Data-Intensive Workflow Optimization based on Application Task Graph Partitioning in Heterogeneous Computing Systems

Saima Gulzar Ahmad1, Chee Sun Liew1, M. Mustafa Rafique2, Ehsan Ullah Munir3, Samee U. Khan4
1Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur, Malaysia
2IBM Research, Dublin, Ireland
3Department of Computer Science, COMSATS Institute of Information Technology, Wah Cantt, Pakistan
4Department of Electrical and Computer Engineering, North Dakota State University, USA

Abstract—Stream based data processing model is proven to be an established method to optimize data-intensive applications. Data-intensive applications involve movement of huge amount of data between execution nodes that incurs large costs. Data-streaming model improves the execution performance of such applications. In the stream-based data processing model, performance is usually measured by throughput and latency. Optimization of these performance metrics in heterogeneous computing environment becomes more challenging due to the difference in the computing capacity of execution nodes and variations in the data transfer capability of communication links between these nodes. This paper presents a dual objective Partitioning based Data-intensive Workflow optimization Algorithm (PDWA) for heterogeneous computing systems. The proposed PDWA provides significantly reduced latency with 47% improvement in the throughput as compared to the approach when workflows are not partitioned.

Keywords-Workflow optimization; Partitioning task graph; Heterogeneous computing; Stream-data processing.

I. INTRODUCTION

The developments in e-Science has produced several challenges to the scientific community in last few decades. The emerging technologies have provided opportunities to unveil unexplored scientific domains. Research activities, such as, large-scale simulations, extensive experimentations, and sensors networks produce enormous amount of data. Data generation and collection speed has exceeded the data processing speed and the volume of data is continuously increasing. A recent study reports that 1 petabyte (PB) of astronomical data is accessible for public and is growing annually by 0.5 PB [1]. This deluge of data has created big challenges, such as, storage, handling, manipulation, and extracting meaningful information from the produced data.

Scientists are struggling to manage the data-intensive applications, these efforts can be categorized in three different domains. (a) hardware design or architectural improvements [2], [3], (b) new storage and memory management techniques [4], [5], and (c) algorithms to optimize data-intensive applications [6], [7]. The science of workflows has emerged to simplify the complex scientific processes by step-wise representation in the form of workflows [8]. 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 [9]. Stream computing is usually 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. The behavior of each task in such applications constitute the following three stages and repeats for each instance of the data: (i) the task get inputs of the data instances, (ii) processes the data, and (iii) pass 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 paper, we adopt the stream-based data processing model and propose a dual objective Partitioning based Data-intensive Workflow optimization Algorithm (PDWA). 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 [10]–[12] consider homogeneous execution environment for data-intensive optimization without incorporating the system heterogeneity. We show that the proposed approach provides considerably better schedule with lower latency and improved throughput. We validate the proposed algorithm using synthesized and real-world workloads [13], and show the performance advantages of the proposed algorithm.

The rest of the paper is organized as follows. Section II summarizes the relevant literature. Section III defines the research problem. The proposed algorithm is presented in Section IV. The performance analysis is presented in Section V. Finally, Section VI concludes the paper.

II. RELATED WORK

Stream-based workflows have been optimized in various ways. Throughput Constrained Latency Optimization heuristic (TCLO) [12] reduces the latency while satisfying the throughput constraint by using a combination of pipelined, task, and data parallelism. These three parallelism cause communication and computational overheads. Moreover, TCLO generates two schedules, a non-pipelined and pipelined schedule and selects a schedule that meets the throughput requirements. The generation of two schedules causes an increase in the time complexity of algorithm. Non-pipelined schedule is generated by a list-based heuristic that prioritizes tasks based on the descending order of bottom-level [14] values. Generating pipelined schedule is a three phase process. First phase is Satisfying Throughput Heuristic (SSTH) that constructs a schedule to satisfy the throughput constraint; second phase is Processor Reduction Heuristic (PRH) that reduces the number of processors used in the previous phase; and third phase is Latency Minimization Heuristic (LMH) that minimizes the latency. PDWA simultaneously optimizes latency and throughput, hence it has reduced time complexity than TCLO. Two Recently proposed algorithms, Data Parallel Replication Mechanism (DPRM) and Task copy Replication Mechanism (TCRM) [10], exploit data and task parallelism, respectively. In the former methodology data parallelism enhances the throughput and the later reduces latency by replication of tasks. Multiple data and tasks replicas increase the complexity and overhead of the execution nodes. Moreover, the algorithm is designed for coarse-grained applications in which communication costs are negligible and the execution environment is homogeneous while PDWA addresses communication-intensive applications in heterogeneous computing system. An approach for streaming-based application is proposed in [11], that considers network topology graph to optimize its configuration with maximum throughput. A theoretical analogy between loads on communication links and the resistances in an electrical circuit is used to define a metric, Kirchhoff Index (KI), as a proxy for the throughput of the network. The algorithm partitions undirected, bi-directional, weighted graphs of networks to optimize KI, which ultimately provides network topology with improved throughput. This algorithm is not suitable for directed graphs, in addition it is designed for the homogeneous environment. Multiple studies [15]–[18] optimize the latency and throughput of stream-based data on different execution platforms, e.g., grid or cloud. Service-based execution platforms include Quality of Service (QoS) based constraints. Applications that execute in service-based computing environment optimize the QoS parameters constrained. In [19], two sets of dual algorithms are proposed. First pair, B-RATE and B-SWAP, maximizes throughput under budget constraint, and the second pair, TP-RATE and TP-SWAP, minimizes execution cost under throughput constraint. These algorithms do not optimize latency directly. Similarly, in [20], execution cost is optimized for the stream of input data in cloud environment. However, our proposed algorithm optimizes the data-intensive workflows by providing pipelined schedules with significantly reduced latency and improved throughput. The salient feature of partitioning the workflow plays key role in the performance of our proposed algorithm.

