Hindawi Publishing Corporation
Journal of Computer Networks and Communications
Volume 2014, Article ID 529835, 15 pages
http://dx.doi.org/10.1155/2014/529835

Research Article

Bandwidth-Aware Scheduling of Workflow Application on Multiple Grid Sites

Harshadkumar B. Prajapati[1] and Vipul A. Shah[2]

[1] *Department of Information Technology, Dharmsinh Desai University, Nadiad, Gujarat 387001, India*
[2] *Department of Instrumentation and Control Engineering, Dharmsinh Desai University, Nadiad, Gujarat 387001, India*

Correspondence should be addressed to Harshadkumar B. Prajapati; harshad.b.prajapati@gmail.com

Received 16 May 2014; Revised 29 July 2014; Accepted 5 August 2014; Published 11 September 2014

Academic Editor: Rick Stevens

Bandwidth-aware workflow scheduling is required to improve the performance of a workflow application in a multisite Grid environment, as the data movement cost between two low-bandwidth sites can adversely affect the makespan of the application. Pegasus WMS, an open-source and freely available WMS, cannot fully utilize its workflow mapping capability due to unavailability of integration of any bandwidth monitoring infrastructure in it. This paper develops the integration of Network Weather Service (NWS) in Pegasus WMS to enable the bandwidth-aware mapping of scientific workflows. Our work demonstrates the applicability of the integration of NWS by making existing Heft site-selector of Pegasus WMS bandwidth aware. Furthermore, this paper proposes and implements a new workflow scheduling algorithm—Level based Highest Input and Processing Weight First. The results of the performed experiments indicate that the bandwidth-aware workflow scheduling algorithms perform better than bandwidth-unaware algorithms: Random and Heft of Pegasus WMS. Moreover, our proposed workflow scheduling algorithm performs better than the bandwidth-aware Heft algorithms. Thus, the proposed bandwidth-aware workflow scheduling enhances capability of Pegasus WMS and can increase performance of workflow applications.

1. Introduction

Grid computing [1] enables executing performance demanding applications efficiently by exploiting distributed resources in a collaborative manner [2]. Many research projects, a few examples that include LIGO [3], Montage [4], BLAST [5], and WIEN2K [6], try to solve their computing problems by making computation demanding applications composed of reusable executables. Various systems such as Pegasus WMS [7], Askalon [8], Kepler [9], Karajan [10], Taverna [11], and Triana [12] have been used by such projects to execute computation demanding applications. Moreover, the systems that are open-source, freely available, well documented, and actively updated attract attention of many researchers and users. Main objective of this paper is to increase performance of workflow applications in Grid environment.

Grid applications having data dependencies are called workflow applications [13], which can be represented as a Directed Acyclic Graph (DAG). A workflow scheduler respects the dependencies among the tasks in the prepared output schedule and a workflow executor executes these tasks as per the arranged order on the chosen resources. The scheduling [14] aspect in Grid computing [15] is involved in two different entities: local resource scheduler and application scheduler. Workflow scheduling [16], an application scheduling, in Grid is complex and challenging, as the workflow scheduler has to decide which resources will execute which tasks, what will be the order of the tasks, how to respect the data dependencies, and how to minimize the makespan of the workflow application. The network performance of the chosen resources affects the makespan of a workflow application. Pegasus WMS is widely used workflow management system for scheduling of scientific workflows; however, it does not address the issue of scheduling tasks based on the network bandwidth among resources. Due to this limitation,

the workflow scheduling capability of Pegasus WMS remains underutilized.

The work on *Data-aware scheduling in Grid computing* in [17] discusses the importance of the data-aware data placement scheduler to enhance the performance of data intensive Grid applications by enabling efficient and reliable access of data. Their work focuses on end-to-end throughput optimization and data reliability through queuing, managing, scheduling, and optimization of data placement jobs. Their work estimates the speed of the storage servers based on monitored bandwidth, latency, and the number of hops information and uses the information in data placement decision. However, in their work, the mechanism of getting the speed information of storage servers, the crucial part of the work, is not paid much attention.

The work on *network bandwidth-aware job scheduling for computational Grids* in [18] presents network bandwidth-aware resource broker that matches users' requests with the best resources based on resource information and network information. Their work uses Ganglia monitoring toolkit [19] for monitoring statuses of resources and Network Weather Service (NWS) [20] to monitor network related information. Their work provides fresh information about resources, monitored by Ganglia and NWS, to users in secure manner. Their work demonstrates the application of bandwidth-aware job scheduling for scheduling of independent tasks, not for workflow applications. Our proposed work focuses on scheduling of workflow applications, which is more complex as compared to scheduling of independent tasks applications.

Major contributions of this paper are as follows. (1) The paper provides the implementation of Java programming interface for accessing NWS. (2) It develops the integration of NWS in Pegasus WMS, which we are the first to attempt, and develops existing Heft algorithm of Pegasus WMS bandwidth-aware. It also implements original HEFT [21] algorithm as bandwidth aware. (3) It proposes and implements a new workflow scheduling algorithm called *Level based Highest Input and Processing Weight First* (LHIPWF). (4) It provides the evaluation of the proposed bandwidth-aware workflow scheduling algorithms by presenting results of experiments performed on our Grid testbed.

This paper is structured as follows. Section 2 briefly discusses NWS and provides the implementation of Java access to NWS. Section 3 discusses the problem of workflow scheduling in Grid, discusses importance of bandwidth-aware workflow scheduling, presents the integration of bandwidth-aware workflow scheduling in Pegasus WMS, and develops Heft of Pegasus WMS and original HEFT as bandwidth aware. Section 4 proposes a new workflow scheduling algorithm called *Level based Highest Input and Processing Weight First* (LHIPWF) and provides the details of its implementation. Section 5 focuses on evaluation of workflow scheduling algorithms using experiments performed on our Grid testbed. The section briefly presents the used Grid testbed; it discusses test workflows and their preparation; it presents the experimental results; and it provides the comparison of the performance of workflow scheduling algorithms. Section 6 discusses the work and applicability of the proposed work. Finally, Section 7 presents the conclusion of the presented work and also discusses direction for further research work.

2. Java Access to Network Weather Service

In this section, we first describe Network Weather Service (NWS) and then discuss our Java implementation to access NWS. We use the Java access in performing bandwidth-aware workflow scheduling, which is presented in the next section.

2.1. Network Weather Service. NWS [20] provides dynamic characteristics of networked computing resources. Moreover, it can also forecast [22] performance of resources. The resource monitoring and performance prediction provided by NWS are useful to higher level services to enable scheduling and QoS guarantees in Grid or cluster computing environments. The implementation of NWS is based on Unix/Linux platform using TCP/IP socket based communication among distributed processes, which form internal components of NWS. NWS is made of four component systems: (1) *Persistent State* process, (2) *Name Server* process, (3) *Sensor* process, and (4) *Forecaster* process. The *Persistent State* process stores measurements in a persistent storage. It also provides access of measurements to their users. The *Name Server* process is a name-directory of the system that keeps mapping between higher level names and low-level contact information. The *Sensor* process periodically collects measurements from configured resources. The *Forecaster* process predicts the performance of the specified resources.

