WORKFLOW OPTIMIZATION IN DISTRIBUTED COMPUTING ENVIRONMENT FOR STREAM-BASED DATA PROCESSING MODEL

SAIMA GULZAR AHMAD

Faculty of Computer Science and Information Technology
UNIVERSITY OF MALAYA
KUALA LUMPUR

2017

WORKFLOW OPTIMIZATION IN DISTRIBUTED COMPUTING ENVIRONMENT FOR STREAM-BASED DATA PROCESSING MODEL

SAIMA GULZAR AHMAD

THESIS SUBMITTED IN FULFILMENT
OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2017

UNIVERSITI MALAYA

ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: (I.C./Passport No.:)

Registration/Matrix No.:

Name of Degree:

Title of Project Paper/Research Report/Dissertation/Thesis ("this Work"):

Field of Study:

I do solemnly and sincerely declare that:

(1) I am the sole author/writer of this Work;
(2) This work is original;
(3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
(4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
(5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
(6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature Date

Subscribed and solemnly declared before,

Witness's Signature Date

Name:
Designation:

ABSTRACT

With the advancement in science and technology numerous complex scientific applications can be executed in heterogeneous computing environment. However, the bottle neck is efficient scheduling algorithms. Such complex applications can be expressed in the form of workflows. Geographically distributed heterogeneous resources can execute such workflows in parallel. This enhances the workflow execution. In data-intensive workflows, heavy data moves across the execution nodes. This causes high communication overhead. To avoid such overheads many techniques have been used, however in this thesis stream-based data processing model is used in which data is processed in the form of continuous instances of data items. Data-intensive workflow optimization is an active research area because numerous applications are producing huge amount of data that is increasing exponentially day by day.

This thesis proposes data-intensive workflow optimization algorithms. The first algorithm architecture consists of two phases *a*) workflow partitioning, and *b*) partitions mapping. Partitions are made in such a way that minimum data should move across the partitions. It enables heavy data processing locally on same execution node because each partition is mapped to one execution node. It overcomes the high communication costs. In the mapping phase, a partition is mapped on that execution node which offers minimum execution time. Eventually, the workflow is executed. The second algorithm is a variation in first algorithm in which data parallelism is introduced in each partition. Most compute intensive task in each partition is identified and data parallelism is applied to that task. It reduces the execution time of that compute intensive tasks. The simulation results prove that proposed algorithms outperform from state of the art algorithms for variety of workflows. The datasets used for performance evaluation are synthesized as well as workflows derived from real world applications. The workflows derived from real world

applications include Montage and Cybershake. Synthesized workflows were generated with different sizes, shapes and densities to evaluate the proposed algorithms. The simulation results shows 60% reduced latency with 47% improvement in the throughput. Similarly, when data parallelism is introduced in the algorithm the performance of the algorithm improved further by 12% in latency and 17% in throughput when compared to PDWA algorithm. In the real time stream processing framework the experiments were performed using STORM with a use-case data-intensive workflow (EURExpressII). Experiments show that PDWA outperforms in terms of execution time of the workflow with different input data size.

ABSTRAK

Dengan kemajuan dalam bidang sains dan teknologi, pelbagai aplikasi saintifik yang kompleks boleh dilaksanakan dalam persekitaran pengkomputeran heterogen. Walau bagaimanapun, kesuntukan ialah algoritma penjadualan yang cekap. Aplikasi yang kompleks sebegini boleh dipersembahkan dalam bentuk aliran kerja. Sumber heterogen yang teragih boleh melaksanakan aliran kerja ini secara selari. Ini memperbaikan pelaksanaan aliran kerja. Dalam aliran kerja data intensif, pergerakan data besar-besaran berlaku di seluruh nod-nod pelaksanaan. Ini menyebabkan overhed komunikasi yang tinggi. Untuk mengelakkan overhed pelbagai teknik telah digunakan. Namun begitu, dalam tesis ini model pemprosesan data berasaskan penstriman digunakan di mana data diproses dalam bentuk data item diproses dalam aliran berterusan. Pengoptimunan aliran kerja data intensif adalah bidang penyelidikan yang aktif kerana banyak aplikasi menghasilkan data yang besar semakin meningkat dengan pesat dari hari ke hari.

Tesis ini mencadangkan algoritma pengoptimunan aliran kerja data intensif. Seni bina algoritma pertama terdiri daripada dua fasa *a*) pemetakan aliran kerja, dan *b*) pemetaan pemetakan. Pemetakan dibuat di mana data yang bergerak di seluruh pemetakan adalah minimum. Ia membolehkan pemprosesan data yang banyak pada nod pelaksanaan yang sama kerana setiap pemetakan dipetakan kepada satu nod pelaksanaan. Ia mengatasi kos komunikasi yang tinggi. Dalam fasa pemetaan, pemetakan yang dipetakan pada nod pelaksanaan yang menawarkan masa pelaksanaan yang minimum. Akhirnya, aliran kerja yang dilaksanakan. Algoritma kedua adalah variasi algoritma pertama di mana keselarian data diperkenalkan dalam setiap pemetakan. Kebanyakan tugas pengiraan intensif dalam setiap pemetakan dikenal pasti dan keselarian data digunakan untuk tugas tersebut. Ia mengurangkan masa pelaksanaan itu tugas pengiraan intensif. Keputusan simulasi membuktikan bahawa algoritma yang dicadangkan melebihi jangkauan algoritma-algoritma

terkini untuk aneka aliran kerja. Dataset digunakan untuk penilaian prestasi termasuk yang disintesis dan yang diperolehi daripada aplikasi aliran kerja dunia sebenar. Aliran kerja yang diperolehi daripada aplikasi dunia sebenar termasuk Montage dan Cybershake. Aliran kerja disintesis telah dihasilkan dengan saiz, bentuk serta kepadatan yang berbeza untuk menilai algoritma yang dicadangkan. Keputusan simulasi menunjukkan masa pedam dikurangkan sebanyak 60% dengan peningkatan 47% dalam daya pemprosesan. Begitu juga, apabila keselarian data diperkenalkan dalam algoritma prestasi algoritma bertambah baik sebanyak 12% dalam kependaman dan 17% dalam daya pemprosesan berbanding dengan algoritma PDWA. Dalam rangka kerja pemprosesan aliran masa nyata, eksperimen telah dijalankan dengan menggunakan STORM dengan aliran kerja data intensif kes kajian (EURExpressII). Eksperimen menunjukkan bahawa PDWA melebihi jangkauan dari segi masa pelaksanaan aliran kerja dengan saiz data input yang berbeza.

ACKNOWLEDGEMENTS

This dissertation has been made possible with the supervision of Dr. Chee Sun LIEW. I would like to pay my heartiest gratitude to him for his priceless guidance and encouragement that gives me confidence to complete this thesis despite of many obstacles. I am also thankful to my co-supervisor Dr. Ehsan Ullah Minur for his immense help, invaluable advice and moral support. Without the help of my supervisors and their support I probably would not be able to complete this dissertation.

In addition to my supervisors, I would like acknowledge the support I received during my research work from Dr. Samee Ullah Khan and Dr. Mustafa Rafique. They provided valuable guidance for my research publications. Not to forget my lab peer, Mr. Kheng Ghee NG, who did great efforts with me to perform the evaluation of proposed work in real-world frame work. I thank him for his collaboration in this research and hope to continue work with him even after I finish PhD.

I would not forget my funding agencies. This research was supported by the Ministry of Education, Malaysia (FRGS FP051-2013A and UMRG RP001F-13ICT). I am obliged to Dr. Chee Sun LIEW to facilitate this work by arranging these funds.

Finally, I am thankful to my family for their understanding over the years. My deepest gratitude goes to my parents for their unconditional love, acceptance and moral support.

This thesis is dedicated to my beloved parents and children

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

A revolution has been observed in the emerging science and the way how technology has been used in the last two decades. Despite the facts that the development in science and technology has solved many multidisciplinary and complex problems, it has also introduced many challenges. Initially the workflows were associated with business processes only. The concept has been used by the scientific community and scientists started modeling complex experiments and applications as workflows. Major difference between business and scientific workflows is that the business workflows are mostly task-oriented and control-driven while scientific workflows can be data-driven as well as control-driven (Shields, 2007). Large-scale experimentation and extensive simulations in modern science are continuously generating huge amount of data. Such complex processes are comprised of sequences of steps, which lead to the science of workflow design, management and execution. Workflows help to make such tedious and data-intensive processes manageable by modeling its steps in proper sequence. Scientific Workflow is a term that refers to the activity of defining the sequence of tasks needed to manage any computational process. Tasks take input from preceding tasks or from data resources and carry out predefined computations on the data. The output of the tasks are then passed to the successor tasks.

Workflows are managed by Workflow Management Systems (WMS), a comprehensive description of number of WMS can be viewed in (LIEW et al., 2016). The basic components of WMS include workflow composition, optimization, execution, and provenance. At first an abstract workflow, a high level workflow is composed. The logical sequence in which the workflow steps will be executed is referred as abstract workflow.

The resources are not binded with the tasks at this stage. The workflow management system finds and map appropriate resources to finalize the execution of workflow tasks, the resulting workflow is called concrete workflow. The later stages include provenance mechanism which keeps the history of workflow data that is useful in the resource mapping phase in determining the optimization approaches and parameters. The provenance data is important for future relevant experimentation and analysis (LIEW, 2012). Workflows can be expressed using various languages like DISPEL (LIEW et al., 2013), BPEL (Slominski, 2007) and YAWL (van der Aalst & ter Hofstede., 2005). Some workflows management systems have their own workbench to compose and design workflows such as Taverna[1].

1.1 Data-Intensive Workflows

Scientific community is experiencing a TSUNAMI of data, that needs to be manipulated. For example, Pan-STARRS[2], the Panoramic Survey Telescope and Rapid Response System has 4 X 4.1 giga-pixel resolution digital cameras that loads nearly 700 new databases which store nightly detected objects each day, merges 50,000 databases with existing 12 offline databases each week and captures greater than one petabytes of raw data and generates one terabytes into the catalog databases each year. In addition, a recent study reports (Berriman & Groom, 2011) several hundreds of petabytes of astronomical data is gathered each day and it is growing quickly day by day.

The science of workflows has made convenient to manage and handle such a big data easily. Data-intensive applications in astronomy, geology, bio-informatics, bio-medical science,and e-commerce need special consideration for enactment. These applications have led researchers to design and develop data-intensive workflow management sys-

[1]http://www.taverna.org.uk/download/workbench

[2]http://pan=starrs.ifa.hawaii.edu

tems. We can find a study and comparisons of these systems presented in (Yu & Buyya., 2005.; Deelman et al., 2009) in a comprehensive way. In contrast to task-oriented workflows, aspects like data storage, data movement, communication and computation costs must be considered in data-intensive workflow enactment. Workflows can be classified into two groups, *a*) control-driven, and *b*) data-driven workflows. In control-driven workflows the precedence of tasks is based on the shift of control. Workflows are represented as a sequence of processes and each parent process needs to be completed before the start of child process. Data-intensive applications are modeled as data-intensive workflows. Hence, these are mostly data-driven and the dependencies between the activities represent the flow of data. In data-intensive workflows the data workload is significantly higher than computational workload. That is why, data-intensive workflow execution requirements are different than control-driven workflows. Since, in data-intensive workflow huge amount of data is being processed therefore, data transfer cost between execution nodes, data storage cost, data processing cost, the resource interconnection bandwidth, resource buffering capability, and many other aspects are of major concerns in data-intensive workflows enactment. Data-intensive workflows can be modeled in various ways but commonly workflows are modeled as Directed Acyclic Graphs (DAGs) (Xu et al., 2013) which is a graphical representation of tasks and data flow. The vertices or nodes represent tasks and edges show the precedence of the tasks and data flow. Successor tasks cannot start execution until predecessor task provide enough data for its execution. However, in the task-based workflows the successor task need complete input data to start its execution once predecessor has been executed.

In recent years streaming model of workflow has gained immense popularity in scientific community, especially in data-intensive applications. For instance, sensor-based scientific experimentation produces live streaming data that require minimum latency and maximum throughput to avoid loss of data and its deletion. Same parameters are crucial

for the enactment of workflows which involves data stored in databases after prior experimentation. The workflow enactment can be made even better using data streaming model of workflows (Liew et al., 2010). In the context of data streaming the continuous stream of data is a chain of data items produced from any scientific activity. In workflows predefined computations are carried out on each data item of a data stream. It is a unidirectional transformation process which means that data can only move downstream. Output of upstream tasks is the input of downstream tasks. In data-intensive applications the concept of data streaming has emerged to be highly useful because processing large scale data on execution node at a time incurs high computation cost and limitations of the computing capacities of resources is also a bottle neck. Computation of large amount of data in the form of data streams causes an inherent parallelism which enhances the performance of data-intensive workflow execution. In addition data streaming model reduces overhead by reducing *a*) I/O to the disk, and *b*) instantiation cost of the workflow nodes.

Scheduling in data-intensive WMS has key importance because the performance of WMS depend on it to the significant extent. We can consider it in two parts *a*) mapping, and *b*) scheduling in execution phase of a WMS. Mapping is a logical distribution of tasks for resources while scheduling is a concrete plan and sequence in which task will be executed on certain resources. This research work revolves around the scheduling of data-intensive workflows and address the workflow optimization by improving techniques of scheduling workflows for more than one performance metrics.

1.2 Motivation

Data-intensive science has emerged as a fourth paradigm (Collins, 2009) during the last decade. It happened because of the exponential rise in data growth not only by the scientific community but also by social networks like Google, twitter, facebook etc. Few example of data-intensive applications are as follows

- "IBM" claims that "every day, we creates 2.5 quintillion bytes of data, 90% of the data in the world today has been created in the last two years (Aniello et al., 2013).

- DOMO, a business intelligence company recently reported (Aniello et al., 2013) that 3125 photos are added on Flickr, 34722 likes are sent on Facebook, more than 100,000 tweets are done on Twitter, each minute.

- Data is gathering so easily and quickly that has exceed the speed of its processing and management. Astronomers are collecting huge amount of data not only by volume but also with increasing complexity and verity. According to the predictions made in a study (Berriman & Groom, 2011), one petabytes of astronomical public data is electronically accessible and this volume is growing at 0.5 petabytes per year.

- Large Synoptic Survey Telescope (LSST)[3], is 8.4m large synoptic survey telescope. It will survey the sky deeply in multiple colors and explore the mysteries of sky with its three billion pixel camera. This telescope will produce enormous volume of data, 20 terabytes per night leading to a database of 60 petabytes of raw data over ten years. The predictions of data size growth has been made due to emerging projects. Similar anticipations are reported for upcoming project ALMA[4].

- A major decision in astronomy was taken in 2012 when it was decided to install Square Kilometer Array[5] (SKA) in the deserts of South Africa and Australia. SKA was installed to know the secrets of the world with world's largest radio telescope. Raw data will be available for public project website[6] in the form of zip files. The

[3]http://www.lsst.org/lsst/

[4]http://www.eso.org/sci/facilities/alma.html

[5]https://www.skatelescope.org/

[6]https://www.skatelescope.org/

research and scientific studies behind the location of the SKA high and mid frequency telescopes in South Africa data is 8.7 gigabytes Zip files, while data of low frequency telescope in Australia is 16 gigabytes Zip files. The data obtained by SKA is such a huge size that collected data in a single day would take nearly millions of years to play back on an iPod. SKA central computer will have the processing power of about one hundred million PCs. The dishes of the SKA will produce 10 times the global Internet traffic.

Big data is usually characterized by with four "Vs" that is high Volume, Variety, Velocity and Value of data. In order to handle each "V", workflow management encompasses many challenges. Value is mostly considered in business oriented scenarios. These challenges correspond to different dimensions of data-intensive workflow optimization. There is a continuous development in the infrastructure of computing systems and this process will continue to progress to tackle upcoming challenges of modern scientific development in future.

Now-a-days stream computing is the efficient solution to manipulate and compute information from such big data. The response time of big data stream applications is always required to be the minimum for gigabytes of live data streams and petabytes of archived simulated data streams. Big data stream applications modeled as data-intensive workflows are aimed to be processed with minimum latency. The main challenge in data-intensive workflow execution is to achieve reduced latency when execution nodes have to process heavy data streams.

In this research work, we propose the data-intensive workflow optimization (scheduling) algorithms by using stream-based data processing model. Scientific data-intensive workflows are used for simulations and experiments. The proposed work aims to reduce the latency or execution time and enhance the throughput.

1.3 Problem Statement and Research Objectives

The problem statement of this dissertation is stated as follows. The stream-based data processing model is a proven smart method to enhance the performance of data-intensive workflow scheduling. The latency and throughput of these workflows can be further improved by using data parallelism phenomenon. On these grounds, we propose algorithms that provide reduced latency and enhanced throughput as compared to other state of the art algorithms. It is proved by simulations and implementations in a real-time streaming framework.

This dissertation revolves around the following objectives.

- **Obj.1:** To propose and validate the scientific workflow optimization algorithm that reduces schedule length/execution time.
- **Obj.2:** To propose and validate data-intensive workflow optimization algorithm for stream-based data processing model that optimizes (reduce) latency and (increases) throughput.
- **Obj.3:** To apply and validate data-parallelism phenomena in order to enhance further the performance of proposed algorithm in Obj.2.
- **Obj.4:** To evaluate the proposed work in real-time streaming framework(STORM) using real-application based workflow.

We have divided our research work into four phases to achieve the above research objectives, which are also depicted in Fig. 1.1. In the phase 1, the literature of scientific workflows is reviewed . To make the concepts clear, an algorithm is proposed that optimizes the makespan of scientific workflows. The proposed algorithm named Hybrid Genetic Algorithm (HGA) is evaluated and verified using scientific workflows. In the next phase, the research work is narrowed down to data-intensive scientific workflows.

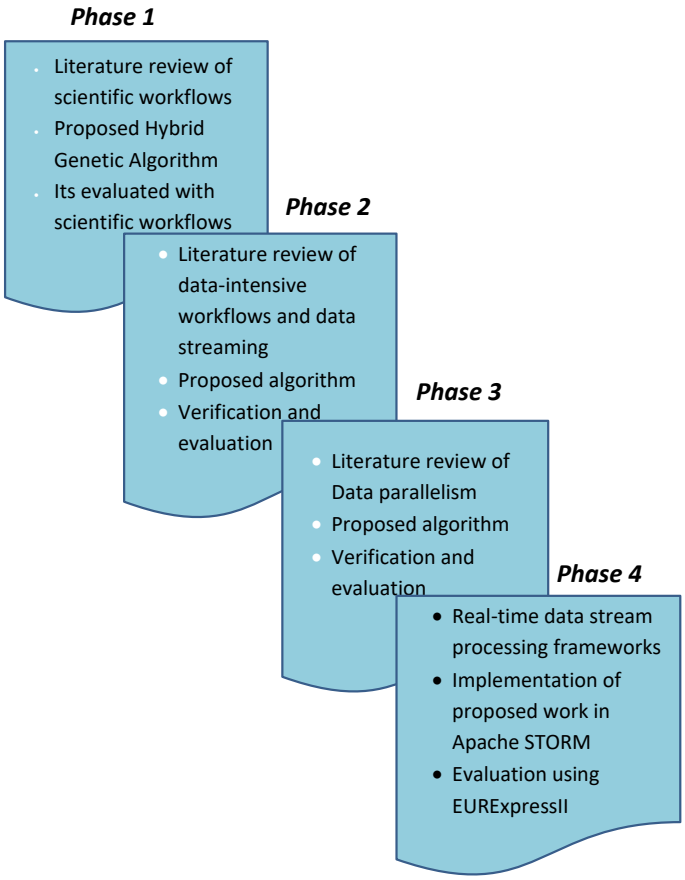


Figure 1.1: Research Phases

The literature of data-intensive workflows, its scheduling algorithms and stream-based data processing model is reviewed. A Partitioning based Data-intensive Workflow optimization Algorithm (PDWA) is proposed that reduces latency and enhances throughput of data-intensive workflows. PDWA is evaluated with the variety of workloads and performance results show its effectiveness. In the phase 3 of this research, data parallelism is introduced in the proposed algorithm to reduce the computation time of the data-intensive workflows. The algorithm is evaluated and simulation results prove its better performance as compared to other state of the art algorithms. Finally, the fourth phase is validation of the proposed work in real-time stream processing framework. Number of stream processing tools are studied and Apache STORM is selected to be the best candidate to carry out experiments. A use-case workflow (EURExpressII) is used as workload. The performance of the proposed work is evaluated in real-world environment with real-world

data-intensive workflow and results authenticated its better performance.

1.4 Contributions

The contributions of this thesis are summarized as follows.

1. A Hybrid Genetic Algorithm (HGA) is proposed that optimizes makespan of scientific workflows. It's architecture is a hybrid approach based on a heuristic and genetic algorithm. In addition to the makespan, HGA also provides load balanced schedules. Its performance results show that the proposed algorithm outperforms for synthesized as well as scientific workflows.

2. A Partitioning-based Data-intensive Workflow optimization Algorithm (PDWA) that optimize data-intensive workflows by partitioning the workflow in order to achieve minimum inter-partition data movement. Since, each partition is mapped on one resource therefore, intra-partition data movement cost is zero. PDWA reduces latency and enhances throughput by adapting stream-based data processing model to optimize data-intensive workflows. Its performance is significantly better as compared to the workflow optimization algorithm without partitioning.

3. Introduced data parallelism in the proposed data-intensive optimization algorithm. Data parallelism is applied on the most compute-intensive task of each partition. This strategy significantly reduced the computation overhead. The simulation results prove that the proposed algorithm not only reduced latency but also enhanced throughput. It outperforms the state of the art algorithms.

4. Implementation of proposed work in real-time data stream processing framework that is Apache STORM based cluster. The cluster is established in Openstack, Virtual Private Cloud (VPC) network to ensure the connectivity between master node and remote worker nodes. The experiments are carried out on this cluster with a

use-case workflow, EURExpressII which is a data-intensive workflow derived from real application. The experiment results show that proposed work outperforms the default STORM optimizer in terms of reduced execution time and better speedup as well as efficiency.

1.5 Thesis Organization

This section presents the organization of this dissertation which is divided into six chapters. Chapter 2 is the literature review, Scientific workflow scheduling using hybrid approach is presented in Chapter 3, Chapter 4 is about partitioning based algorithm to optimize data-intensive workflows for stream processing, it also describes how the algorithm performance is improved by introducing data parallelism. The algorithm presented in Chapter 4 is implemented in STORM, the experiments using STORM based cluster and the results are discussed in Chapter 5. Chapter 6 presents the conclusion and future research directions in this important research area.

CHAPTER 2

LITERATURE REVIEW

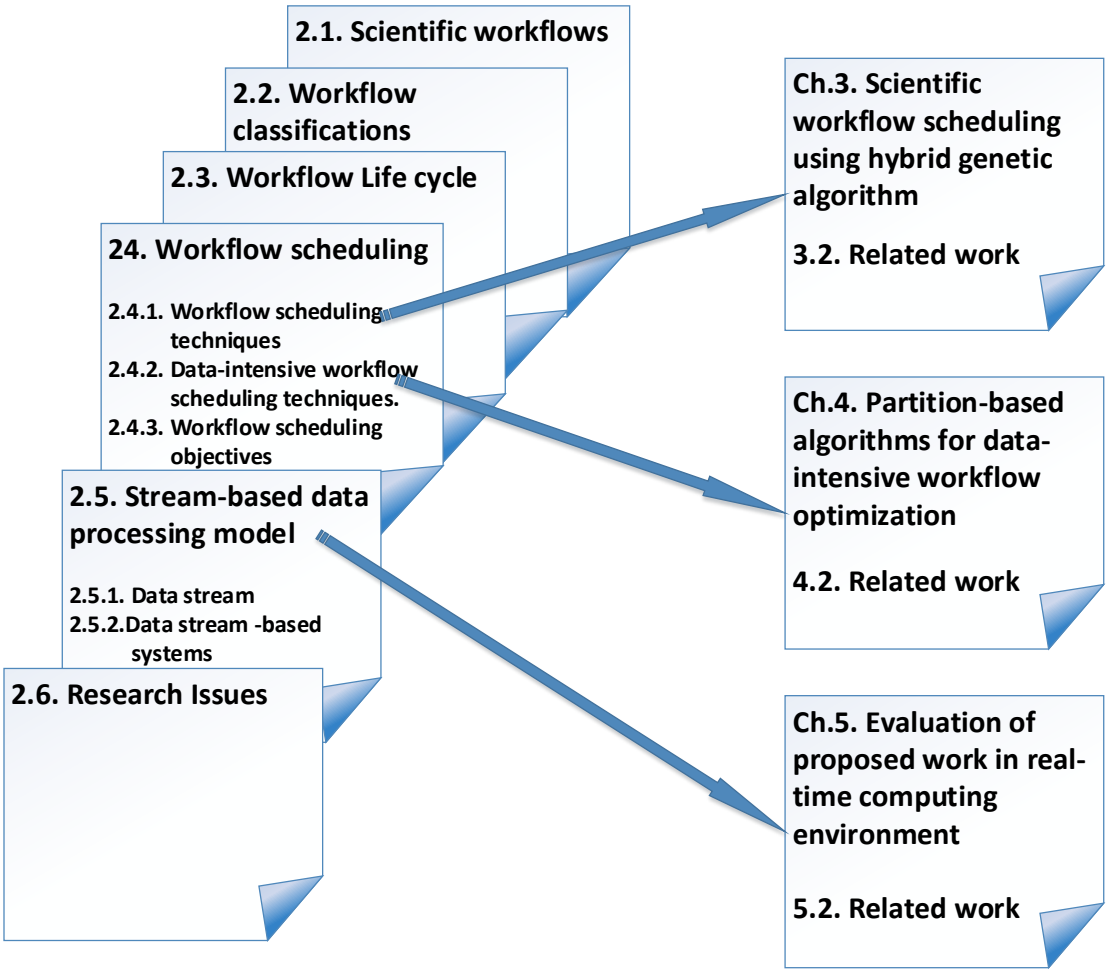


Figure 2.1: Literature Review: An Overview

Fig. 2.1 presents an overview of the literature that is reviewed in this research. In this chapter, section 1 introduces scientific workflows and its different patterns. In section 2 we present a taxonomy that classify the workflows into different categories. Section 3, briefly describes the life cycle of workflows. Workflow scheduling is presented in section 4 and a taxonomy showing different strands of workflow scheduling found in literature are cited and discussed. In addition, workflow scheduling, data-intensive scheduling and workflow scheduling objectives are also discussed in section 4. The concepts of Stream-based data processing model is presented in section 5. This chapter end-up with the research issues of data-intensive workflow scheduling in section 5.

We have covered the literature review in a general perspective in Chapter 2, but in this thesis specific related work is spread over all upcoming chapters. The detailed related study about workflow scheduling is presented in Chapter 3, Section 3.2. Similarly, the data-intensive workflow optimization algorithms are reviewed in Chapter 4, Section 4.2. In Chapter 5, Section 5.2 state of the art big data processing tools are discussed and STORM in detail, which is meant for processing data streams.

2.1 Scientific Workflows

Workflows have simplified the execution and processing of complex scientific applications by introducing a step-wise approach which defines the precedence of the tasks and flow of data. Workflows are sequences of tasks or operations, that can be represented graphically as well as in the form of script. Graphically, the tasks are usually represented as workflow nodes and in the script these are the processes or the jobs for example in the form of XML document. In literature, the most common graphical way of the workflow representation is Directed Acyclic Graphs (DAGs) (Couvares et al., 2007). The nodes in the DAGs represent tasks/activities/data or data items and vertices show the precedence constraint. Workflows can also be expressed as Petri Net (Hoheisel & Alt, 2007) in which

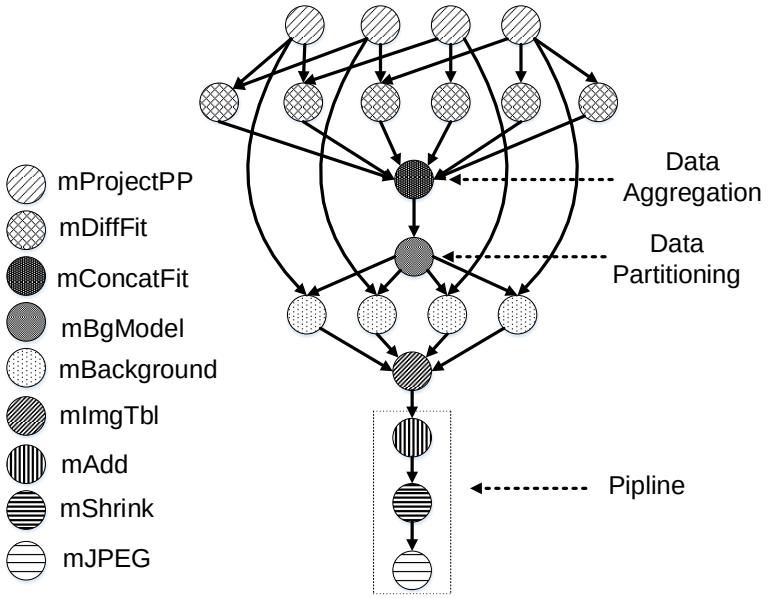


Figure 2.2: Scientific Workflow: Montage (Deelman et al., 2005)

data is represented as tokens and processes as transitions. Other ways of representation include scripting language and modeling language (UML) diagram (de Carvalho et al., 2010). In this research work workflows are modeled as DAGs, therefore in the rest of our discussion workflow and DAGs will be used interchangeably.

Montage is an example of scientific workflow, it is an astronomical image mosaic engine created by NASA that is used to generate a mosaic of the sky. The input astronomical images are combined to form the final mosaic. The geometry of the final mosaic is determined by the input images that can be represented as a workflow. Fig. 2.2 illustrates the structure of a small size montage having 20 nodes. The size of the workflow is proportional to the number of input images. Various examples of scientific workflows can be viewed in (I. J. Taylor et al., 2007). There are different patterns in the workflow structure which depends on the nature of application which is modeled as workflow. Some common workflow patterns that usually occur are discussed below (der Wijngaart & Frumkin, 2002). The NASA Advanced Super-computing (NAS) parallel benchmarks are designed for the performance evaluation of parallel and distributed systems as these

benchmarks are the workflow patterns that are most common in workflows. These are obtained from computational fluid dynamics (CFD) applications. They are in a set of four benchmarks that are Embarrassingly Distributed(ED), Helical Chain(HC), Visualization Pipeline(VP), and Mixed Bag(MB). Each of these consists of computational tasks achieved from NAS parallel Benchmarks (NPB). They symbolize the typical applications that run on the heterogeneous systems like grid, cloud, or any computing system.

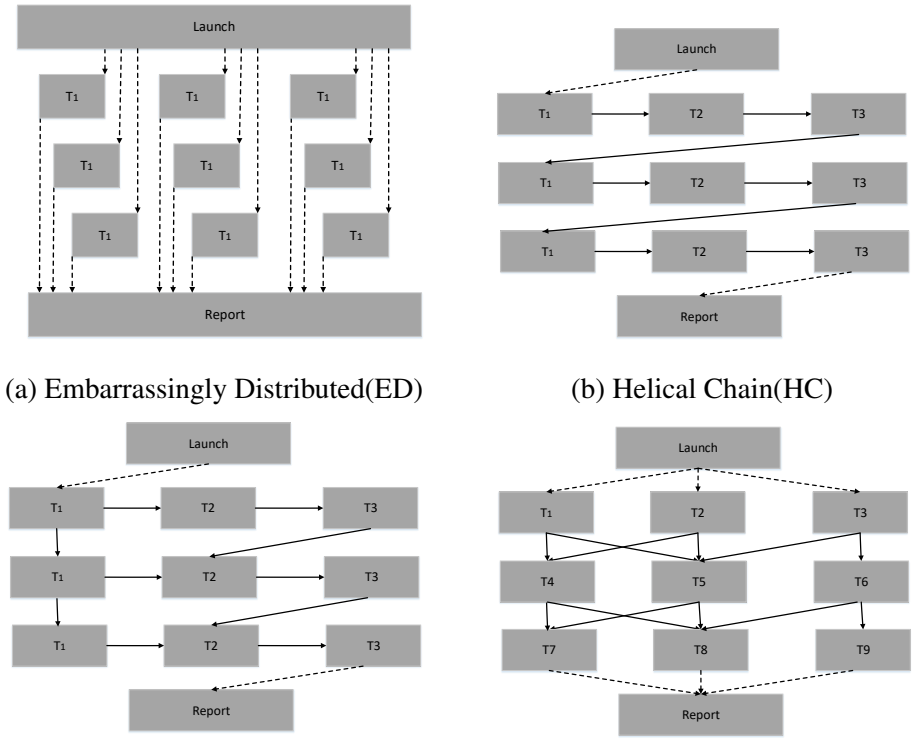
1. ED represents an important class of workflow applications called parameter studies, in which same program is run several times independently but with different set of input parameters. The structure is shown in Fig. 2.3a.

2. HC represents long chains of sequential computations such as simulations in series of tasks. It is the execution of repeating a process or set of processes/tasks one after another. It is presented in Fig. 2.3b.

3. VP is compound of different structured processes. It contains some degree of parallel as well as sequential flow of tasks. Its structure is presented in Fig. 2.3c.

4. The structure of MB is similar to VP but the degree of parallelism, and heterogeneity increases. There is aggregation and split nodes as well. The structure of MB is shown in Fig. 2.3d.

For all the figures of ED, HC, VP, and MB the dotted lines represents the control flow while the solid lines represents the control and data flow. Russell et al. (2005, 2006) provide vast and detailed studies of data flow and control flow patterns in a workflows. The complex and large workflows consist of usually such patterns that occur in different combinations. In Fig. 2.2, we can see different workflow patterns as labeled. Data aggregation is a common pattern in which data from all predecessor task are aggregated in a successor



(a) Embarrassingly Distributed(ED) (b) Helical Chain(HC)

(c) Visualization Pipeline(VP) (d) Mixed Bag(MB)

Figure 2.3: Different Workflow Patterns.

task, similarly data partitioning is splitting of data into successor tasks. Pipeline pattern is tasks in processing data in a sequence one after another also found in VP and MB.