III. PROBLEM FORMULATION

Based on the nature of the applications, scientific workflows have different characteristics. Data-intensive workflows have gained a lot of attention of scientific community due to the immense growth of the data as discussed in Section I. This paper addresses the workflow applications in which communication costs are comparatively higher than computation costs. The higher communication costs are associated with the movement of large data among the execution nodes. In this paper, we present an algorithm that optimizes latency and throughput of the data-intensive workflows. The execution environments of these workflows such as, grid, cloud etc., are usually heterogeneous in nature. Therefore, the execution environment is also assumed to be heterogeneous in this paper. We consider the heterogeneity in terms of different computing capacity \( (E_v) \) of the execution nodes, which are fully connected with high speed communication links.

The workflows are modeled as Directed Acyclic Graph (DAG) [21]. 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, \( L \), between them. 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 includes 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.

A. 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 is 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:

\[
TP_{comm}(L_{ij}) = Bw(L_{ij})/ \sum_{e=1}^{n} W(L_{ij}e)
\]  

(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:

\[
TP_{comm} = \min(TP_{comm}(L_{ij}))
\]  

(2)

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

\[
TP_{comp}(u_i) = E_s(u_i)/ \sum_{i=1}^{n} W(u_i)
\]  

(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:

\[
TP_{comp} = \min(TP_{comp}(u_i))
\]  

(4)

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

\[
TP = \min(TP_{comm}, TP_{comp})
\]  

(5)

B. Latency Estimation of Pipelined Schedule

Latency is defined as the time spend 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:

\[
L = t_{d_n} - t_{d_{n-1}}
\]  

(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.

\( \text{Fig. 1: An example DAG and execution environment.} \)

\( \text{Table I: Heterogeneity model of three execution nodes.} \)

<table>
<thead>
<tr>
<th>Network Links</th>
<th>Data Transfer Capability</th>
<th>Execution Nodes</th>
<th>Computing Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_1</td>
<td>10</td>
<td>u_1</td>
<td>7</td>
</tr>
<tr>
<td>l_2</td>
<td>100</td>
<td>u_2</td>
<td>5</td>
</tr>
<tr>
<td>l_3</td>
<td>1000</td>
<td>u_3</td>
<td>3</td>
</tr>
</tbody>
</table>

IV. THE PROPOSED ALGORITHM

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. 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 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. 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 I. 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. 1a are given in Table II.

The pseudocode of the proposed algorithm is presented in Algorithm 1. The DAG is partitioned using the Partition algorithm, shown in Algorithm 2. 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 2. \( N_{max} \) is the maximum possible number of application tasks in a partition, that is determined by the
Table II: Application tasks completion time of DAG shown in Fig. 1a.

<table>
<thead>
<tr>
<th>Tasks (v_i)</th>
<th>CT(u_1)</th>
<th>CT(u_2)</th>
<th>CT(u_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11</td>
<td>15</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>14</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>16</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td>11</td>
<td>19</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>14</td>
<td>13</td>
</tr>
<tr>
<td>8</td>
<td>11</td>
<td>15</td>
<td>10</td>
</tr>
</tbody>
</table>

Algorithm 1: Partitioning based Data-intensive Workflow Optimization Algorithm (PDWA).