NWS supports measurements of the following variables: fraction of CPU availability time, connection time of TCP socket, end-to-end network latency of TCP, and end-to-end network bandwidth of TCP. NWS implements CPU availability sensor as a passive sensor, which itself does not measure any characteristics, rather relies on `vmstat` utility. The network sensor is implemented as an active sensor, which explicitly measures network performance related measures by using active measurement probes. We deploy NWS on each Grid site of our testbed; refer to Section 5.1 for details of the testbed. Each Grid site needs to run two daemon processes of NWS: memory and sensor; moreover one Grid site needs to run *Name Server* process. In our Grid testbed, we choose `grid-b` to run *Name Server* on it in addition to memory and sensor daemon processes.

2.2. Implementation of Java Access to NWS. Java access to NWS service is implemented by providing Java wrapper on `nws_extract`. Figure 1 shows relationship among classes of implemented Java access to NWS. The `NetworkInformationCollection` class is used by a client, whereas other classes `NetworkInformation`, `MeasurementRecord`, `NwsCommand`, and `NwsCommandExecutor` are used by `NetworkInformationCollection` class to carry out measurement related activities. The `NwsCommand` class generates required command string for getting bandwidth measurements or latency measurements. `NwsCommandExecutor` creates a `nws_extract` process using Java's process class and waits for result of measurements. The `NetworkInformation` class holds bandwidth and latency

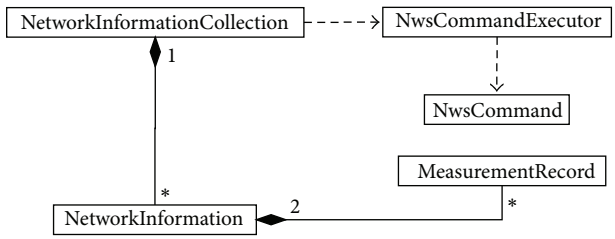

Figure 1: Relationship among the classes of implemented Java access to NWS.

information for a pair of source machine and destination machine. The `NetworkInformation` class holds all records of bandwidth and latency measurements performed using `NwsCommandExecutor`.

Multiple measurements records are collected for each pair of source machine and destination machine, which are used to calculate average bandwidth and average latency value, to smoothen out abrupt increase or decrease in value of measurement. Moreover, for each measurement, either bandwidth or latency, both the measured value and the forecast value are collected from the output of `nws_extract` process. A Java client of NWS retrieves measurement information using the object of the `NetworkInformationCollection` class. The `NetworkInformationCollection` class provides various getters methods to access bandwidth, forecast-bandwidth, latency, and forecast-latency between a pair of requested source and destination.

3. Bandwidth-Aware Workflow Scheduling and Its Implementation

Before discussing bandwidth-aware workflow scheduling and its implementation, we present the problem of workflow scheduling in Grid along with the related formula used in scheduling a workflow.

3.1. The Problem of Workflow Scheduling in Grid. The problem of workflow application scheduling involves two entities: the workflow itself and the set of resources that are used to execute the tasks of the workflow. Most workflow scheduling systems in Grid use Directed Acyclic Graph (DAG) for representing a scientific workflow. We use Pegasus WMS, which also uses DAG for workflow representation, for scheduling scientific workflows. Moreover, in workflow scheduling in Grid, the set of Grid sites becomes the set of candidate resources on which to schedule the tasks of the workflow or DAG. The problem can be represented as shown in the following.

A DAG is $G = (T, E)$, in which $T = \{T_i, i = 1, \ldots, n\}$ is the set of vertices representing the tasks of a workflow and $E = \{e_{i,j}, (i, j) = \{1, \ldots, n\}X\{1, \ldots, n\}\}$ is the set of edges representing precedence constraints and data communication between pairs of two tasks: T_i and T_j. If there are m Grid resources in a Grid system, then the set of resources is $R = \{R_k, k = 1, \ldots, m\}$. In Grid, each Grid resource can have different processing speed and different data communication speed, which result in a heterogeneous system.

There are two special tasks in a scientific workflow: the entry task and the exit task. Let T_{entry} represent the entry task and let T_{exit} represent the exit task. Moreover, let $\text{AST}(T_i)$ represent the actual start time of the task T_i and let $\text{AFT}(T_i)$ represent the actual finish time of the task. The general objective of DAG application scheduling is to minimize the makespan, which is Makespan = $\text{AFT}(T_{\text{exit}}) - \text{AST}(T_{\text{entry}})$. This problem of DAG application scheduling in Grid environment is NP-complete [23], due to which researchers try to devise heuristics based solution.

Let $w_i(R_j)$ indicate the estimated execution time to complete the task T_i on the resource R_j. The average execution cost of a task T_i is calculated as

$$\overline{w_i} = \frac{\sum_{j=1}^{|R|} w_i\left(R_j\right)}{|R|}. \tag{1}$$

Let a task T_i be scheduled on a resource R_m and let a successor task T_k be scheduled on a resource R_n. The amount of data communication between the tasks T_i and T_k is represented as $\text{data}(i, k)$. Let the bandwidth between two resources R_m and R_n be represented as $\text{BW}(m, n)$. The data communication cost between the two tasks T_i and T_k is calculated as

$$c(i,k) = \begin{cases} \dfrac{\text{data}(i,k)}{\text{BW}(m,n)}, & \text{if } m \neq n, \\ 0, & \text{if } m = n. \end{cases} \tag{2}$$

The average data communication cost between two tasks T_i and T_k is computed as

$$\overline{c(i,k)} = \frac{\text{data}(i,k)}{\overline{\text{BW}}}, \tag{3}$$

where $\overline{\text{BW}}$ is the average bandwidth among the resources that are considered for scheduling.

Let $\text{EST}(T_i, R_j)$ represent the earliest start time of the task T_i on the resource R_j and $\text{EFT}(T_i, R_j)$ the earliest finish time of the task T_i on the resource R_j. The EST of the entry task is calculated as

$$\text{EST}\left(T_{\text{entry}}, R_j\right) = 0. \tag{4}$$

For other tasks, ESTs and EFTs are computed recursively as

$$\begin{aligned} \text{EST}\left(T_i, R_j\right) &= \max\Bigg(\text{AvailableTime}\left(R_j\right), \\ &\qquad \max_{T_p \epsilon \text{pred}(T_i)} \left[\text{AFT}\left(T_p\right) + c\left(p, i\right)\right]\Bigg), \\ \text{EFT}\left(T_i, R_j\right) &= w_i\left(R_j\right) + \text{EST}\left(T_i, R_j\right). \end{aligned} \tag{5}$$

3.2. Bandwidth-Aware Workflow Scheduling. Though a workflow management system uses Grid resources in a collaborative manner for a workflow application, it needs to choose the best resources among available ones. Pegasus WMS has been deployed by many research projects; however, it is not able to fully utilize its scheduling capability due to unavailability of the integration of bandwidth monitoring infrastructure. Moreover, the workflow scheduling problem in distributed environment being NP-complete [23] in nature attracts the attention of many researchers.

