Juve, G. and E. Deelman, *Scientific workflows and clouds.* Crossroads, 2010. 16(3): p. 14-18.

5. Bharathi, S., et al. *Characterization of scientific workflows*. in *Workflows in Support of Large-Scale Science, 2008. WORKS 2008. Third Workshop on*. 2008. IEEE.

6. Kosar, T. and M. Livny, *A framework for reliable and efficient data placement in distributed computing systems.* Journal of Parallel and Distributed Computing, 2005. 65(10): p. 1146-1157.

7. Weiss, A., *Computing in the clouds.* networker, 2007. 11(4).

8. Foster, I., et al. *Cloud computing and grid computing 360-degree compared*. in *Grid Computing Environments Workshop, 2008. GCE'08*. 2008. Ieee.

9. Lohr, S., *Google and IBM join in 'cloud computing' research.* New York Times, 2007. 8.

10. Ricadela, A., *Computing heads for the clouds.* Business Week, 2007.

11. Juve, G., et al., *Characterizing and profiling scientific workflows.* Future Generation Computer Systems, 2013. 29(3): p. 682-692.

12. Brantner, M., et al. *Building a database on S3*. in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. 2008. ACM.

13. Buyya, R., C.S. Yeo, and S. Venugopal. *Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities*. in *High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on*. 2008. Ieee.

14. Moretti, C., et al. *All-pairs: An abstraction for data-intensive cloud computing*. in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*. 2008. IEEE.

15. Zhao, Y., et al. *Opportunities and challenges in running scientific workflows on the cloud*. in *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2011 International Conference on*. 2011. IEEE.

16. Hoffa, C., et al. *On the use of cloud computing for scientific workflows*. in *eScience, 2008. eScience'08. IEEE Fourth International Conference on*. 2008. IEEE.

17. Ji Liu, E.P., Patrick Valduriez, Marta Mattoso, *Parallelization of Scientific Workflows in the Cloud*. 2014.

18. Hey, A.J., S. Tansley, and K.M. Tolle, *The fourth paradigm: data-intensive scientific discovery*. Vol. 1. 2009: Microsoft Research Redmond, WA.

19. Gilbert, S. and N. Lynch, *Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services.* ACM SIGACT News, 2002. 33(2): p. 51-59.

20. Birman, K., Q. Huang, and D. Freedman, *Overcoming the 'D'in CAP: Using Isis2 to Build Locally Responsive Cloud Services.* Computer, 2011: p. 50-58.

21. Agrawal, D., et al., *Data management challenges in cloud computing infrastructures*, in *Databases in Networked Information Systems*. 2010, Springer. p. 1-10.

22. Yuan, D., et al., *A data placement strategy in scientific cloud workflows.* Future Generation Computer Systems, 2010. 26(8): p. 1200-1214.

23. Stewart, R.J., P.W. Trinder, and H.-W. Loidl, *Comparing high level mapreduce query languages*, in *Advanced Parallel Processing Technologies*. 2011, Springer. p. 58-72.

24. Grossman, R. and Y. Gu. *Data mining using high performance data clouds: experimental studies using sector and sphere*. in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2008. ACM.

25. *NIST Big Data Working Group*. Available from: http://bigdatawg.nist.gov/.

26. Brown, D.A., et al., *A case study on the use of workflow technologies for scientific analysis: Gravitational wave data analysis*. 2006: Springer.

27. Mohan, A., et al. *A NoSQL Data Model for Scalable Big Data Workflow Execution*. in *Big Data (BigData Congress), 2016 IEEE International Congress on*. 2016. IEEE.

28. Demchenko, Y., et al. *Addressing big data issues in scientific data infrastructure*. in *Collaboration Technologies and Systems (CTS), 2013 International Conference on*. 2013. IEEE.

29. Eaton, C., et al., *Understanding big data: Analytics for enterprise class hadoop and streaming data.* FREE ebook, 2012.

30. Kashlev, A. and S. Lu. *A System Architecture for Running Big Data Workflows in the Cloud*. in *Services Computing (SCC), 2014 IEEE International Conference on*. 2014. IEEE.

31. Wang, J., et al., *Big data applications using workflows for data parallel computing.* Computing in Science & Engineering, 2014. 16(4): p. 11-21.

32. Lotfy, A.E., et al., *A middle layer solution to support ACID properties for NoSQL databases.* Journal of King Saud University-Computer and Information Sciences, 2016. 28(1): p. 133-145.