**Input:** Given an application task graph \( G(E,V) \).

**Output:** \( S \) of a Near Optimal Solution.
1. Read the graph: let \( E \) is the set of \( e \) edges and \( V \) is the set of \( v \) application tasks;
2. Partition\((E,V)\); // Algorithm 2
3. Mapping\((P)\) Partitions; // Algorithm 3
4. Return optimal solution \( S \)

The optimized partitioned DAG is mapped to the execution nodes by using the method shown in Algorithm 3. 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 \( P_n \).

\[ CI_{u_j}(P_n) = \sum_{v_i \in P_n} CT(v_i) \]  \hspace{1cm} (7)

where, \( v_i, \{1 = 2, \ldots, m\} \), 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. 3a. The schedule of the algorithm without partitions (AWOP) is shown in Fig. 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.

\[ TP_{comp} \] of PDWA for execution nodes \( u_1 \) and \( u_3 \) is 0.155 and 0.066, respectively, which is computed using Eq. 3. \( TP_{comm} \) for PDWA is 27.77. Since only communication link \( l_3 \) is used to transfer data between execution nodes \( u_1 \) and \( u_3 \), the minimum value of \( TP_{comp} \) is 0.066. By using Eq. 5, the overall throughput of the system is 0.066 for PDWA. Similarly, \( TP_{comp} \) of AWOP is 0.33 for node \( u_1, 0.33 \) for \( u_2 \) and 0.052 for \( u_3 \), which gives minimum \( TP_{comp} \) of

Algorithm 2: Workflow Partitioning Algorithm.

**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.
1. Compute the threshold edge weight \( e_{th} = \sum_{i=1}^{n} e_i/n \);
2. \( N_{pmax} = v \times f_p \);
3. Duplicate the entry tasks \( v_e \);
4. while pool of non-partitioned tasks is not empty do
5. let \( e_{ij} \) be the edge weight between task node \( v_i \) & \( v_j \);
6. if \( (e_{ij} > e_{th}) \) then place \( v_i \) & \( v_j \) in partition \( P_k \);
7. delete tasks \( v_i \) & \( v_j \) from the pool of tasks to be partitioned;
8. if \( (N_p = N_{pmax}) \) next partition \( P_{k+1} \);
9. endw
10. The application task graph splits into \( P_n \) partitions;
11. */* Optimization of partitions */*
12. while Inter-partition edge weights \( e_{ij} > e_{th} \) do
13. Shift the task \( v_i \) or \( v_j \) to any other partition such that inter-partition edge weight always less than \( e_{th} \);
14. endw
15. Return \( P_n \) partitions;

Algorithm 3: Partitions Mapping Algorithm.

**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.
1. for \( u_1 \) to \( u_n \) do
2. Compute the \( CI \) of each partition using equation 7;
3. endfor
4. for \( P_1 \) to \( P_n \) do
5. Determine the minimum \( CI \) value among the execution nodes i.e., \( CI_{min}(u_i) \);
6. if \( (u_i \) is Idle) then Map partition \( P \) to \( CI_{min}(u_i) \);
7. endfor
8. Return \( M \) mapped \( P \) partitions;
0.052 for node $u_3$. $TP_{comm}$ of link $l_2$ is 4.16 and $l_3$ is 13.69. Therefore, $TP_{comm}$ is 4.16. Finally, system throughput, $TP$, is 0.066 for PDWA and 0.052 for AWOP, using Eq. 5. PDWA provides 27% higher throughput as compared to AWOP. This example shows that the latency of the proposed algorithm is less with reasonable throughput as compared to AWOP.

V. PERFORMANCE ANALYSIS AND DISCUSSION

In this section, we present an evaluation of the proposed algorithm. Extensive simulations are carried out with the synthesized workflows of different characteristics and Gaussian elimination data-flow graphs [13]. These workflows are produced randomly by workflow generator [13], [14] in batches of 50 workflows. The performance metrics used to compare the results are average latency and throughput of these batches of workflows and the results are compared with AWOP. The advantage of the synthesized workflows is that a wide range of workflows patterns can be generated to analyze the performance. The parameters that control the synthesized workflow characteristics are:

- $\alpha$: It determines the shape of the workflows;
- out-degree: It is the number of successors of a node;
- communication cost to computation cost ratio (CCR): It selects the nature of workflow.

These attributes are discussed in detail in the following section and corresponding simulation results are presented. We conducted simulations with workflows obtained from linear equation system of Gaussian elimination [13], and present the achieved results.