The bandwidth among the resources of a single Grid site, that is, single organization, is larger than the bandwidth among various Grid sites, which are connected using Internet. Therefore, in a multi-Grid sites environment, data movement becomes an important issue for scheduling of a scientific workflow. In a workflow, the output file(s) generated by one task works as input file(s) for its dependent tasks. To reduce the data communication cost, the workflow scheduler needs to schedule tasks on the Grid sites that can complete intersite data transfers as fast as possible. If the workflow scheduler is data aware and bandwidth aware, the performance of a workflow application can be improved. Pegasus WMS is open-source, freely available, well documented, and actively updated workflow management system for scheduling of scientific workflows; however, it is not bandwidth aware. We have attempted to address this issue to enhance the workflow scheduling capability of Pegasus WMS.

3.3. Implementation of Bandwidth-Aware Workflow Scheduling. To implement bandwidth-aware workflow scheduling, we first studied the source code of the Java classes of Pegasus WMS that are responsible for carrying out scheduling of tasks of a workflow. Moreover, we also studied and understood flow of control of Pegasus WMS for running the site selection algorithm by running Pegasus WMS on submit site, using its source code, under debug environment provided by NetBeans 7.0.1 [24], as the available literature on Pegasus WMS does not address the implementation at depth. We modify `pegasus.jar` file of Pegasus WMS to embed bandwidth-aware workflow scheduling algorithms in it. This section describes the development of the bandwidth-aware workflow scheduling and workflow scheduling algorithms.

We present an example of `NwsAwareHeft`, an extension of the Heft algorithm of Pegasus WMS, in Listing 1 to explain how bandwidth-aware workflow scheduling is embedded in Pegasus WMS. Pegasus WMS plans jobs to the Grid sites, the logical names used in Site Catalog; however, the NWS service does not know meaning of these logical names. Therefore, the mapping from a Grid site name to its machine name is needed for initializing NWS and performing measurements about these machines. We embed a `mGridSitesToMachinesMap` data member, an instance of `HashMap <String, String>`, in the `NwsAwareHeft` class to store the mapping between Pegasus WMS's Grid site name and corresponding machine name. In `public void initialize (PegasusBag bag)` method of `NwsAwareHeft` class, `mMachinesNIMap` and `mGridSitesToMachinesMap` instances are added in `PegasusBag bag`, which are used by workflow scheduling algorithm that is encapsulated in algorithm class under `edu.isi.pegasus.planner.selector.site.nwsawareheft` package.

We add bandwidth awareness in the `Algorithm` class of Heft of Pegasus WMS to create bandwidth-aware Heft site-selector algorithm. Similarly, we implement bandwidth-aware original HEFT algorithm [21]. Major changes are in the calculation of upward rank, calculation of downward rank, calculation of data transfer time, and calculation of earliest finish time (EFT). In these calculations, we use measured bandwidth values among Grid sites and data sizes of input/output files, which are specified in .dax file, an XML file that represents an abstract workflow in Pegasus WMS.

4. Proposed Workflow Scheduling Algorithm

We propose a new workflow scheduling algorithm called *Level based Highest Input and Processing Weight First* (LHIPWF). This section presents this algorithm and shows its implementation in Pegasus WMS.

4.1. Level Based Highest Input and Processing Weight First Algorithm. The proposed scheduling algorithm, *Level based Highest Input and Processing Weight First* (*LHIPWF*), is shown in Algorithm 1. As the algorithm is of type list scheduling, it has two phases: the first phase, on the lines 2–4, assigns the priority to each task of the DAG and the second phase, on the lines 7–18, is the scheduling loop. To calculate the priority, LHIPWF algorithm uses the average bandwidth and the average computing power of resources. As the priority in LHIPWF is not recursively defined, as opposed to what was done in HEFT, there is no need to traverse the tasks of a DAG in any specific order, either topological or bottom-order. As shown on the line 3, the priority of a task is defined as $priority(T_i) = (computationAmount(T_i)/averagePPower) + (totalInputSize/averageBW)$, in which *totalInputSize* is calculated by summing the data communication amount from each predecessor of the task to the task itself. The priority measure is inspired from the following observation. If a task having a large data transfer is selected first for choosing the required Grid site, then there is a high probability that such task gets the best Grid resource, possibly minimizing the data communication cost, and such task does not become a bottleneck task to its successors. Furthermore, a task that requires large computation should get higher priority as compared to tasks having smaller computation. Therefore, the proposed scheduling algorithm considers both communication costs and computation cost in assigning priorities to the tasks of the workflow.

As part of the scheduling loop, on the lines 7–18, the highest priority ready task is selected, on the line 8, and it is assigned to the resource that finishes the task earliest, on the line 9. Once a task is scheduled, its immediate ready successors are found, on the line 11, and these unscheduled ready tasks are temporarily kept in *tempSet*, on the line 12, until *priorityQueue* is nonempty. When *priorityQueue* becomes empty during the scheduling loop, that is, when all the tasks of the current level (batch) have already been scheduled, the tasks from *tempSet* are transferred into

(1) Calculate average bandwidth (*averageBW*) and average processing power (*averagePPower*) of resources
(2) **for** each task T_i in DAG **do**
(3) $priority(T_i) \leftarrow computationAmount(T_i)/averagePPower + totalInputSize/averageBW$
(4) **end for**
(5) Initialize priority queue with the entry task (T_{entry}) of DAG
(6) $currentBatchSize \leftarrow 1$
(7) **while** there is an unscheduled task in the *priorityQueue* **do**
(8) select highest priority task (T_{hp}) from *priorityQueue*
(9) allocate the task T_{hp} to the resource R_j that minimizes the EFT(T_{hp}, R_j).
(10) $currentBatchSize \leftarrow currentBatchSize - 1$
(11) $readySuccessors \leftarrow ReadySuccessors(T_{hp})$
(12) **add** *readySuccessors* into *tempSet*
(13) **if** $currentBatchSize = 0$ **then**
(14) **add** all tasks present in *tempSet* into *priorityQueue*
(15) $currentBatchSize \leftarrow |tempSet|$
(16) $tempSet \leftarrow \phi$
(17) **end if**
(18) **end while**

Algorithm 1: *Level based highest input and processing weight first (LHIPWF)* algorithm.

priorityQueue, on the line 14; next, *currentBatchSize* is initialized with the number of tasks in *tempSet*, on the line 15; and then *tempSet* is cleared to hold the ready successor tasks of the next level, on the line 16. In the proposed algorithm, priority is given to the task that has the highest cost including input data-transfer cost and processing cost; moreover, this algorithm decides scheduling of the ready tasks on level-by-level basis; therefore, its name is Level based Highest Input and Processing Weight First.

4.2. Implementation of LHIPWF Algorithm in Pegasus WMS. The implementation of LHIPWF in Pegasus WMS consists of the `NwsAwareLHIPWF` class in the `edu.isi.pegasus.planner.selector.site` package and `Algorithm`, `LHIPWFBag`, `Processor`, and `Site` classes in the package: `edu.isi.pegasus.planner.selector.site.nwsawarelhipwf`. However, as compared to the implementation of HEFT, there are no changes in the `Site` and the `Processor` classes. The `Algorithm` class implements the algorithm presented in Algorithm 1. `LHIPWFBag` is similar to `HeftBag` with addition of the `mPriority` data member and associated methods and modifications in `add` and `get` methods. The `computePriority` method computes the priority of a task using measured average bandwidth value and the amount of data communication to the considered task from its predecessor tasks. For holding tasks in `priorityQueue`, we use `PriorityQueue` class of `java.util` package.