33. Lakshman, A. and P. Malik, *Cassandra-a decentralized structured storage system. 2009.* Cited on, 2015: p. 17.

34. Chodorow, K., *MongoDB: the definitive guide*. 2013: " O'Reilly Media, Inc.".

35. Abouelhoda, M., S.A. Issa, and M. Ghanem, *Tavaxy: Integrating Taverna and Galaxy workflows with cloud computing support.* BMC bioinformatics, 2012. 13(1): p. 77.

36. Emeakaroha, V.C., et al., *Managing and optimizing bioinformatics workflows for data analysis in clouds.* Journal of grid computing, 2013. 11(3): p. 407-428.

37. Vöckler, J.-S., et al. *Experiences using cloud computing for a scientific workflow application*. in *Proceedings of the 2nd international workshop on Scientific cloud computing*. 2011. ACM.

38. Wu, Z., et al., *A market-oriented hierarchical scheduling strategy in cloud workflow systems.* The Journal of Supercomputing, 2013. 63(1): p. 256-293.

39. De Oliveira, D., et al. *Scicumulus: A lightweight cloud middleware to explore many task computing paradigm in scientific workflows*. in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. 2010. IEEE.

40. *OpenStack*. Available from: https://www.openstack.org/.

41. *Amazon Elastic Compute Cloud (EC2)*. Available from: http:////aws.amazon.com/ec2/.

42. *FutureSystems: Digital Science Center, School of Informatics and Computing, Indiana University | FutureSystems Portal*. Available from: https://portal.futuresystems.org/.

43. Kosar, T., et al., *Data placement in widely distributed environments.* Advances in Parallel Computing, 2005. 14: p. 105-128.

44. Kayyoor, A.K., A. Deshpande, and S. Khuller, *Data placement and replica selection for improving co-location in distributed environments.* Energy (Joules), 2012. 1000: p. 1500.

45. Chervenak, A., et al. *Data placement for scientific applications in distributed environments*. in *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing*. 2007. IEEE Computer Society.

46. Ghemawat, S., H. Gobioff, and S.-T. Leung. *The Google file system*. in *ACM SIGOPS operating systems review*. 2003. ACM.

47. Borthakur, D., *The hadoop distributed file system: Architecture and design.* Hadoop Project Website, 2007. 11(2007): p. 21.

48. *Pegasus | Workflow Management System*. Available from: http://pegasus.isi.edu/index.php.

49. Deelman, E., et al., *Pegasus: Mapping large-scale workflows to distributed resources*. 2006: Springer.

50. Deelman, E., et al., *Pegasus: A framework for mapping complex scientific workflows onto distributed systems.* Scientific Programming, 2005. 13(3): p. 219-237.

51. *Nimbus*. Available from: http://www.nimbusproject.org/.

52. *Eucalyptus*. Available from: https://www.eucalyptus.com/.

53. Çatalyürek, Ü.V., K. Kaya, and B. Uçar. *Integrated data placement and task assignment for scientific workflows in clouds*. in *Proceedings of the fourth international workshop on Data-intensive distributed computing*. 2011. ACM.

54. Er-Dun, Z., X. Xing-Xing, and C. Yi. *A data placement strategy based on genetic algorithm for scientific workflows*. in *Computational Intelligence and Security (CIS), 2012 Eighth International Conference on*. 2012. IEEE.

55. Rodriguez, M.A. and R. Buyya. *A Responsive Knapsack-Based Algorithm for Resource Provisioning and Scheduling of Scientific Workflows in Clouds*. in *2015 44th International Conference on Parallel Processing*. 2015.

56. Yu, J., R. Buyya, and C.K. Tham. *Cost-based scheduling of scientific workflow applications on utility grids*. in *e-Science and Grid Computing, 2005. First International Conference on*. 2005. Ieee.

57. Abrishami, S., M. Naghibzadeh, and D.H.J. Epema, *Cost-Driven Scheduling of Grid Workflows Using Partial Critical Paths.* IEEE Transactions on Parallel and Distributed Systems, 2012. 23(8): p. 1400-1414.

58. Wieczorek, M., A. Hoheisel, and R. Prodan, *Towards a general model of the multi-criteria workflow scheduling on the grid.* Future Generation Computer Systems, 2009. 25(3): p. 237-256.

59. Malawski, M., et al. *Cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds*. in *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*. 2012.