2.2 Workflow Classifications

In this section, we present workflow classification based on their structure, types of workflow applications, data processing models and objectives. We discuss each type as follows

1. **Structure:** The structure of the workflows depend on the nature of its application. Based on the structure of workflows, the types can be *a*) sequential, *b*) parallel, and *c*) choice Several applications are expressed as sequence of tasks one after another forming a sequential(linear) workflow. In a sequential workflow there is a series of tasks that are executed in a sequential manner when the parent task is completed (Agrawal et al., 2010). In the parallel structure of workflows some tasks can be executed in parallel rather than sequentially. Most of the workflows have

parallelism that can cause parallel execution of tasks ultimately reduces workflow execution time. In case of choice, at run time a task is chosen to execute when its all requirements are complete.

2. **Application:** Workflows can be classified as control-driven and data driven workflows as reported in (Shields, 2007). In the control-driven workflows the precedence of tasks is based on the shift of control. The workflows are represented as sequence of processes and each parent process needs to be completed before the start of child process. The data driven-workflows support data-intensive applications and the dependencies between the activities represents the flow of data. I. Taylor et al. (2007) presents examples of data-driven workflows. In hybrid class, both approaches are used as appropriate but workflows are usually biased as data or control-driven for instance in (Laszewski et al., 2007). Sometimes the data flow is explicit and control flow is implicit or vise-versa.

3. **Data Processing:** Based on the input type workflows are categorized into two classes. The workflows can be driven by a single input and successor tasks depend solely on the output of their predecessors. The output generated from predecessors serve as input to the next level of task hence, successor tasks can't start execution unless predecessor are completed. Another type of workflows are those which take input in the from of continuous data stream (LIEW, 2012) such as the output of the sensors or simulators. Since, the output of each activity is a sequence of outputs therefore, if the successor gets enough input to start its execution then the successors can start execution.

4. **Objectives:** The fourth classification differentiates between task-based and service-based workflows as defined by Glatard et al. (2007). It is job oriented workflow in which set of jobs represented by nodes are mapped on the set of resources. Work-

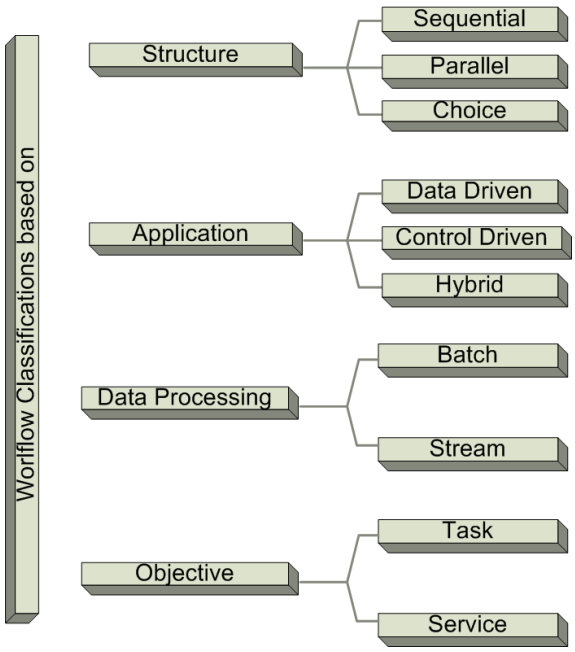


Figure 2.4: Workflow Classifications

flow is based on computing tasks which require complete dataset to start execution. On the other hand tasks are hidden under web services. Services are invoked on the execution of each node and execution can start without knowing the complete dataset.

The above discussion of workflow classification is summarized in Fig. 2.4.

2.3 Workflow Life Cycle

A workflow has a complete life cycle as presented in Fig. 2.5. There are three basic stages of workflow life cycle as discussed below.

- **Workflow Composition:** Initially, the workflows are developed from the data available in workflow libraries or databases. This data is usually the outcome of the previous executions of the workflows with different parameters or scenarios (provenance data). It can be the complex scientific experiments and simulation data saved in databases. The data and software components are obtained from databases. This information is used to develop a high level workflow called as abstract workflow us-

ing a particular representation. The workflow tasks are not binded to the resources at this stage, only the data and software components are identified. For instance, in Pegasus (Deelman et al., 2015) wings[1] is responsible for workflow composition. Wings helps the selection of workflow templates and data (from data repositories) to create workflow instances, which are also known as abstract workflows. Similarly, Taverna (Wolstencroft et al., 2013) provides an easy way for domain experts to find and design workflows through a workbench.

- **Mapping:** This stage maps workflow tasks on the physical resources from the execution environment. The abstract workflow converts into a complete execution plan resulting in a concrete workflow. Selection of appropriate resources and allocation to the workflow tasks can affect the performance of the workflow execution. Hence, optimized mapping strategies may help in improving the overall performance. For instance, Pegasus (Deelman et al., 2015) maps the workflow tasks on the execution resources to create the executable workflow, which has all the execution specifications, that is the data to be used and its location, the computing resources selected for execution, and the required data movements. The mapping process relies on three important catalogs, *a*) site catalog, which describes the compute resources known as the sites, that are used to run the workflow. A site can be cluster, virtual machine in cloud, or local machine. Pegasus works with heterogeneous and distributed computing environments, *b*) replica catalog, it is used for data discovery to resolve input/output files in the workflow. It keeps mapping logical files ids to physical file ids, and *c*) transformation catalog, which maps logical transformation to physical executable on the system. This thesis focus optimization of mapping and discussed with details in chapter 4.

[1] http://www.isi.edu/ikcap/wings/

- **Execution:** It performs the mapped workflow enactment in the execution environment and monitors the executions. This stage schedules the tasks on the mapped resources and collects the resulting data. The output and provenance data is then saved in the workflow repositories which may then accessed again to compose a workflow. Provenance mechanism that keeps the history of workflow data which is also useful for the resource mapping phase in determining the optimization approaches and parameters. In addition, provenance data is important for future relevant experimentation and analysis. Pegasus (Deelman et al., 2015) is a workflow planner and has no capability to execute workflow. However, it can run on various execution engines for example, Condor DAGMan[2] and Globus[3]. The input to the execution engine is a concrete workflow for instance in Condor DAGMan file is input for execution.

Mapping and execution are the optimization stages of workflow. Since, mapping and scheduling of workflow tasks are performed in these stages therefore, the performance of workflow execution depends on it. In this thesis, these stages of workflow life cycle is optimized that enhances the execution performance.

Workflow composition, mapping and execution for complex scientific applications is a tedious process, therefore scientific community have proposed Workflow Management Systems(WMSs) to automate the procedure. Some examples of WMSs are Pegasus (Deelman et al., 2015), Taverna (Wolstencroft et al., 2013), Kepler (Ludäscher et al., 2006), Swift (Y. Zhao et al., 2007), Trident (Barga et al., 2008), Meandre (Llora et al., 2008), and KNIME (Berthold et al., 2009). Each WFM has its own mechanism to compose, map and execute the workflows on resources, however the overall cause is to optimize the management and execution of the workflows.

[2]http://www.cs.wisc.edu/condor/dagman/

[3]http://toolkit.globus.org/toolkit/

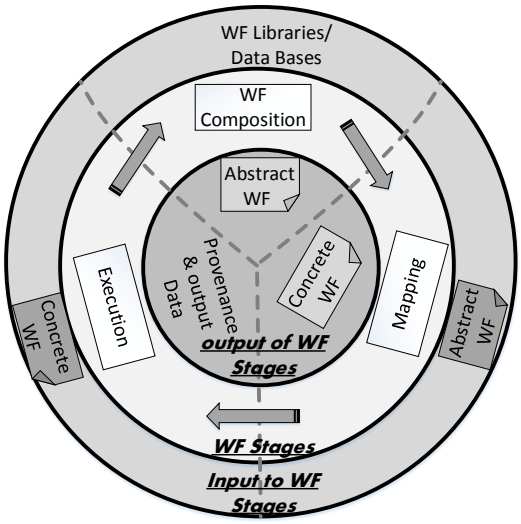


Figure 2.5: Workflow Life Cycle

2.4 Workflow Scheduling

As discuss in previous sections, the performance of workflow execution depends on it mapping and scheduling on appropriate resources therefore, the its optimization can enhance the system efficiency. In this section, we discuss various aspects that workflow scheduling involves. In grid, workflow scheduling is the allocation of workflow tasks to distributed available resources, while in case of cloud resources are in the form of services because a cloud can provide Iaas(Infra structure as a service), Paas(Platform as a service) or Saas(Software as a service). Efficient scheduling is crucial for the optimal workflow execution on every execution platform. Scheduling exploits the content of parallelism in workflows to enhance the performance but according to the Amdahl's law the advantage of parallelism is limited by the sequential part of the workflow. However, efficient scheduling plays a vital role in the workflows execution. Scheduling is affected by various types of parallelism. Task Parallelism is the simultaneous execution of different tasks on the same data item. It helps to reduce latency in workflows. Pipelined parallelism is the simultaneous execution of two different tasks of same workflow on same data item. It helps in enhancing the throughput. Replicated parallelism also helps in increasing the throughput as the different copies of same task executes on different data

items concurrently. This is useful when the resources are more in number than workflow tasks. Data parallelism corresponds to the parallelism present within the task inherently. Many processing elements execute the same task for same data item. Data parallelism is a mechanism in which multiple instances of task run simultaneously on different datasets.

1. **Objectives:** Most of the scheduling algorithms are single objective that optimize the run time and makespan like (Topcuoglu & Wu, 2002; Daoud & Kharma, 2008a). However there are numerous multi objective algorithms available in literature that optimize the parameters based on the requirements or application (Wieczorek et al., 2009).

2. **Scheduling Strategy:** Scheduling policies can be grouped into performance driven, market driven or trust driven. In performance driven strategies the scheduling aim is to improve the performance of workflow execution like execution time, throughput or cost algorithms like MinMin, MaxMin, Suffrage (Braun et al., 2001) focus to optimize the performance of the workflows. Market driven strategies focus the market oriented parameters like budget, deadline, reliability, profit or other QoS based aspects (Lee et al., 2012). Trust driven scheduling policies perform the resource allocation based on their trust levels. By using this type of scheduling strategies the scheduling reliability and security increases (S. Zhao & Lo, 2005).

3. **Execution Platform:** Based on the execution model, workflow scheduling can be differentiated by homogeneous and heterogeneous environments. In homogeneous execution platforms the resources have identical characteristics like in clusters (Wieczorek et al., 2005), while in heterogeneous environment, resources differ from each other based on execution time, storage capacity, communication time and I/O read time. Grid and cloud are example of heterogeneous environment (Khan, 2012). Scientists have experimented heterogeneous clusters and clusters developed

in grid or cloud environment which have characteristics of previous both categories (Issa et al., 2013).

4. **Scheduling Decision:** Scheduling decision can be local or global depends upon the scheduling policy. In local scheduling decision the scheduling strategy will apply for only one task or a part of workflow, which may effect the overall performance of the workflow. However, the global workflow scheduling algorithms apply for entire workflow scheduling. It is observed that global workflow scheduling policies provides better performance results as compared to local.

5. **Dynamism:** We can differentiate scheduling algorithms based on dynamism. When the scheduling process carried out at runtime then scheduling is called dynamic (Bansal et al., 2011). The resources and the jobs to be scheduled are not known prior to the scheduling phase. It is possible that jobs are accumulated to form a batch and then scheduled. This type of scheduling is also called as non deterministic scheduling. In case of static scheduling all the information regarding tasks and resources are already in hand prior to scheduling. Most important aspect of scheduling workflows is its validation.

6. **Performance Estimation:** We have classified the workflow scheduling based on performance estimations methods. Researcher used various methods to evaluate the performance of new scheduling algorithms in existing literature. Simulations based validation involve simulators like cloudsim, gridsim, simgrid on the other hand some researchers also used real-time data to analyze their work, which shows the behavior of algorithm in real environment. Simulation are useful to check the performance for large datasets. In case of analytical modeling the performance of workflow on given set of resources is predicted based on the analytic metric. In

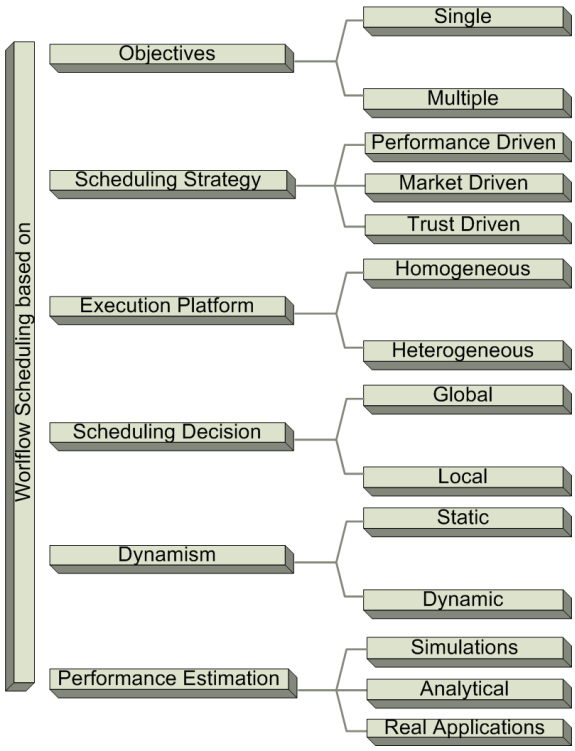


Figure 2.6: Taxonomy of Workflow Scheduling

some situations the hybrid approach is used. The workflow scheduling classifications discussed in this section are summarized in Figure 2.6.

2.4.1 Workflow Scheduling Techniques

In the literature there are number of methods which have been adapted for solving workflow scheduling problem. We have proposed a taxonomy of various scheduling policies found in literature. We have divided the scheduling strategies into three groups. *a*) heuristics, *b*) probabilistic search,and *c*) hybrid approaches.

- Scheduling is an NP-Hard problem and heuristics are mostly used to solve such problems. Heuristics are time effective solution for specific scenario of problem space. Heuristics are further of various types most common is list based scheduling heuristics, which generates a priority list of tasks based on some specific criteria and then tasks are allocated to resources (Topcuoglu & Wu, 2002). Resources allocation is also dependent on the attribute used to select resources for tasks. Some

workflow application are communication intensive, task duplication based heuristics use the task duplication technique to multiple resources to avoid the setup and communication cost (Agarwal & Kumar, 2009). Many clustering-based heuristics are also available in literature (Y. Zhang et al., 2008). In clustering heuristics tasks are clustered based on certain criteria to enhance the over all performance of workflow.

- Probabilistic search is another broad category of scheduling algorithms which include genetic algorithms, Simulated Annealing (Abdulal et al., 2012) and ant colony or swarm optimization. Working principle of these algorithms are from real world. Genetic Algorithms are well suited for large problem space and by starting from initial population after number of generations the algorithm converge to an optimal solution (Omara & Arafa, 2010). Lot of attention has been paid to ant colony (Srikanth et al., 2012) and swarm optimization (L. Zhang et al., 2006) to solve the scheduling problem. In both techniques the natural behavior of ants and swarm are adapted to solve the scheduling problem.

- In hybrid approaches the above mentioned strategies are combined to enhance the performance of algorithm like in (Guo-ning et al., 2010a) genetic algorithm is hybridized with simulated annealing while in (Daoud & Kharma, 2011) heuristic and genetic algorithm is hybridized. J. Liu et al. (2008); Srikanth et al. (2012) use the combination of ant colony and genetic algorithm.

The taxonomy of workflow scheduling strategies is presented in Fig. 2.7.

2.4.2 Data-intensive Workflow Scheduling Techniques

The key rationale behind data-intensive workflow management is the storage and movement of large volume of data. Data storage and movement incur huge cost. With the

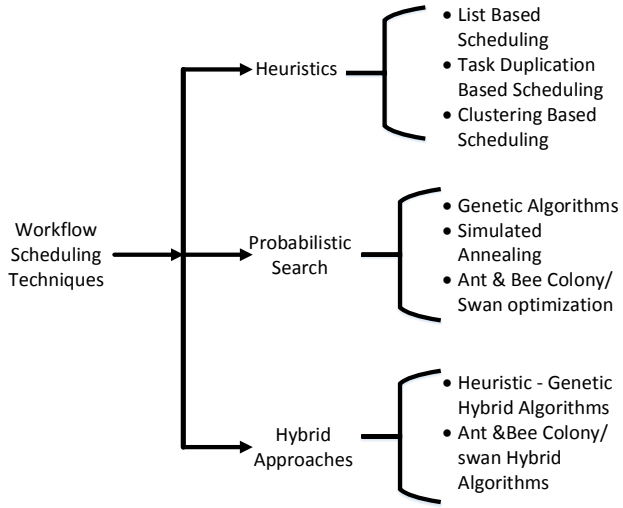


Figure 2.7: Taxonomy of Workflow Scheduling Techniques

emergence of big data, scientists have to focus in various domains to overcome the challenges. We can broadly group these research efforts in three major domains that include modifications in *a*) architectures, *b*) programming models and *c*) data-intensive workflow optimization strategies. In (Givelberg et al., 2011) authors proposed a concept of architecture for data-intensive computer based on data-intensive operating system that can manage data in Petabytes. The proposed architecture can be run with high performance computing clusters. The architecture exploits the parallel access (read/write) of massive databases through an intermediate layer of operating system that runs on different servers. Local data processing mechanism during workflow execution has been proposed in (Reimann et al., 2011), which improves the optimization of the data-intensive workflow execution.

MapReduce (Dean & Ghemawat, 2008) is widely used as a parallel and distributed programming model for data-intensive applications. Hadoop framework is most common implementation of MapReduce that has been used extensively by researchers to handle data-intensive applications in scientific systems. Researchers exploited Hadoop framework in many ways for example, G-Hadoop is a MapReduce based framework proposed in (L. Wang et al., 2013) that enable distributed data-intensive computing across multiple

clusters. Hadoop is integrated in Kepler (Ludäscher et al., 2006) an existing workflow management system as described in (J. Wang et al., 2009). This integration facilitate users to compose and execute MapReduce applications in Kelper.

This section describes the data-intensive management and scheduling techniques, also presented in Fig. 2.8. The data-intensive workflow scheduling techniques can be grouped into four major categories, *a*) data locality, *b*) data transfer, and *c*) data footprints. Each category is described as follows citing the previous work which has been done in that area.

1. **Data Locality:** In data-intensive workflow execution large amount of data is required to move between the execution nodes that causes significant delays depending on the size of data and network capacity. Therefore, most of the data-intensive workflow scheduling techniques target on the optimization of the data transfer by exploiting the data locality. This strategy can be further classified into *a*) spatial clustering, *b*) task clustering, and *c*) worker centric.

    - **Spatial clustering:** This scheme develops the workflow based on the spatial relationship of files in the input dataset. The clusters of jobs are created depending on the spatial proximity. The clustered jobs are then submitted to the execution nodes on the same site. Hence, the data-intensive workflow scheduling in this technique improves the data reuse and reduces the data transfer between the execution sites, that reduces the network traffic. It also enhances the execution performance.

      Meyer et al. (2006) presents an algorithm for spatial clustering that make use of data locality technique through dynamic replication of data and scheduling the workflow tasks in such a way that reduces the number of replicas and the data files transfers when workflow is executed. Quality of Service (QoS)

aware Workflow Language(QoWL) is developed in (Brandic et al., 2006), which is the extension of Business Process Execution Language(BPEL). It provides the facility to the users to customize (define) preferences of the execution location affinity for jobs/tasks with high security and legal constraints. QoS parameters restrict the system to move sensitive data from agreed domains for confidential applications.

- **Task clustering:** In this technique small tasks are grouped together in order to reduce the data movement between computing nodes. In the grouped tasks the data/files transfers between computing nodes are minimum. Hence, the tasks within one group can access data locally. This technique reduces the transfer of intermediate data files between tasks that are grouped to single computing node. In addition it also reduces the overhead of running small tasks individually.

  In (G. Singh et al., 2005), the authors restructures the workflow in order to reduce the data dependencies. Independent tasks of same level in workflow are clustered, but it is not necessary that tasks in none cluster will be executed on same computing node. The authors present workflow performance in two scenarios, *a*) tasks are submitted centrally by a master node, and *b*) tasks are submitted by multiple and distributed nodes. In the first case, the complete workflow is submitted and executed by single host. To increase the execution performance in the distributed job submission scheme, there are multiple job submission hosts and slave nodes. The workflow is restructured with multiple clusters at each level. The clusters and submission hosts at each level are always same in number. The scheduler on the host selects the suitable worker nodes for the execution of submitted jobs.

Pandey et al. (2009) apply the concept of task scheduling for a workflow based on medical application. The clustering is based on execution time of tasks, data transfer,and level. The most compute-intensive tasks are executed without clustering and the rest of the tasks are clustered, that enhances the makespan of the workflow execution.

- **Worker centric:** In this category of workflow scheduling technique, locality of interest present in the execution environment is exploited to schedule the workflow tasks. For instance, Ko et al. (2007) presents an algorithm, that allocates the workflow tasks to the workflow node upon receiving a request from worker node. The algorithm determines the weight of unscheduled tasks and submit to suitable node. The weight calculation involves the data files already present on the worker node and any additional data required to perform the task on that particular worker node. This algorithm reduces the transfers of data files and increases the chance of using same data which has been used in the past. The performance measured in terms of number of files transfers between the node and the results show better performance of proposed algorithm.

2. **Data Transfer:** In literature, several techniques are available that reduces the time of data transfer. These are *a*) data parallelism, *b*) data streaming, and *c*) data throttling.

    - **Data Parallelism:** It refers to a process of computation of data fragments simultaneously with minimum performance loss. This technique processes independent data items on different computing nodes by replicating the workflow task. In (Glatard et al., 2008), a workflow engine MOTEUR is presented. The proposed algorithms that combine well-defined data composition strate-

gies and fully parallel execution. Tasks and data is scheduled in such a way that most datasets are computed by independent resources without violating the precedence constraints. The performance of the proposed research work is carried out using a medical imaging application that run on the Enabling Grids for E-Science EU IST project [4] (EGEE) grid.

The technique of data parallelism and related work is discussed in detail in Chapter 4, section 4.4 and section 4.2.

- **Data Streaming:** In data streaming real-time real time streams of data generated by simulations and experiments are processed at high throughput, low latency and robust way. Bhat, Parashar, & Klasky (2007); Bhat, Parashar, Liu, et al. (2007) address data-streaming and design self-managing data-streaming service that enables efficient data transport in scientific workflow execution. The proposed system provides adaptive buffer management schemes as well as QoS management policies. The proposed work shows that online streaming can significantly effect the performance of workflow execution. The detailed literature review of scheduling algorithms for data-streams are presented in Chapter 4, section 4.2.

- **Data Throttling:** It is the process to control and select the data rate that can be delayed in the transfer from one compute node to the other. The data-intensive tasks have to wait for long time for the generation of data from predecessors and to transfer that large amount of data. Data throttling is a mechanism in which such data-intensive tasks are identified and further delays are caused purposefully by using slow network link to transfer data. The delay time can be used by computation nodes for the execution of tasks with high priority or

[4] http://www.eu-egee.org

urgency.

The limitation of existing systems are identified in (Park & Humphrey, 2008), that are firstly, no control on the data arrival time, and secondly, rate of data transfer between nodes. In this research work, a data throttling-based framework is proposed that controls that rate of data transfer between workflow tasks. The scheduler in the proposed system determines before hand that the data when/where will be executed and the data transfer rate also. It also specifies the data transfer delays in the system. The delays caused by this scheme helps to create balance between the execution time of workflow branches by reducing the extra usage of bandwidth that results efficient execution. Directed Acyclic Graph MANager (DANMan[5]) is a workflow execution engine used in WMS, Pegasus (Deelman et al., 2015). Pegasus uses DAGMan to execute the executable workflow. Data throttling phenomena is also used in this system.

3. **Data Footprints:** WMS use several methods to use data footprints. These methods can be classified as *a*) cleaning jobs, *b*) restructuring of workflow, and *c*) data placement and replication.

    - **Cleaning Jobs:** This mechanism is used in workflow execution to remove the data files from the resources which are no longer in use. The workflow tasks are scheduled to those nodes which have enough storage capacity to store input and output data files. In short, the scheduling is done based on the storage capability of compute resources.

      Two algorithms proposed in (Ramakrishnan et al., 2007) reduces the data footprints of the workflows. In the first algorithm, the data files are cleaned up

[5]http://www.cs.wisc.edu/condor/dagman/

that are no longer in use or being transferred (as data is replicated to multiple nodes) to permanent storage. In the second algorithm, an improvement in terms of reduced number of clean up jobs is done. It reduces the system to establish and replication of data to multiple nodes. Hence, it reduces the data footprints but as a consequence the workflow execution.

A storage constrained algorithm is proposed in (G. Singh et al., 2007), that that schedules data-intensive workflows on storage constrained resources. In the proposed algorithm, resource disk space availability is considered and then based on the resources computing power priorities are generated. The algorithm looks for the suitable resource that can accommodate the data files for the task to be scheduled on it. If no such resource is available then algorithm halts. The algorithm allocates that task to the resource which offers earliest finish time. Earliest finish time is the sum of data transfer time and execution time of the task. Eventually, the algorithm cleans up all the unwanted files from the resources.

- **Restructuring of Workflow:** The workflow structure determines the data footprints. Restructuring of the workflow is the change in the workflow structure that effects input/output data is placed, transferred, replicated, and cleaned up. Usually, task clustering and workflow partitioning are used to restructure the workflow in order to minimize the data transfer, increase in the data re-use, to balance the storage and compute resources, and so on.

  Pegasus (Deelman et al., 2015) has a characteristic to map and schedule the portions of workflow by partitioning it. Deelman et al. (2015) proposed a level-based partitioning algorithm. The level of the workflow is its depth, hence the tasks of each level are partitioned. In algorithm (Blythe et al., 2005),

Pegasus waits for the execution of preceding workflow partition and then map the dependent partition. The precedence constraint is maintained even after the partitioning. The partition level failure recovery is also carried out in this algorithm.

- **Data Placement and Replication:** In this category of data-intensive workflow scheduling technique the data is strategically placed before and during the execution. Data replication is common method to make data available on multiple resources. In data-intensive workflow scheduling data replication may or may not be useful based on the optimization objectives of the workflow.

  A scheduler Stork for data placement in grid is proposed in (Kosar & Livny, 2004). In Stork, data placement is considered as a separate job apart from data computational jobs. In *classads*, *i.e.* job description file, users clearly define the data placement jobs. Stork is used in DAGMan to manage data placement jobs. The precedence between computational jobs handled by Condor and Stork jobs are not violated as defined in workflow. Stork classify data placement jobs into three categories, *a*) transfer jobs, transfer data files between physical locations, *b*) allocation jobs, allocate bandwidth of network and data storage space at the destination resources, and *c*) release jobs, release the resources which were allocated before.

2.4.3 Workflow Scheduling Objectives

Workflow optimization involves various objective functions depending on the nature of workflow application and modeling. The data-streams produced by simulations and experiments are delivered at low latency and high throughput, therefore the processing system at the receiving end must be synchronized with the incoming data speed. Hence, in

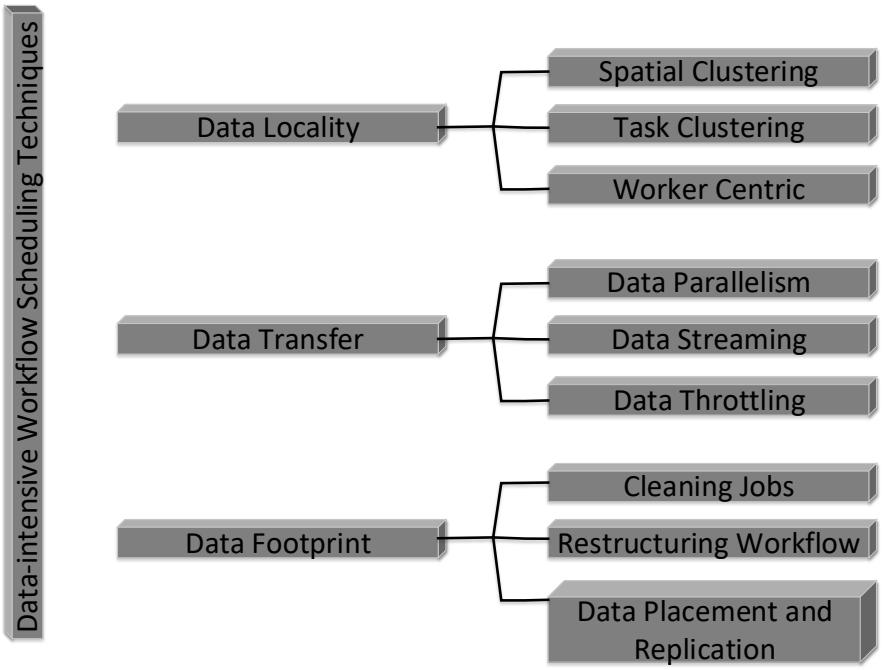


Figure 2.8: Taxonomy of Data-intensive Workflow Scheduling Techniques

the processing such data-streams in data-intensive workflows require minimum latency and maximum throughput for efficient system. Latency and throughput is defined as follows:

- Latency is defined as the maximum time a dataset spends in the system. Latency is directly related to the data size and can be reduced by deploying the processing element with reasonable computing power (Vydyanathan et al., 2011).

- Throughput is the number of data items processed per unit time. The reciprocal of throughput is period which is the time required to enter two consecutive datasets entering the system (Guirado et al., 2013).

The overall workflow execution time also termed as makespan is always optimized to the minimum (Munir et al., 2013). In data-intensive workflows massive data storage, its movement and data management between nodes are primary issues. Methods to achieve less expensive, low latency and energy efficient data storage is one of fundamental objectives in data-intensive workflows (BrykMaciej et al., 2016). In addition, transfer of

the massive data between processing elements involves huge data transfer costs, causes delay and increase in the network traffic (Prajapati & Shah, 2014). In case of processing (execution) node failure useful data may loss. Therefore, fault tolerance, redundancy and reliability are also crucial in data-intensive workflow scheduling (L. Zhao et al., 2010).

In service oriented environments like, cloud Quality of Service (QoS) is the prime importance factor. Each user is competing for earliest and cheapest response. Therefore, execution time (makespan) and cost are the main objective functions in cloud computing. Budget is a user defined parameter, it is the cost that a user can spend for the service used (Arabnejad & Barbosa, 2014a). In some cases, users want job completions at some pre-defined time called as deadline. Deadline-based scheduling complete workflow execution before user given deadlines (Luo et al., 2015). The deadline can be at the task level or at the workflow level.Optimal resource utilization reduces the cost of workflow execution. Objective function of load balancing is usually associated in this regard (K. Wang et al., 2016). Some objectives are from user perspective and some are from resources perspective. For instance, load balancing and optimized resource utilization are the workflow scheduling objectives from resource perspective.

In literature there are number of articles regarding scheduling multiple workflows in grid and cloud environment (Hirales-Carbajal et al., 2012; Tsai et al., 2012) but to the best of our knowledge no review paper is available for data-intensive workflow scheduling with stream-based data processing model and its performance perspective. This domain has still lot of research potential. Another relevant survey (Yu & Buyya, 2005) proposed a taxonomy of workflow management system characteristics. The author also discussed the scheduling aspect in workflow management systems in grid and a survey is presented in the paper which lists various characteristics of workflow management systems including the scheduling characteristics.

In data-intensive workflow execution large volumes of data movements among execution nodes incur high communication costs and causes network delays. With increasing amount of data sizes the network delays are increasing, therefore, time delays also increase to retrieve data from data storage to the execution nodes. To overcome this issue data should be executed or processed in place. In other words *bring the code to the data* contrary to the traditional approach to *bring the data to the code* (Kaisler et al., 2013). In addition to the data transportation issues, timely data processing is another major challenge of data-intensive workflows. Fast computations and efficient data executions require extensive parallel processing and multi-dimensional algorithms for data-intensive workflows. Data storage and management is a key challenge while dealing data-intensive workflows. Data storage can adversely effect the performance of data-intensive workflows if not managed in an effective manner. Avoiding data replication and deletion of unwanted data might help in reducing costs of acquiring extra storage space. Stream-based data processing model is an effective way to overcome this problem because data processing model produces an inherent parallelism that enhance the performance of data-intensive workflow execution moreover, large bulk of data is not required to be replicated at each execution node. However, buffering between each compute node of workflow is challenging because the size of output data is unpredictable and different for each application (Benoit et al., 2014). To determine the size of buffer is challenging and when data exceeds the buffer size, data must be stored in local disk that may cause delays in data retrieval and storage. These issues becomes more crucial when processing live stream of data. Mismatch in data processing and data arrival speed would cause loss of data. Data security, reliability, scalability, heterogeneity, data complexity and number of other domains need to be addressed for effective data-intensive workflow performance (Wen et al., 2015).

2.5 Stream-Based Data Processing Model (SDPM)

In connection oriented communication a data stream is a sequence of digitally encoded coherent signals (data packets) used to transmit and receive information that is in process of being transmitted.

At higher abstraction level, streaming data is the data that is generated continuously by thousands of data sources, which is send in data records simultaneously and in small sizes. Streaming data includes a wide verity of data such as a log file generated by customers using mobile or web applications, e commerce purchases, in-game player activity, information of social networks, financial trading floor or geo-spatial services and telemetry from connected devices or instruments in data centers. This data needs to be processed sequentially and incrementally on a record by record basis or over sliding time windows and used by wide verity of analytic including correlation, aggregation, filtering and sampling. Information derived from such analysis gives companies visibility into many aspects of their business and customer activity.