5. Evaluation of Bandwidth-Aware Workflow Scheduling

In this section, we first provide information about test environment and test applications. Next, we provide experimental results. Finally, we provide comparison of the performance of considered workflow scheduling algorithms.

5.1. Grid Testbed for Experiments. We have developed a Grid testbed to carry out experimental evaluation of the proposed bandwidth-aware workflow scheduling. Figure 2 shows the components used in the Grid testbed. The testbed is made of the following software:

(i) Globus toolkit 5.2.3 [25] as Grid middleware,
(ii) Network Time Protocol (NTP) [26] for time synchronization,
(iii) DAGMan [27] as DAG job execution engine,
(iv) HTCondor (7.8.7) [28] as a Local Resource Manager,
(v) Condor-g [29] for performing job submission to LRM using GRAM,
(vi) Pegasus WMS (4.1.0) to enable workflow scheduling,
(vii) Network Weather Service (NWS) [20] to monitor network performance,
(viii) Dummynet for bandwidth control.

We have four Grid sites having varying bandwidth. Each Grid site has two processing elements. Therefore, each Grid site can execute two computing tasks at a time. Table 1 shows role and bandwidth configuration of each Grid site used in our developed Grid testbed. To find out bandwidth among the Grid sites using NWS, we use default configuration for performing measurements of bandwidth and latency, in which the NWS transmits 64 KB data as four 16 KB messages with socket buffer size of 32 KB [20]. The network sensors of NWS perform periodic measurements once in 10 to 60 seconds, which are stored in a circular buffer with time-stamped values [20]. The forecaster chooses best history size to do each prediction. When the bandwidth information is required by the Pegasus WMS, before performing mapping of the tasks of a workflow, we use average of 20 measurements as bandwidth between a pair of two machines, though

```
public class NwsAwareHeft extends Abstract {
   private NetworkInformationCollection mMachinesNIMap;
   private HashMap<String,String> mGridSitesToMachinesMap;
   public void initialize ( PegasusBag bag ) {
      super.initialize ( bag );
      initializeNwsInformation ();
      bag.add (PegasusBag.NWS_INFO, mMachinesNIMap);
      bag.add (PegasusBag.SITES_TO_MACHINES_MAP,
mGridSitesToMachinesMap);
      mHeftImpl = new Algorithm ( bag );
   }
   private void initializeNwsInformation () {
      mMachinesNIMap=new NetworkInformationCollection ();
      mGridSitesToMachinesMap=getGridSitesToMachinesMap ();
      String
nwsNameServer=getNwsNameServerNameFromConfigFile ();
      String [] resources=getMachinesNames ();
      mMachinesNIMap.init (nwsNameServer, resources);
      mMachinesNIMap.performMeasurements ();
   }
...
   private String getMachineName (String gridSiteName) {
      String machineName="";
      String urlPrefix="";
      SiteCatalogEntry scEntry=mSiteStore.lookup (gridSiteName);
      if (scEntry!=null) {
         HeadNodeFS headNodeFS=scEntry.getHeadNodeFS ();
         if (headNodeFS!=null) {
            FileServer fileServer =
headNodeFS.selectStorageSharedFileServer ();
            urlPrefix=fileServer.getURLPrefix ();
            if (urlPrefix!=null) {
               machineName=parseMachineName (urlPrefix);
            }
         }
      }
   return machineName
   }
...
}
```

Listing 1: Parts of the source code showing how access to NWS is embedded in the Heft site-selector algorithm. We create a new Java class NwsAwareHeft for bandwidth-aware Heft site-selector algorithm. This listing shows only important fragments of modified or added part in the original Heft algorithm of Pegasus WMS. The added/modified code is shown in bold-face to distinguish it from the already available code in Pegasus WMS's Heft class.

the number of measurements to be considered is configurable in our implemented java access to NWS.

For performing experiments of workflow scheduling and execution in Pegasus WMS, it is important to understand the working of Pegasus WMS. Therefore, we briefly discuss the working of Pegasus WMS. Pegasus WMS is a workflow execution and management software for workflow jobs that are represented as Directed Acyclic Graph. It manages dependencies of jobs. It can allow use of one or more Grid sites for the execution of the jobs of a workflow application. Pegasus WMS can use any Grid resource or Grid site that can be accessed using GRAM. Pegasus WMS uses, on the workflow submit site, Condor-g for job submission and Condor DAGMan for execution of a DAG; see Figure 2. Pegasus WMS does planning of the jobs of a workflow, represented in.dax format, and generates a concrete workflow in form of.dag file and Condor-g submit files and passes them to DAGMan for execution.

DAGMan itself cannot decide which Grid site runs which jobs, though it can utilize any available machine of the cluster once the job is submitted to a particular Grid site. DAGMan cannot take decision about which Grid site is best to run a particular job. In this regard, Pegasus WMS complements DAGMan for supporting scheduling and execution of the jobs of a workflow on Grid resources. Pegasus WMS chooses appropriate Grid resources for execution of jobs, which

Table 1: Role and bandwidth configuration of each Grid site used for workflow scheduling using pegasus WMS.

Grid-site	Head-node	Role of site	Available bandwidth
ddu_grid-b	grid-b.it2.ddu.ac.in	Submitter, executor	1024 kbit/s
ddu_grid-v	grid-v.it2.ddu.ac.in	Executor	512 kbit/s
ddu_grid-m	grid-m.it2.ddu.ac.in	Executor	256 kbit/s
ddu_ca	ca.it2.ddu.ac.in	Executor	128 kbit/s

Table 2: Information of three test workflows.

	10-node WF	41-node WF (large data comm.)	41-node WF (small data comm.)
Task runtime (seconds)	70 to 120 S	243 to 357 S	243 to 357 S
Cumulative runtime	961 S	12219 S	12219 S
Avg. runtime	96.1 S	298.02 S	298.02 S
Task data communication	3547228 to 4667402 B	9879000 to 15698940 B	987900 to 1569894 B
Cumulative data Comm.	59241448 B	873346380 B	87334638 B
Avg. data comm.	4231532 B	12300653 B	1230065 B
CCR	≈0.82	≈0.65	≈0.065

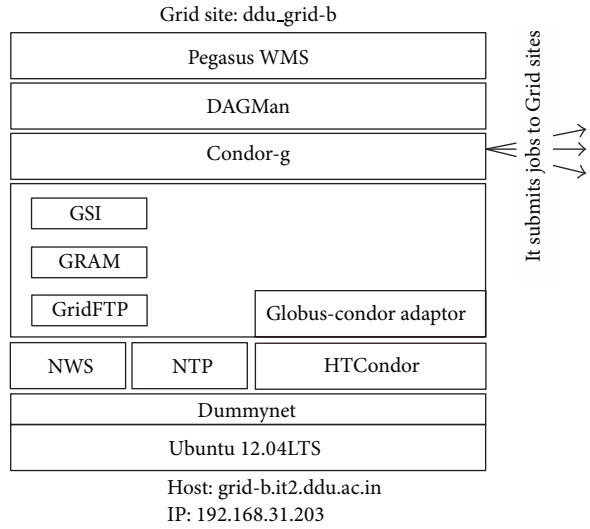

Figure 2: Components in our Grid testbed.

is done by its site-selector algorithm, and generates codes for file movement, data cleanup, data registration, and so forth. Moreover, Pegasus WMS provides monitoring of status of a running workflow and also provides provenance and performance related information.