60. Yu, J. and R. Buyya, *Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms.* Scientific Programming, 2006. 14(3-4): p. 217-230.

61. Lin, C. and S. Lu. *Scheduling scientific workflows elastically for cloud computing*. in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. 2011. IEEE.

62. Lin, C. and S. Lu. *SCPOR: An elastic workflow scheduling algorithm for services computing*. in *Service-Oriented Computing and Applications (SOCA), 2011 IEEE International Conference on*. 2011. IEEE.

63. Arabnejad, H. and J.G. Barbosa, *List Scheduling Algorithm for Heterogeneous Systems by an Optimistic Cost Table.* IEEE Transactions on Parallel and Distributed Systems, 2014. 25(3): p. 682-694.

64. Topcuoglu, H., S. Hariri, and M.-y. Wu, *Performance-effective and low-complexity task scheduling for heterogeneous computing.* IEEE transactions on parallel and distributed systems, 2002. 13(3): p. 260-274.

65. Durillo, J.J., H.M. Fard, and R. Prodan. *Moheft: A multi-objective list-based method for workflow scheduling*. in *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*. 2012. IEEE.

66. Li, X., et al. *An improved max-min task-scheduling algorithm for elastic cloud*. in *Computer, Consumer and Control (IS3C), 2014 International Symposium on*. 2014. IEEE.

67. Bochenina, K., et al., *A clustering-based approach to static scheduling of multiple workflows with soft deadlines in heterogeneous distributed systems.* Procedia Computer Science, 2015. 51: p. 2827-2831.

68. Deldari, A., et al. *A clustering approach to schedule workflows to run on the cloud*. in *2016 Eighth International Conference on Information and Knowledge Technology (IKT)*. 2016.

69. Yang, T. and A. Gerasoulis. *A fast static scheduling algorithm for DAGs on an unbounded number of processors*. in *Proceedings of the 1991 ACM/IEEE Conference on Supercomputing (Supercomputing '91)*. 1991.

70. MadadyarAdeh, M. and J. Bagherzadeh. *An improved ant algorithm for grid scheduling problem using biased initial ants*. in *Computer Research and Development (ICCRD), 2011 3rd International Conference on*. 2011. IEEE.

71. Wen, X., M. Huang, and J. Shi. *Study on resources scheduling based on ACO allgorithm and PSO algorithm in cloud computing*. in *Distributed Computing and Applications to Business, Engineering & Science (DCABES), 2012 11th International Symposium on*. 2012. IEEE.

72. Ge, Y. and G. Wei. *GA-based task scheduler for the cloud computing systems*. in *Web Information Systems and Mining (WISM), 2010 International Conference on*. 2010. IEEE.

73. Zhao, C., et al. *Independent tasks scheduling based on genetic algorithm in cloud computing*. in *Wireless Communications, Networking and Mobile Computing, 2009. WiCom'09. 5th International Conference on*. 2009. IEEE.

74. Wu, Z., et al. *A revised discrete particle swarm optimization for cloud workflow scheduling*. in *Computational Intelligence and Security (CIS), 2010 International Conference on*. 2010. IEEE.

75. Ebrahimi, M., et al. *BDAP: A Big Data Placement Strategy for Cloud-Based Scientific Workflows*. in *2015 IEEE First International Conference on Big Data Computing Service and Applications*. 2015.

76. Ebrahimi, M., et al., *Task And Data Allocation Strategies for Big Data Workflows.* International Journal of Big Data (IJBD), 2015. 2(2): p. 28-42.

77. Ebrahimi, M., et al. *TPS: A task placement strategy for big data workflows*. in *2015 IEEE International Conference on Big Data (Big Data)*. 2015.

78. Mohan, A., et al. *Scheduling big data workflows in the cloud under budget constraints*. in *Big Data (Big Data), 2016 IEEE International Conference on*. 2016. IEEE.

79. Wang, X., et al., *Optimizing the makespan and reliability for workflow applications with reputation and a look-ahead genetic algorithm.* Future Generation Computer Systems, 2011. 27(8): p. 1124-1134.

80. Durillo, J.J., V. Nae, and R. Prodan. *Multi-objective workflow scheduling: An analysis of the energy efficiency and makespan tradeoff*. in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*. 2013. IEEE.

81. Tangudu, V. and M. Mishra. *Estimating makespan using double trust thresholds for workflow applications*. in *Proceedings of the CUBE International Information Technology Conference*. 2012. ACM.