2.5.1 Data Stream

Data streams are continuous instances of data items that need to be processed instantly, if D is a data then data stream can be represented as $d_1, d_2, d_3, \therefore$, where d_1, d_2, d_3 and so on are continuous instances of items of data D, which are being generated by a data source. Moreover, the sequence and arrival speed of these data items cannot be controlled by the processing nodes. The definition of data stream is taken from Encyclopedia of data base systems (L. Liu & Ozsu., 2009) to be extracted from data bases. It is suitable for data generated for from modeling and simulations. A data item may be composed of relational tuple, raw data packets or pieces of text.

2.5.2 Data Stream-based System

The data stream is processed through sequence of procedures usually called as tasks. The continuous stream of data items are being processed (transferred) each task in workflow. The up-stream task (predecessor) performs certain operation on data and passes to down-stream task (successor). The direction of data flow cannot be changed. The transformation is done by each task independently and self-contained, although there are data dependencies between them. There must be a source of data stream called entry task and a sink called exit task that passes data out of the system. Further explanations of SDPM and workflow optimization algorithms for SDPM will be presented in Chapter 4.

Data stream processing has great capability to perform data-intensive computation. Large scale data impose challenges on the infrastructure to *transmit* entire data into, from the system and between tasks, *compute* large amount of data with limited resources and *store* the raw data, intermediate results as well as final results. These challenges can be resolved using stream data processing model, as it is easier to transfer, compute, and store smaller continuous instances of data. A comprehensive survey (Muthukrishnan, 2005) covers number of algorithms and applications used in various domains, *e.g.* network traffic monitoring, text mining, and real-time streaming applications on the web.

Stream processing provide the advantage of parallel execution where the tasks are independent of each other. With the advancement in multi-core architectures and distributed computing it has become possible to do parallel executions. Software systems, *e.g.* Imagine (Khailany et al., 2001), and Streamware (Gummaraju et al., 2008), are developed to support streaming applications. Further discussion on parallelism in data streams processing is presented in Chapter 4. Stream processing model reduces the overhead of input/output read write as well as processing element instantiation overhead. Therefore, we have adapted stream-based data processing model in this research.

There are different tools to process big data. Either data is processed in batches or in data streams. Apache STORM is an open source stream-based data processing real-time framework that is used to process unbounded streams of data. In Chapter 5, Section 5.2 various big data processing tools are reviewed and compared with STORM. Detailed description of STORM architecture and concepts are presented in Chapter 5, Section 5.3.

2.6 Research Issues

In data-intensive workflow optimization the main concern is the storage, transfer, and process large data. The existing infrastructures and algorithms are not smart enough to deal with these concerns as data increases. Scientific community is working hard to overcome the challenges caused by increasing data but due to the rapidly increasing data still these challenges are bottle neck .

Most of the research work focused task-based data processing model for scheduling and managing data-intensive workflows. The reason might be due to the amount of data considered is not too large. Stream-based data processing model has great capability to optimize data-intensive workflow execution. With the increasing amount of data generation and gathering, it is essential to focus the data models that can help in processing such large data in optimized manner.

In globally distributed resources, data transfer from one computing site to another cost extremely high. Stream-based data processing model also help in this regard. In addition to that, scheduling schemes must be proposed that can minimize the data transfer overheads. The growing data causing these research issues to challenges for the scientific community. These issues become more challenging when the distributed computing environment comprises with different capability resources. This research work addresses these issues by proposing data-intensive optimizing algorithms using stream-based data processing model.

In this chapter, we have discussed the scientific workflows and different workflow patterns. Based on the literature review workflows are classified and a taxonomy is developed. Each category is discussed citing related work. Workflow life cycle is discussed briefly. The workflow scheduling is presented with the literature review of its classification on different criterion. Various workflow scheduling techniques and data-intensive workflow scheduling techniques are reviewed. Workflow scheduling objectives found in literature are briefly discussed. This chapter also presents the concept of data stream processing model and eventually the research issues in data-intensive workflow scheduling.

CHAPTER 3

SCIENTIFIC WORKFLOW SCHEDULING USING HYBRID GENETIC ALGORITHM

This chapter presents preliminary research and investigation to the data-intensive workflow optimization. In this chapter we present an evolution based approach to solve the workflow scheduling problem. The proposed work is evaluated with the scientific workflows and resulting behaviors of workflows are observed and discussed. The architecture and performance of proposed hybrid approach is discussed followed by the discussion of the performance results achieved with different types of workflows.

3.1 Introduction

Workflow scheduling is a well-known NP hard problem (Ahmad et al., 2012), which is studied extensively by the scientists to enhance the workflow execution performance. There is a lot of diversity in the approaches to schedule workflows but evolution based approach gained lot of popularity in the last two decades. These approaches provide promising results for even large problem spaces. The performance of evolutionary algorithms can be further improved if these are supported by any high performance heuristic (L. Singh & Singh, 2013). The evolutionary approaches include genetic algorithms, ant-colony optimization, particle swarm optimization and bee colony optimization. However, genetic algorithms are comparatively more extensively focused in research communities. In addition, it can be hybridized with heuristics to enhance the performance of workflow scheduling. In this chapter we presents a Hybrid Genetic Algorithm (HGA) to solve the workflow scheduling problem. Mostly, the execution environment for workflow executions are homogeneous like clusters but in this research we adopted more realistic (practical) computing environments that is heterogeneous computing platforms like grid,

cloud, virtual private network (VPN) etc. The proposed algorithm provides optimized schedules of workflows with the significantly reduced makespan because of the load balancing. The performance of HGA is evaluated with different types of workflows and the comparative results show that HGA outperforms several state of the art approaches.

3.2 Related Work

In literature, numerous approaches are available to solve the workflow scheduling problem. Fig. 3.1 presents a taxonomy that highlights the major paradigms that are being explored by the scientific community. We can broadly group the workflow scheduling algorithms into *a*) Heuristics and *b*) Meta-heuristics. Heuristics are algorithms that provide a reasonable solution for specific conditions, however meta-heuristics guarantee an optimized solution. Meta-heuristics are the evolutionary algorithms including genetic algorithms. List based heuristics are most popular in literature for workflow scheduling. As presented in Fig.3.1 some renowned heuristics include Heterogeneous Earliest Finish Time (HEFT) (Topcuoglu et al., 2002), Critical Path On Processor (CPOP) (Topcuoglu et al., 2002), Performance Effective Task Scheduling (PETS) (Ilavarasan et al., 2005), Modified Critical Path (MCP) (Samriti et al., 2012) and Longest Dynamic Critical Path (LDCP) (Daoud & Kharma, 2008b). The algorithm of list based scheduling heuristics mostly consist of two phases. The first phase prioritizes the tasks based on weights, maintains a queue for ready tasks, and sorts the tasks in a priority queue. The second phase picks a task from the queue, submits it to the available resource and repeats until all tasks are assigned to the processors. Mostly, the resource is selected for task assignment based on some criteria such as Earliest Finish Time (EFT) in HEFT algorithm. The priority criteria of few list based algorithms are shown in the Table 3.1.

In clustering heuristics, tasks of a workflow are mapped on unlimited number of clusters. At each step, the tasks that are selected to cluster can be any task not nec-

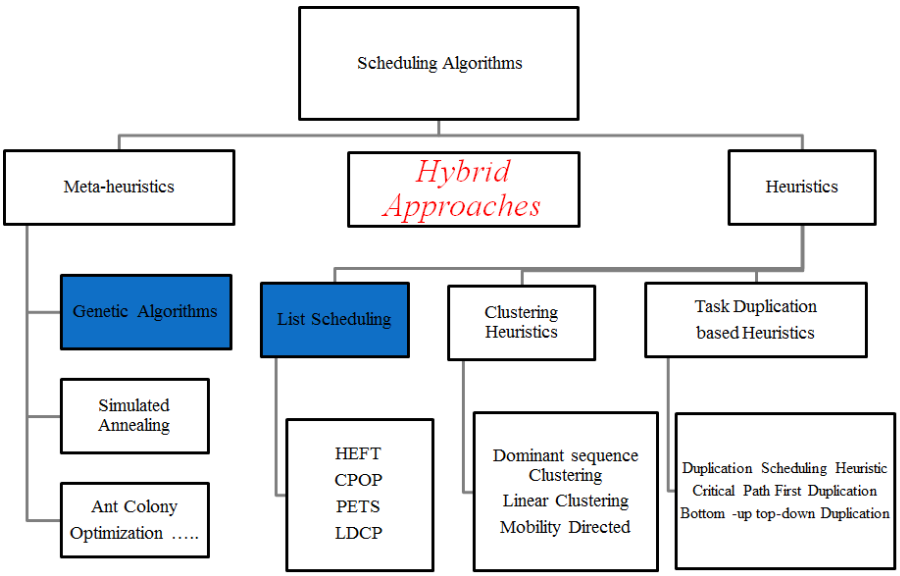


Figure 3.1: Taxonomy of the Workflow Scheduling Algorithms

Heuristics	**Task Prioritization Criteria**
HEFT	Up-word Rank (b level)
CPOP	Up-word Rank (b level) Down-word Rank (t-level)
PETs	Average computation cost (ACC)+ Data transmit cost (DTC) + Data receive cost (DRC)
MCP	As Late as possible (ALAP) = Critical Path Length – upward rank

Table 3.1: Different Priority Criteria of List Based Scheduling Heuristics

essarily the ready task. The iterative process in these category of heuristics refines the previous clustering by merging some clusters. A clustering heuristic requires following steps to generate the final schedule. *a*) Cluster merging step, so that the finally generated cluster match the available number of processors, *b*) Cluster mapping step, to map cluster on certain processor and *c*) A task prioritization queue within each cluster. Few examples of classic clustering heuristics are Domain Sequence Clustering (DSC) (Yang & Gerasoulis, 1994), Linear Clustering Method (Kim & Browne, 1988), and Mobility Directed (Wu & Gajski, 1990). In the category of task duplication heuristics some tasks are copied/duplicated redundantly to more than one processor to reduce the inter-process communication over head. These type of heuristics differ from each other by the selection strategy of tasks for duplication. The algorithms in this group are mostly designed for un-

bounded number of processors and the complexity of such heuristics are comparatively much higher than other groups of heuristics.

Another major group of algorithms are meta-heuristics that works well for specific scenarios and for larger problem spaces. Meta-heuristics are well suited for such problems and workflow scheduling is one of these research problems. In literature, we find extensive research work based on workflow scheduling using meta-heuristics. It includes variety of algorithms that are nature inspired, *e.g.* ant colony optimization (Srikanth et al., 2012), bee colony optimization (Taheria et al., 2013), particle swan optimization (Awad et al., 2015), simulated annealing (Guo-ning et al., 2010b) and genetic algorithms (Ahmad et al., 2012). However, genetic algorithms gained comparatively high popularity to solve task scheduling problem. The working of a genetic algorithm is an analogy of human evolution. The basic architecture of genetic algorithm is presented in Algorithm 1. Genetic algorithms start with a set of solution called initial population. Each solution is called as chromosome, which is evaluated based on a fitness function. In task scheduling problem, the fitness function is usually the schedule length (makespan) or it reciprocal. Hence, ultimately the fitness function determines which chromosome (solution) is comparatively better than other. As shown in Algorithm 1 the rest of the procedures of genetic algorithms are in a while loop of stopping criteria that shows that the loop continues until the stopping criteria is met. Mostly the stopping criteria is the number of generations or a reference fitness function. Within the loop the evaluated chromosomes are selected based on stochastic methods such as roulette wheel section method, binary selection method etc. for genetic operations. Genetic operators include crossover and mutation that are applied over selected chromosome to obtain better solutions and search the rest of the problem space. There is lot of diversity in crossover operation such as single point, double point crossover etc and similarly mutation can be single point, double point or random. The process continues until the solution converges to an optimal solution.

```
Initial Population Generation();
while stopping criteria not met do
    Selection();
    Crossover();
    Mutation();
    Evaluation();
end
Return best schedule();
```

Algorithm 1: Basic Genetic Algorithm

Below is a brief comparison of heuristics and genetic algorithms.

- Genetic algorithms start with a number of solutions in hand in the form of initial population while heuristics tend to navigate to a single solution.
- Genetic algorithms search the problem space depending upon the quality of genetic operators while heuristics limit the search space. Heuristics usually provide a feasible solution in some specific scenario of problem space.
- Genetic algorithms are well suited for problems with large search space while heuristics work well for relatively small scale search space problems.
- Genetic algorithms are recursive in nature, while heuristics are iterative.
- The time complexity of genetic algorithms is normally higher as compared to heuristics.
- Genetic algorithms converge to a best solution after certain generations while heuristic construct a schedule or solution using some predefined criteria.

In literature, there is another class of algorithm that is the combination of heuristics and genetic algorithms that can be grouped as hybrid algorithms. Various hybrid approaches can be found in literature however, our work focuses on a hybrid approach that uses both a heuristic and a genetic algorithm to try to reach an optimal (makespan) solution

for workflow scheduling. Various hybrid algorithms have been proposed by combining heuristics with genetic algorithms. A Hybrid Successor Concerned Heuristic-Genetic Scheduling (HSCGS) algorithm was presented in (C. Wang et al., 2012), in which the authors, combined a heuristic and a GA. In the first phase, the heuristic named Successor Concerned List Heuristic Scheduling (SCLS) was employed to generate a schedule. The SCLS is a list-based heuristic in which the priority list of tasks was formed based on the up-ward rank and successor of tasks. In the second phase the schedule generated by SCLS heuristic is incorporated in a GA. After number of generations, the algorithm converges and provides a schedule with reasonable schedule length. Another recently proposed algorithm, Performance Effective Genetic Algorithm (PEGA) (Ahmad et al., 2012) optimizes makespan using a hybrid approach. However, the time complexity of PEGA is high. The PEGA only optimizes makespan while the HGA reduces makespan as well as provides a load balanced schedule. In addition, a heuristic is also incorporated in the HGA that enhances the performance of the HGA by directed search.

A recently proposed GA, Multiple Priority Queues Genetic Algorithm (MPQGA) (Xu et al., 2014) exploits multiple queues of the tasks (priority lists) in a GA. The chromosomes are represented by the priority lists produced for the DAG by b-level, t-level, and sum of both parameters. The b-level ($rank_b(v_n)$), and t-level ($rank_t(v_n)$) values of each task are calculated by Eq. 3.2, and Eq. 3.1 respectively.

$$rank_t(v_n) = max_{v_{m \in pred(v_n)}} \{\overline{w_n} + \overline{d_{nm}} + rank_t(v_m)\}, \tag{3.1}$$

where $\overline{w_n}$ is the average execution cost, $\overline{d_{nm}}$ is the average communication cost between nodes n and m, and v_m is the predecessor of v_n. The t-level (downward rank) of each of the task is calculated by the above recursive function given in Eq. 3.1 and a task priority list is generated by an ascending order of the corresponding values of t-level.

The initial population consists of these priority queues based chromosomes and the mapping of tasks on the processors is performed based on earliest finish time parameter. Fitness for each chromosome is then computed using roulette wheel method and fit chromosomes are selected for genetic operations. Single point crossover and swap mutation operation comprises the genetic operations of the MPQGA. Our proposed algorithm differs from the MPQGA by the chromosome representation because in the HGA chromosome length is twice the number of tasks in a DAG and half of the chromosome consists of the random processor allocation to each of the task and the other half represents the priority list of tasks within the DAG. The HGA dominates the MPQGA from its two-fold crossover and mutation operations that are twice as efficient as compared to the traditional genetic operations. Load balancing is an additional strength of our proposed algorithm. Daoud and Kharma have proposed a two phase algorithm, named the Hybrid Heuristic Genetic Algorithm (H2GS) (Daoud & Kharma, 2011). In the first phase, schedules in the form of chromosomes are generated by employing the Longest Dynamic Critical Path (LDCP) heuristic. The LDCP generates schedules and thereafter such schedules are used in the initial population of a customized GA, called the Genetic Algorithm for Scheduling (GAS). The produced schedules work as catalyst and support GAS during the second phase to reach the resulting schedule. A two-dimensional chromosomal representation used in the GAS and the customized operators are used to search the problem space. In our proposed technique, we have used a comparatively less complex one-dimensional direct chromosome representation. The genetic operators (crossover and mutation) are modified to a two-fold operators that enhance the process of search to pursue an optimal (makespan) solution. Based on the aforementioned features, the HGA has a capability to arrive efficiently at the best solution.

In Salimi et al. (2014) an improvement of well-known multi-objective GA, NSGA-II is presented. In the market oriented grid environment, the price for the workflow execution is an important constraint. Therefore, in addition to makespan, price is also considered as the objective function in the proposed algorithm. A fuzzy system is implemented to optimize the third objective that is load balancing. The proposed algorithm improved these parameters but the complexity is increased considerably. However, the improvement in the makespan, and load balancing can be achieved by the HGA with much less time.

3.3 Proposed Hybrid Approach

The proposed algorithm is a hybrid genetic algorithm (HGA) that is a combination of a heuristic and genetic algorithm. The proposed algorithm has similar architecture to genetic algorithm however, a seed chromosome is added to the initial population to direct the search towards optimal solution. The pseudo code of the proposed algorithm is presented in Algorithm 2. The detailed explanation of the algorithm is as follows.

3.3.1 Initial Population and Chromosomal Representation

The algorithm HGA gets the population size N_p and number of generations N_g as input. N_g is taken as the stopping criteria. However, the initial population of size $N_p - 1$ is generated randomly at first step. The remaining one solution (chromosome) is added in the pool obtained from a renowned heuristics HEFT (Topcuoglu et al., 2002). The chromosomal representation used is direct and has two parts. The length of chromosome is double the number of tasks in a workflow. The left half represents the resource allocation and its length is equal to the number of nodes in the DAG. The genes represent the processor numbers from 1 to P, where P is the maximum number of available processors. The size of the second half is also the number of nodes within a DAG, which represents the sequence or order of the tasks to be scheduled. The precedence constraints must not

Input: N_p Population Size,
α Elitism Rate,
β Mutation Rate,
N_g Number of Generations.
Output: S Near Optimal Solution.
Initial population generation of size $N_p - 1$;
Seed HEFT schedule as a chromosome in N_p;
for $i = 1$ *to* N_g **do**
    /* Evaluation */
    for $j = 1$ *to* N_p **do**
        Compute fitness value of each chromosome;
    end
    /* Elitism */
    Number of Elite Chromosomes $E = \alpha \times N_p$;
    Select E Chromosomes having best fitness values as N_E;
    Selection Routine as shown in Algorithm 3;
    Crossover Routine as shown in Algorithm 4;
    Mutation Routine as shown in Algorithm 5;
    /* Next generation */
    /* N_s are the selected chromosomes */
    $N_p = N_E + N_s$;
    Load Balancing Routine as shown in Algorithm 6;
end
Return Near optimal (makespan) solution S;

Algorithm 2: Hybrid Genetic Algorithm

be violated; otherwise the chromosome will be an invalid chromosome and will not represent a correct schedule. Each chromosome represents a valid schedule. An example chromosome is shown in Fig. 3.2, and the corresponding schedule on three processors in the form of Gantt Chart is shown in Fig. 3.3. The first half is randomly generated, while the second half is determined by the up-ward rank of tasks using Eq. 3.2. The sequence of tasks in the example chromosome shown in Fig. 3.2 will be in the descending order of the b-level. The task priority list will be {1, 3, 4, 6, 2, 7, 5}. If the tasks are mapped on the processors according to the processor allocation shown in the first half of chromosome, then the resulting schedule length will be 51 time units. The corresponding schedule is shown in Fig. 3.3.

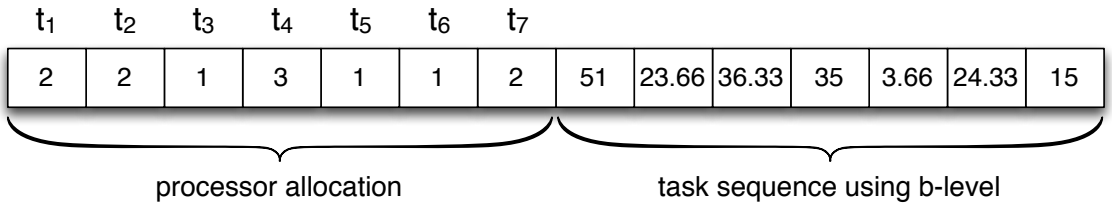


Figure 3.2: Chromosome representation.

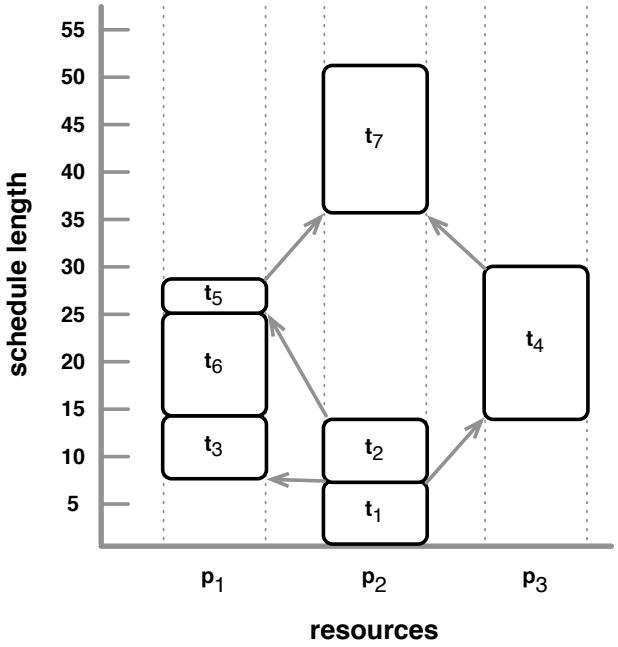


Figure 3.3: Corresponding schedule of example chromosome shown in Fig. 3.2.

$$rank_b(v_n) = \overline{w_n} + max_{v_{m \in succ(v_n)}} \{\overline{d_{nm}} + rank_b(v_m)\}, \tag{3.2}$$

where $\overline{w_n}$ is the average execution cost (it is the amount of time required to execute a task on execution node), $\overline{d_{nm}}$ is the average communication cost (it is the cost of data transfer from a parent to child node in a workflow) between nodes n and m, and v_m is the successor of v_n. The up-ward rank of each of the task is calculated by the above recursive function and a task priority list is generated by a descending order of the corresponding upward rank values.

3.3.2 Evaluation

Evaluation step is to determine the fitness of each chromosome. The fitness of chromosome any chromosome x depends on the fitness function given below.

$$f(x) = c/makespan(x), \tag{3.3}$$

where c is a constant and the makespan is defined as:

$$makespan(x) = F.T(t_{exit}), \tag{3.4}$$

with $F.T(t_{exit})$ being the finish time of exit node. In case there are more than one exit task, the makespan is defined by Eq. 3.5:

$$makespan(x) = max(F.T(t_i)), i = \{1,2,3,\ldots,n\}. \tag{3.5}$$

Hence, the fitness of a chromosome is higher which has short makespan. According to the fitness function defined in Eq. 3.3 the chromosomes with lower makespan are evaluated to be more fit.

3.3.3 Selection

In HGA, the fit chromosomes are selected by binary tournament selection method. In which couple of chromosomes are randomly picked up from the initial population and their fitness values are compared and the chromosome with higher fitness is selected and transferred to selected pool of chromosomes N_s. This processes continues for all initial population. The selected chromosomes are further used for genetic operations. The selection routine is presented in Algorithm 3.

Pick two chromosomes at random from initial population;
for $k = 1$ *to* N_g **do**
    if $f(x) < f(y)$ **then**
        Select chromosome x as N_s;
    end
end

Algorithm 3: Selection Routine

3.3.4 Genetic Operators

In genetic algorithms, genetic operators are key to get optimal solution. In Elitism the chromosomes with highest fitness values are copied to next generation. The parameter elitism rate (α) determines the number of chromosomes with highest fitness that will be copied to next generation. Elitism eliminates the chance of losing best solutions in the next generations.

Crossover is the core part of genetic algorithm which is an analogy of mating process. In the proposed algorithm two fold crossover is introduced that explores more problem space. Two fold crossover is single and double point crossover at a time with same parents and among the resulting four off-springs, two are copied to next generation that have high fitness values. The second half of the chromosome is not used in crossover as it represents the precedence of task execution, therefore only first half that is processor allocation part is involved in crossover and mutation. Fig. 3.4 illustrates the single point crossover, in which a crossover point is chosen randomly between $1 \ldots n$, where n is the number of tasks in a workflow. Two chromosomes cut off at the crossover point and exchange their parts. Similarly, in double point crossover two points are selected randomly between $1 \ldots n$. The part of chromosome between the two crossover points is exchanged between parent chromosomes as shown in Fig. 3.5 and two new off-springs are formed. The crossover process is applied to all selected chromosomes in N_s that can be in a couple (pair), it is shown in Algorithm 4.

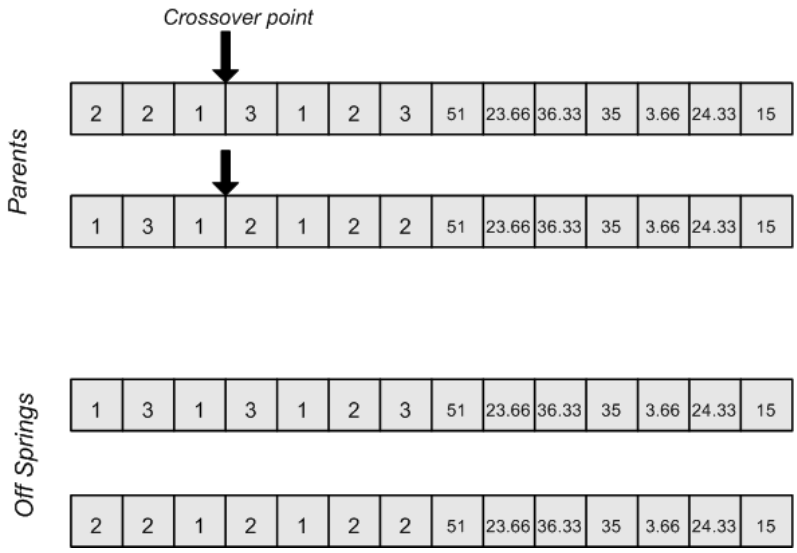


Figure 3.4: Single Point CrossOver.

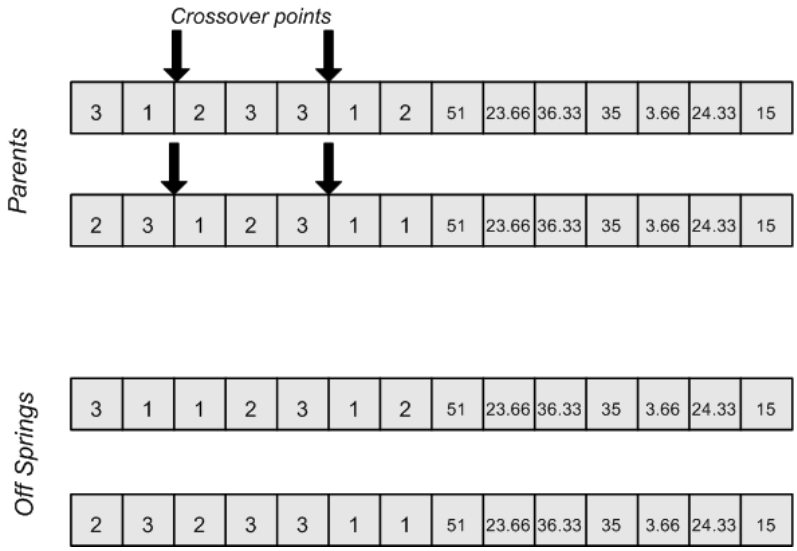


Figure 3.5: Double Point CrossOver.

Mutation genetic operation is usually carried out with comparatively less probability as compared to crossover. It explores the rest of the problem space that is spared by crossover. Mutation produces new chromosome (solution) that can provide better fitness and help to reach optimal solution. In the proposed algorithm, single and double point mutation is applied on the selected chromosomes with the probability of β that is an input for this algorithm. Similar to crossover single and double point mutation is applied to the parent chromosome and two different off-springs are generated. The mutation points

Data: Number of crossover $C = N_s/2$
for $k = 1$ *to* C **do**
    Randomly select two chromosomes x_a and x_b for mating;
    Off springs by single point crossover is x_c and x_d;
    Off springs by double point crossover is x_e and x_f;
    Compute fitness values of x_c, x_d, x_e and x_f;
    Select two best off springs as N_s;
end

Algorithm 4: Crossover Routine

are randomly selected between $1 \ldots n$ and the selected processor allocation gene is also changed at random. The phenomena of single and double point mutation is illustrated in Fig. 3.6 and Fig. 3.7 respectively. The fitness values of both off-springs are computed and compared and the chromosome with higher fitness is copied to next generation. The procedure of mutation is presented in Algorithm 5.

The chromosomes obtained from elitism, crossover,and mutation constitute next generation N_p. It must be considered that number of chromosomes within each generation always remain the same.

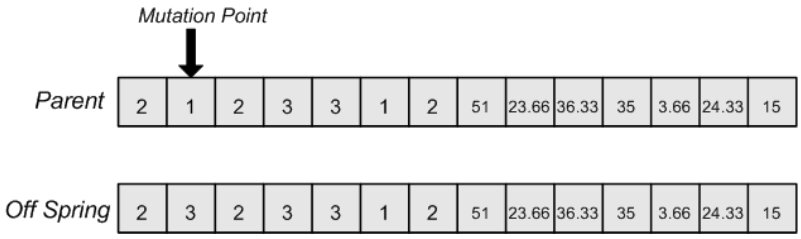


Figure 3.6: Single Point Mutation.

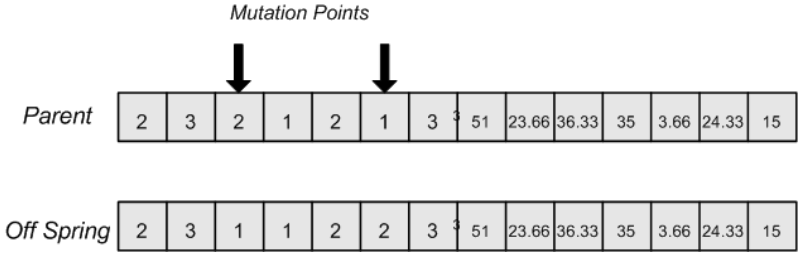


Figure 3.7: Double Point Mutation.

Data: Number of mutations $M = \beta x N_s$
for $l = 1$ *to* M **do**
    Randomly select a chromosome x_i from N_s;
    Perform single point mutation off spring = x_j;
    Perform double point mutation off spring = x_k;
    Compute fitness of x_j and x_k;
    Select offspring with better fitness value as N_s;
end

Algorithm 5: Mutation Routine

3.3.5 Load Balancing in HGA

The resources must be utilized in such a fashion so that the resources should neither be overloaded nor stay idle for a long time. The load balancing algorithm distributes the tasks among the processors in such a way that all of the processors complete the job with the minimum difference in the finish time. At the end of each iteration, load balancing is carried out that enhances the quality of the schedules in terms of schedule length by balancing the load across all of the processors.

The load balancing parameter (LB) is defined in Eq.3.6, that is used in load balancing algorithm to determine the quality of the load balance in each of the schedule:

$$LB = S_{Length} - min\{FT_1, FT_2, FT_3, \ldots, FT_n\}, \tag{3.6}$$

where S_{Length} being the schedule length of the corresponding chromosome and FT_n is the finish time of processor n. The greater the LB value the worse is the load balancing. The pseudo code of the routine is given in Algorithm 6. The complexity of the proposed algorithm is $nm + n^2$, where n is the number of generations and m is the population size. Increase in any of these two factors, the runtime of HGA will also increase.

Calculate *LB* parameter for all chromosomes using Eq. 3.6;
Select 50% of chromosome with higher *LB* values;
forall the *selected chromosomes* **do**
    Identify the overloaded processor p_l from chromosome y_{ol};
    Randomly select a processor p_i such that $p_i \neq p_l$;
    Replace the p_{ol} by p_n at two places;
    A new chromosome x_{lb} is formed;
    if $f(x_{lb}) > f(y_{ol})$ **then**
        Replace x_{lb} by y_{ol};
    end
end

Algorithm 6: Load Balancing Routine

3.4 Simulation Results and Discussion

In this section the datasets (workflows) used for evaluation HGA and simulation results are discussed.

3.4.1 Data Sets

Researchers have been working on the workflow patterns [1] and many benchmarks are available for the performance evaluation of new algorithms. As an example, the NAS Grid Benchmarks[2] (NGB) are designed for the performance evaluation of parallel and distributed systems. The benchmarks include four classes of problem obtained from computational fluid dynamics (CFD) applications: Embarrassingly Distributed (ED), Helical Chain (HC), Visualization Pipeline (VP), and Mixed Bag (MB) (der Wijngaart & Frumkin, 2002). These benchmarks represent the typical applications that run on the heterogeneous systems like grid. Each of these consists of computational tasks achieved from NAS parallel Benchmarks (NGB). They symbolize the typical applications that run on the heterogeneous systems like grid. ED represents an important class of applications called parameter studies, in which same program is run several times independently but with different set of input parameters. HC represents long chains of sequential compu-

[1] www.workflowpatterns.com

[2] www.nas.nasa.gov/publications/npb.html

tations. It is the execution of repeating a process one after another. VP is compound of different structured processes. It contains some degree of parallel as well as sequential flow of tasks. The structure of MB is similar to VP but the degree of parallelism, and heterogeneity increases. The selected datasets, *i.e.* Montage (Deelman et al., 2005), CyberShake (Deelman et al., 2015), and Gaussian Elimination are complex and large workflows that consist of most of the NGB classes of problem. These are real-world workflows that demonstrate realistic execution behavior on distributed environment. Together with the synthesized workflows that are generated randomly, we have a comprehensive set of test loads to analyze the performance of the proposed algorithm.