5.2. Test Workflows. In mapping based WMS, such as Pegasus, the knowledge of input/output data size and runtime of each job is needed to evaluate workflow scheduling algorithms. However, for real workflows, such information might not be available directly, though they could be estimated or predicted external to WMS. At present, Pegasus WMS has no component that directly provides runtime information. Moreover, we would like to emphasize that the researchers that are not involved in working with any real scientific workflow applications find it very difficult to understand the tasks and input/output files of real workflows. Due to the above-mentioned limitation of Pegasus WMS and the reason, we prepare test workflows having tasks of desired runtime values and input/output data sizes. We develop a way of preparing test workflows, which is inspired from the black-diamond workflow of Pegasus WMS and the work presented in [31]. The work in [31] presents the generation of synthetic workflows that can be used only on simulator. However, using the `pegasus-keg`, we prepare test workflows that can be executed on a real Grid environment. Using `pegasus-keg`, it is possible to create a job that runs for the specified runtime and generates the output data (file) of the specified size. We discuss preparing test workflows next.

The `pegasus-keg`, which comes along with the installation of Pegasus WMS, is Kanonical Executable for Grids. The `pegasus-keg` command can take any number of input files and can generate any number of output files. Two interesting features of the command enable its use to form test workflows of desired characteristics. These two features are being (1) able to generate large output file using `-G` option and (2) able to do heavy computation for specified duration using `-T` option. Moreover, the `pegasus-keg` command has `-a` option to specify the name of an application. This option can be used to form different jobs having different names using the same `pegasus-keg` command.

While preparing an abstract workflow in DAX format, we use the `size` attribute of the `uses` child tag of the `job` tag of DAX to specify the size of input or output file and the `runtime` profile key of pegasus namespace inside the `job` tag to specify the runtime of the job. Each job is made to exhibit computation on computing node (CPU) for the specified runtime through `-T` option to the `pegasus-keg` command. For each job, the transformation catalog holds the mapping from the logical job name to the physical`pegasus-keg` command. Similarly, the specified `size` value of output file is also passed to the `pegasus-keg` command using `-G` option in order to generate the output data of the needed size; see Listing 2.

We prepare abstract workflows, in .dax format, having tasks of desired runtime values and input/output data sizes, using `pegasus-keg`. Table 2 shows the information about

```
...
<job id="j9" namespace="test-10" name="J" version="4.0">
    <argument>-a J -T 94 -i <file name="data.gj"/>    <file name="data.hj"/>    <file
name="data.ij"/> -G 1000000 -o <file name="data.j"/>    </argument>
    <profile namespace="pegasus" key="runtime">94</profile>
    <uses name="data.gj" link="input" transfer="false" register="false" size="4345853"/>
    <uses name="data.hj" link="input" transfer="false" register="false" size="3618930"/>
    <uses name="data.ij" link="input" transfer="false" register="false" size="4172567"/>
    <uses name="data.j" link="output" transfer="true" register="true" size="1000000"/>
</job>
...
    <child ref="j1">
        <parent ref="j0"/>
    </child>
    <child ref="j2">
        <parent ref="j0"/>
    </child>
    <child ref="j3">
        <parent ref="j0"/>
    </child>
    <child ref="j4">
        <parent ref="j0"/>
    </child>
    <child ref="j5">
        <parent ref="j0"/>
    </child>
    <child ref="j6">
        <parent ref="j2"/>
    </child>
    <child ref="j7">
        <parent ref="j1"/>
        <parent ref="j3"/>
        <parent ref="j5"/>
    </child>
    <child ref="j8">
        <parent ref="j1"/>
        <parent ref="j3"/>
        <parent ref="j4"/>
    </child>
    <child ref="j9">
        <parent ref="j6"/>
        <parent ref="j7"/>
        <parent ref="j8"/>
    </child>
</adag>
```

Listing 2: Parts of the .dax file for the test workflow having 10 tasks, emphasizing how each task of workflow is created using pegasus-keg command and the structure of the overall workflow showing dependencies among the tasks.

three test workflows. These test workflows have different characteristics to evaluate workflow scheduling and execution. We vary the number of tasks in workflows. Moreover, an important factor for evaluating workflow scheduling is varying communication to computation (CCR) ratio, for which we consider three possibilities: high CCR (0.82), medium CCR (0.65), and low CCR (0.065). The first test workflow has 10 tasks. As the second test workflow, a large workflow having 41 tasks is prepared. This workflow has structure of task-graph of molecular dynamics code, which is available in [32]. The third test workflow is similar to the earlier one. However, the amount of data communication between each pair of task is small.

5.3. Experimental Results. To evaluate and compare the performance of the proposed bandwidth-aware workflow scheduling algorithm, we execute each workflow application using different site-selector algorithms. Experiments are performed using `Random`, `Heft`, `NwsAwareHeft`, `NwsAwareOHeft`, and `NwsAwareLHIPWF` site-selector algorithms.

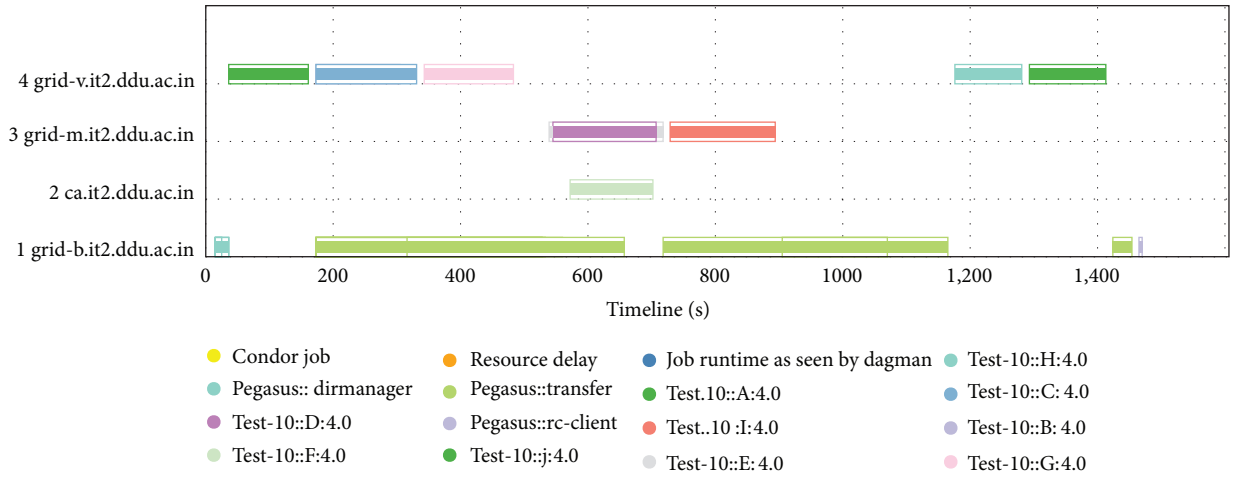

FIGURE 3: The output schedule for the workflow of 10 tasks scheduled using Heft site-selector provided by Pegasus WMS.