A. Results with Different Sizes of Workflows

We associate the size of a workflow with the number of application nodes in a workflow. We generate workflows with 20, 40, 60, 80, and 100 application nodes. We can simulate the data-intensive synthesized workflows if the values of CCR is selected greater than 1. The communication cost is higher in data-intensive workflows due to large data transfer between execution nodes, therefore, CCR values of greater than one simulates this specific category of workflows. Hence, in our simulations, CCR=10 is fixed to simulate data-intensive workflows. Since the synthesized workflows are generated randomly, therefore, the out-degree of each application node is also selected at random. Similarly, the shape parameter, $\alpha = 1$, is selected for normal shaped workflows. Average latency and throughput of 50 batches
of workflows is observed for both algorithms that partition the graph (PDWA), and the algorithm that does not partition (AWOP). The obtained results are presented in Fig. 4. The comparative results show that PDWA outperforms significantly by providing lower latency and higher throughput. Fig. 4a shows that the average latency of PDWA reduces sharply as compared to AWOP algorithm with increasing number of nodes, which shows that PDWA is highly scalable. Overall, PDWA provides 60% lower average latency for the workflows with 100 node as compared to AWOP. We observe similar results for throughput, as shown in Fig. 4b. Overall, PDWA shows 47% better average throughput as compared to AWOP for 100 nodes workflows. Although the improvement is not the same for all sizes of workflows but overall PDWA consistently shows higher throughput values as compared to AWOP.

B. Results with Different Shapes of Workflows

The parameter $\alpha$ determines the shape of synthesized workflows. If $\alpha = 1$ balanced workflows are produced that are neither two short nor two long. Dense workflows with short length and higher parallelism can be generated by selecting $\alpha >> 1$. Similarly, if $\alpha << 1$ the longer workflows with lower degree of parallelism can be obtained. In our evaluation, we analyze the behavior of PDWA for different shapes of workflows by fixing the other parameters (CCR=10, application nodes=20, out-degree=random) and by varying $\alpha$. With these workflows, the results of average latency and throughput is shown in Fig. 5. PDWA shows minimum average latency for $\alpha = 1$ however the latency is comparatively greater for the dense workflows because of the higher degree of parallelism between the application tasks when they are grouped in partitions that still require large inter-partition data movements. However, the average latency of PDWA is improved by approximately 66%, as shown in Fig. 5a, for $\alpha \leq 1$ as compared to AWOP. Similar results are observed for average throughput, and are shown in Fig. 5b. The average throughput of workflows with $\alpha > 1$ is less than the values achieved from AWOP, however PDWA outperforms AWOP with significant improvements in the average throughput.
C. Results with Different CCR Values

The proposed algorithm is designed to optimize the latency and throughput metrics for data-intensive workflow applications. The synthesized workflows with $CCR >> 1$ simulates the data-intensive workflows and if $CCR << 1$ compute intensive workflows can be obtained. When the $CCR = 1$ the workflows obtained are neither data-intensive nor compute intensive. We conducted experiments with different values of CCR while $\alpha = 1$, and results are shown in Fig. 6. Fig.6a shows that PDWA performs better with increasing values of CCR. This implies that PDWA suits well for data-intensive workflows. The average latency is reduced by 67% as compared to AWOP for CCR=10. Similar results are achieved for throughput, and are shown in Fig. 6b.

D. Gaussian Elimination

The Gaussian elimination workflow pattern is generated for the Gaussian elimination algorithm. It is characterized by the parameter $m$ that determines the number of nodes in the graph. For any given value of $m$, the number of nodes in the workflow is given by:

$$n = \frac{m^2 + m - 2}{2}, \quad (8)$$

We conducted experiments with different values of $m$ that provides Gaussian elimination workflow of varying sizes. The results are presented in Fig. 7. The Gaussian elimination workflows are selected for $m = 5, 7, 9, 11, 13, 15$. The number of nodes for each value of $m$ can be determined by Eq. 8. Fig. 7a shows that the performance of PDWA is significantly better than AWOP with 13%, 16%, and 6.6% improved average latency for $m = 11, 13, 15$, respectively. The average throughput is also recorded and is shown in Fig. 7b. The performance of the proposed algorithm is consistent and better as compared to AWOP with 50% maximum improvement in the average throughput.

VI. CONCLUSION

In this paper, a new approach, PDWA, is presented to optimize the data-intensive workflow applications. 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. PDWA performs significantly well for data-intensive workflows and reduces the latency and increases the throughput due to optimized partitioning and mapping. The proposed approach is evaluated with synthesized workflows of different characteristics. The evaluation of PDWA shows that it provides significantly reduced latency with improved throughput. Similar performance is observed for Gaussian elimination data-flow graphs of varying sizes.

The future work will include further improvements in the algorithm to incorporate wider range of workflows. We aim to implement the algorithm in a real-world heterogeneous computing environment and evaluate its performance with larger and complex workloads.

ACKNOWLEDGMENT

The work presented in this paper is supported by the Ministry of Education Malaysia (FRGS FP051-2013A and UMRG RP001F-13ICT).

REFERENCES