Table 3.2: Characteristics of datasets

Workflows	**Type**	**Source**	**Nature**	**Nodes**	**Shape**
Montage	Real	(Juve et al., 2013)	Regular	25,50,100	Fixed 3.8
CyberShake	Real	(Juve et al., 2013)	Regular	30,50,100	Fixed 3.9
Gaussian Elimination	Simulated	Generated	Regular	14,20, ...104,119	Fixed 3.13
Random	Synthesized	Generated	Random	20,40,60, 80,100	Varying

3.4.2 Montage and CyberShake Workflows

Montage is an astronomical image mosaic engine created by NASA that is used to generate a mosaic of the sky. The input astronomical images are combined to form the final mosaic. The geometry of the final mosaic is determined by the input images that can be represented as a workflow. Fig. 3.8 illustrates the structure of a small size montage having 20 nodes. In Montage, there are a lot of jobs with short execution time, such as mProjectPP, mDiffFit, mBackground, and mJpeg that are required to be executed on a number of different data items. On the other hand, some jobs such as mConcatFit, mBgModel, and mAdd take much longer time to execute. The CyberShake workflow is used by the Southern California Earthquake Center (SCEC). The CyberShake workflow

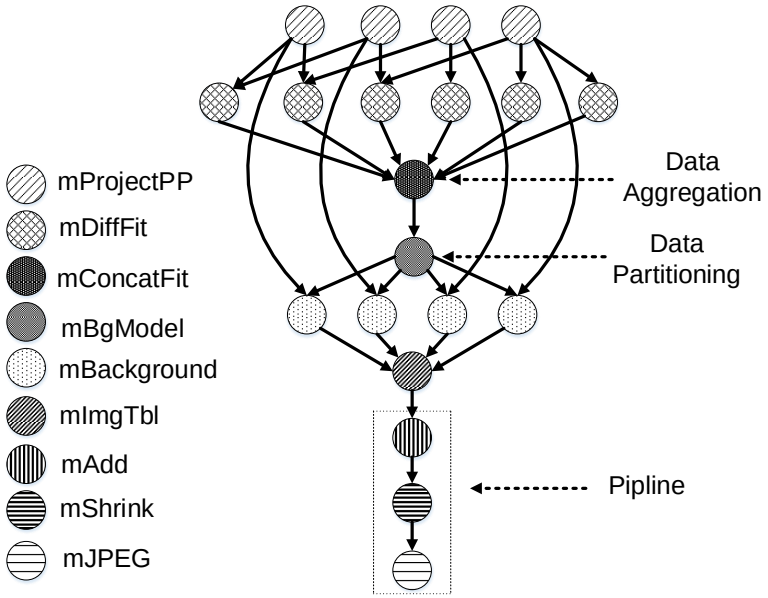


Figure 3.8: Workflow benchmark: Montage (Deelman et al., 2005)

is used to identify the earthquake hazards within a region. CyberShake is a relatively simple workflow but can handle large datasets. As an example, a small 20 node CyberShake workflow is shown in Fig. 3.9. CyberShake is compute-intensive as well as data-intensive workflow. The details regarding the characteristics of both workflows can be obtained from (Juve et al., 2013), in which the authors provided the details of the execution of six diverse workflows including Montage and CyberShake. Therefore, these workflows with different characteristics make them highly suitable to be used for the validation of our proposed work.

3.4.3 Performance Evaluation

In this section the performance of the proposed algorithm is analyzed. The HGA is evaluated by using datasets with diverse characteristics and the achieved results are compared with the following selected algorithms. We selected heuristics (MCP and HEFT), a generic evolutionary algorithm (PEGA), and recently proposed hybrid genetic algorithms (MPQGA and HSCGS) for comparative analysis with the proposed algorithm. These

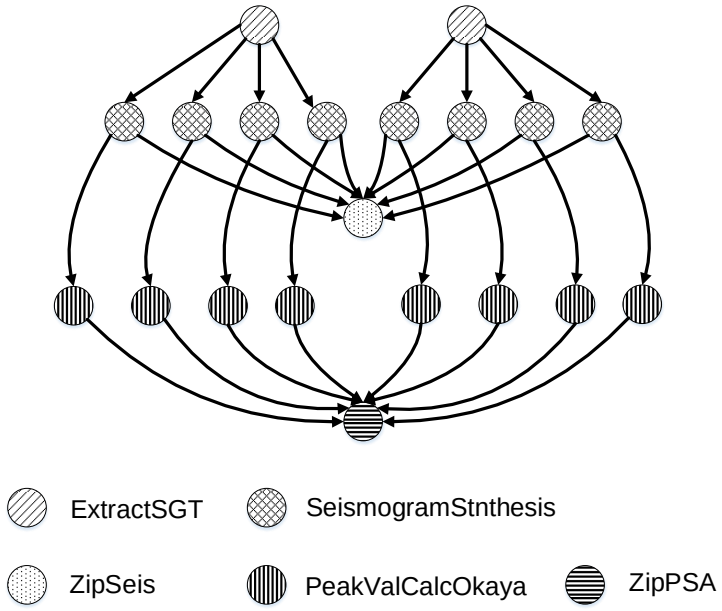


Figure 3.9: Workflow benchmark: CyberShake (Deelman et al., 2005)

selected algorithms based on different approaches provide sound grounds to study and compare the behavior of the HGA. We have selected Average Schedule Length (ASL) of 1000 runs as a performance metric. The range bars for ASL of all algorithms shows a 95% of the confidence interval for corresponding ASL. This shows that for any other workflow of similar type, the schedule length of that workflow would be in the given interval with 95% of certainty. In some of the bar charts the confidence interval is not distinguishable from the mean value for the scale used in those graphs.

The proposed algorithm is evaluated by simulations in a target system that is heterogeneous. The resources as well as the network links both are heterogeneous in the execution environment. Since the tasks are different based on the type of the workloads, therefore the heterogeneity of execution nodes and tasks both are considered in the heterogeneous execution times of each task on execution nodes. Similarly, the heterogeneity of network links and the edges are implicitly considered as the varying communication costs mentioned on the edges. After number of simulations, the most suitable parameters for the proposed algorithm that provides best results with the crossover and mutation proba-

bilities are 0.8 and 0.02 respectively. The population size and the number of generations both attributes are taken as 100 to simplify the simulations.

HEFT is a well-known heuristic that provides good schedules. We seed HEFT schedule in the initial population of HGA that accelerates the process to reach an optimal (makespan). We have made simulations with and without HEFT solution in the initial population of HGA. The results are given in Table 3.3. The results show that the HEFT solution in the initial population accelerates the runtime of the algorithm.

Table 3.3: Comparision of results with and without HEFT Seed in Initial Population

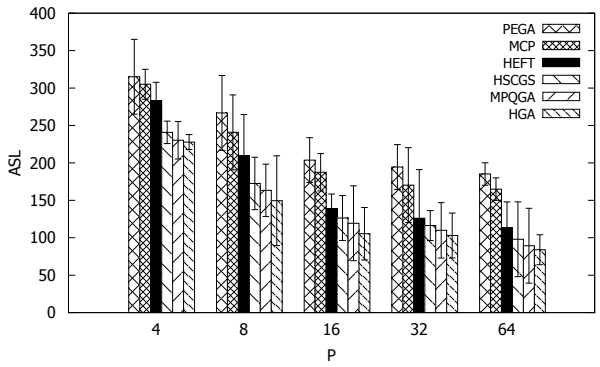
Workflow Nodes	**HGA with HEFT (sec)**	**HGA without HEFT (sec)**
100	6.55	7.94
500	30.66	39.99
1K	72.11	80.07
2K	143.52	159.45

We have performed extensive simulations using Montage and CyberShake workflows to evaluate the behavior of the HGA. We obtained the data of these benchmark workflows from (Deelman et al., 2015). The obtained data was the complete details of previous executions of Montage and CyberShake. All of the algorithms were tested under the same conditions to observe the comparative results. The performance metric used for the performance results is ASL for 1000 runs. In the plots the horizontal axis represents number of processors (P). The plots in Fig. 3.10 and Fig. 4.21 show the behavior of algorithms in terms of ASL when the number of processors increase. The HGA outperformed the state of the art algorithms, with its performance becoming better as the processors increase. The average percentage improvement of the HGA over PEGA, MCP, HEFT, HSCGS, and MPQGA was 73.97%, 59.55%, 29.85%, 12.58%, and 6.32%, respectively for 30 nodes CyberShake (Fig. 3.10a).

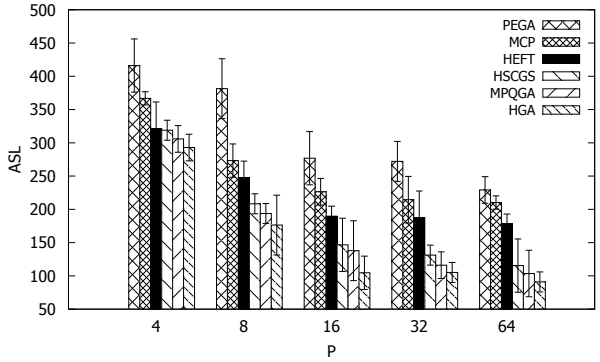
Similar results were obtained for CyberShake of 50, and 100 nodes with improved performance as the size of workflow increased. The proposed algorithm showed 19.53% and 25.05% improvement over the HSCGS, and 29.85% and 45.99% from the HEFT for 50 (Fig. 3.10b), and 100 (Fig. 3.10c) nodes CyberShake respectively. However, the HGA significantly outperforms the PEGA and MCP in case of the CyberShake workflows. The HGA outperformed the MPQGA by 11.3% for 50 nodes, and 13.86% for 100 nodes Cyberhake workflows. The better performance of the HGA with the increasing size in CyberShake workflows proves the scalability of the HGA.

Similar experiments were carried out with the Montage workflows of 25, 50, and 100 nodes. The results in Fig. 4.21 show a better performance of proposed algorithm, as compared to the other five algorithms. In case of the 25 nodes (Fig 3.11a) Montage workflow the HGA is better than the PEGA by 32.07%, MCP by 16%, HEFT by 6.7%, and HSCGS by 4.97%. However, minimum average percentage improvement was 5.4% and 8.56% over MPQGA for 50(Fig 3.11b) , and 100 (Fig 3.11c) nodes Montage workflows respectively. For the rest of the algorithms the average percentage improvement is comparatively high. It is noticeable that the performance of proposed algorithm is better for CyberShake workflows as compared to Montage workflows. Both types of workflows have different characteristics, CyberShake is data-intensive workflow as compared to Montage workflow.

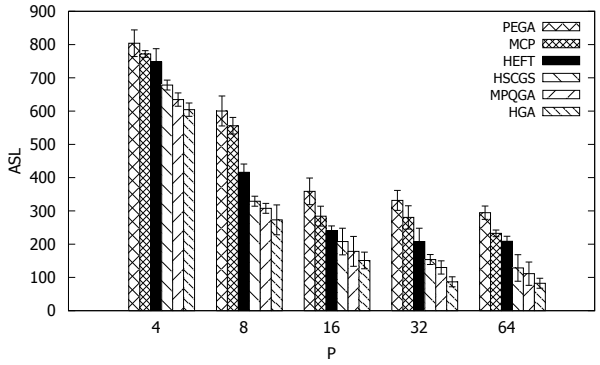
The bar charts of Fig. 3.12 show the overall performance of the CyberShake and Montage workflows of different sizes. The average schedule lengths obtained from proposed algorithm are less considerably than the other five algorithms. The HGA performance improved from MPQGA by 6.54%, HSCGS by 7.99%, HEFT by 10.73%, MCP by 17.7%, and PEGA 29.33% for Cybershake workflows (Fig. 3.12a). In case of the Montage workflows (Fig. 3.12b) average percentage improvement of HGA over MPQGA is 5.46%, and rest of the results are also similar to CyberShake workflow. The proposed



(a) 30 nodes.



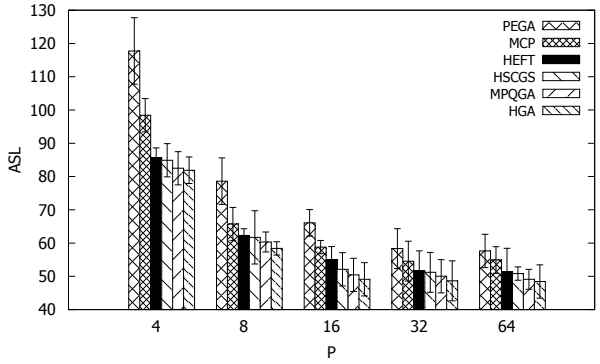
(b) 50 nodes.



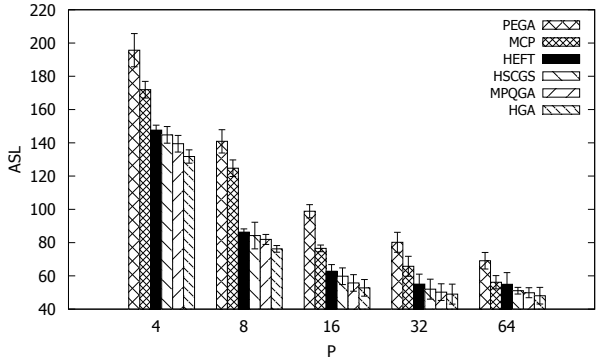
(c) 100 nodes.

Figure 3.10: Performance of CyberShake workflows.

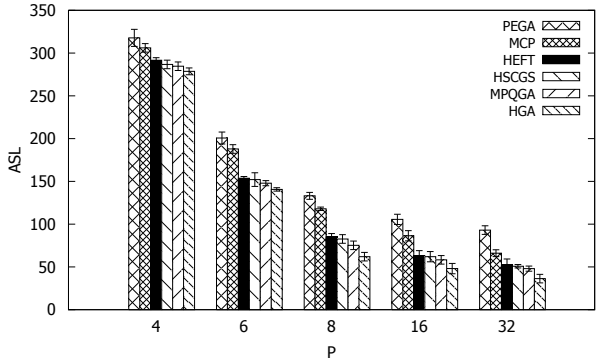
algorithm outperforms aforementioned algorithms, however the performance is significantly better than PEGA. In the proposed algorithm the heuristic accelerates the process to search an optimal (makespan) schedule and modified genetic operators help to search the problem space efficiently. These features dominate the HGA among other algorithms and it outperforms completely.



(a) 25 nodes.



(b) 50 nodes.



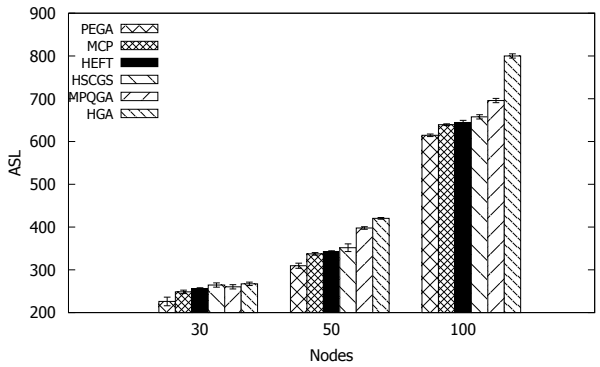
(c) 100 nodes.

Figure 3.11: Performance of Montage workflows.

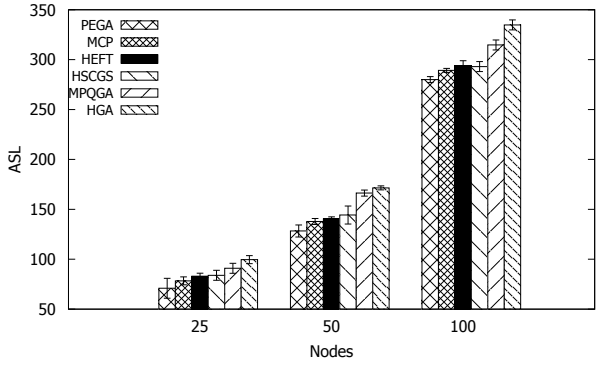
3.4.4 Gaussian Elimination

The Gaussian elimination algorithm generates a repeating pattern as shown in Fig. 3.13. The parameter m determines the size of the workflow. As an example, a small 20 nodes Gaussian elimination data flow for $m = 6$ is shown in Fig. 3.13 to illustrate the structure. Number of nodes for any matrix size can be calculated by using Eq. 3.7.

$$n = (m^2 + m - 2)/2, \tag{3.7}$$



(a) CyberShake



(b) Montage

Figure 3.12: Performance of CyberShake and Montage workflows of different sizes.

where n being the number of nodes within the graph.

The parameter m is also called as matrix size which determines the number of nodes in Gaussian Elimination workflow by using Eq. 3.7. Greater the value of parameter m higher the number of nodes in the workflow. We used different values of parameter m and generated workflows of various sizes. The values of parameter m used in our experimentation are from 5 to 15, which generated workflows of suitable sizes for our simulations. HGA and aforementioned algorithms were experimented with these workflows. Fig. 3.14a shows the plot of average schedule length with the increase in the workflow size determined by the matrix size (m) taken along x-axis. We can see that as the size of workflows become bigger the average schedule lengths increase because the execution time of bigger workflows will be higher. The plot in Fig. 3.14a shows a significant average improvement of 19.6% of HGA over PEGA, which is considerable improvement as compared to other algorithms. The HGA is better than MCP by 7.86%, HEFT by

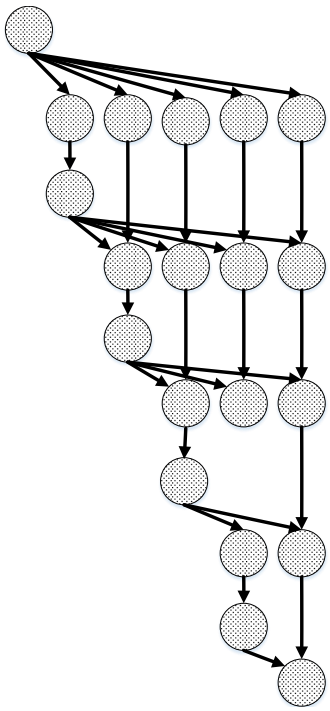


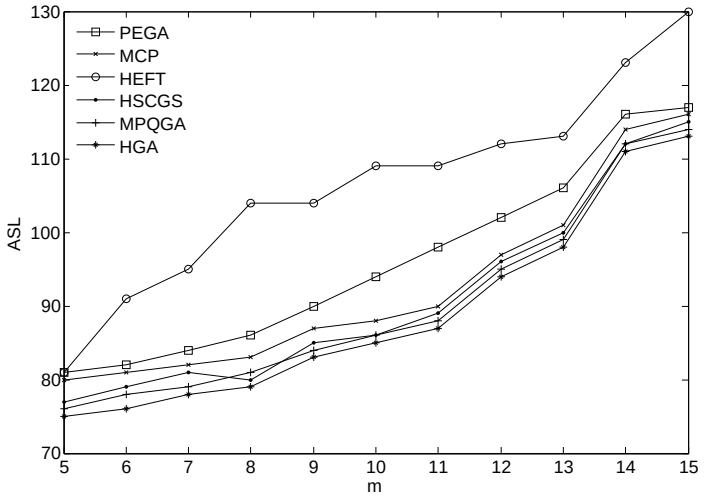
Figure 3.13: Gaussian elimination workflow for matrix size 6 (20 nodes).

4.07%, HSCGS by 2.14%, and MPQGA by 1.32%. The performance of the HGA with increase in the processors was also investigated and the results obtained are presented in bar chart 3.14b. The bar chart shows that as compared to MPQGA, and HSCGS, the proposed algorithm did not performed well for less number of processors but as processors increase the results of HGA became better. The HGA results are approximately 28% better than PEGA while about 7% better as compared to HEFT on the average. Proposed algorithm showed considerable improvement for Gaussian Elimination workflow.

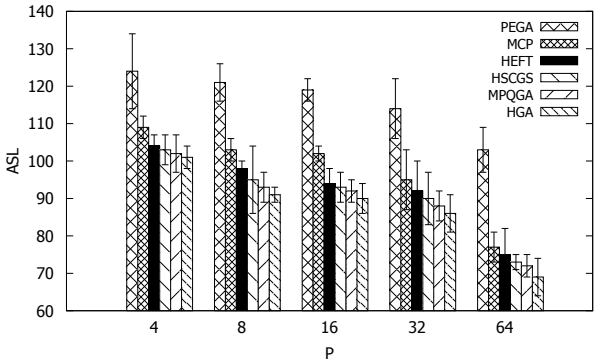
3.4.5 Synthesized Workflows

The synthesized workflows can be generated randomly (Topcuoglu et al., 2002) based on the following parameters and these are used for performance evaluation of the HGA.

- Workflow Size (n): The parameter n is the number of nodes in a workflow that represents the size of a workflow. Various sizes of random workflows using different values of n given in the following set were used in the performance evaluation of the HGA. $n = \{20, 40, 60, 80, 100\}$.



(a) increasing matrix size (m).



(b) increasing processors.

Figure 3.14: Performance results with Gaussian Elimination workflow.

- Shape Parameter (α): The shape of workflow can be handled by the parameter α. If $\alpha < 1$, then longer workflows with less parallelism are generated. If $\alpha > 1$, the small size workflows with higher parallelism are generated. If $\alpha = 1$, the workflows of balanced size are generated that are neither long nor short. The HGA is evaluated with three different sizes of workflows, with $\alpha = \{0.1, 1, 2\}$.

- Communication to Computation Ratio (CCR): The parameter CCR determines whether the workflow is computation-intensive or communication-intensive. If CCR > 1, then the workflow is communication-intensive. If CCR < 1, then the workflow is compute-intensive. If CCR = 1 then the workflow is neither communication nor computation-intensive. The values of CCR considered in experimental evaluation were 0.1, 1, and 10.

- The number of resources is represented by the parameter P, we use the number of resources as 2^x, where x = {2, 3, 4, 5, 6}.

- Out degree: This parameter determines the number of edges going out from a node. Because the workflows were random, the out degree was chosen at random.

- Range percentage of computation costs on processors (β): It determines the heterogeneity factor for processor speed. Its higher value causes significant difference in the computation costs of tasks and lower value indicates similar or equal computation costs of tasks. The values of β used in simulations are 0.1, 0.5, and 1.

Variety of random workflows with different characteristics were generated to analyze the performance of proposed algorithm. Fig. 3.15 presents the behavior of algorithms for random 100 nodes workflow when processors increase. We must note that as the resources increase the average schedule lengths decrease up to certain extent, which indicates that execution times can be reduced by exploiting parallelism in workflows but it is limited due to the serial content in workflows. The HGA showed 50.6% over PEGA, 44.57% over MCP, 22.76% over HEFT, 13% over HSCGS, and 6.78% over MPQGA average percentage improvement. Sets of workflows 5k, 10k, and 15k with different characteristics as mentioned above were generated and simulation results are achieved with CCR values of 0.1, 1, and 10. The results are shown in Fig. 3.16. The HGA showed significant improvement over MPQGA, HSCGS, HEFT, PEGA, and MCP for the results achieved for CCR values of 1 and 10. However, the performance of the proposed algorithm is not convincing for CCR value 0.1. This indicates that HGA performs well for communication-intensive workflows.

The load balancing feature in the proposed algorithm is also evaluated by comparing the results achieved from algorithm with and without load balancing. Results are presented in Fig. 3.17. The plot shows the strength of load balancing feature of *HGA* as

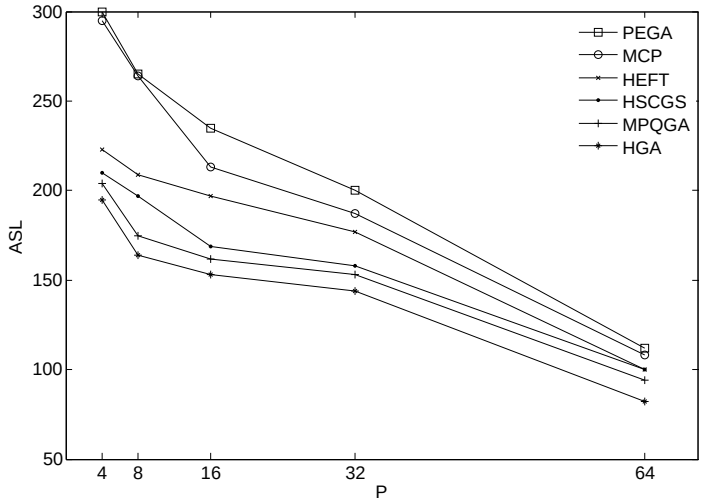
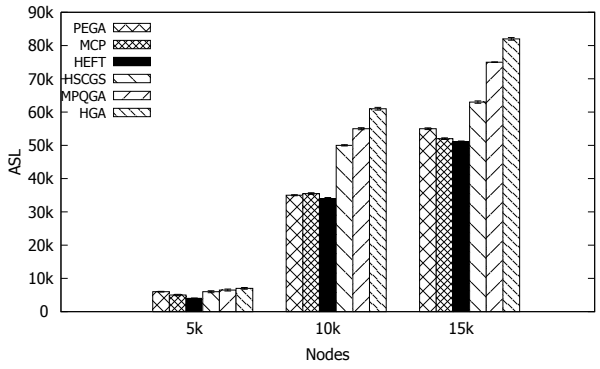
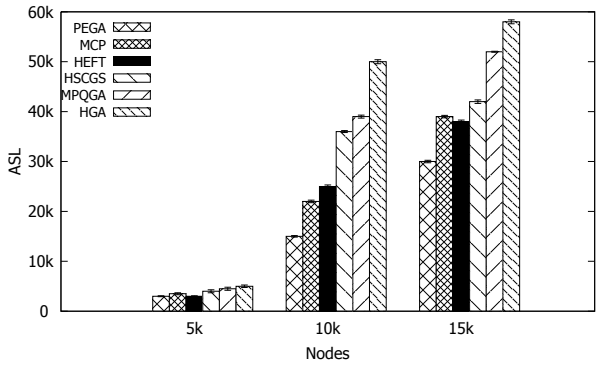


Figure 3.15: Performance results with random workflows with increasing number of processors.

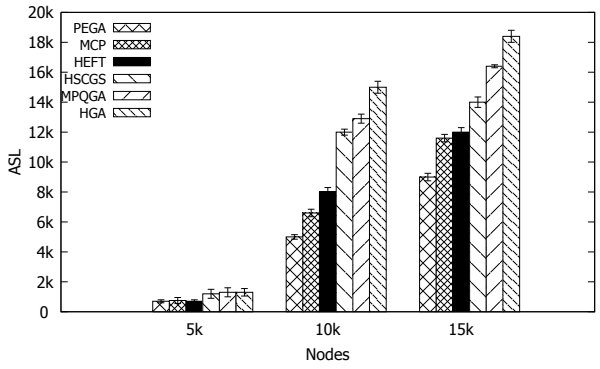
the average schedule lengths are better with load balancing. The *HGA* outperformed and considerable better results proved the supremacy of proposed algorithm over the other five algorithms. The time complexity of *HGA* is $O(nm+n^2)$,where n is number of generations and m is the population size. Overall the time complexity of heuristics is relatively low as compared to evolutionary algorithms for instance in the evaluation of HGA, $HEFT$ a famous heuristic is used to compare the results. Its time complexity for low density workflows is $O(vp)$ and $O(v^2p)$ for dense workflows, where v is the number of tasks in the workflow and p is the number of processor of execution environment. Which is less as compared to HGA. On the other hand, the time complexity of genetic algorithm *MPQGA* is $O(gener \times n^2 \times m)$, where *gener* are the number of generations, n is the number of task in a workflow and m is the number of processors in a system. Although it is in the order of two but product of these parameters shows that *MPQGA* is more time complex. The proposed algorithm has high complexity as compared to heuristics for instance $HEFT$ but low as compared to *MPQGA*. Since the proposed algorithm is designed for the static scheduling, therefore the limitation of high complexity of HGA might not affect the system performance.



(a) CCR 0.1



(b) CCR 1



(c) CCR 10

Figure 3.16: Performance of synthesized workflows.

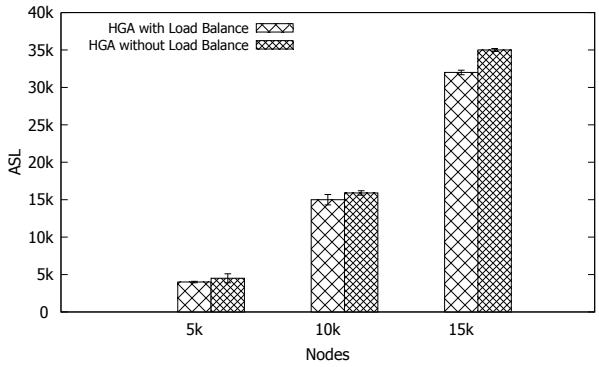


Figure 3.17: Performance results of HGA with and without load balancing with increasing number of nodes.

3.5 Conclusion

In this chapter, a genetic evaluation based approach is modified and a new hybrid genetic algorithm (HGA) for workflow scheduling is presented. The proposed algorithm seeds the Heterogeneous Earliest Finish Time (HEFT) based schedule in the initial population that guides the algorithm to reach an optimal (makespan) schedule in fewer generations. Rigorous search with two fold crossover and mutation operators cover the large problem space and enhances the HGA performance. The proposed algorithm optimizes workflow schedule length with an additional feature of load balancing that ensures the optimized resource utilization. The scheduling algorithms with different approaches are compared with the HGA. The simulations with variety and different sizes of datasets show the diversity and scalability of proposed algorithm. The results prove that the HGA outperforms and the quality of schedules is better by reduced schedule lengths. The simulation with different communication to computation ratio (CCR) shows that proposed algorithm performs well for workflows with CCR values greater than 1, that is communication-intensive workflows.

As discussed in chapter 2, the scientific communities are facing data deluge problem in conducting their experiments. The data-intensive aspect has forced them to model data-intensive applications in workflows. Stream-based data processing mechanism is an effective way to optimize the data-intensive workflows. In the next chapter, we focus optimization of data-intensive workflows using stream-based data processing model.

CHAPTER 4

PARTITION BASED ALGORITHMS FOR DATA-INTENSIVE WORKFLOW OPTIMIZATION

This chapter presents two proposed algorithms to optimize data-intensive workflows. First approach is a workflow partitioning based algorithm that reduces latency and enhances throughput of data stream based applications. The second algorithm is the extension of first approach, in which data parallelism is deployed to improve the latency and throughput. The simulation results prove that the second approach is not only an improvement of first algorithm but also outperforms many state of the art algorithms. The description of algorithms and discussion on simulation results are organized in the later sections.

4.1 Introduction

Scientists are struggling to manage the data-intensive applications, these efforts can be categorized in three different domains: *a*) hardware design or architectural improvements (Givelberg et al., 2011; Reimann et al., 2011), *b*) new storage and memory management techniques (Dean & Ghemawat, 2008; L. Wang et al., 2013), and *c*) algorithms to optimize data-intensive applications (Jung & Kettimuthu, 2013; Shibata et al., 2010). The science of workflows has emerged to simplify the complex scientific processes by step-wise representation in the form of workflows (Laszewski et al., 2007). The developments in the aforementioned areas optimize the execution of data-intensive applications in different ways. Traditionally, the data-intensive workflow optimization problem is addressed by using a number of techniques, but data stream computing is proven to be a well established concept in data-intensive workflow optimization in which data stream consists of continuous instances of data items (Han, Liew, et al., 2011). Stream computing is usu-

ally associated with real-time continuous data from sensors, audio/video systems, and dynamic social network. Moreover, this concept can also be used for processing archived data, *e.g.*, a single query to a database can invoke a stream of data items for processing, where a stream of data consists of different instances of data (Atkinson et al., 2012). The behavior of each task in such applications constitute the following three stages and repeats for each instance of the data: *a*) the task gets inputs of the data instances, *b*) processes the data, and *c*) passes the output processed data. The continuous stream of input data can be either from predecessors or an I/O read operation. The output processed data can be passed on to the successors or I/O write operation or store the data in the memory. The streaming model of data processing introduces an inherent parallelism within the workflow. At the execution level, if processed data items are sufficient to start the execution of a successor task, the successor task can start its execution. This reduces the waiting time of the successor tasks,which significantly reduces the execution time of the application. In stream-based applications, throughput and latency are two main metrics to measure the performance of an application execution.

In this research, we adopt the stream-based data processing model and propose a dual objective Partitioning based Data-intensive Workflow optimization Algorithm (PDWA) see section 4.3. The proposed algorithm optimizes data-intensive workflow by providing low latency schedules with reasonable throughput. PDWA partitions the application task graph in such a way that inter-partition data movement is minimum. Large amount of data movement among execution nodes incurs high overhead in the execution cost of data-intensive applications. The optimized partitions in PDWA ensure that inter-partition data movement is the lowest. Each partition is mapped to the execution node that provides minimum execution time, which reduces the latency. Furthermore, we leverage partial task duplication to further reduce the latency. We consider a heterogeneous computing system in which the execution nodes and communication links have different computing

and communication capacities, respectively. Most of the existing work (Guirado et al., 2013; Hackett et al., 2013; Vydyanathan et al., 2011) consider homogeneous execution environment for data-intensive optimization without incorporating the system heterogeneity. We have also proposed the Improved PDWA (I-PDWA), which shows better results in terms of reduced latency and improved throughput for details see section 4.4. We show that both proposed approaches provide considerably better schedule with lower latency and improved throughout. We validate the proposed algorithms using synthesized and real-world workloads (Arabnejad & Barbosa, 2014b),and show the performance advantages of the proposed algorithms.