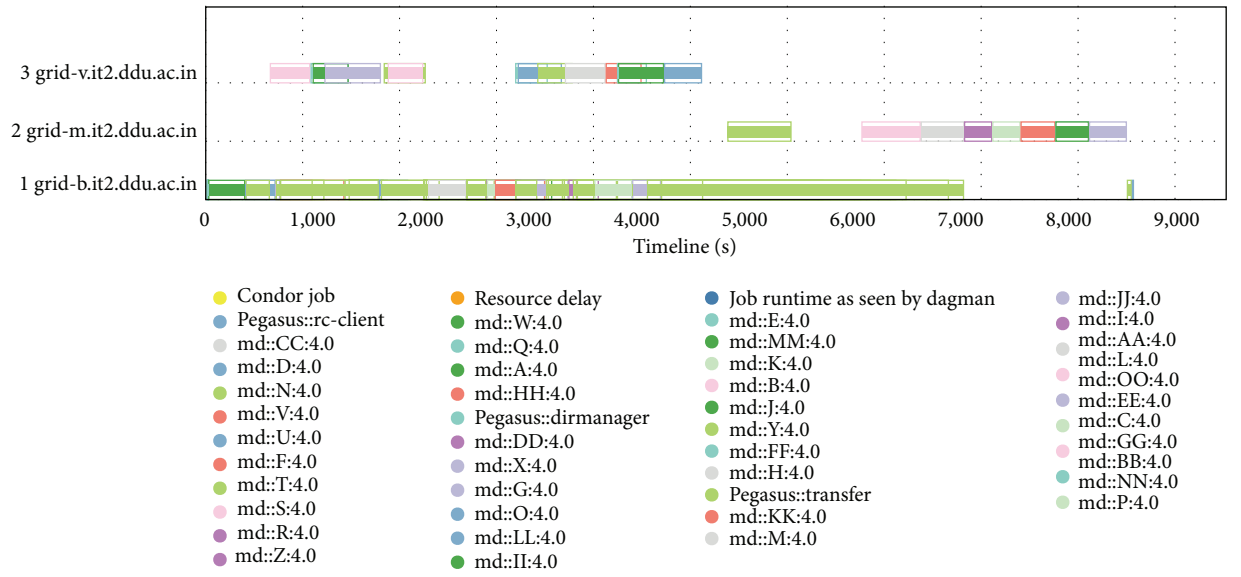

FIGURE 4: The output schedule for the workflow of 41 tasks with large data communication scheduled using bandwidth aware original Heft site-selector implemented by us.

`Random`, and `Heft` are existing site-selector algorithms of Pegasus WMS, whereas `NwsAwareHeft` and `NwsAware-OHeft` are bandwidth-aware site-selector algorithms that are implemented by us and `NwsAwareLHIPWF` is bandwidth-aware site-selector algorithm that is proposed and implemented by us. The site-selector algorithm is changed by changing the `pegasus.selector.site property` in `/home/gtuser/.pegasusrc` file on `ddu_grid-b` site. We have gathered the results by executing 15 experiments (for 5 algorithms and 3 workflow applications) on the testbed in which executing workflows took time ranging around 44 minutes to as large as around 6.5 hours.

We briefly present the methodology of gathering the results from Pegasus WMS. When a workflow application finishes its execution, we prepare various charts by running `pegasus-plots` on the workflow directory. The output schedule is extracted from the outputs of `pegasus-plots` command, which is available as *host over time chart*. Figure 3 shows the output schedule for Heft site-selector algorithm for the workflow of 10 tasks. Figure 4 shows the output schedule for the workflow of 41 tasks with large data communication for bandwidth-aware original Heft site-selector implemented by us. Figure 5 shows the output schedule for the workflow of 41 tasks with small data communication for bandwidth-aware LHIPWF proposed and implemented by us. Due to space limitation, we do not present output schedule of each.

5.4. Comparison of the Performance of Workflow Scheduling Algorithms. Comparison of the performance of workflow scheduling algorithms is discussed here. The comparison results include comparison of total data communication time, execution-time taken, and Speedup achieved by each

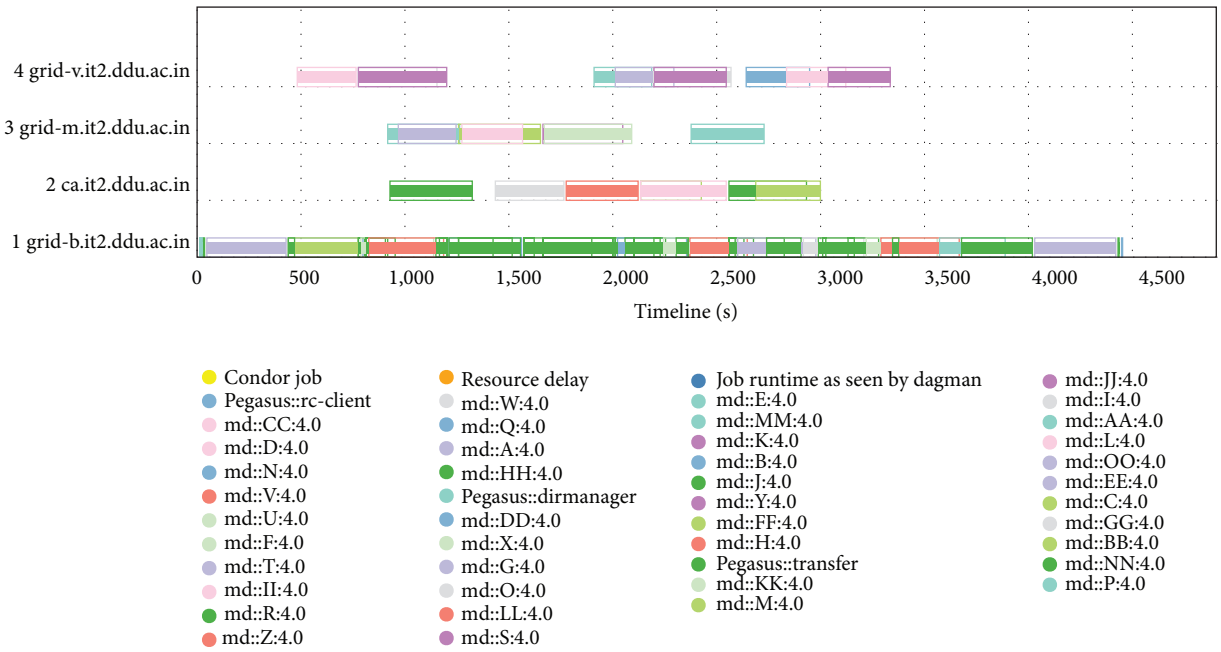

FIGURE 5: Output schedule for the workflow of 41 tasks with small data communication scheduled using LHIPWF site-selector proposed and implemented by us.

workflow scheduling algorithm for the three workflows. The total data communication time of the workflow execution is indicated by the total time taken by `pegasus::transfer` task, which we extract from *invocation breakdown chart* generated using `pegasus-plots`. We extract the execution-time of the run of a workflow using `pegasus-statistics`. The Speedup indicates how speedily a workflow runs when running it on multiple Grid sites. The Speedup of the workflow run is calculated by dividing the serial execution time by execution-time of the workflow on multiple Grid sites.