82. Weise, T., *Global optimization algorithms-theory and application.* Self-Published, 2009.

83. Venugopal, S. and R. Buyya, *An SCP-based heuristic approach for scheduling distributed data-intensive applications on global grids.* Journal of Parallel and Distributed Computing, 2008. 68(4): p. 471-487.

84. Yang, Y., et al. *An algorithm in SwinDeW-C for scheduling transaction-intensive cost-constrained cloud workflows*. in *eScience, 2008. eScience'08. IEEE Fourth International Conference on*. 2008. IEEE.

85. Abrishami, S., M. Naghibzadeh, and D.H. Epema, *Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds.* Future Generation Computer Systems, 2013. 29(1): p. 158-169.

86. Rodriguez Sossa, M. and R. Buyya, *Deadline based Resource Provisioning and Scheduling Algorithmfor Scientific Workflows on Clouds.* 2014.

87. Liu, L., et al. *A Survey on Workflow Management and Scheduling in Cloud Computing*. in *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*. 2014. IEEE.

88. *Google App Engine: Platform as a Service*. Available from: https://cloud.google.com/appengine/docs.

89. *Microsoft Azure: Cloud Computing Platform & Services*. Available from: http://azure.microsoft.com/en-us/.

90. Tan, P.-N., M. Steinbach, and V. Kumar, *Introduction to data mining*. Vol. 1. 2006: Pearson Addison Wesley Boston.

91. *Montage - Image Mosaic Software for Astronomers*. Available from: http://montage.ipac.caltech.edu/index.html.

92. Berriman, G.B., et al. *Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand*. in *Astronomical Telescopes and Instrumentation*. 2004. International Society for Optics and Photonics.

93. Graves, R., et al., *CyberShake: A physics-based seismic hazard model for southern California.* Pure and Applied Geophysics, 2011. 168(3-4): p. 367-381.

94. *Southern California Earthquake Center*. Available from: http://www.scec.org/.

95. Callaghan, S., et al. *Reducing time-to-solution using distributed high-throughput mega-workflows-experiences from SCEC CyberShake*. in *eScience, 2008. eScience'08. IEEE Fourth International Conference on*. 2008. IEEE.

96. Deelman, E., et al. *Managing large-scale workflow execution from resource provisioning to provenance tracking: The cybershake example*. in *e-Science and Grid Computing, 2006. e-Science'06. Second IEEE International Conference on*. 2006. IEEE.

97. *USC Epigenome Center*. Available from: http://epigenome.usc.edu/.

98. *Illumina | Sequencing and array-based solutions for genetic research*. Available from: http://www.illumina.com/.

99. *Maq: Mapping and Assembly with Qualities*. Available from: http://maq.sourceforge.net/.

100. *LIGO project, LIGO—laser interferometer gravitational wave observatory*. Available from: https://www.ligo.caltech.edu/.

101. Althouse, W.E. and M.E. Zucker, *LIGO: The laser interferometer gravitational-wave observatory.* Science, 1992. 256(5055): p. 325.

102. Livny, J., et al., *High-throughput, kingdom-wide prediction and annotation of bacterial non-coding RNAs.* PloS one, 2008. 3(9): p. e3197.

103. *DAGMan - Directed Acyclic Graph Manager*. Available from: http://research.cs.wisc.edu/htcondor/dagman/dagman.html.

104. *DATAVIEW*. Available from: http://dataview.org/.

105. A. Kashlev, S.L., A. Chebotko, *A System Architecture for Running Big Data Workflows in the Cloud*. 2014.

106. Hull, D., et al., *Taverna: a tool for building and running workflows of services.* Nucleic acids research, 2006. 34(suppl 2): p. W729-W732.

107. Ludäscher, B., et al., *Scientific workflow management and the Kepler system.* Concurrency and Computation: Practice and Experience, 2006. 18(10): p. 1039-1065.

108. Freire, J., et al., *Managing rapidly-evolving scientific workflows*, in *Provenance and Annotation of Data*. 2006, Springer. p. 10-18.

109. Zhao, Y., et al. *Swift: Fast, reliable, loosely coupled parallel computation*. in *Services, 2007 IEEE Congress on*. 2007. IEEE.

110. Lin, C., et al. *Service-oriented architecture for VIEW: A visual scientific workflow management system*. in *Services Computing, 2008. SCC'08. IEEE International Conference on*. 2008. IEEE.