4.2 Related Work

Number of approaches are found that optimize latency and throughput of streaming applications but most of them are designed for homogeneous computing environment. In Guirado et al. (2013) coarse-grained applications in which communication costs are negligible and algorithms are designed to reduce the latency of such applications. The authors proposed two different approaches to optimize both latency and throughput of streaming applications *a*) Data Parallel Replication Mechanism (DPRM), *b*) Task Copy Replication Mechanism (TCRM). In the former algorithm, data parallelism is exploited to enhance the throughput while in the later task replication technique is used to reduce latency. The task and data replications increases the complexity and the overhead of the algorithms. However, the algorithm proposed in this paper optimizes both (latency and throughput) performance indicators with relatively simple approach. In addition to this, the algorithm is designed for heterogeneous computing environment. Similarly, Vydyanathan et al. (2011) proposed algorithm that generates pipeline schedules for streaming applications but using two different approaches. The algorithm minimizes latency by satisfying throughput constraint and vise versa. The authors exploited pipeline, task, and data parallelism that

Table 4.1: Comparison of proposed algorithms with related algorithms.

Algorithm	Parallelism	Workflow Partitioning	Data Processing Model	Optimization Metric	Execution Environment	References
I-PDWA	Task,Data, Pipelined	Yes	SDPM	Latency, Throughput	Heterogeneous	Proposed Algorithm
PDWA	Task, Pipelined	Yes	SDPM	Latency, Throughput	Heterogeneous	Proposed Algorithm
AWOP	-	No	TDPM	Latency	Heterogeneous	(Munir et al., 2013)
PEFT	-	No	TDPM	Latency	Heterogeneous	(Arabnejad & Barbosa, 2014b)
TCRM	Task	Yes	SDPM	Latency	Homogeneous	(Guirado et al., 2013)
TCLO	Task,Data, Pipelined	Yes	SDPM	Latency, Throughput	Homogeneous	(Vydyanathan et al., 2011)

increases the algorithm complexity and the target system is homogeneous.

Same problem is addressed in numerous algorithms which are for specific execution platform like Cloud (Issa et al., 2013). In these algorithms, along with latency or throughput time and cost of computation are crucial to minimize. It is due to the cloud system that is based on "resources on demand" and "pay as you go" policy. Similarly many approaches based on different execution platforms (grid, cloud, clusters etc) are found in literature. A scheduling heuristic for stream based application for grid is proposed in (Agarwalla et al., 2007). In this paper authors proposed a dynamic scheduler named streamline, that is adaptive in nature and performs scheduling in three phases *a*) stage prioritization phase, *b*) resource filtering phase, and *c*) resource selection phase. In first phase, a priority list of tasks is generated and then resources that are capable of performing tasks are filtered and lastly resources are allocated for tasks. The scheduler is designed specifically for grid environment and implemented on Globus toolkit.

A recent research effort by Hackett et al. (2013) proposes an approach for stream based applications. Authors leverage network topology graph for optimizing throughput. Analogy has been used between communication links and electrical circuit. Based on this analogy the load on the communication links are considered as resistance in elec-

trical circuit, which is called Kirchhoff Index (KI). KI metric is used as a proxy for the optimization of throughput. The research entirely targets homogeneous computing environment. Several other research studies (Gu & Wu, 2010; Agrawal et al., 2009; Gu et al., 2012) are found in literature but they propose solutions either for homogeneous computing environment or for specific scenarios. The algorithm proposed in this paper differs from these approaches in two respects *a*) the algorithm is a balance between task and data parallelism that optimizes large variety of workflows, *b*) the computing environment is heterogeneous in addition to the TPGs. Table 4.1 presents a comparative overview of related algorithms that differentiate the proposed algorithm from other algorithms.

4.3 Partition Based Approach

The data-intensive workflows are characterized by the high communication costs due to large data movement among execution nodes. The workflows are modeled as Directed Acyclic Graph (DAG) (Xu et al., 2013). A DAG, $G(V,E)$, consists of a set of vertices, V, and edges, E. Each vertex represents a process (application task) that an input stream of data instances undergo, while the edges show the precedence of processes and the direction of the data flow. The execution environment, $H(U,L)$, consists of a set of compute nodes, U, and communication links, L_{ij} between nodes i and j. In heterogeneous computing systems, computation nodes have different computation capacity, E_s. We assume that all computation (execution) nodes are fully connected with each other through bi-directional high speed communication links. The cost model of a pipelined schedule for data-intensive workflows include throughput, TP, and latency, L. Both metrics are closely related to each other with a trade-off between them. Following section outlines an approach to estimate throughput and latency of the pipelined schedule.

4.3.1 Throughput Estimation of Pipelined Schedule

Since the execution environment consists of two components, *i.e.*, the execution nodes, and the communication links between these nodes, therefore, the system throughput is based on both these components. The two parts of throughput are termed as communication throughput, TP_{comm}, and computation throughput, TP_{comp}. We estimate TP_{comm} of L_{ij} between execution nodes, u_i and u_j, as:

$$TP_{comm}(L_{ij}) = Bw(L_{ij}) / \sum_{e=1}^{n} W(L_{ij}e) \tag{4.1}$$

where, $Bw(L_{ij})$ is the data transfer capacity of link L_{ij} between nodes u_i and u_j. $\sum_{e=1}^{n} W(L_{ij}e)$ is the data transfer load on the link L_{ij}, e represents the edges of application graph mapped to the links L. The communication throughput of communication links is given by:

$$TP_{comm} = min(TP_{comm}(L_{ij})) \tag{4.2}$$

Similarly, the computation throughput, TP_{comp}, of an execution node is defined as:

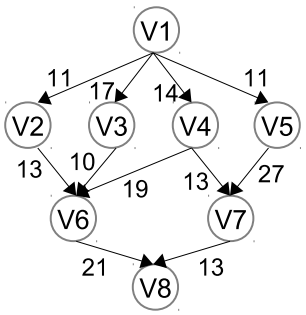
$$TP_{comp}(u_i) = E_s(u_i) / \sum_{i=1}^{n} W(u_i) \tag{4.3}$$

where, $E_s(u_i)$ is the execution speed of compute node u_i and $\sum_{i=1}^{n} W(u_i)$ is the computation load on the node u_i. The computation throughput of execution nodes is given by:

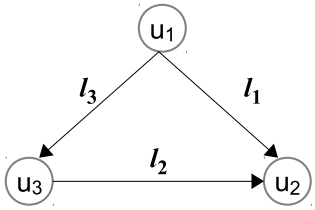
$$TP_{comp} = min(TP_{comp}(u_i)) \tag{4.4}$$

where, $i = \{1, 2, 3, \ldots, n\}$. The system throughput will be minimum between TP_{comm} and TP_{comp} and is given as:

$$TP = min(TP_{comm}, TP_{comp}) \tag{4.5}$$



(a) Example DAG with 8 application nodes $(v_1, v_2, \ldots, v_8)$ and 11 edges $(e_1, e_2, \ldots, e_{11})$.



(b) Execution environment of 3 execution nodes (u_1, u_2, u_3) connected with links (l_1, l_2, l_3).

Figure 4.1: An example DAG and execution environment.

4.3.2 Latency Estimation of Pipelined Schedule

Latency is defined as the time spent by an instance of the data in the system. Let D represents a stream of input data and data instances are $\{d_1, d_2, \ldots, d_n\}$. Latency is given by:

$$L = t_{d_n} - t_{d_{n-1}} \tag{4.6}$$

where, t_{d_n} and $t_{d_{n-1}}$ is the time when d_n and d_{n-1}, the consecutive instances of a data stream, complete their processing in the system, respectively. The difference between their completion times is the latency of the pipelined schedule.

4.3.3 Algorithm Architecture

The proposed algorithm consists of two phases. *a*) partitioning phase, and *b*) mapping phase. In the partitioning phase, the algorithm partitions the workflow based on pre-defined maximum possible number of tasks in single partition. This grouping of tasks in

Input: Given an application task graph $G(E,V)$.
Output: S of a Near Optimal Solution.
Read the graph: let E is the set of e edges and V is the set of v application tasks;
Partition(E,V) ; `// Algorithm 8`
Mapping(P Partitions) ; `// Algorithm 9`
Return optimal solution S

Algorithm 7: Partitioning based Data-intensive Workflow Optimization Algorithm (PDWA).

Input: E: Set of e edges; V: Set of v application tasks.
Output: P: Partitions of application task graph. Each partition contain m number of tasks.
Compute the threshold edge weight $e_{th} = \sum_{i=1}^{n} e_i/n$;
$N_{p_{max}} = v * f_p$;
Duplicate the entry tasks v_e;
while *pool of non-partitioned tasks is not empty* **do**
  let e_{ij} be the edge weight between task node v_i & v_j;
  if $(e_{ij} > e_{th})$ then place v_i & v_j in partition P_k;
  delete tasks v_i & v_j from the pool of tasks to be partitioned;
  if $(N_p = N_{p_{max}})$ next partition P_{k+1};
end
The application task graph splits into P_n partitions;
`/* Optimization of partitions */`
while *Inter-partition edge weights* $e_{ij} > e_{th}$ **do**
  Shift the task v_i or v_j to any other partition such that inter-partition edge weight always less than e_{th};
end
Return P_n partitions;

Algorithm 8: Workflow Partitioning Algorithm.

each partition is carried out in deterministic way. A threshold edge weight is determined, which is considered as the mean value of all edge weights. These initial partitions are then optimized in such a way that inter-partition data movement is minimum.

4.3.3.1 Illustration with an example

For a given data-intensive application, modeled as a DAG, $G(V,E)$, PDWA minimizes the latency with reasonable throughput for a stream-based data processing model. An example DAG is shown in Fig. 4.1a to illustrate the proposed methodology. The application tasks are represented as vertices $(v_1, v_2, \ldots, v_8)$, and edges $(e_1, e_2, \ldots, e_{11})$ show the dependencies between them. The edge weight shows the time required to transfer the

Input: P Partitions of application task graph and each partition contain almost equal m number of tasks.
Output: M mapped partition on the set of R resources.
for u_1 *to* u_n **do**
  Compute the CI of each partition using equation 4.8;
end
for P_1 *to* P_n **do**
  Determine the minimum CI value among the execution nodes *i.e.*, $CI_{min}(u_i)$;
  if (u_i is Idle) then Map partition P to $CI_{min}(u_i)$;
end
Return M mapped P partitions;

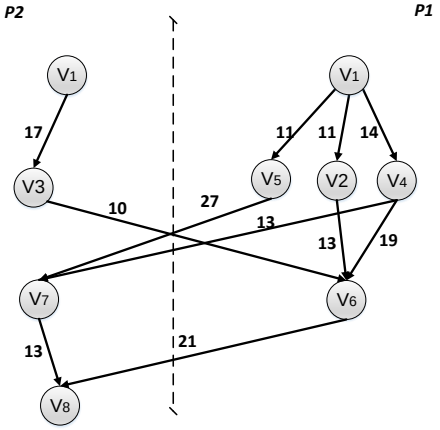
Algorithm 9: Partitions Mapping Algorithm.

data between the pair of execution nodes. PDWA is designed for heterogeneous computing environment where the computing capacity of execution nodes and the data transfer capability of the communication links between the nodes are heterogeneous. Consider an execution environment $H(U,L)$ as shown in Fig. 4.1b that consists of three execution nodes (u_1,u_2,u_3) that are fully connected through communication links (l_1,l_2,l_3). We assume that the baseline computing capacity is 1 GHz and each execution nodes have 7, 5 and, 3 cores, respectively, as shown in Table 4.2. The basic data transfer capability of communication links is 1 Gb/sec, and data transfer rate of each link is 10, 100, 1000 Gb/sec, respectively. $CT(v_i)$ represents the computation time of application task v_i. The computation times of all application tasks of the example DAG shown in Fig. 4.1a are given in Table 4.3. The pseudocode of the proposed algorithm is presented in Algorithm **??**. The DAG is partitioned using the Partition algorithm, shown in Algorithm 8. The DAG is split into suitable number of partitions such that the inter-partition data movement is minimum. We incorporate partial task duplication in PDWA. Partial task duplication only duplicates the entry tasks, which helps to reduce the latency of the schedule. The threshold edge weight, e_{th}, is the minimum allowed edge weight between partitions, which is 15 in the example. e_{th} is determined by the statement 1 of Algorithm 8. $N_{p_{max}}$ is the maximum possible number of application tasks in a partition, that is determined by

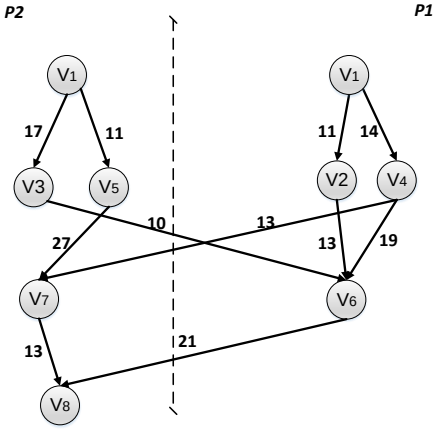
the product of vertices v and partitioning factor f_p. We assumed $f_p = 0.7$ that provides suitable partitions. As a result of Algorithm 8, the example DAG will be split into two parts. Fig. 4.2a shows the initial partitions in which two edges, e_{57} and e_{68}, with weights that exceeds e_{th}. Lines 10 to 12 of Algorithm 8 perform iterations until all inter-partitions edges have weights less than e_{th}. Fig. 4.2 illustrates that after two iterations, the DAG is partitioned such that there is no edge that violates the condition. In the first iteration, the application node 5 is shifted from partition P_1 to P_2, and during the second iteration v_8 is moved to P_1. Finally, the required condition is satisfied and the DAG is divided into two optimized partitions where, P_1 consists of application tasks {1, 2, 4, 6, 8}, and partition, P_2, consists of application tasks {1, 3, 5, 7}. The optimized partitioned DAG is mapped to the execution nodes by using the method shown in Algorithm 9. We define computation index, $CI_{u_j}(P_n)$, of execution node, u_j, for partition, P_n, a criteria to select execution node for partition P_n.

$$CI_{u_j}(P_n) = \sum_{v_i \in P_n} CT(v_i) \tag{4.7}$$

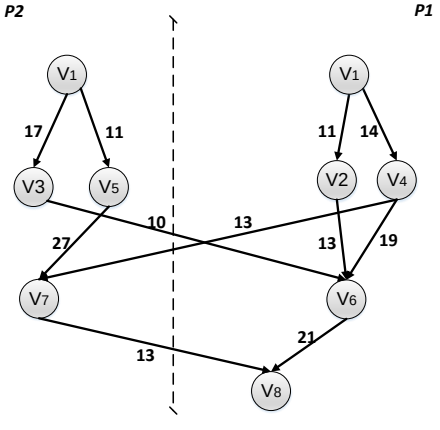
where, $v_i, \{i = 1, 2, \ldots, m\}$, if there are m vertices in a DAG, are the application tasks that are grouped to the partition P_n. Each partition is mapped to the execution node which gives minimum CI. In the example DAG, the $CI(P_1)$ is 55, 48, and 45, for the execution nodes u_1, u_2, and u_3, respectively. Similarly, the $CI(P_2)$ is 45 for u_1, 50 for u_2, and 55 for u_3. The minimum CI of partition P_1 is for execution node u_3, so P_1 is mapped on u_3. Similarly, P_2 is mapped to u_1. The pipelined schedule with latency 76 is obtained by PDWA, which is shown in Fig. 4.3a. The schedule of the algorithm without partitions (AWOP) is shown in Fig. 4.3b, and its latency is 88. The proposed algorithm outperforms with significant improvement in the schedule latency. It must be noted that PDWA performs better while utilizing fewer number of execution nodes.



(a) Initial partitions.



(b) Partitions after first iteration.



(c) Optimized partitions achieved after second iteration.

Figure 4.2: PDWA partitioning process for DAG shown in Fig. 4.1a.

Table 4.2: Heterogeneity model of three execution nodes.

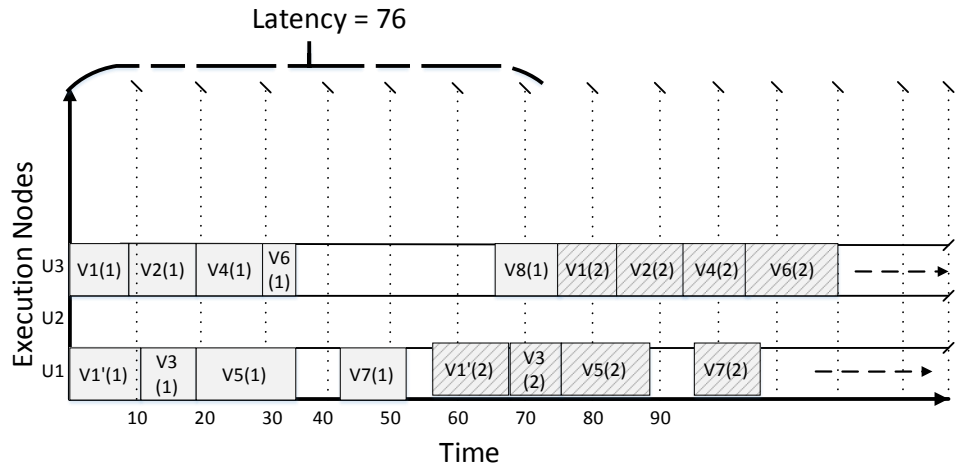
Network Links	Data Transfer Capability	Execution Nodes	Computing Capacity
l_1	10	u_1	7
l_2	100	u_2	5
l_3	1000	u_3	3

Table 4.3: Application tasks completion time of DAG shown in Fig. 4.1a

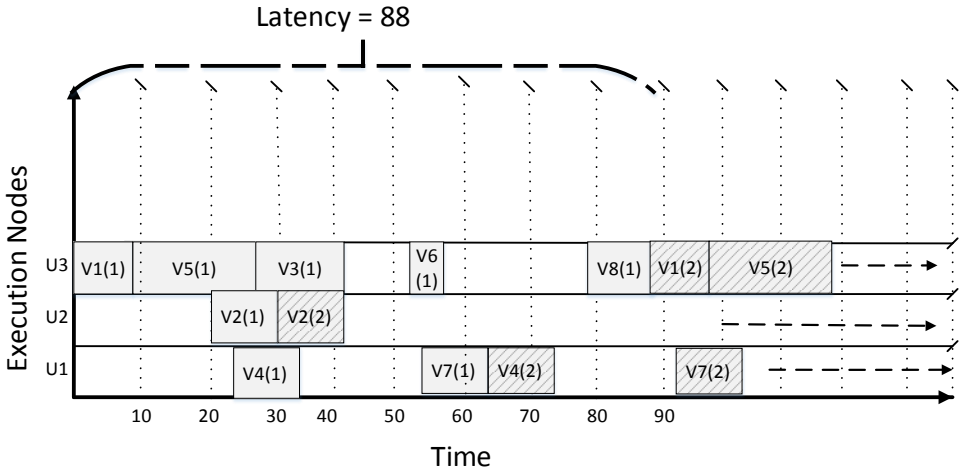
Tasks (v_i)	$CT(u_1)$	$CT(u_2)$	$CT(u_3)$
1	11	13	9
2	10	15	11
3	9	12	14
4	11	16	10
5	15	11	19
6	12	9	5
7	10	14	13
8	11	15	10

4.4 Data Parallelism Based PDWA (I-PDWA)

In data stream processing there can be three types of parallelism, *i.e.* *a*) task parallel execution, *b*) data parallel execution,and *c*) pipeline execution. Lets discuss these types with the help of example. Assuming Task Parallel Graph (TPG) shown in Fig.4.4 as an example and v_1, v_2, v_3, and v_4 have expected execution time for three identical execution nodes are 5, 10, 15, and 20 respectively and edge weights e_1,e_2, e_3, and e_4 as 2, 4, 6, and 8 respectively, we illustrate these execution options. *Task parallel execution:*Task parallelism can be deployed to those tasks that have no dependencies between them. Two tasks are called independent of each other if the output data stream never reaches the input stream of the other. Such independent tasks can be executed in parallel on different execution nodes instead of the execution of these tasks sequentially that causes longer time to execute. For the example in Fig. 4.4 the resulting task execution will be as shown in Fig. 4.5. In this example tasks v_2 and v_3 are independent of each other, therefore these tasks execute in parallel on two different execution nodes. Task v_2 can start execution as soon as task v_1 is completed because both tasks are on same execution node, however the delay caused in the start of task v_3 is because it is submitted to different execution



(a) Pipelined schedule of PDWA.



(b) Pipelined schedule of AWOP.

Figure 4.3: Pipelined schedules of example in Fig. 4.2.

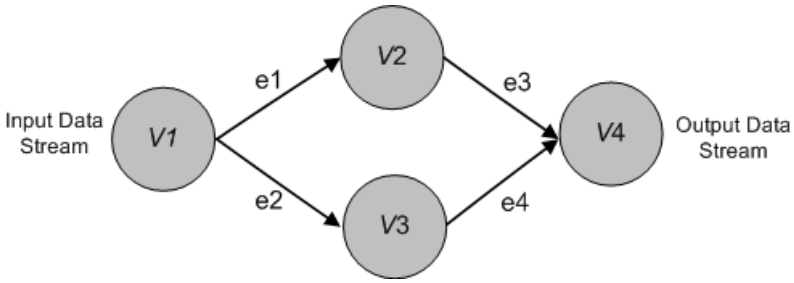


Figure 4.4: Example Task Parallel Graph

node and the delay is due to the edge between tasks v_2 and v_3 that is $e_2 = 4$. The overall execution time is 51 time units as shown in Fig. 4.5.

*Data parallel execution:*Data parallelism is the parallel execution of data items with no data dependencies between them. In data parallel execution, data is chopped into

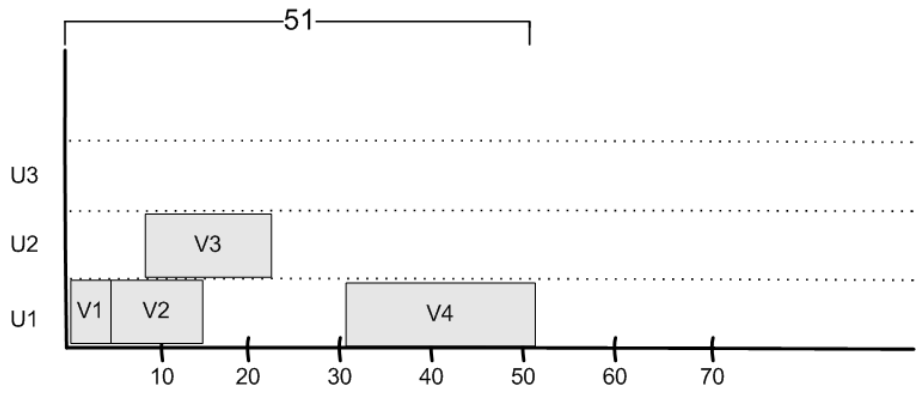


Figure 4.5: Task Parallel Execution

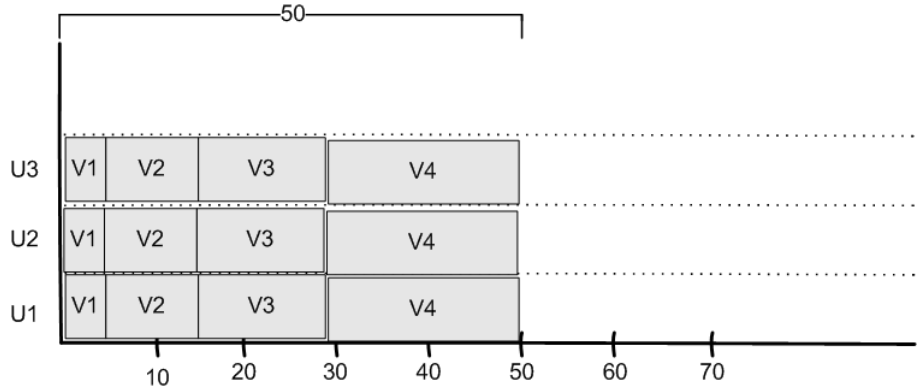


Figure 4.6: Data Parallel Execution

number of independent chunks of data and tasks are replicated to execution nodes that process these independent chunks of data in parallel. For the example in Fig. 4.4, the data parallel execution is shown in Fig. 4.6. Since, there are three execution nodes (u_1, u_2, u_3), therefore all the tasks are replicated on these execution nodes and the input data is also chopped into three independent chunks. The completion time in data parallel execution is 50 time units as shown in Fig. 4.6.

*Pipelined execution:*The task graph is mapped by taking into account the iterative behavior. A pipeline schedule will be developed and synchronous stages will be produced on all execution nodes as shown in Fig. 4.7. The input data is in the form of stream of different data items therefore, in Fig. 4.7 the tasks blocks without pattern present the execution sequence of first data item and the blocks with pattern represent second data item,

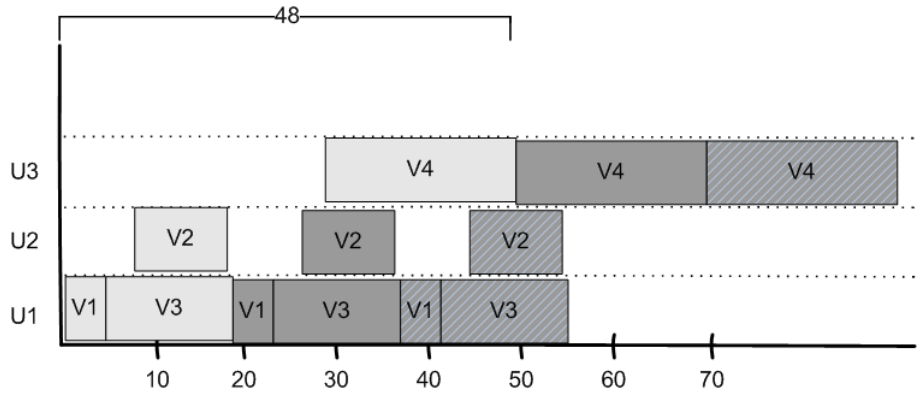


Figure 4.7: Pipelined Execution

similarly a pipeline execution continues for complete input data stream. The completion time is 48 time units in this scenario.

Task parallelism and pipelined execution features are already integrated in PDWA, however we have incorporated data parallelism in Improved PDWA (I-PDWA) presented in this paper. We apply the data parallelism technique to the most compute-intensive tasks of a workflow that reduces the latency of data-intensive workflow execution as compared to PDWA. In PDWA, partial task duplication is used in which the initial (entry) task is duplicated to multiple execution nodes that reduces the latency. We can further optimize latency by introducing data parallelism. In data parallelism mechanism different data items are processed by the same task, that are replicated on different execution nodes. The parallel execution of different chunks of data reduces the execution time. Fig. 4.8 depicts the Data Parallel Technique (DPT) using an example task with execution time of 40 time units and the amount of data processed is 20 MB. It is assumed that the task data parallel capabilities and each data item of input stream can be divisible into equal parts. As shown in Fig. 4.8 data is chopped into 4 pieces and 5 MB of each data chunk is processed by replicating the task T as $\{T', T'', T''', T''''\}$. Then the execution time of task T is reduced to 10 time units by the factor of 4. This concept is exploited in I-PDWA to optimize latency.

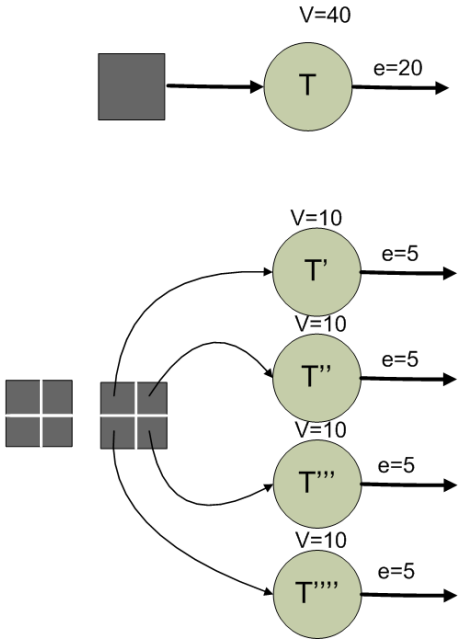


Figure 4.8: Data Parallel Technique

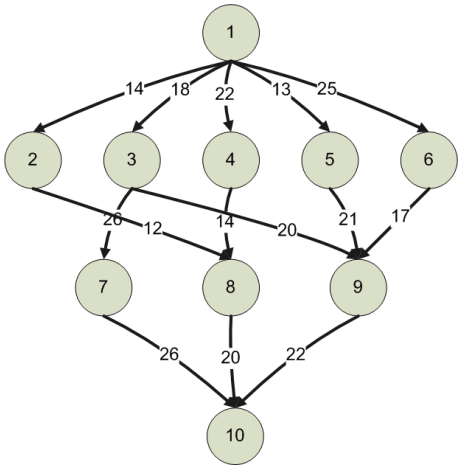


Figure 4.9: An example workflow to illustrate the proposed algorithm.

4.4.1 Illustration with an Example

We illustrate the proposed algorithm with an example workflow shown in Fig. 4.9 that is required to be executed on three fully- connected node target system as shown in Fig. 4.10. We consider the heterogeneity in the following two ways: *a*) The execution tasks are different, therefore, they require different execution times to process the input stream of data. Table 4.4 shows the completion time of all execution tasks across all execution nodes, *b*) The target system (execution environment) is also heterogeneous in which the

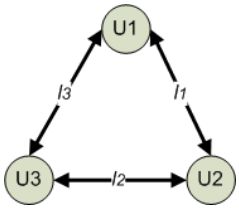


Figure 4.10: Execution environment for example workflow shown in Fig 4.9.

Table 4.4: Application tasks completion time of DAG shown in Fig. 4.9.

Tasks (v_i)	$CT(u_1)$	$CT(u_2)$	$CT(u_3)$
1	19	14	6
2	19	12	8
3	17	8	3
4	13	11	5
5	16	13	9
6	15	11	7
7	16	10	9
8	14	11	5
9	20	18	12
10	21	16	7

Table 4.5: Heterogeneity model of three execution nodes.

Execution Nodes	**DPS**	**Communication Links**	**DTC**
u_1	2	l_1	10
u_2	4	l_2	100
u_3	6	l_3	1000

Data Processing Speed (DPS) of each execution node (u_1, u_2, u_3) is different. Moreover, the Data Transfer Capacity (DTC) of communication links (l_1, l_2, l_3) between these nodes are different. Table 4.5 presents the characteristics of the execution environment. The baseline DPS of each execution node is assumed to be 1 GHz and the numbers in the second column of Table 4.5 are the number of cores in each execution node. Similarly the baseline DTC is assumed to be 1 Gb/Sec and the links l_1, l_2, l_3 have multiple of 10, 100, 1000 of baseline DTC.

The pseudo code of the proposed algorithm is given in Algorithm 10. A workflow modeled as TPG and execution environment TEE is read by the proposed algorithm. Then the TPG is partitioned based on the Algorithm 11. Entry task is duplicated in first two partitions, which reduces the latency significantly. The Algorithm 11 partitions TPG

Input: Given an application task graph $TPG(V,E)$.
Output: S of a Near Optimal Solution.
Read the graph: let V is the set of v application tasks & E is the set of e edges;
Partition(V,E) ; `// Algorithm 11`
Mapping(P Partitions) ; `// Algorithm 12`
DPT(P Partitions, $TEE(U,L)$) ; `// Algorithm 13`
Return optimal solution S

Algorithm 10: Data Parallel Technique based Partitioning based Data-intensive Workflow Optimization Algorithm (DPT-PDWA).

Input: E: Set of e edges; V: Set of v application tasks.
Output: P: Partitions of application task graph. Each partition contain n number of tasks.
N_{p_i} = 0; i = 1;
n = The number of tasks in the task graph;
Compute the threshold edge weight $e_{th} = \sum_{i=1}^{n} e_i/n$;
$N_{p_{max}} = v_n * f_p$; // Maximum possible number of tasks in one partition
Duplicate the entry tasks v_e to first two partitions;
while *($i <= n$)* **do**
  if($N_{p_j} <$ = $N_{p_{max}}$)
  Assign task v_i to Partition p_j;
  $i = i + 1$; //number of application task
  N_{p_j} = N_{p_j} + 1; // Number of tasks in p_j partition
  else
  $p_j = p_j + 1$; // Next Partition
  N_{p_j} = 0; // Reset the number of tasks in a partition
end
The application task graph splits into P_n partitions;
`/* Optimization of partitions */`
while *Inter-partition edge weights* $e_{ij} > e_{th}$ **do**
  Shift the task v_i or v_j to any other partition such that inter-partition edge weight always less than e_{th};
end
Return P_n partitions;

Algorithm 11: Workflow Partitioning Algorithm.

and iteratively process until the inter-partition edge weights is less than the threshold edge weight (e_{th}). e_{th} is assumed to be the average of the edge weights of entire workflow *i.e.* $e_{th} = \sum_{i=1}^{n} e_i/n$. In the example workflow e_{th} is 19. After a number of experiments, it is selected that partition factor f_p should be 0.5 in order to achieve suitable partitions. Maximum possible number of tasks in a partition is given by $N_{p_{max}} = v_n * f_p$ *i.e.* 5 in the example workflow. The TPG is partitioned according to Algorithm 11 and the resulting partitions obtained are $P_1 = \{1,2,4\}, P_2 = \{1,6\}$, and $P_3 = \{3,5,7,8,9,10\}$. In the map-

Input: P Partitions of application task graph and each partition contain almost equal m number of tasks.
Output: M mapped partition on the set of R resources.
for u_1 *to* u_n **do**
  Compute the CI of each partition using equation 4.8;
end
Determine the minimum CI value among the execution nodes *i.e.*, $CI_{min}(u_i)$;
Make a queue of partition Q_P to be mapped based on the descending order of their $CI_{min}(u_i)$;
while Q_p *not empty* **do**
  if (u_i is Idle)
  Map partition from queue Q_{Pj} to $CI_{min}(u_i)$;
  else
  Map partition Q_{Pj} to u_i that offer next lowest computation time;
  Delete the partition Q_{Pj} from the queue;
end
Return M mapped P partitions;

Algorithm 12: Partitions Mapping Algorithm.