Figure 6 compares the total data communication time of the considered workflow scheduling algorithms for the workflow of 10 tasks. The results show that bandwidth-aware workflow scheduling algorithms BW-Aware-Heft, BW-Aware-original-Heft, and BW-Aware-LHIPWF spend less time in performing data communication as compared to bandwidth-unaware algorithms: Random and Heft.

Figure 7 compares the total data communication time of the workflow scheduling algorithms for the workflow of 41 tasks with large data communication among the tasks of the workflow. From the results, it can be seen that BW-Aware-Heft, BW-Aware-original-Heft, and BW-Aware-LHIPWF spend less time in performing data communication as compared to Random and Heft algorithms. Moreover, BW-Aware-LHIPWF consumes minimum time in data communication as compared to others. Figure 8 compares the total data communication time of the workflow scheduling algorithms for the workflow of 41 tasks with small data communication among the tasks of the workflow. BW-Aware-LHIPWF consumes minimum time in data communication as compared to others. Moreover, due to very small CCR

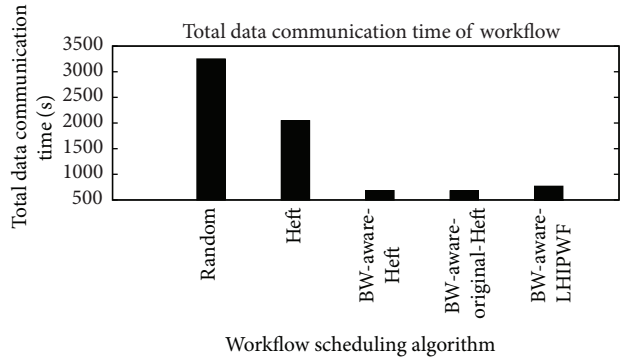

FIGURE 6: Comparison of the total data communication time for the workflow of 10 tasks.

value, there are not comparatively much variations in the performance.

Figure 9 compares the execution time of the considered workflow scheduling algorithms for the workflow of 10 tasks. From the comparison graph, it can be seen that Random algorithm performs very badly taking execution time of 2651 seconds. Heft takes 1475 seconds for execution of the workflow. Bandwidth-aware workflow scheduling algorithms perform better than Random and Heft. Bandwidth-aware Heft, shown as BW-Aware-Heft in the graph, uses 1073 seconds for finishing execution of the workflow and bandwidth-aware original Heft, shown as BW-Aware-original-Heft in the graph, uses 1045 seconds for finishing execution of the workflow. Our proposed algorithm LHIPWF, shown as BW-Aware-LHIPWF in the graph, takes only 939 seconds for

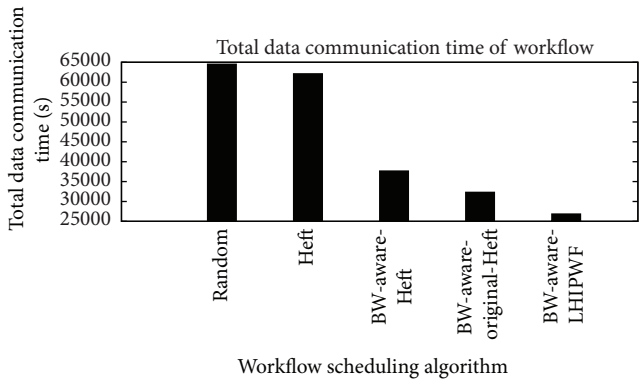

Figure 7: Comparison of the total data communication time for the workflow of 41 tasks with large data communication.

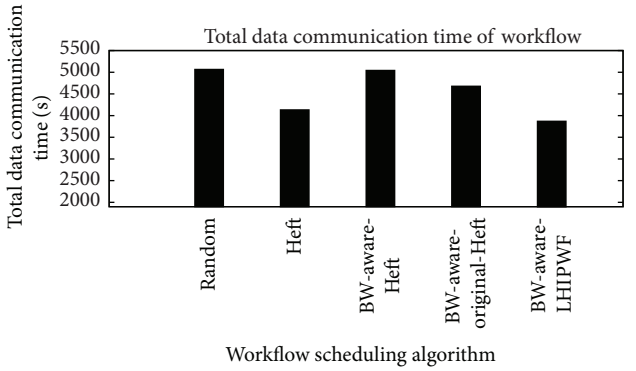

Figure 8: Comparison of the total data communication time for the workflow of 41 tasks with small data communication.

executing the workflow. On comparing our proposed algorithm with other bandwidth-aware algorithms, we find that our proposed algorithm, LHIPWF, performs 10% better than bandwidth-aware original Heft, that is, BW-Aware-original-Heft, and 12% better than bandwidth-aware Heft, that is, BW-Aware-Heft. On comparing LHIPWF algorithm with bandwidth-unaware algorithms, we find that BW-Aware-LHIPWF performs 65% better than Random and 36% better than Heft.

Figure 10 compares the Speedup of the workflow scheduling algorithms for the workflow of 10 tasks. From the graph of Speedup, it can be seen that only our proposed workflow scheduling algorithm, bandwidth-aware LHIPWF, achieves Speedup greater than 1. Random algorithm performs very badly. Heft provided better Speedup than Random, but its Speedup is less than 1. BW-Aware-Heft and BW-Aware-original-Heft performed approximately equally and tried to reach Speedup value of 1. Our proposed algorithm, LHIPWF, achieves Speedup value of 1.024. These observed low Speedup values are due to high CCR value among the tasks of the workflow.

Figure 11 compares the execution time of the workflow scheduling algorithms for the workflow of 41 tasks with large data communication. For the large workflow with

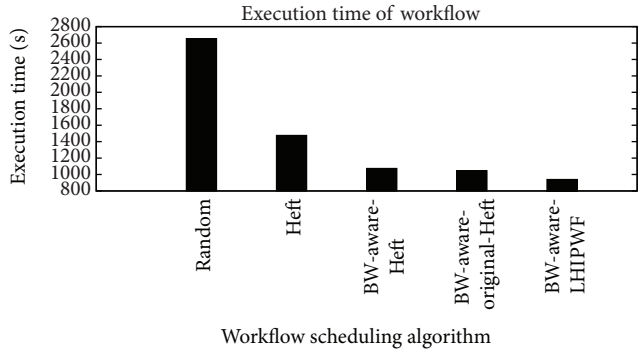

Figure 9: Comparison of the execution time for the workflow of 10 tasks.