111. *Dropbox*. Available from: https://www.dropbox.com.

112. *Google Drive*. Available from: https://www.google.com/drive/.

113. *Microsoft OneDrive*. Available from: https://onedrive.live.com.

114. Jayyousi, T.W. and R.G. Reynolds, *Extracting Urban Occupational Plans Using Cultural Algorithms [Application Notes]*. IEEE Computational Intelligence Magazine, 2014. 9(3): p. 66-87.

115. Reynolds, R.G., *An overview of cultural algorithms.* Advances in Evolutionary Computation, 1999.

116. Ali, M.Z. and R.G. Reynolds, *Cultural algorithms: a Tabu search approach for the optimization of engineering design problems.* Soft Computing, 2014. 18(8): p. 1631-1644.

117. Kobti, Z., R.G. Reynolds, and T. Kohler, *Agent-based modeling of cultural change in swarm using cultural algorithms.* Funded by NSF grant BCS-0119981, 2004.

118. Reynolds, R.G. and B. Peng. *Cultural algorithms: modeling of how cultures learn to solve problems*. in *Tools with Artificial Intelligence, 2004. ICTAI 2004. 16th IEEE International Conference on*. 2004. IEEE.

119. Reynolds, R.G. and B. Peng, *Knowledge learning and social swarms in cultural systems.* Journal of Mathematical Sociology, 2005. 29(2): p. 115-132.

120. Ali, M.Z., N. Awad, and R.G. Reynolds. *Balancing search direction in cultural algorithm for enhanced global numerical optimization*. in *Swarm Intelligence (SIS), 2014 IEEE Symposium on*. 2014. IEEE.

ABSTRACT

DATA PLACEMENT AND TASK MAPPING OPTIMIZATION FOR BIG DATA WORKFLOWS IN THE CLOUD

by

MAHDI EBRAHIMI

August 2017

Advisor: Dr. Shiyong Lu and Dr. Song Jiang

Major: Computer Engineering

Degree: Doctor of Philosophy

Data-centric workflows naturally process and analyze a huge volume of datasets. In this new era of Big Data there is a growing need to enable data-centric workflows to perform computations at a scale far exceeding a single workstation's capabilities. Therefore, this type of applications can benefit from distributed high performance computing (HPC) infrastructures like cluster, grid or cloud computing.

Although data-centric workflows have been applied extensively to structure complex scientific data analysis processes, they fail addressing the big data challenges as well as leveraging the capability of dynamic resource provisioning in the Cloud. The concept of “big data workflows” is proposed by our research group as the next generation of data-centric workflow technologies to address the limitations of existing workflows technologies in addressing big data challenges.

Executing big data workflows in the Cloud is a challenging problem as workflow tasks and data are required to be partitioned, distributed and assigned to the cloud execution sites. In running such big data workflows in the cloud distributed across physical locations, the workflow execution time and cost efficiency highly depends on the initial placement of tasks and datasets across the multiple virtual machines in the Cloud.

In this dissertation, I propose BDAP strategy (Big Data Placement strategy) for data placement and TPS (Task Placement Strategy) for task placement, which improve workflow performance by minimizing data movement across multiple virtual machines in the Cloud during the workflow execution. In addition, I 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. In this dissertation, I 1) formalize data and task placement problems in workflows, 2) propose a data placement algorithm that considers both initial input dataset and intermediate datasets obtained during workflow run, 3) propose a workflow scheduling strategy to minimize the workflow execution cost once the deadline is provided by user and 4) perform extensive experiments in the distributed environment to validate that our proposed strategies provide an effective data and task placement solution to distribute and place big datasets and tasks into the appropriate cloud virtual machines within reasonable time.

AUTOBIOGRAPHICAL STATEMENT

Mahdi Ebrahimi is currently a PhD candidate in the Big Data Research Lab at Wayne State University under the supervision of Dr. Shying Lu. His main research interest is in the field of big data management, with the focus on big data placement and task mapping optimization for cloud-based workflows. His broader interests include big data, data science, cloud workflow security, cloud computing, Internet of Things (IoT), and multi-objective optimization. He has published several research articles in peer-reviewed international journal and conferences, including International Journal of Big Data (IJBD), IEEE International Conference on Big Data (IEEE BigData), IEEE International Conference on Big Data Computing Service and Application (IEEE BigDataService), IEEE International Congress on Big Data (IEEE BigData Congress), and others. He is a member of IEEE and ACM.