Input: P Mapped Partitions of application task graph. Each Mapped Partition contain almost equal m number of tasks. Each execution node has n number of cores.
Output: M DPT applied to bottle neck task
for P_1 *to* P_n **do**
  Determine the bottle neck task T_{BN} in each partition;
  Determine the execution node u_i where T_{BN} is mapped;
end
There are n cores in the execution node then Divide the input data to n parts **for** ui_1 *to* ui_n **do**
  Map each chunk of data to each core
end
Return DPT applied to T_{BN};

Algorithm 13: DPT Algorithm.

ping phase (see Algorithm 12) each partition is mapped to one execution node, therefore within a partition the cost of data movement is zero. We define a criteria to map partition on the execution node named as Computation Index ($CI_{u_j}(P_n)$), which shows the CI of the execution node u_j of partition P_n. CI is given by:

$$CI_{u_j}(P_n) = \sum_{v_i \in P_n} CT(v_i) \tag{4.8}$$

where, $i = 1, 2, \ldots, m$, if there are m vertices in a DAG, are the application tasks that are

grouped to the partition P_n. Each partition is mapped to the execution node that gives the minimum CI. Hence, according to the above definition in this example $CI_{u_1}(P_1)$ is 51, $CI_{u_2}(P_1)$ is 37 and $CI_{u_3}(P_1)$ is 19. Similarly, for partition P_2, $CI_{u_1}(P_2)$ is 34, $CI_{u_2}(P_2)$ is 25 and $CI_{u_3}(P_2)$ is 13. $CI_{u_1}(P_3)$ is 104, $CI_{u_2}(P_3)$ is 76 and $CI_{u_3}(P_3)$ is 45, for partition P_3. According to Algorithm 12 line 5, $CI_{min}(u_i)$ for all partitions are determined and a priority queue Q_p is generated for the assignment of partitions to the execution nodes as shown in line 6. Q_p is generated based on the descending order of $CI_{min}(u_i)$ values. Therefore, in the example Q_p will be P_3, P_1, and P_2. So, based on Algorithm 12 line 6 to 12 P_3 will be mapped to u_3, P_1 will be mapped to u_2 and P_2 will be mapped to u_1. At this stage, we apply DPT to bottleneck tasks T_{bn}. These are the tasks with maximum execution times in each partition. In this example, task v_1 is most compute intensive (T_{bn}) in partition 1 as well as in partition 2, while v_9 is T_{bn} in partition 3. Therefore, computation time of T_{bn} is 19, 14, and 12 on execution nodes u_1, u_2, and u_3 respectively. From Table 4.5 we can observe that execution node u_1 is dual core, u_2 is quad-core and u_3 has six cores. On each T_{bn} tasks we apply DPT, as discussed in Section 4.4. The execution time of task v_1 on u_1 is reduced to half (*i.e.* 9.5 time units) similarly the execution time of v_1 on u_2 is reduced to one fourth (*i.e.* 3.5 time units) and the execution time of v_9 is decreased by six times that is 2 time units. After applying DPT the resulting pipelined schedule obtained is shown in the Fig. 4.11 with latency of 56.5 time units while by applying PDWA on the same workflow the latency is 77 as shown in Fig. 4.12. In both figures the plain blocks represent first instance of data item and the dotted pattern filled boxes show second instance of data item. The arrow represents pipelined schedule for further stream of data. In the similar pipelined fashion, whole stream of data is processed in the target system. I-PDWA outperforms not only from latency but also in throughput values as the throughput of PDWA is 0.013 and proposed algorithm is 0.0176. The proposed algorithm provides lower latency and high throughput and shows about 36% improvement in both

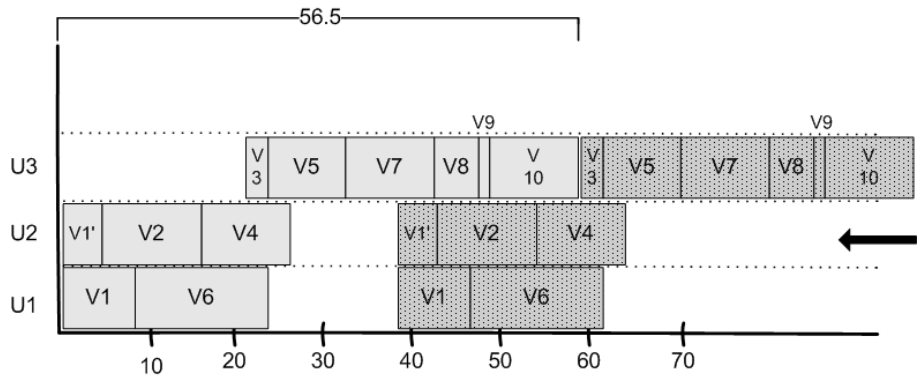


Figure 4.11: Corresponding schedule of example workflow shown in Fig. 4.9 using I-PDWA.

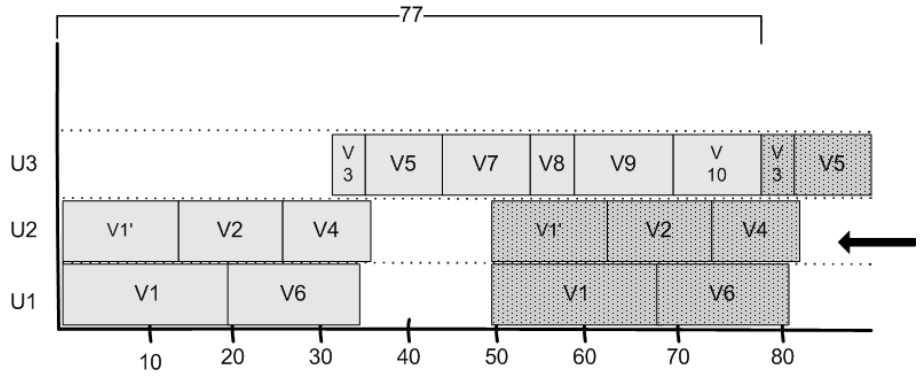


Figure 4.12: Corresponding schedule of example workflow shown in Fig. 4.9 using PDWA.

performance metrics as compared to PDWA.

4.5 Simulation Results and Discussion

In this section we evaluate the performance of I-PDWA with variety of datasets. We selected synthesized workflow (Ahmad et al., 2012) with diverse characteristics in addition using real-world applications to evaluate the performance of the proposed algorithm. The real-world applications that we used in our evaluation are Montage and Cybershake (Deelman et al., 2009). We performed 1000 runs of each simulation and calculated average latency and throughput which are discussed in detail in the following section. The selected datasets include almost all workflow patterns that are usually considered as benchmarks to study the performance of the given algorithm.

4.5.1 Performance Evaluation Using Synthesized Workflows

Synthesized workflows with different workflows characteristics are generated to evaluate the proposed algorithm. The characteristics of the synthesized workflows depend on the following parameters.

1. **Workflow Size (n)** The parameter n selects the number of nodes in a workflow. Higher the value of n bigger will be the size of the workflow. We generated the workflows of different size to evaluate the proposed algorithm. The following set n shows the sizes of workflows that is used in simulations $n = \{2k, 4k, 6k, 8k, 10k\}$.

2. **Communication to Computation Ratio (CCR)** This parameter determines weather the workflow is communication intensive or compute intensive. If $CCR > 1$ then the workflow is communication intensive. If $CCR < 1$ then the workflow is compute intensive and for $CCR = 1$, the workflow is neither communication nor compute intensive.

3. **Shape Parameter** (α) This parameter determines the shape of the workflow. If $\alpha < 1$, longer workflows with less parallelism are generated. If $\alpha > 1$, smaller workflows with higher parallelism are generated, however if $\alpha = 1$ then a balanced workflow which is neither long nor short in length is generated.

4. **Out-Degree** This parameter determines the number of edges going out of a node. A workflow will be dense if the value of it's out-degree is high. Since the synthesized workflows are generated randomly hence, the out-degree is selected randomly for each node.

5. **Number of Processors (P)** This parameter selects the number of processors. We selected the number of processors as 2^x where $x = \{2, 3, 4, 5, 6\}$.

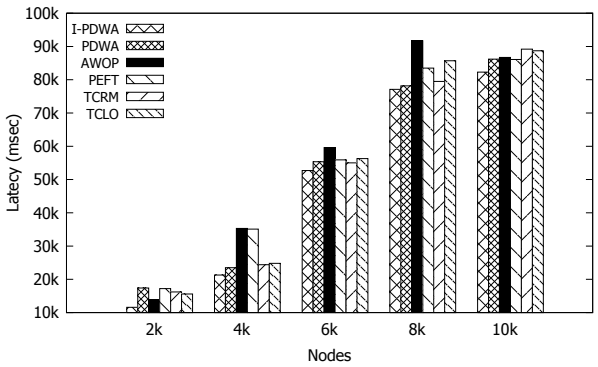


Figure 4.13: Impact on latency with increasing workflow size using synthesized workflows.

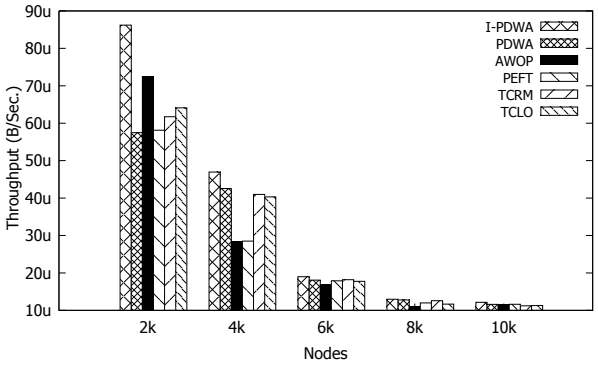


Figure 4.14: Impact on throughput with increasing workflow size using synthesized workflows.

6. **Heterogeneity Factor** (β) It determines the heterogeneity of the processor speed. Higher value in β causes significant difference in the execution costs of tasks for each processor while the lower value causes similar values of execution costs. Following values of β are used in our simulations, $\beta = \{0.1, 0.5, 1\}$.

We performed different simulations to evaluate the performance and effectiveness of the proposed algorithm. We generated a dataset of 100 random workflows with different characteristics as discussed above and the same dataset is used as input under the same conditions to the proposed algorithm, PDWA (Ahmad et al., 2014), AWOP (Munir et al., 2013), PEFT (Arabnejad & Barbosa, 2014b), TCRM (Guirado et al., 2013), and TCLO (Vydyanathan et al., 2011) and measure the latency and throughput for each algorithm. The results obtained show that the proposed algorithm outperforms all other

algorithms that we have used in this evaluation as shown in Fig. 4.13 and Fig. 4.14. The proposed algorithm shows 6.3% improvement in latency and similar improvement in throughput as compared to PDWA.

Similarly, a set of synthesized random workflows of three different types is generated by varying the CCR parameter. The purpose of this experiment is to evaluate the proposed algorithm for different types of workflows. The results of synthesized workflows with $CCR > 1$ are shown in Fig. 4.15. The communication intensive workflows show 9.15% improvement in the latency over PDWA. The results show that the proposed algorithm significantly performed better as compared to the other algorithms and performance becomes better as the size of workflows increases. Similar results were achieved for the workflows with $CCR = 1$, shown in Fig. 4.16. The proposed algorithm shows 7.7% improvement as compared to PDWA. However, the proposed algorithm shows reduced performance for the compute intensive workflows with $CCR < 1$ as shown in Fig. 4.17. This is because the proposed algorithm is mainly designed for communication intensive workflows and optimized partitions specifically minimizes the inter-partition data movements. We also evaluated the proposed algorithm with workflows of different shapes by varying the shape parameter α. The simulation results are shown in Fig. 4.18, Fig. 4.19, and Fig. 4.20. We observe that the proposed algorithm performs better for longer ($\alpha < 1$) and medium ($\alpha = 1$) shaped workflows. The proposed algorithm shows 2.76% and 0.237% improvement over PDWA respectively. Conversely, for short workflows having higher parallelism ($\alpha > 1$) the proposed algorithm shows reduced performance. The reason of reduced performance is due to high parallelism in the workflows with $\alpha > 1$. In the proposed algorithm, the partitions can not be optimized because of too many edges, therefore, the inter-partition edge weights are higher that causes the low performance of proposed algorithm.

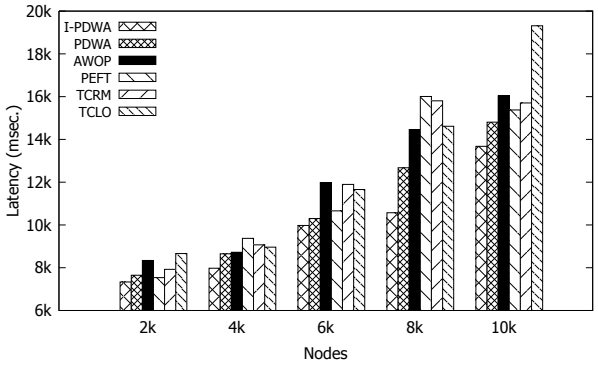


Figure 4.15: Impact on latency using synthesized workflows having CCR=10.

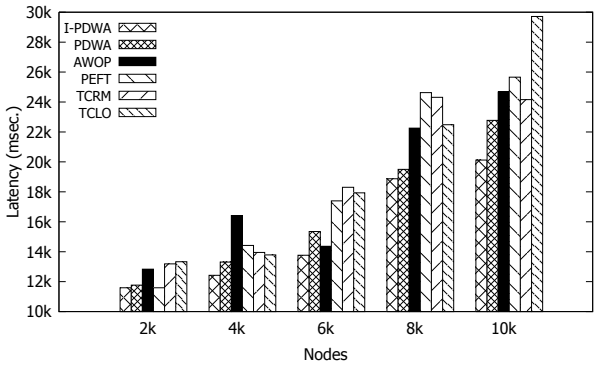


Figure 4.16: Impact on latency using synthesized workflows having CCR=1.

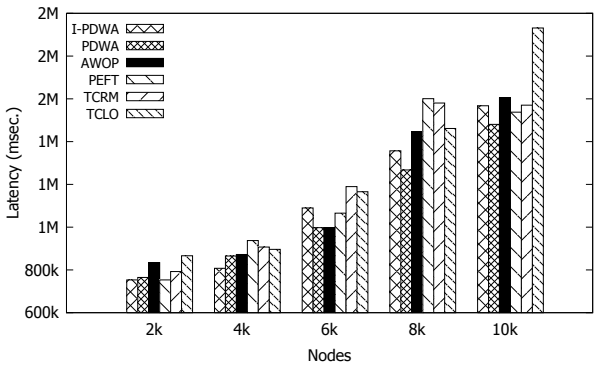


Figure 4.17: Impact on latency using synthesized workflows having CCR=0.1.

4.5.2 Performance Evaluation Using Real-World Applications

Montage (Deelman et al., 2009) is an astronomical mosaic engine created by NASA that is used to generate a mosaic of a sky. The input astronomical images are combined together to find the final mosaic. The geometry of the mosaic depends on the input images. The process can be represented in the form of a workflow. There are some jobs in Montage

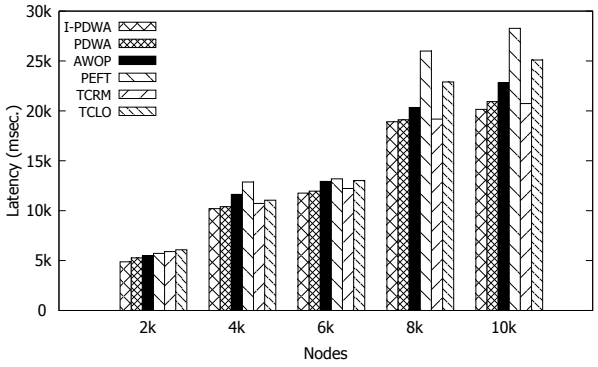


Figure 4.18: Impact on latency using synthesized workflows having $\alpha = 0.1$.

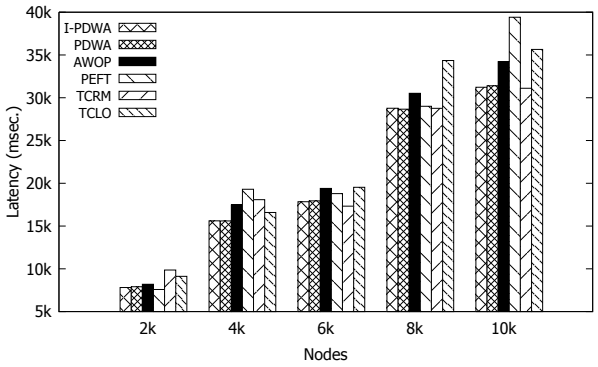


Figure 4.19: Impact on latency using synthesized workflows having $\alpha = 1$.

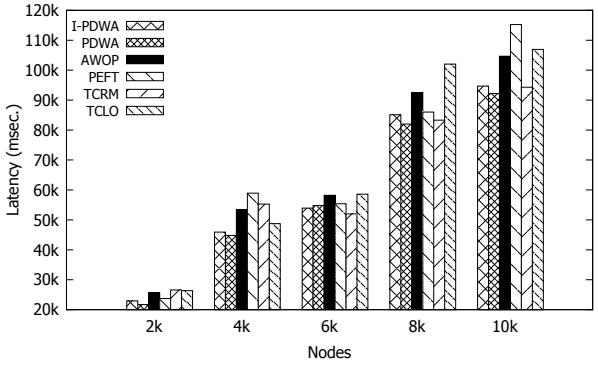


Figure 4.20: Impact on latency using synthesized workflows having $\alpha = 2$.

workflow that have very short execution time and few job take longer to complete. The structure of Montage workflow with twenty nodes is illustrated in Fig. 4.21. We use the publicly available workflow data from Pegasus Workflow Gallery[1] for this experiment, and executed Montage with 25, 50, and 100 nodes in our simulations. The simulation

[1]https://confluence.pegasus.isi.edu/display/pegasus/Workflow+Data

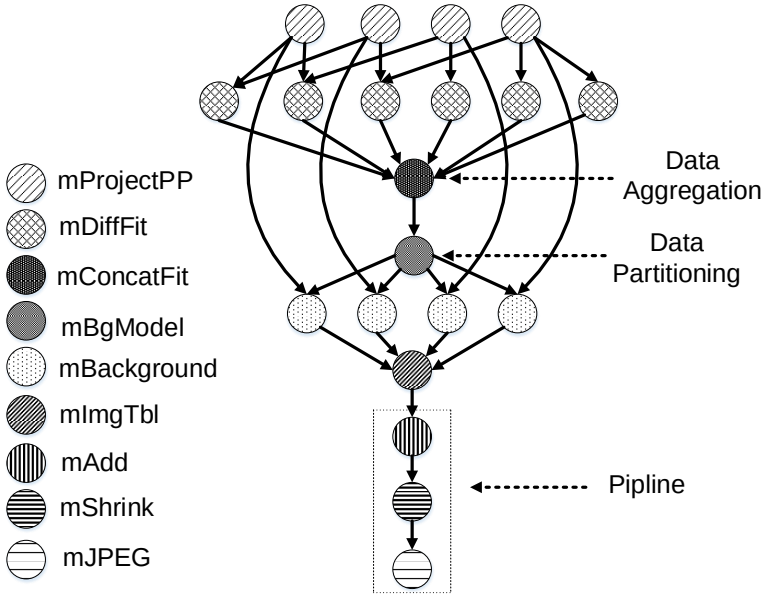


Figure 4.21: Twenty nodes Montage workflow.

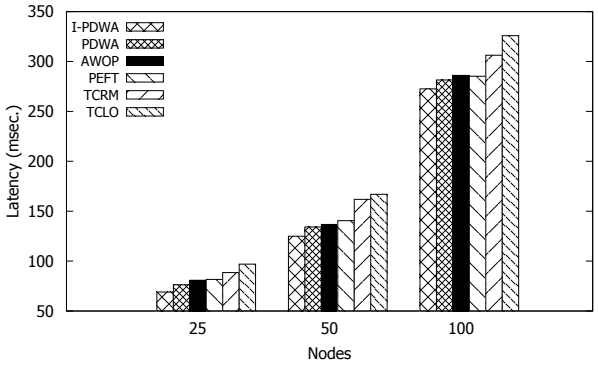


Figure 4.22: Impact on latency with 25, 50, and 100 nodes Montage workflows.

results of latency and throughput with different sizes of Montage workflow are shown in Fig. 4.22 and Fig. 4.23. The results show that proposed algorithm shows better performance as compared to the algorithms used in our evaluation and shows 5.46% reduced latency in comparison with PDWA. Similarly, the proposed algorithm outperforms other algorithms in terms of throughput and provides 13% improved throughput over PDWA.

Cybershake (Deelman et al., 2009) is used by the Southern California Earthquake Center (SCEC) to identify the earthquake within the region. Cybershake is relatively simpler workflow but it can handle large sizes of datasets, therefore it is a compute as

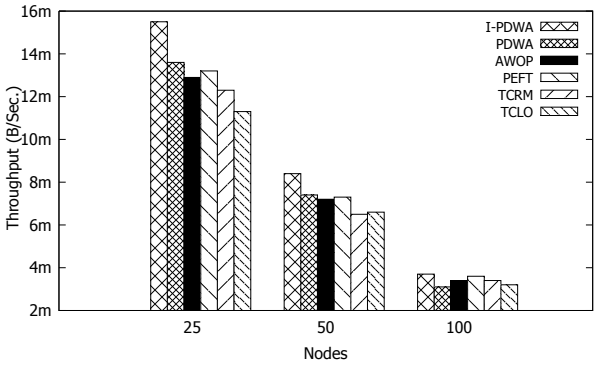


Figure 4.23: Impact on throughput with 25, 50, and 100 nodes Montage workflows.

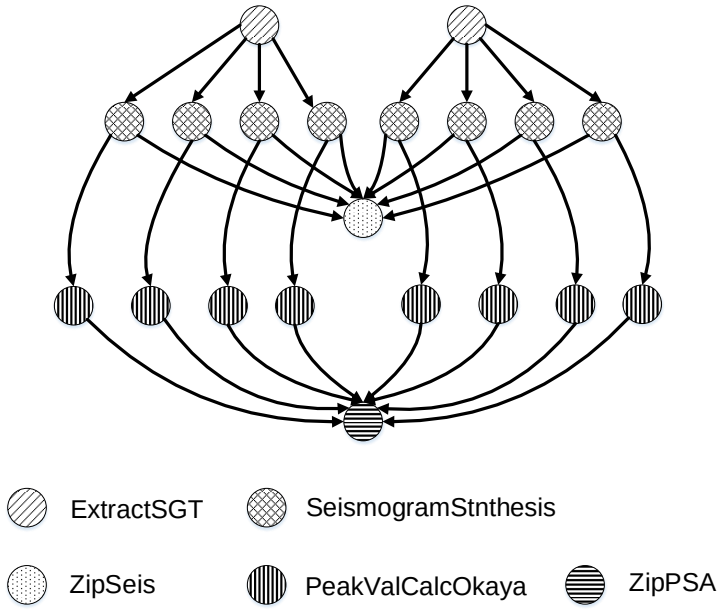


Figure 4.24: Twenty nodes Cybershake workflow.

well as data intensive workflow. Fig. 4.24 shows the structure of a small size twenty nodes Cybershake workflow. We selected different sizes of Cybershake workflow for the performance evaluation of the proposed algorithm. Latency and throughput of 30, 50, and 100 nodes Cybershake workflows were determined for the proposed and other five algorithms. The simulation results of latency and throughput are presented in Fig. 4.25 and Fig. 4.26 respectively. The results show that the proposed algorithm provides 6.5% and 9.37% better results for both latency and throughput respectively as compared to PDWA. The results are even more better for other five algorithms.

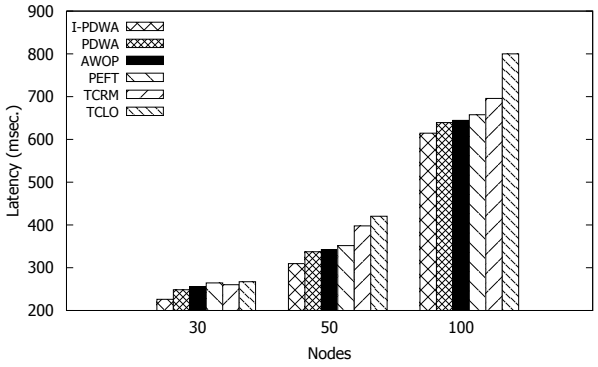


Figure 4.25: Impact on latency with 30, 50, and 100 nodes Cybershake workflows.

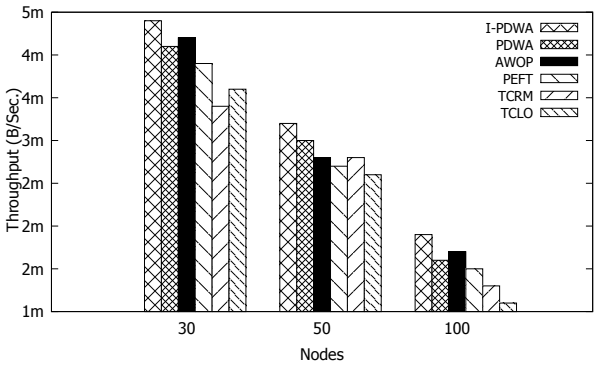


Figure 4.26: Impact on throughput with 30, 50, and 100 nodes Cybershake workflows.

In this research work, the stream-based data processing model is used, which differs from real-time data streams. The data-intensive workflow scheduling is not online or run time scheduling which is called dynamic scheduling. Hence, the proposed algorithms are static scheduling algorithms that perform scheduling with all the system information prior to the execution of workflow. In this category of scheduling algorithms the scheduling overhead does not cause any problem or inefficiency in the execution as well as performance of the system. On the other hand, the scheduling overhead in dynamic scheduling can cause serious issues like data loss if there is a scheduling overhead. In this scenario, the delay caused by the scheduling overhead makes unsynchronized data processing and data processing rate. That effect adversely on the system performance and useful data might loss.

The proposed algorithms are heuristics hence the time complexity is low. In $I-PDWA$ the time complexity is determined by asymptotic analysis. Since, the architecture of the algorithm consists of three phases *a*) Partitioning, *b*) Mapping, and *c*) DPT. Based of asymptotic analysis the time complexity of partitioning phase is $O(n)$, where n is the number of nodes of workflow. $O(n+u_n)$ is for mapping phase and time complexity of DPT is $O(n+ui_n)$, where u_n is the number of execution node in the computing environment and ui_n is the number of core in an execution nodes used for data parallelism. Hence, the overall time complexity is $O(n+u_n+ui_n)$. From the expression it is clear that the time complexity depends on workflows size and execution environment. Bigger the workflow and greater the number of execution node with more core the time complexity increases. There is always a compromise between time complexity and the algorithm optimization parameters i.e. latency and throughput in this case.

4.6 Conclusion

In this chapter, we presented algorithms to optimize data-intensive workflows. The algorithm splits the workflow into suitable partitions in such a way that the inter-partition communication cost is minimum. Each partition is mapped to one selected execution node, therefore, intra-partition data-movement is eliminated, which reduces the overheads of large data movement. Moreover, the execution environment is heterogeneous that makes the optimization more challenging. In the mapping phase of PDWA, each partition is assigned to the execution node that offers minimum execution cost. Improvement in PDWA (I-PDWA) is also presented in this chapter. In which data parallelism is applied to most most compute intensive task in each partition. This addition in PDWA improves the results to significant extent, reduces the latency and increases the throughput due to optimized partitioning, mapping, and data parallel technique. The proposed approach is evaluated with synthesized and real-world applications of various characteristics. The

evaluation of the proposed algorithm shows that it provides significantly reduced latency with improved throughput. Next chapter is about the implementation of these algorithms in real time SDPM execution platform *i.e.* STORM.

CHAPTER 5

EVALUATION OF PROPOSED ALGORITHM USING A REAL-WORLD USE CASE IN STORM

In this chapter, the proposed algorithm as presented in Chapter 4 is implemented in real-time stream-based execution platform, *i.e.* STORM, which is an open source frame work to process unbounded streams of data. In order to evaluate the performance we adopted real-world use case (EURExpressII workflow). The performance is measured in terms of execution time of the workflow.n addition, this chapter presents detailed discussion on related workflow execution platforms and highlights the key features of STORM. Also, it explains the selected use-case and experimental setup. Finally, it presents the detailed results and discuss the key outcomes of experiments.

5.1 Introduction

The data-intensive workflow execution is optimized in PDWA by partitioning the workflow. The detailed explanation of the algorithm is given in Chapter 4. Initially, the performance of the algorithm was evaluated through simulations. The results presented in Chapter 4 proved that PDWA performs better for data-intensive workflows as compared to the algorithm which did not partition the workflow. The characteristics of real-world workflows and environment cannot be fully simulated therefore, we setup the experiments in real-world environment and evaluate our algorithm with real-world workflow. In this chapter, the proposed algorithm PDWA is fully implemented in STORM and evaluated in real data stream based framework with a use case from real-world. Since, the experimental setup is STORM based cluster therefore, I-PDWA couldn't be implemented in homogeneous environment. However, we have introduced data parallelism in the workflow at certain levels. After vast study of various frameworks STORM was selected to implement

PDWA and evaluated the proposed algorithm with real-world workflow (EURExpressII). It is a data-intensive workflow described in forthcoming section in this chapter.

In our research work, the proposed algorithm is evaluated with simulations (see Chapter 4) as well as in real time framework. Both evaluation methodologies have their own benefits. Simulations allow the designer to determine the correctness and efficiency of the designed algorithm before it is implemented/deployed in real system. Consequently, merits and efficiency of the proposed work can be studied and can consider alternative designs. In addition to that the design and evaluation can be carried out without using real computing resources. Thus, simulations provides an initial low cost evaluation and design solution, that allow to measure algorithm's efficiency and even compare with state of the art algorithms. Simulations also provide an advantage to test the proposed algorithm at extreme conditions such as in this research, the proposed algorithm can be evaluated with different shapes, sizes, types of workflows, number of execution nodes and input data size at even maximum and minimum levels. This helps to determine the limitations and complexity of the proposed algorithm. In addition, the behavior of proposed algorithm also needs to be observed with real-world data and in real-time framework. The performance of proposed algorithm is analyzed by its implementation in real time framework and with data-intensive workflow based on real world application. These experiments help to analyze the proposed algorithm (PDWA) performance in the real system.

5.2 Execution Platforms

There is an escalating interest found in the scientific research on the big data stream processing. In literature we find number of frameworks that support batch mode of data processing however, few for data streams. We discuss these frameworks briefly as follows.

1. **Spark:** Apache Spark is a fast engine for processing large scale data. It is about 100 times faster than Hadoop MapReduce in memory, or about 10 times faster on disk. Spark takes MapReduce to next level with less expensive shuffles in the data processing with capabilities like in-memory storage and near real-time processing. Its performance is several times faster than other big data frameworks. Spark also evaluates lazy evaluation of big data queries, which helps with optimization of the steps in data processing workflows. It provides a higher level of API to improve developer productivity and a consistent architect model for big data solutions. Spark holds intermediate results in memory rather then writing them to disk which is very useful especially when you need to work on same datasets multiple times. It is designed to be an execution engine that works both in-memory and on-disk. Spark operators perfrom external operations when data does not fit in-memory. Spark can be used for processing datasets that is larger than the aggregate memory in a cluster. Spark will attempt to store as much as data in memory and then spill to disk. It must be considered for data and use cases to access memory requirements with this in-memory data storage, spark comes with performance advantage.

   Spark Architecture: It includes following three main components

   a) **Data Storage:** Spark uses HDFS file system for data storage purposes. It works with any Hadoop compatible data source including HDFS, HBase, Cassandra, etc.

   b) **API:** The API provides the application developers to create spark based applications using a standard API interface. Spark provides API for Scala, Java, and Python programming languages.

   c) **Resource Management:** Spark can be deployed as a stand alone server or it can be on a distributed computing framework like Mesos or Yarn.

Mainly, in spark batch processing is used for data processing however, micro-batching is also used in spark streaming. Since, spark is not solely used for stream processing moreover it was difficult to plugin custom scheduler, therefore it is not used in our experiments.

2. **Hadoop:** It is an open source software framework for storing and processing big data in a distributed fashion on large clusters of commodity hardware. Essentially, it accomplish two tasks *a*) massive data storage and *b*) faster processing. Hadoop framework can store huge amounts of data by breaking the data into blocks and storing it in clusters of lower cost commodity hardware. Hadoop also processes large amounts of data in parallel across clusters in the form of batches. It is low cost, high computing power, scalable, and have flexible storage. It has inherent data protection and self healing capabilities.

   Hadoop architecture consists of two main components, *a*) HDFS and *b*) MapReduce. HDFS is a java based distributed file system that can store all kinds of data without prior organization. MapReduce is a software programming model for processing large datasets in parallel.

3. **HBase:** Apache HBase is a non-relational (NoSQL) database that runs on the top of HDFS. It is an open source database that provides real-time read/write access to those large datasets. HBase scales linearly to handle huge datasets with billions of rows and millions of columns and it easily combines data sources that use a wide variety of different structures and schemas. HBase is naively integrated with Hadoop and works seamlessly alongside other data access engines through YARN. Since, HBase is integrated with Hadoop so data is processed in batches.

   Apache HBase provides random, real-time access to data in Hadoop. It was created for hosting very large tables making it a great choice to store multi-structured or

sparse data. Users can query HBase for a particular point in time, making "flash-back" queries possible. These following characteristics make HBase a great choice for storing semi-structured data like log data and then providing that data very quickly to users or applications integrated with HBase. Considering the batch processing behavior, we did not selected HBase in these experiments.

4. **S4:** S4 is a general purpose distributed, scalable, fault-tolerant, pluggable platform that allows programmers to easily develop applications for processing continuous unbounded streams of data. S4 fills the gap between complex proprietary system and batch-oriented open source computing platforms. S4 is high performance computing platform that hides the complexity inherent in parallel processing system from the application programmer.

   S4 is not selected for the experimental platform for this research because of its architecture as it is more like event-driven instead of data-driven platform. S4 uses processing elements as its basic computational unit and a new process will be instantiated for each value of the key attribute in other words its kind of auto scaling for each different key attribute. These processing elements are then distributed evenly across the processing nodes. The processing nodes are like the logical hosts for processing elements and can do any kind of work. Basically, there is no preset number of processing elements and the generated processing elements are not known initially. When the processing elements are instantiated then these are required to schedule on the processing nodes.

5. **STORM:** STORM [1] is an open source frame work to process unbounded streams of data. Apache Storm is developed by Nathan Marz at BackType, which is later acquired by Twitter. Now-a-days, some organizations are using Apache Storm, for

[1] http://storm.apache.org/

instance, Yahoo!, Alibaba, Baidu, Groupon and so on. Apache Storm is a real-time distributed stream data processing engine. Its characteristics are as below:

- Scalable: Nodes can be easily added or removed from the Storm cluster without disrupting existing data flows through Storm topologies.
- Resilient: Fault-tolerance is crucial to Storm as it is often deployed on large clusters, in which hardware components can fail. The Storm cluster must continue processing existing topologies with a minimal performance impact.
- Extensible: Storm topologies may call arbitrary external functions (*e.g.* looking up a MySQL service for the social graph), and thus needs a framework that allows extensibility.
- Efficient: Since Storm is used in real-time applications; it must have good performance characteristics. Storm uses a number of techniques, that includes keeping all its storage and computational data structures in memory.
- Easy to Administer: Early warning tools are needed to quickly point out the source of problems as they arise.

In literature, few research articles has been found related to STORM that addresses data-intensive workflow optimization and the article (Aniello et al., 2013) is one of them. In this article the authors presents a scheduling scheme for STORM that enhances the performance of different STORM typologies. The algorithm has two parts, first part works offline and allocate resources based on the structure of the topology while the second part supports the resource allocation and works at run time. The second part of the algorithm supports the previous deployments by monitoring online and rescheduling at runtime. It reduces the latency of input data stream. However, topology based scientific workflows are not being discussed, moreover the execution environment is a STORM

cluster of 8 worker nodes (homogeneous computing environment) while in this paper we analyze the scheduler in a heterogeneous STORM cluster.

In another recent research work, DaweiSun et al. (2015) proposed a framework for real-time resource scheduling for big data streams but energy efficiency is mainly focused. This framework is aided utilizing an energy efficient heuristic and critical path within the data stream graph. The algorithm achieve a trade-off between computation and response time. This algorithm does not perform well for data-intensive workflows, in addition to this the performance evaluation of the proposed work is carried out on a 16 virtual machine in data centers. Since the virtual machines of same data center are used, therefore the distributed and heterogeneous execution environment is ignored. Moreover, overall system latency is not measured neither analyzed.

In a similar proposed work (Rychly et al., 2014), a stream based scheduling algorithm is designed for heterogeneous cluster. It does not focus the latency of entire topology, however the proposed work optimizes the cluster utilization.

5.3 STORM

In this section presents the details of STORM architecture, fault tolerance and tuple grouping in STORM.

5.3.1 STORM Architecture

According to the physical view, STORM consists of following parts, as presented in Fig. 5.1.

- Nimbus node: It is a master node and similar to the Hadoop JobTracker. It distributes jobs and launches workers across the cluster. It also monitors jobs and reallocate workers as required.

- ZooKeeper node: It communicates and coordinates the Storm cluster.

- Supervisor node: It communicates with Nimbus through Zookeeper. It starts and stops workers according to Nimbus.

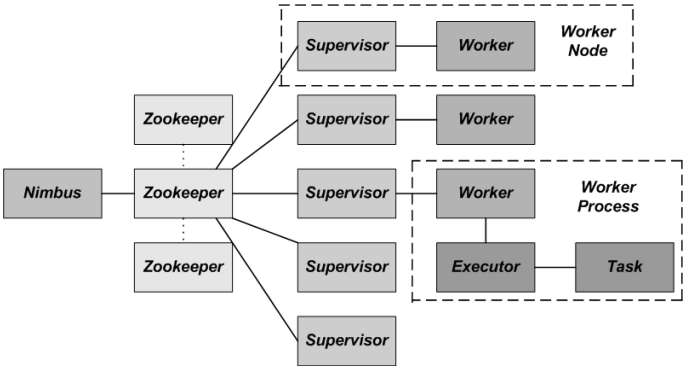


Figure 5.1: STORM Physical View (Evans, 2015)

Similarly, the conceptual view of STORM is shown in Fig. 5.2. The basic STORM data processing architecture consists of streams of tuples flowing through topologies (Solaimani et al., 2014).

- Spout: Source of input tuples streams for the topology. It can read from external data source.

- Bolt: Processor units with most crucial or analysing logic. It can process any number of input tuples streams and produce any number of output streams to the next set of bolts downstream.

- Topology: Network of Spouts and Bolts. It is a directed graph where the vertices represent computation nodes and the edges represent the data flow between the computation components. It runs indefinitely when it is deployed.

Clients submit topologies to a master node, which is called the Nimbus. Nimbus is responsible for distributing and coordinating the execution of the topology, where the actual work is done on slave nodes. Nimbus is also responsible for scheduling the topologies on the worker nodes and monitoring the progress of the tuples flowing through the topology.

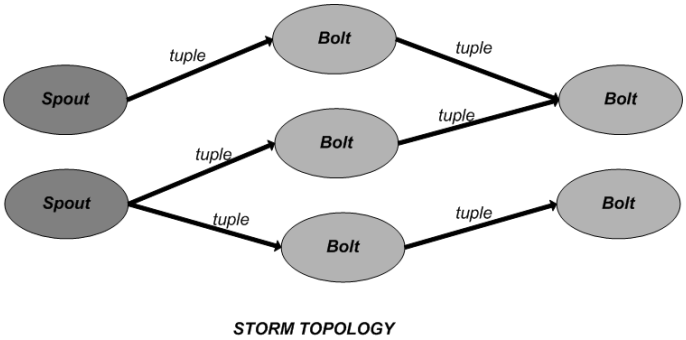


Figure 5.2: STORM Conceptual View (Evans, 2015)

Nimbus plays a similar role as the "JobTracker" in Hadoop, and act as interface between the user and the Storm system. Nimbus is an Apache Thrift service and Storm topology definitions are Thrift objects. To submit a job to the Storm cluster (*i.e.* to Nimbus), the user describes the topology as a Thrift object and sends that object to Nimbus. With this design, any programming language can be used to create a Storm topology. As part of submitting the topology, the user uploads the user code as a JAR file to Nimbus. Nimbus uses a combination of the local disk(s) and Zookeeper to store state about the topology.

5.3.2 Fault Tolerance in STORM

All coordination between Nimbus and the Supervisors is done through Zookeeper. Furthermore, Nimbus and the Supervisor daemons are fail-fast and stateless, and all their state is kept in Zookeeper or on the local disk(s) to assure the Storm's resilience. If the Nimbus service fails, then the workers still continue to make forward progress. In addition, the Supervisors restart the workers if they fail. However, if Nimbus is down, then users cannot submit new topologies. Also, if running topologies experience machine failures, then they cannot be reassigned to different machines until Nimbus is revived. The supervisor runs on each Storm node. It receives assignments from Nimbus and spawns workers based on the assignment. It also monitors the health of the workers and respawns them if necessary. Each worker node runs one or more worker processes. At any point in

time a single machine may have more than one worker processes, but each worker process is mapped to a single topology. More than one worker process on the same machine may be executing different part of the same topology. STORM based system is redundant. There are 4 possibilities of failure and the handling methods of STORM are discussed as following

- If the Nimbus daemon fails, the task processing is still continued, but without topology life cycle operations and reassignment facilities.
- If the Supervisor daemon fails, the task processing is still continued, but the assignment is never synchronized.
- If the worker process fails, the Supervisor daemon will restart the worker process and continue processing the tasks.
- If the remote slave nodes fail, the Nimbus will reassign the tasks to other cloud machines in the same cluster to continue the task processing.

5.3.3 Tuple Grouping Strategies in STORM

Each worker process runs a JVM, in which it runs one or more executors. Executors are made of one or more tasks. The actual work for a bolt or a spout is done in the task. A task is an instance of a spout or a bolt. A task is strictly bound to an executor because that assignment is currently static. Thus, tasks provide intra-bolt/intra-spout parallelism, and the executors provide intra-topology parallelism. Worker processes serve as containers on the host machines to run Storm topologies. With each spout or bolt, a set of tasks are running in a set of executors across machines in a cluster. Data is shuffled from a producer spout/bolt to a consumer bolt (both producer and consumer may have multiple tasks). Main partitioning strategies of Storm:

- Shuffle grouping, which randomly partitions the tuples.

- Fields grouping, hashes on a subset of the tuple attributes/fields.
- All grouping, which replicates the entire stream to all the consumer tasks.
- Global grouping, which sends the entire stream to a single bolt.
- Local grouping, which sends tuples to the consumer bolts in the same executor.

The default STORM scheduler is based on round robin strategy designed for even allocations therefore, called as even scheduler (Aniello et al., 2013). In the first phase it iterates through the topology executors, grouped by component, and allocates them to the configured number of workers in a round-robin fashion. In the second phase the workers are evenly assigned to worker nodes, according to the slot availability of each worker node. This scheduling policy produces workers that are almost assigned an equal number of executors, and distributes such workers over the worker nodes at disposal so that each one node almost runs an equal number of workers.

5.4 EURExpressII

The EURExpressII project (Han, van Hemert, & Baldock, 2011) aims to build a transcriptome-wide atlas of gene expression for developing mouse embryo established by RNA in situ hybridisation. The project annotates images of the mouse embryos by tagging images with terms from the ontology for mouse anatomy development. The data consists of mouse embryo image files and an annotation database (in MySQL) that describes the images. In this project, 4 TB of images have been produced and 80% of the annotation is done manually by human curators. Based on 600 MB that we have received, we will produce multiple classifiers where each classifier recognize a gene expression from a set of 1,500 anatomical components to classify the remaining 20% of images (85,824 images) automatically. The overall EURExpressII automated annotation task is divided into

3 stages: training, testing, and deployment. Both testing and training stage are performed in a workflow. Dataset is split into 2 parts: for training a classifier and for testing.

The block diagram of workflow is shown in Figure 5.3. Initially, raw image file and annotation database is read, image is then scaled to standard size of 320×200 pixels. The noise is removed from the images by applying median filter. Features Generation, using wavelet transformation, generate the image features as matrices of wavelet coefficients. 64,000 features are generated per image of 320×200 pixels. In features extraction, it reduces the features set by selecting the representative features for constructing classifiers using Fisher Ratio analysis (Fisher, 1936). In our experiment, 24 most significant features are extracted from 64,000 features generated in feature generation stage. The classifier design stage build a separate classifier for each anatomical feature which takes image features as input and outputs a rating of "not detected", "possible", "weak", "moderate" or "strong" for anatomical features such as eyes, nose, etc. The evaluation stage, test the classifier built in classifier design step against a partition of the data not used in the preceding steps but already classified.

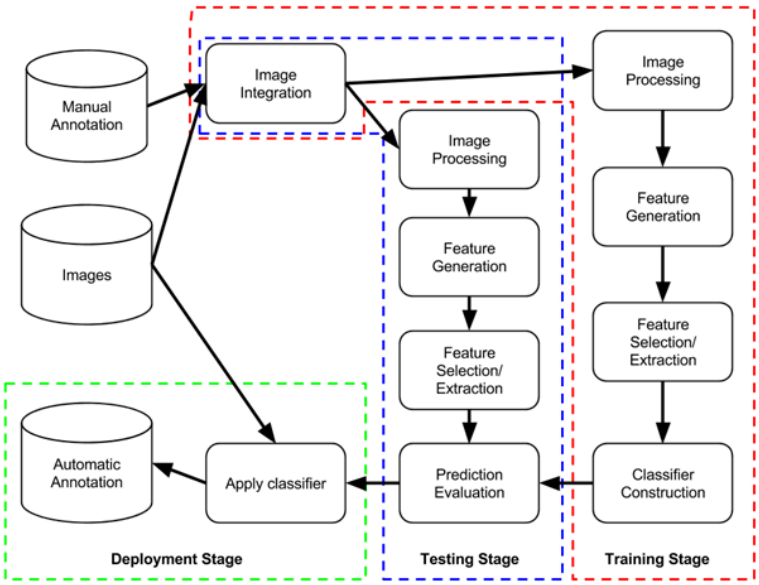


Figure 5.3: EURExpressII Workflow Block Diagram (Han, van Hemert, & Baldock, 2011)

5.5 Computing Environment for Experiments

The system architecture of the proposed solution, which demonstrates the logical design, is shown in Fig. 5.4.

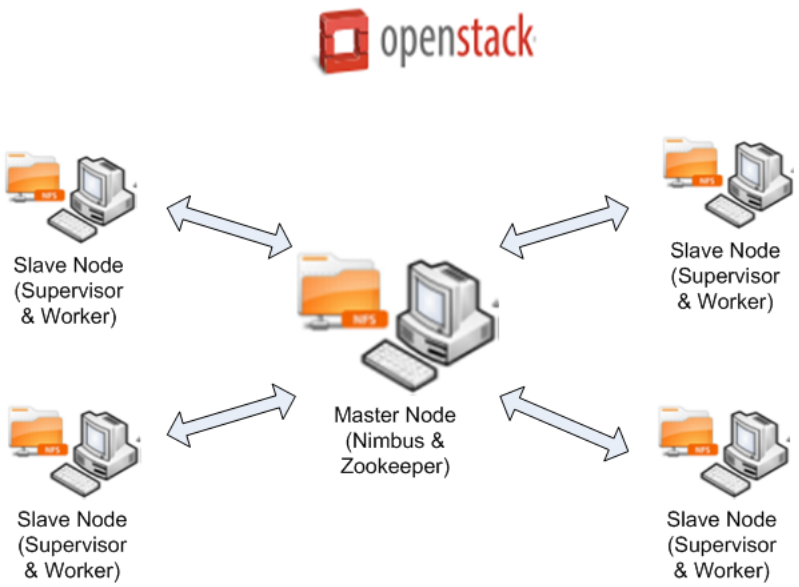


Figure 5.4: Computing Environment for Experiments

The system is developed in OpenStack Virtual Private Cloud (VPC) network, to ensure the inter-connectivity between the master node and the remote slave nodes. In the STORM cluster of proposed system Nimbus and Zookeeper are together on the master node and the Supervisors and worker nodes are on same slave nodes. The Nimbus at the master node would submit the assigned workflow tasks in the form of topology to the Supervisor of the remote slave nodes for processing purposes. The Supervisor spawns workers based on the assignments from the Nimbus. The Nimbus, Supervisor and workers would produce the log files encompass the detail of the overall process that can be used for error tracing as well.

The experiment is implemented using Apache Storm 0.9.6. A Storm topology is created and deployed on our computational platform. The topology is executed and performance data is collected. The execution is repeated with different data sizes *e.g.*,

pre-process different numbers of image files over a range of 800 to 8000. The results collected for subsequent analysis. The workflow converted to STORM topology as shown in Fig. 5.5.

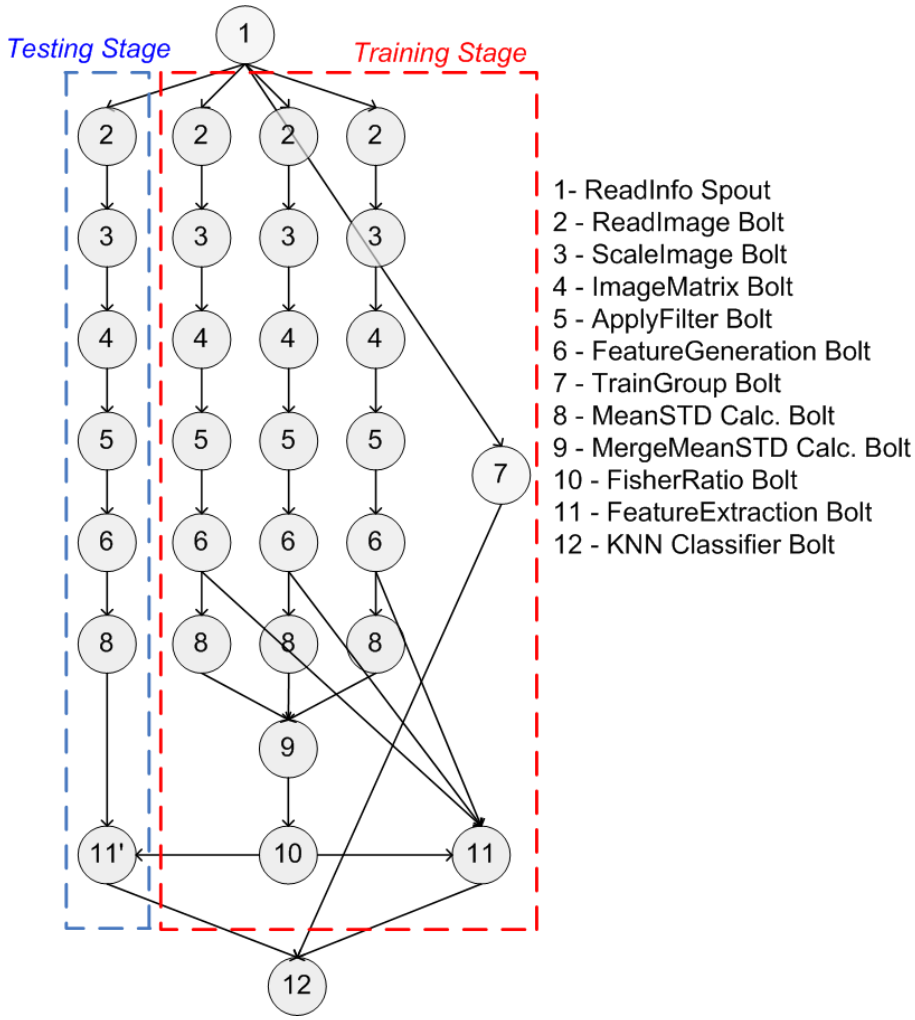


Figure 5.5: EURExpressII STORM Topology

The hardware infrastructure comprises 5 Virtual Machines with 2 Intel Xeon Processor Core 2.2 GHz with 8 GB RAM connected using a 10/100 Mbps switch. One of the virtual machine act as the Apache Zookeeper and Storm Nimbus to keep track the status of the Storm supervisor and scheduling the tasks to the supervisors. While the other four act as Storm supervisor to run the task given to them. All of these virtual machines are run on Dell PowerEdge R820 Rack Server. The software infrastructure include five virtual machines are running on CentOs 6 $(Linux - version2.6.32 - 573.18.1.el6.x86 - 64)$.

Table 5.1: Communication Cost Table of EURExpressII Workflow for 8000 input images shown in Fig. 5.5.

Edges (predecessor,successor)	Data Size(Mb)
(1,2)	1.77
(2,3)	400
(1,7)	5.31
(3,4)	250
(4,5)	190
(5,6)	190
(6,8)	360
(6,11)	360
(7,12)	0.02
(8,9)	7
(9,10)	7
(10, 11)	0.00112
(10, 11')	0.00112
(11,12)	1.27
(11',12)	0.53

In order to run Apache Storm, all of the VMs are installed with Apache Storm 0.9.6 and prerequisite software, such as Java 7.

5.6 Results and Discussion

This section describes the experimental results, and discuss the comparative observations with default STORM scheduler and benchmark. Benchmark represents the average execution time when the workflow is executed on local (single) node. The performance of workflow execution is measured in terms of execution time because the structure of the workflow is pipelined from task 1 to 9 but due to the data aggregation at node 9 and 12 latency and through put can not be measured. There are following parameters that we have used to compare the performance of PDWA.

- *Average Execution Time(AET)*

  The performance is measured in terms of Average Execution Time(AET). It is defined as

$$AET = \sum_{n=1}^{m} ET_n/n \tag{5.1}$$

where, ET_n is the execution time of the single run of workflow, m is the maximum number of runs and n is the number of execution of workflow.

- *Speedup*

  Speedup is another performance metrics that is considered in performance evaluation. It is defined as the ratio between the sequential execution time that is the cumulative execution cost of all workflow tasks on single (local) node to the parallel execution of the workflow.

$$Speedup = \sum_{v_i=1}^{n} ET_{v_i}(u_1)/ET(u_j) \tag{5.2}$$

  where, $ET_{v_i}(u_1)$ is the sum of execution time of n workflow tasks v_i on single processing node u_1 and $ET(u_j)$ is the execution time achieved when workflow is executed on u_j execution nodes, where, $j = 1,2,3,\ldots$ that cause parallel executions.

- *Efficiency*

  The third performance metric is the efficiency, which is defined as

$$Efficiency = Speedup/n, n = 1,2,3\ldots \tag{5.3}$$

  where, n is the number of execution nodes.

The experiments was repeated for number of times and AET, speedup and efficiency is determined. Fig.5.6 represents the PDWA ouput in terms of AET for different datasizes (Number of Images = {800, 1600, 3200, 4800, 6400, 8000}). The achieved results are plotted against number of worker (execution) nodes. In Fig. 5.6 it must be noted that AET increases with increasing datasize. The graph pattern is almost similar for all datasets that shows the stability of the proposed algorithm. AET reduces at the cost of

increasing worker nodes but this behaviors of PDWA is more significant for large datasets as compared to small input datasets.

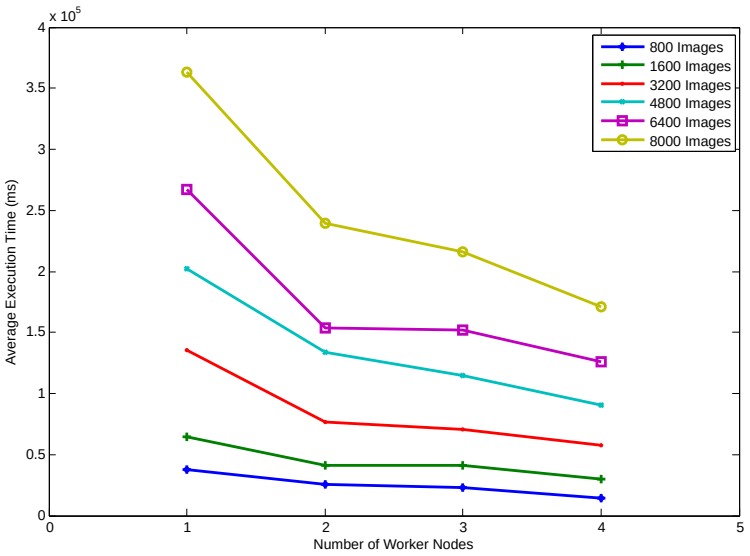


Figure 5.6: Average execution time of PDWA for execution nodes.

Speedup in the execution of EURExpressII using PDWA implemented in STORM is presented in Fig. 5.7. For each plot it is observed that there is an abrupt rise in speedup for 2 and 4 worker nodes, which shows the high parallelism in workflow execution when workflow is executed on 2 and 4 worker nodes. It is caused due to the four (even number of) data pipelines as shown in Fig. 5.5. The results of speedup is not significant when the workflow is executed on 3 worker nodes.

Fig. 5.8 shows comparative results of AET achieved by PDWA and STORM default scheduler. AET increases with the increase in datasize as the big amount of data needs more time to process or compute. PDWA outperformed STORM default scheduler. It is significant for 2 and 3 worker nodes as shown in Fig. 5.8a and Fig. 5.8b respectively, while less significant improvement as compared to plot of 4 worker nodes. The plot of benchmark is AET when all tasks are executed sequentially one single (local) node. It is obvious from Fig. 5.8 that PDWA shows considerable improvement as compared to the benchmark. The percentage improvement of PDWA over STORM default scheduler and

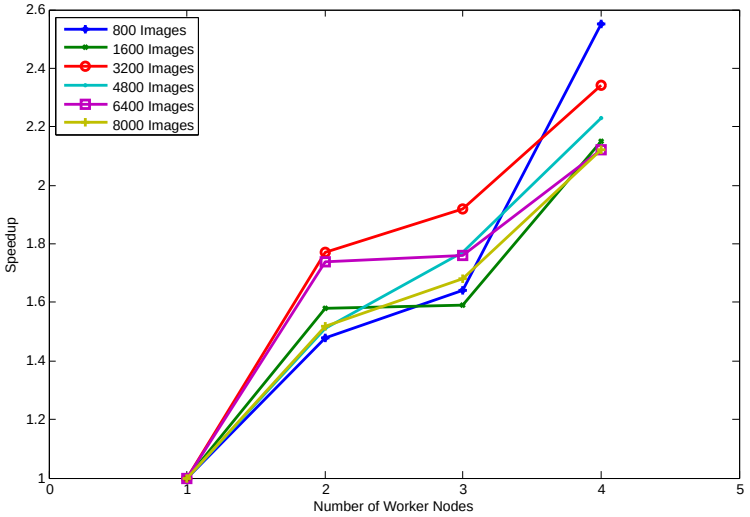


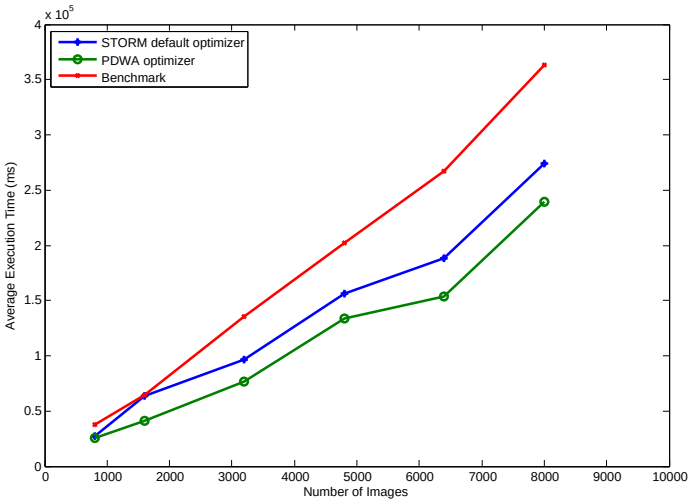
Figure 5.7: Speedup of PDWA for execution nodes.

Table 5.2: The percentage improvement of PDWA.

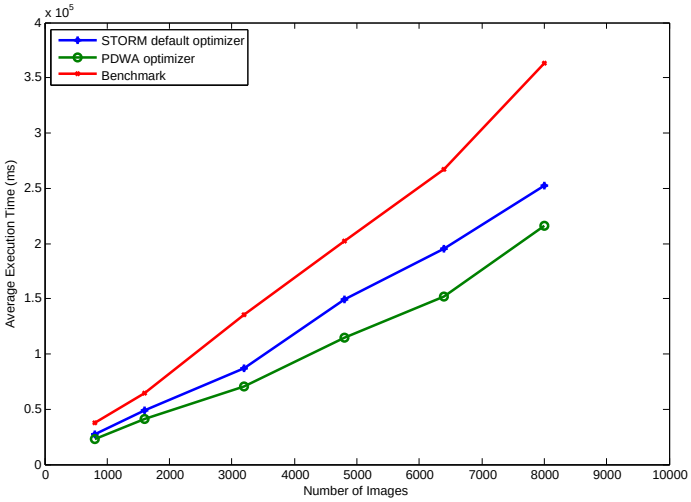
Images	Worker nodes	SDS	Benchmark
800	2	7.77%	32.37%
8000	2	12.57%	34.04%
800	3	16.59%	53.01%
8000	3	14.37%	40.56%
800	4	20.12%	60.86%
8000	4	13.67%	52.9%

benchmark for 800 and 8000 input images are tabulated in Table 5.2. Table shows that the percentage improvement of PDWA as compared to STORM default scheduler and benchmark. PDWA shows significant improvement in AET when workflow was executed on 4 worker nodes. The suitable partitions produced by PDWA due to the structure of EURExpressII causes considerable improvement in AET.

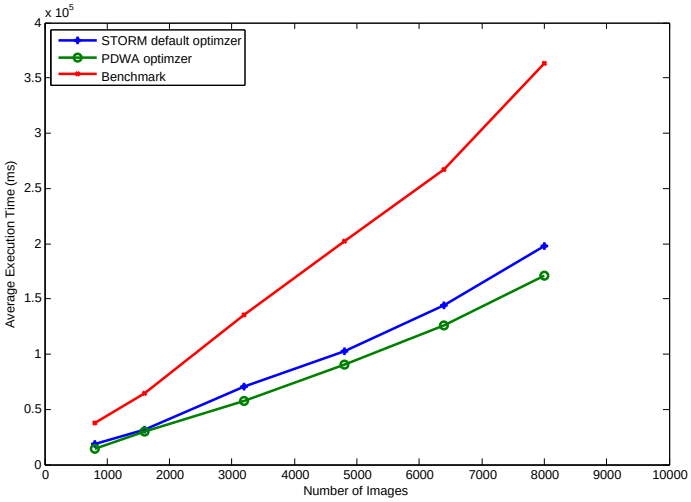
Performance metric that is used for comparison is efficiency that is defined in Eq. 5.3. The results of efficiency for 2, 3, and 4 worker nodes are presented in Fig. 5.9a, Fig. 5.9b, and Fig. 5.9c respectively. The bar charts shows that PDWA is more efficient as compared to STORM default scheduler. As shown in Fig. 5.9 PDWA outperformed for all sizes of datasets and shows significant improvement of 25% for 800 input images for 2 worker nodes, however for 8000 nodes the improvement in the efficiency over bench-



(a) Worker nodes = 2.

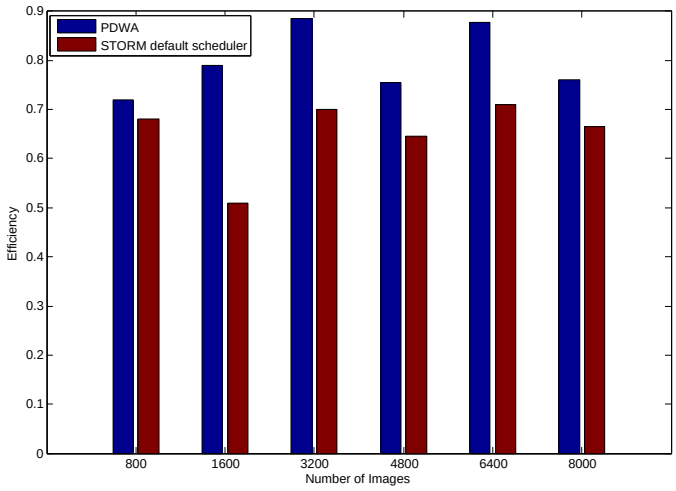


(b) Worker nodes = 3.

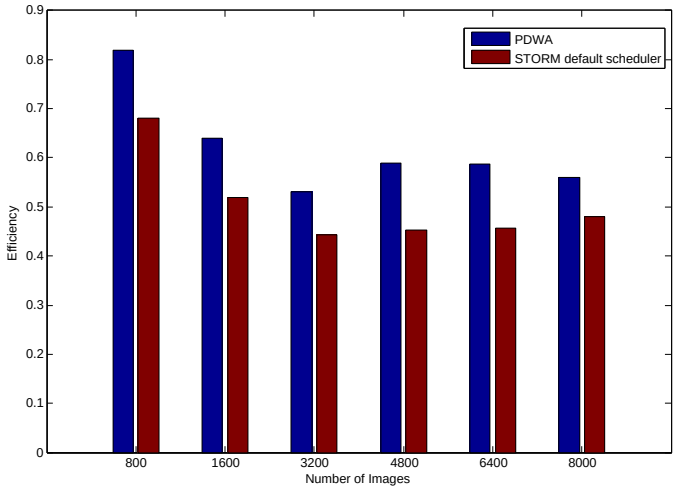


(c) Worker nodes = 4.

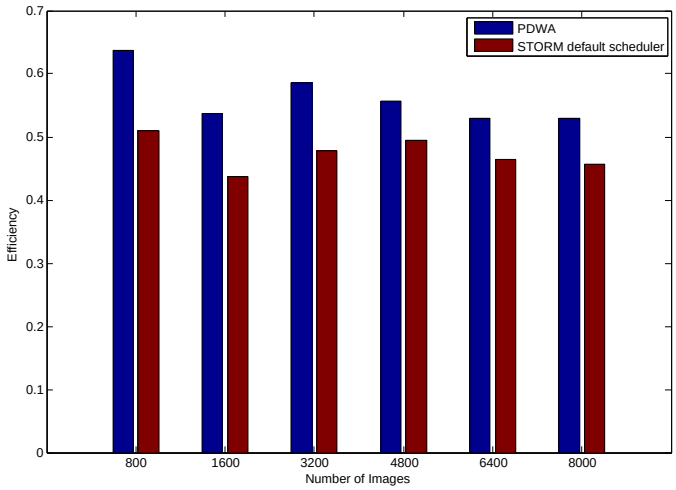
Figure 5.8: Comparative AET of PDWA and STORM default scheduler for increasing datasets.



(a) Worker nodes = 2.



(b) Worker nodes = 3.



(c) Worker nodes = 4.

Figure 5.9: Comparative Efficiency of PDWA and STORM default scheduler.

mark is 15.82%. Similarly, PDWA shows improvement in efficiency while processing 800 and 8000 images over STORM default scheduler and benchmark for 3 worker nodes is 20.58% and 16.66% respectively. For the similar number of images PDWA efficiency is 5.88% higher than STORM default scheduler and 14.29% from benchmark. PDWA is proved to be better than STORM default scheduler due to reduced AET and enhanced efficiency.

5.7 Conclusion

In this chapter, a STORM based computing environment is developed to analyze the performance of our proposed algorithm PDWA. In previous chapter the performance of the proposed algorithm was analyzed through simulations but in this chapter the algorithm is implemented and tested in real world framework. The experiments were performed with workflow of EURExpressII workflow, a use case derived from real world application. The performance of PDWA implemented in STORM was investigated with varying range of dataset size and on different number of execution node. The execution performance was measured in average execution times and speed up of the workflow. The results show that PDWA outperforms to the significant extent as compared to the STORM default scheduler. In future, we aim to extend these experiments to be performed with comparatively complex and more data-intensive workflows. Moreover, we also plan to observe the execution behavior of PDWA in heterogeneous execution node with different computing capacities.

CHAPTER 6

CONCLUSION AND FUTURE WORK

This chapter concludes the thesis by summarizing what we have done in this research work in Section 6.1. It then suggests possible future research direction in Section 6.2.

6.1 Summary and contributions to knowledge

The exponential rise in the growth of data has exposed new challenges to the scientific communities. The complexity of scientific experiments and the heterogeneity of computing environments have opened a vast research area to work in. In this thesis data-intensive scientific applications (modeled as workflows) are focused for the performance optimization. This research is just like a drop added in the big ocean of data-intensive research.

We have presented a overview of complex scientific workflows and how data deluge has changed the research directions. Chapter 2 presented workflow optimization by covering related work regarding workflow classifications, scheduling and performance indicators. Literature review of workflow optimization algorithms *i.e.* heuristics and meta-heuristics specifically genetic algorithms and hybrid approaches were reviewed in Chapter 3, Section 3.2. Then we narrow down the research to data-intensive workflow optimization. We adapted Stream based Data Processing Model (SDPM) to enhance the performance of data-intensive workflows. SDPM was discussed in Section 2.5 and related optimization algorithms are reviewed in Chapter 4, Section 4.2. Finally, real-time stream processing platform (STORM) and other relevant frameworks were reviewed in Chapter 5.

In Chapter 3 the proposed a Hybrid Genetic Algorithm (HGA) for workflow optimization in heterogeneous computing environments was described. It is an evolution

based approach that reduces the execution time (Schedule length) of workflows. Major features of HGA are *a*) Hybrid algorithm (combination of a heuristic and genetic algorithm), *b*) Modified genetic operator (crossover & mutation), and *c*) Load balancing across execution nodes. Simulation results show that HGA outperforms number of state of the art algorithms for verity of input workflows. The performance of the proposed algorithm was evaluated with synthesized workflows as well as workflows derived from real-world applications (Montage, Cybershake, Gaussian elimination). The overall performance proved that HGA generate schedules with lower schedule lengths and load is also balanced among resources.

The core part of the thesis lies in Chapter 4. It presented data-intensive optimization algorithms (PDWA, I-PDWA). This chapter mainly focused on data-intensive workflow optimization and data streaming model. The algorithm PDWA optimizes data-intensive workflows in terms of reduced latency and enhanced throughput. Salient feature of the algorithm include partitioning of workflow in such a way that inter-partition data movement is minimum. PDWA overcomes the communication overhead caused by the heavy data movement between execution nodes. After suitable partitions each partition is mapped on that execution node that offer minimum execution time for that particular partition, it reduces the latency. I-PDWA is an enhanced version of PDWA in which an additional feature of data parallelism is applied to the most compute intensive task in each partition. This added feature further improved the performance of PDWA. Simulation results proved that PDWA is better than non-partitioning algorithm and I-PDWA performed even better than PDWA and other state of the art streaming algorithms. The simulations were carried out with verity of datasets both synthesized and real-world based workflows.

In Chapter 5, we have implemented PDWA in real-time stream processing system (STORM) and experiments were performed with a use case workflow (EURExpressII). The experiments were performed with different input data sizes and number of execu-

tion nodes. A homogeneous execution environment was developed in Openstack. The results show that PDWA outperformed STORM default scheduler. Finally, in this chapter we conclude the dissertation by summarizing it and by listing our contributions and suggesting future research directions.

The contributions of this dissertation are summarized below:

- a comprehensive literature review on workflows, data-intensive workflows and optimization algorithms,
- a Hybrid Genetic Algorithm (HGA) to optimize workflow execution,
- a Partition based Data-intensive Workflow optimization Algorithm (PDWA) to optimize data-intensive workflows using stream based data processing model,
- an Improved PDWA (I-PDWA) that further improved the performance of PDWA,
- a demonstration how PDWA performs with real world data-intensive workflow in real stream processing framework.

6.2 Future Directions

In the course of the work, we have come across many potential research directions that can be build on the work in this thesis.

- This research mainly focus the time based matrices that is latency, execution time, and throughput, however energy optimization is crucial issue. The data centers consume huge amount of power. The total energy consumed by data centers all over the world is equal to the energy consumed by Czech Republic. Therefore, the carbon foot prints on the environment is increasing. Energy efficient scheduling algorithms and policies are required to be developed to address these issues. It must be focused how the performance of the data centers and energy consumption is compromised

while scheduling decisions. The performance of the data centers depends on the usage of the hardware devices by the virtual machine management software depending on the user needs. When more CPUs are used the hardware temperature increases. Due to this increased temperature the performance and hardware might damage, therefore cooling of the data centers is required. Thus, the energy consumption is directly related with the performance of data centers. Most of the cost of operation of data centers is the energy consumed. The energy efficient scheduling algorithms must be adopted to save energy and cooling cost of data centers.

- Scheduling sensitive data-intensive applications, *e.g.* banking data require secured data scheduling and processing. The security of sensitive information is crucial in scheduling applications. The flow of private data and its storage must be ensured in order to built/keep trust of the users. Both data and computations are susceptible to attacks resulting from any intruder. In addition, when carried out on shared resources the data security become a vital issue. Thus, the possibility of exposure and sharing of data on shared resources. These security issues need to be taken into account while scheduling data and tasks as well as mapping them onto resources.

- In this research, we have focused static scheduling in which complete information of resources and tasks are available before scheduling. As discussed in Chapter 2, there are two optimization phases, mapping and execution. This research is the optimization of mapping phase, however run-time optimization can also be desirable where live data streams are processed. Dynamic resources and tasks are managed and optimized by online scheduling. Dynamic scheduling is faster as it schedules on the fly, which is carried out at execution phase of workflow life cycle. In computing environments, like cloud where resources are not fixed in number, dynamic scheduling is effect in such scenarios.

- In this thesis we used the concept of stream-based data processing model, which is usually used for archived data. In static scheduling, archived data from a database can be used as a stream of data because a single SQL query can invoke a chain of data items for instance images from the database. These set of concepts are used in this research, however in future this work can be modified to handle real time live data stream. Run time scheduling algorithms can be designed to serve the purpose that must have high execution/processing speed that need to be synchronized with the data arrival rate. In case of difference in the data processing speed and its arrival rate, data can be lost. There are numerous examples of real time data generation for instance data from sensors, geo-spatial services, information of social networks, e commerce purchases so on and so forth.

- In this thesis, STORM is used a real time framework for the evaluation of proposed algorithm in real-time environment. Recently, few new streaming framework have been developed for instance Flink and Heron that can also be studied and used for analyzing the behavior of proposed algorithm. Algorithm must be transformed and make it compatible for these frameworks to be used as a plugin. EURExpress II is used as a use case for the workload to evaluate the proposed algorithm. Other data-intensive workflows can also be investigated to use as the test workload, for instance workflows from other scientific areas, *e.g.* seismology, life sciences and medicine, and banking.

REFERENCES

Abdulal, W., Jabas, A., Ramachandram, S., & Jadaan, O. A. (2012). Grid computing - technology and applications, widespread coverage and new horizons. In D. S. Maad (Ed.), (p. 89-110). InTech.

Agarwal, A., & Kumar, P. (2009). Economical duplication based task scheduling for heterogeneous and homogeneous computing systems. In *Ieee international advance computing conference, 2009.*

Agarwalla, B., Ahmed, N., Hilley, D., & Ramachandran, U. (2007). Streamline: A scheduling heuristic for streaming applications on the grid. *Multimedia Systems.*, *13*, 69-85.

Agrawal, K., Benoit, A., Dufosse, F., & Robert, Y. (2009). *Mapping filtering streaming applications with communication costs.* (Tech. Rep.). Massachusetts Institute of Technology, USA.

Agrawal, K., Benoit, A., Magnan, L., & Robert, Y. (2010). Scheduling algorithms for linear workflow optimization. In *International symposium on parallel & distributed processing (ipdps).* (p. 1-12).

Ahmad, S. G., Liew, C. S., Rafique, M. M., Munir, E. U., & Khan, S. U. (2014). Data-intensive workflow optimization based on application task graph partitioning in heterogeneous computing systems. In *Fourth international conference on big data and cloud computing.* (p. 129-136).

Ahmad, S. G., Munir, E. U., & Nisar, M. W. (2012). PEGA: A performance effective genetic algorithm for task scheduling in heterogeneous systems. In *Ieee 14th inter-*

national conference on high performance computing and communications (p. 1082-1087).

Aniello, L., Baldoni, R., & Querzoni, L. (2013). Adaptive online scheduling in storm. In *7th acm international conference on distributed event-based systems* (p. 207-218).

Arabnejad, H., & Barbosa, J. G. (2014a). A budget constrained scheduling algorithm for workflow applications. *Journal of Grid Computing*, *12*(4), 665-679.

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

Atkinson, M., Liew, C. S., Galea, M., Martin, P., Krause, A., Mouat, A., ... Snelling, D. (2012). Data-intensive architecture for scientific knowledge discovery. *Distributed And Parallel Databases*, *30*(5-6), 307-324.

Awad, A., El-Hefnawy, N., & Abdel-kader., H. (2015). Enhanced particle swarm optimization for task scheduling in cloud computing environments. *Procedia Computer Science*, *65*, 920-929.

Bansal, S., Kothari, B., & Hota, C. (2011). Dynamic task-scheduling in grid computing using prioritized round robin algorithm. *International Journal of Computer Science Issues.*, *8*, 472-477.

Barga, R., Jackson, J., Araujo, N., Guo, D., Gautam, N., & Simmhan, Y. (2008). The Trident Scientific Workflow Workbench. In *Ieee international conference on escience* (p. 317 - 318). doi: 10.1109/eScience.2008.126

Benoit, A., Nicod, J.-M., & Rehn-Sonigo, V. (2014). Optimizing buffer sizes for pipeline workflow scheduling with setup times. In *Ieee international parallel & distributed processing symposium workshops (ipdpsw)*. IEEE. doi: 10.1109/IPDPSW.2014.77

Berriman, G. B., & Groom, S. L. (2011, December). How will astronomy archives survive the data tsunami? *Communications of the ACM*, *54*(12), 52-56. Retrieved from `http://doi.acm.org/10.1145/2043174.2043190` doi: 10.1145/2043174.2043190

Berthold, M. R., Cebron, N., Dill, F., Gabriel, T. R., Kötter, T., Meinl, T., ... Wiswedel, B. (2009). *KNIME-The Konstanz Information Miner Version 2.0 and Beyond.* ACM SIGKDD Explorations Newsletter 11:26-31.

Bhat, V., Parashar, M., & Klasky, S. (2007). Experiments with in-transit processing for data intensive grid workflows. In *Proceedings of the 8th ieee/acm international conference on grid computing* (p. 193-200). Washington, DC, USA.

Bhat, V., Parashar, M., Liu, H., Kandasamy, N., Khandekar, M., Klasky, S., & Abdelwahed, S. (2007). A self-managing wide-area data streaming service. *Cluster Computing*, *10*(4), 365-383.

Blythe, J., Jain, S., Deelman, E., Gil, Y., Vahi, K., Mandal, A., & Kennedy, K. (2005). Task scheduling strategies for workflow-based applications in grids. In *Proceedings of the fifth ieee international symposium on cluster computing and the grid (ccgrid)* (Vol. 2, p. 759-767). Washington, DC, USA.

Brandic, I., Pllana, S., & Benkner, S. (2006). An approach for the high-level specification of qos-aware grid workflows considering location affinity. *Scientific Programming*, *14*(2,3), 231-250.

Braun, T. D., Siegel, H. J., Beck, N., Boloni, L. L., Maheswaran, M., Reuther, A. I., ... Freund, R. F. (2001). A comparison of eleven static heuristics for mapping a class

of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing.*, *61*, 810-837.

BrykMaciej, P., Malawski, Juve, G., & Deelman, E. (2016). Storage-aware algorithms for scheduling of workflow ensembles in clouds. *Journal of Grid Computing*, *14*(2), 359-378.

Collins, J. P. (2009). *The fourth paradigm: Data-intensive scientific discovery.* (T. Hey, S. Tansley, & K. Tolle, Eds.). Miscosoft.

Couvares, P., Kosar, T., Roy, A., Weber, J., & Wenger, K. (2007). Workflow management in condor. In I. J. Taylor, E. Deelman, D. B. Gannon, & M. Shields (Eds.), *Workflows for e-science.* (p. 357-375). Springer London.

Daoud, M. I., & Kharma, N. (2008a). A high performance algorithm for static task scheduling in heterogeneous distributed computing systems. *J. Parallel Distrib. Comput.*, *68*(4), 399-409.

Daoud, M. I., & Kharma, N. N. (2008b). A high performance algorithm for static task scheduling in heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, *68*(4), 399-409.

Daoud, M. I., & Kharma, N. N. (2011). A hybrid heuristic-genetic algorithm for task scheduling in heterogeneous processor networks. *J. Parallel Distrib. Comput.*, *71*(11), 1518-1531.

DaweiSun, Zhang, G., Yang, S., Zheng, W., Khan, S. U., & Li., K. (2015). Re-Stream: Real-time and energy-efficient resource scheduling in big data stream computing environments. *Information Sciences*, *319*, 92-112.

Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. *Communications of the ACM - 50th anniversary issue*, *51*(1), 107-113.

de Carvalho, E. C. A., Jayanti, M. K., Batilana, A. P., Kozan, A. M. O., Rodrigues, M. J., Shah, J., ... Pietrobon, R. (2010). Standardizing clinical trials workflow representation in uml for international site comparison. *Public Library of Science, One (PLOS One)*, *11*(5), 1-8.

Deelman, E., Gannon, D., Shields, M., & Taylor, I. (2009). Workflows and e-Science: An overview of workflow system features and capabilities. *Future Gener. Comput. Syst.*, *25*(5), 528 - 540.

Deelman, E., Juve, K. V. G., Rynge, M., Callaghan, S., Maechling, P. J., Mayani, R., ... Wenger, K. (2015). Pegasus, a workflow management system for science automation. *Future Generation Computer Systems*, *46*(C), 17-35.

Deelman, E., Singh, G., Su, M.-H., Blythe, J., Gil, Y., Kesselman, C., ... Katz, D. S. (2005, July). Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Sci. Program.*, *13*(3), 219–237. Retrieved from `http://dl.acm.org/citation.cfm?id=1239649.1239653`

der Wijngaart, R. F. V., & Frumkin, M. (2002). *NAS Grid Benchmarks Version 1.0* (Tech. Rep.). NASA Advanced Supercomputing (NAS) Division, NASA Ames Reserach Center, Moffett Field, CA 94035-1000.

Evans, R. (2015, 9-13 March 2015). Apache storm, a hands on tutorial. In *Ieee international conference on cloud engineering (ic2e).* IEEE.

Fisher, R. A. (1936, September). The use of multiple measurements in taxonomic problems. *Annals of Human Genetics*, *7*(2), 179-188.

Givelberg, E., Szalay, A., Kanov, K., & Burns, R. (2011). An Architecture for a Data Intensive Computer. In *Proceedings of the first international workshop on Network-aware data management* (p. 57-64). ACM New York, NY, USA.

Glatard, T., Montagnat, J., Lingrand, D., & Pennec, X. (2008). Flexible and efficient workflow deployment of data-intensive applications on grids with moteur. *International Journal of High Performance Computing Applications*, *22*(3), 347-360.

Glatard, T., Sipos, G., Montagnat, J., Farkas, Z., & Kacsuk, P. (2007). Workflows for e-science: Scientific workflows for grids. In I. J. Taylor, E. Deelman, D. B. Gannon, & M. Shields (Eds.), (p. 279-299). Springer London.

Gu, Y., Shenq, S.-L., Wu, Q., & Dasgupta, D. (2012). On a multi-objective evolutionary algorithm for optimizing end-to-end performance of scientific workflows in distributed environments. In *Proceedings of the 45th annual simulation symposium.*

Gu, Y., & Wu, Q. (2010). Maximizing workflow throughput for streaming applications in distributed environments. In *19th international conference on computer communications and networks (icccn).*

Guirado, F., Roig, C., & Ripoll, A. (2013). Enhancing throughput for streaming applications running on cluster systems. *Journal of Parallel and Distributed Computing*, *73*(8), 1092-1105.

Gummaraju, J., Coburn, J., Turner, Y., & Rosenblum, M. (2008). Streamware:programming general-purpose multicore processors using streams. In *13th international conference on architectural support for programming languages and operating systems, asplos xiii* (p. 297-307).

Guo-ning, G., Ting-lei, H., & Shuai, G. (2010a). Genetic simulated annealing algorithm for task scheduling based on cloud computing environment. In *International conference on intelligent computing and integrated systems (iciss).*

Guo-ning, G., Ting-lei, H., & Shuai, G. (2010b). Genetic simulated annealing algorithm for task scheduling based on cloud computing environment. In *International conference on intelligent computing and integrated systems (iciss)* (p. 60-63).

Hackett, A., Ajwani, D., Ali, S., Kirkland, S., & Morrison, J. P. (2013). A Network Configuration Algorithm Based on Optimization of Kirchhoff Index. In *Ieee 27th international symposium on parallel and distributed processing* (p. 407-417).

Han, L., Liew, C. S., Atkinson, M. P., & van Hemert, J. (2011). A generic parallel processing model for facilitating data mining and integration. *Parallel Computing*, *37*(3), 157-171.

Han, L., van Hemert, J. I., & Baldock, R. A. (2011). Automatically identifying and annotating mouse embryo gene expression patterns. *Bioinformatics*, *27*(8), 1101-1107.

Hirales-Carbajal, A., AndreiTchernykh, Yahyapour, R., Luis, J., Gonzalez-Garcia, Roblitz, T., & Ramirez-Alcaraz, J. M. (2012). Multiple workflow scheduling strategies with user run time estimates on a grid. *J Grid Computing*, *10*, 325-346.

Hoheisel, A., & Alt, M. (2007). Workflows for e-science: Scientific workflows for grids. In I. J. Taylor, E. Deelman, D. B. Gannon, & M. Shields (Eds.), (p. 190-207). Springer London.

Ilavarasan, E., Thambidurai, P., & Mahilmannan, R. (2005). Performance effective task scheduling algorithm for heterogeneous computing system. In *Proceedings of the 4th ieee international symposium on parallel and distributed computing.* (p. 28-38).

Issa, S. A., Kienzler, R., El-Kalioby, M., Tonellato, P. J., Wall, D., & Abouelhoda, R. B. M. (2013). Streaming support for data intensive cloud-based sequence analysis. *BioMed Research International.*, *2013*, 1-16.

Jung, E.-S., & Kettimuthu, R. (2013). An Overview of Parallelism Exploitation and Cross-layer Optimization for Big Data Transfers. In *28th ieee international parallel & distributed processing symposium* (p. 1-6).

Juve, G., Chervenak, A., Deelman, E., Bharathi, S., Mehta, G., & Vahi, K. (2013). Characterizing and profiling scientific workflows. *Future Gener. Comput. Syst.*, *29*(3), 682 - 692.

Kaisler, S., Armour, F., Espinosa, J. A., & Money, W. (2013). Big data: Issues and challenges moving forward. In *International conference on system sciences (hicss)* (p. 995-1004).

Khailany, B., Dally, W. J., Kapasi, U. J., Mattson, P., Namkoong, J., Owens, J. D., ... Rixner, S. (2001, March-April). Imagine: media processing with streams. *IEEE Micro*, *21*(2), 35-46.

Khan, M. (2012). Scheduling for heterogeneous systems using constrained critical paths. *Parallel Computing.*, *38*, 175-193.

Kim, S., & Browne, J. (1988). A general approach to mapping of parallel comutations upon multiprocessor architectures. In *International conference of parallel processing* (Vol. 2, p. 1-8).

Ko, S. Y., Morales, R., & Gupta, I. (2007). New worker-centric scheduling strategies for data-intensive grid applications. In R. Cerqueira & R. H. Campbell (Eds.), (p. 121-142). Springer.

Kosar, T., & Livny, M. (2004). Stork: Making data placement a first class citizen in the grid. In *Proceedings of the 24th international conference on distributed computing systems* (p. 342-349). Washington, DC, USA.

Laszewski, G. V., Hategan, M., & Kodeboyina, D. (2007). Workflows for e-science: Scientific workflows for grids. In I. J. Taylor, E. Deelman, D. B. Gannon, & M. Shields (Eds.), (p. 340-356). Springer London.

Lee, Y. C., Wangb, C., Zomaya, A. Y., & Zhou, B. B. (2012). Profit-driven scheduling for cloud services with data access awareness. *J. Parallel Distrib. Comput.*, *72*, 591-602.

LIEW, C. S. (2012). *Optimisation of the enactment of fine-grained distributed data-intensive workflows*. Edinburgh: The University of Edinburgh.

LIEW, C. S., ATKINSON, M. P., GALEA, M., ANG, T. F., MARTIN, P., & HEMERT, J. I. V. (2016). Scientific workflows: Moving across paradigms. *ACM Computing Surveys*.

Liew, C. S., Atkinson, M. P., van Hemert, J. I., & Han., L. (2010). Towards optimising distributed data streaming graphs using parallel streams. In *High-performance parallel and distributed computing.* (p. 725-736).

LIEW, C. S., Krause, A., & Snelling, D. (2013). The Data Bonanza: Improving Knowledge Discovery in Science, Engineering, and Business. In M. Atkinson et al. (Eds.), (p. 251-268). Wiley- IEEE Computer Society Press.

Liu, J., Chen, L., Dun, Y., Liu, L., & Dong, G. (2008). The research of ant colony and genetic algorithm in grid task scheduling. In *International conference on multimedia and information technology.*

Liu, L., & Ozsu., M. T. (2009). Encyclopedia of database systems. *Springer US*, 638-638.

Llora, X., Acs, B., Auvil, L. S., Capitanu, B., Welge, M. E., & Goldberg, D. E. (2008). Meandre: Semantic-driven data-intensive flows in the clouds. In *Fourth ieee international conference on escience.*

Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., ... Zhao, Y. (2006). Scientific Workflow Management and the KEPLER System. *Concurrency and Computation: Practice and Experience*, *18*(10), 1039-1065.

Luo, H., Yan, C., & Hu, Z. (2015). An enhanced workflow scheduling strategy for deadline guarantee on hybrid grid/cloud infrastructure. *Journal of Applied Science and Engineering*, *18*(1), 67-78. doi: 10.6180/jase.2015.18.1.09

Meyer, L., Annis, J., Wilde, M., Mattoso, M., & Foster, I. (2006). Planning spatial workflows to optimize grid performance. In *Proceedings of the 2006 acm symposium on applied computing. new york, ny, usa* (p. 786-790).

Munir, E. U., Mohsin, S., Hussain, A., Nisar, M. W., & Ali, S. (2013). SDBATS: A novel algorithm for task scheduling in heterogeneous computing systems. In *Ieee 27th international parallel and distributed processing symposium workshops phd forum (ipdpsw)* (p. 43-53).

Muthukrishnan, S. (2005). Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, *1*(2), 117-236.

Omara, F. A., & Arafa, M. M. (2010). Genetic algorithms for task scheduling problem. *Journal of Parallel and Distributed Computing.*, *70*, 13-22.

Pandey, S., Voorsluys, W., Rahman, M., Buyya, R., Dobson, J. E., & Chiu, K. (2009). A grid workflow environment for brain imaging analysis on distributed systems. *Concurrency and Computation: Practice & Experience*, *21*(16), 2118-2139.

Park, S., & Humphrey, M. (2008). Data throttling for data-intensive workflows. In *Proceedings of the 22nd international parallel and distributed processing symposium* (p. 1-11).

Prajapati, H. B., & Shah, V. A. (2014). Bandwidth-aware scheduling of workflow application on multiple grid sites. *Journal of Computer Networks and Communications*, *vol. 2014*, 15 pages. (Article ID 529835) doi: 10.1155/2014/529835

Ramakrishnan, A., Singh, G., Zhao, H., Deelman, E., Sakellariou, R., Vahi, K., ... Samidi, M. (2007). Scheduling data-intensive workflows onto storage-constrained distributed resources. In *Proceedings of the seventh ieee international symposium on cluster computing and the grid* (p. 401-409). Washington, DC, USA.

Reimann, P., Schwarz, H., & Mitschang, B. (2011). Design, Implementation, and Evaluation of a Tight Integration of Database and Workflow Engines. *Journal of Information and Data Management*, *3*(2), 353-368.

Russell, N., ter Hofstede, A., Edmond, D., & van der Aalst, W. (2005). Workflow data patterns: Identification, representation and tool support. In *Proceedings of the 24th international conference on conceptual modeling (er 2005)* (p. 353-368). Berlin, Germany: Springer-Verlag.

Russell, N., ter Hofstede, A., van der Aalst, W., & Mulyar., N. (2006). *Workflow control-flow patterns : A revised view* (Tech. Rep.). BPM Center Report BPM. Retrieved from `BPMcenter.org`

Rychly, M., Skdo, P., & Smrz, P. (2014). Scheduling decisions in stream processing on heterogeneous clusters. In *International conference on complex, intelligent and software intensive systems (cisis)* (p. 614-619).

Salimi, R., Motameni, H., & Omranpour, H. (2014). Task scheduling using NSGA II with fuzzy adaptive operators for computational grids. *J. Parallel Distrib. Comput.*, *74*(5), 2333-2350.

Samriti, Gill, S., Bharadwaj, A., Singh, N., Singh, H., & Singh, J. (2012). Analysis of HLFET and MCP Task Scheduling Algorithms. *International Journal of Modern Engineering Research.*, *2*(3), 1176-1180.

Shibata, T., Choi, S., & Taura, K. (2010). File-access patterns of data-intensive workflow applications and their implications to distributed filesystems. In *Proceedings of the 19th acm international symposium on high performance distributed computing* (p. 746-755).

Shields, M. (2007). Workflows for e-science: Scientific workflows for grids. In I. J. Taylor, E. Deelman, D. B. Gannon, & M. Shields (Eds.), (p. 167-173). Springer London.

Singh, G., Kesselman, C., & Deelman, E. (2005). Optimizing grid-based workflow execution. *Journal of Grid Computing*, *3*(3-4), 201-219.

Singh, G., Vahi, K., Ramakrishnan, A., Mehta, G., Deelman, E., Zhao, H., ... S.Katz, D. (2007). Optimizing workflow data footprint. *Scientific Programming*, *15*(4), 249-268.

Singh, L., & Singh, S. (2013). A survey of workflow scheduling algorithms and research issues. *International Journal of Computer Applications*, *75*(15), 21-28.

Slominski, A. (2007). Workflows for e-science: Scientific workflows for grids. In I. J. Taylor, E. Deelman, D. B. Gannon, & M. Shields. (Eds.), (p. 208-226). Springer.

Solaimani, M., Richardson, Khan, L., & Thuraisingham, B. (2014). Real-time anomaly detection over vmware performance data using storm. In *Ieee 15th international conference on information reuse and integration (iri).* (p. 458-465).

Srikanth, G. U., Maheswari, V. U., Shanthi, P., & Siromoney, A. (2012). Tasks scheduling using ant colony optimization. *Journal of Computer Science*, *8*, 1314-1320.

Taheria, J., Lee, Y. C., Zomaya, A. Y., & Siegel., H. J. (2013). A Bee Colony based optimization approach for simultaneous job scheduling and data replication in grid environments. *Comput. & Oper. Res.*, *40*(6), 1564-1578.

Taylor, I., Shields, M., Wang, I., & Harrison, A. (2007). Workflows for e-science: Scientific workflows for grids. In I. J. Taylor, E. Deelman, D. B. Gannon, & M. Shields (Eds.), (p. 320-339). Springer London.

Taylor, I. J., Deelman, E., Gannon, D. B., & Shields, M. (2007). *Workflows for e-science: Scientific workflows for grids* (I. J. Taylor, E. Deelman, D. B. Gannon, & M. Shields, Eds.). Springer London.

Topcuoglu, H., Hariri, S., & Min-You. (2002). Performance-effective and low complexity task scheduling for heterogeneous computing. *IEEE Trans. on Parallel and Distrib. Syst.*, *13*(3), 260-274.

Topcuoglu, H., & Wu, M.-Y. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems.*, *13*, 260-274.

Tsai, Y.-L., Huang, K.-C., Chang, H.-Y., Ko, J., Wang, E. T., & Hsu, C.-H. (2012). Scheduling multiple scientific and engineering workflows through task clustering and best-fit allocation. In *Ieee eighth world congress on services.*

van der Aalst, W., & ter Hofstede., A. (2005). YAWL: yet another workflow language. *Journal Information Systems*, *30*(4), 245-275.

Vydyanathan, N., Catalyurek, U., Kurc, T., Sadayappan, P., & Saltz, J. (2011). Optimizing latency and throughput of application workflows on clusters. *Parallel Computing*, *37*, 694-712.

Wang, C., Gu, J., Wang, Y., & Zhao, T. (2012). A hybrid heuristic-genetic algorithm for task scheduling in heterogeneous multi-core system. In Y. Xiang, I. Stojmenovic, B. Apduhan, G. Wang, K. Nakano, & A. Zomaya (Eds.), *Algorithms and architectures for parallel processing* (Vol. 7439, p. 153-170). Springer Berlin Heidelberg.

Wang, J., Crawl, D., & llkay Altintas. (2009). Kepler+hadoop: A general architecture facilitating data intensive applications in scientific workflow systems.

Wang, K., Qiao, K., Sadooghi, I., Zhou, X., Li, T., Lang, M., & Raicu, I. (2016). Load-balanced and locality-aware scheduling for data-intensive workloads at extreme scales. *Journal Concurrency and Computation: Practice & Experience*, *28*(1), 70-94.

Wang, L., Tao, J., Marten, H., Streit, A., Ranjan, R., Chen, J., & Chen, D. (2013). G-Hadoop: MapReduce across distributed dat ceneters for data-intensive computing. *Future Generation Computer Systems.*, *29*(3), 739-750.

Wen, Z., Cala, J., Watson, P., & Romanovsky, A. (2015). Cost effective, reliable, and secure workflow deployment over federated clouds. In *Ieee 8th international conference on cloud computing.* IEEE. doi: 10.1109/CLOUD.2015.86

Wieczorek, M., Hoheisel, A., & Prodana, R. (2009). Towards a general model of multi criteria workflow scheduling on the grid. *Future Generation Computer Systems.*, *25*, 237-256.

Wieczorek, M., Prodan, R., & Fahringer, T. (2005). Scheduling of scientific workflows in the askalon grid environment. *ACM SIGMOD Record Journal.*, *34*, 56-62.

Wolstencroft, K., Haines, R., Fellows, D., Williams, A., Withers, D., Owen, S., ... Goble, C. (2013). The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. *Nucleic Acids Research 41*, *W1*, W557-W561.

Wu, M., & Gajski, D. (1990). Hypertool: A Programming Aid for Message Passing Systems . *IEEE Trans. on Parallel and Distrib. Syst.*, *3*, 330-343.

Xu, Y., Li, K., He, L., & Truong., T. K. (2013). A DAG Scheduling Scheme on Heterogeneous Computing Systems Using Double Molecular Structure-based Chemical Reaction Optimization. *Journal of Parallel and Distributed Computing.*, *73*(9), 1306-1322.

Xu, Y., Li, K., Hu, J., & Li, K. (2014). A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues. *Inf. Sci.*, *270*, 255 - 287.

Yang, T., & Gerasoulis, A. (1994). DSC: Scheduling Parallel Tasks on an unbounded Number of Processors. *IEEE Trans. on Parallel and Distrib. Syst.*, *5*(9), 951-967.

Yu, J., & Buyya, R. (2005). A taxonomy of scientific workflow systems for grid computing. *ACM SIGMOD*, *34*, 44 - 49.

Yu, J., & Buyya., R. (2005.). A Taxonomy of Workflow Management Systems for Grid Computing. *Journal of Grid Computing.*, *34*(3), 44-49.

Zhang, L., Chen, Y., & Yang, B. (2006). Task scheduling based on pso algorithm in computational grid. In *International conference on intelligent systems design and applications.*

Zhang, Y., Koelbel, C., & Cooper, K. (2008). Cluster-based hybrid scheduling mechanisms for workflow applications on the grid. In *Ieee fourth international conference on escience.*

Zhao, L., Ren, Y., Xiang, Y., & Sakurai, K. (2010). Fault-tolerant scheduling with dynamic number of replicas in heterogeneous systems. In *12th ieee international conference on high performance computing and communications (hpcc).* (p. 434-441). IEEE. doi: 10.1109/HPCC.2010.72

Zhao, S., & Lo, V. (2005). *Result verification and trust-based scheduling in open peer-to-peer cycle sharing systems* (Tech. Rep.). University of Oregon, USA.

Zhao, Y., Hategan, M., Clifford, B., Foster, I., von Laszewski, G., Raicu, I., ... Wilde, M. (2007). Swift: Fast, reliable, loosely coupled parallel computation. In *Ieee scw* (p. 199-206). doi: 10.1109/SERVICES.2007.63