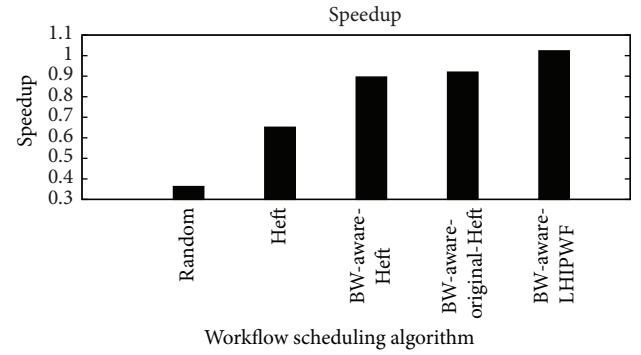

Figure 10: Comparison of the Speedup for the workflow of 10 tasks.

large data communication, Random algorithm takes execution time of 23261 seconds, which is worst among other workflow scheduling algorithms. Heft takes 17256 seconds for execution of the workflow. Bandwidth-aware workflow scheduling algorithms perform better than Random and Heft, which are bandwidth-unaware. Bandwidth-aware Heft, shown as BW-Aware-Heft in the graph, uses 11642 seconds for finishing execution of the workflow and bandwidth-aware original Heft, shown as BW-Aware-original-Heft in the graph, uses 9571 seconds for finishing execution of the workflow. Bandwidth-aware original Heft performs 58.8% better than Random, which is default workflow scheduling algorithm of Pegasus WMS, and 44.5% better than Heft. Our proposed algorithm LHIPWF, shown as BW-Aware-LHIPWF in the graph, takes 8951 seconds, minimum among all, for executing the workflow. On comparing our proposed algorithm with other bandwidth-aware algorithms, we find that our proposed algorithm, LHIPWF, performs 23% better than bandwidth-aware Heft, that is, BW-Aware-Heft, and 6.4% better than bandwidth-aware original Heft, that is, BW-Aware-original-Heft. On comparing LHIPWF algorithm with bandwidth-unaware algorithms, we find that BW-Aware-LHIPWF performs 61.5% better than Random and 48.1% better than Heft.

Figure 12 compares the execution time of the workflow scheduling algorithms for the workflow of 41 tasks with small data communication. For the large workflow with small

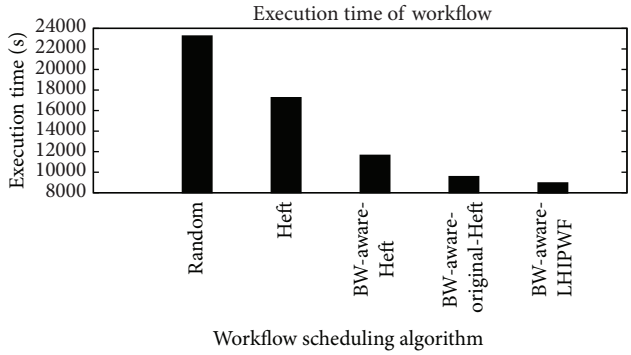

Figure 11: Comparison of the execution time for the workflow of 41 tasks with large data communication.

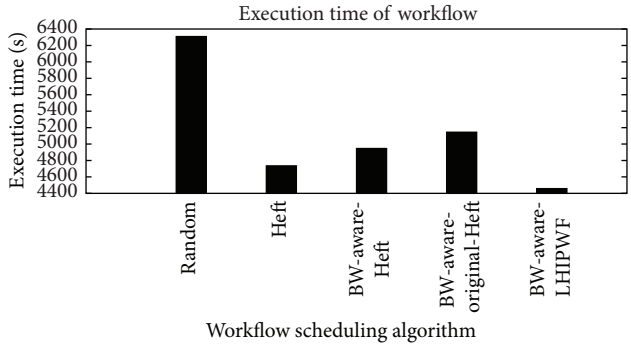

Figure 12: Comparison of the execution time for the workflow of 41 tasks with small data communication.

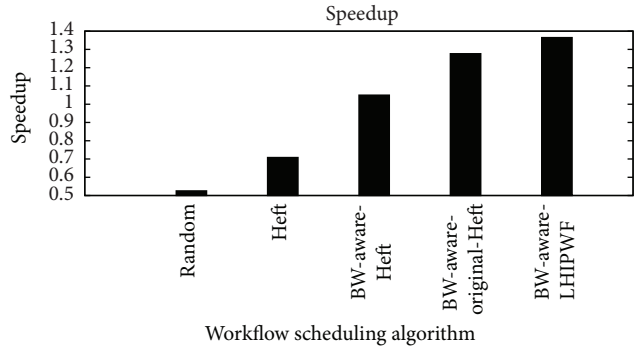

Figure 13: Comparison of the Speedup for the workflow of 41 tasks with large data communication.

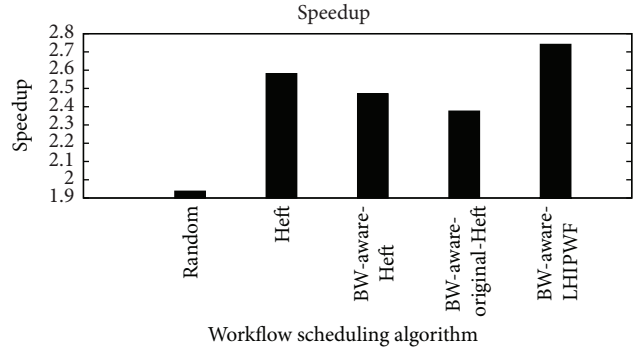

Figure 14: Comparison of the Speedup for the workflow of 41 tasks with small data communication.

data communication, Random algorithm takes execution time of 6308 seconds, which is worst among other workflow scheduling algorithms' execution times. Bandwidth-aware workflow scheduling algorithms perform better than Random. Heft takes 4735 seconds for execution of the workflow. Bandwidth-aware Heft takes 4946 seconds and bandwidth-aware original Heft takes 5144 seconds for finishing execution of the workflow. Bandwidth-aware original Heft performs 18% better than Random, which is default workflow scheduling algorithm of Pegasus WMS. Our proposed algorithm LHIPWF, shown as BW-Aware-LHIPWF in the graph, takes 4458 seconds, minimum among all, for executing the workflow. On comparing our proposed algorithm with other bandwidth-aware algorithms, we find that our proposed algorithm, LHIPWF, performs 9.9% better than bandwidth-aware Heft, that is, BW-Aware-Heft, and 13.3% better than bandwidth-aware original Heft, that is, BW-Aware-original-Heft. On comparing LHIPWF algorithm with bandwidth unaware algorithms, we find that BW-Aware-LHIPWF performs 29.3% better than Random and 5.9% better than Heft.

Figure 13 compares the Speedup of workflow of 41 tasks with large data communication and Figure 14 compares the Speedup of workflow of 41 tasks with small data communication for the workflow scheduling algorithms. On comparing Speedup of these two workflows, it can be seen that when intermediate data communication amount increases, it affects Speedup value. As compared to the workflow with small intermediate data communication, comparatively higher time is spent in performing data transfer in the workflow with large intermediate data communication. Consequently, the high data transfer time increases the execution-time of the workflow and decreases Speedup value. For both considered large workflows, the bandwidth-aware LHIPWF performed best among all others by achieving highest Speedup. For the large workflow with large data communication, Random algorithm, which is provided by Pegasus WMS and is default site-selector algorithm of Pegasus WMS, performed very badly. The ranking of the algorithms, with the best performer first, based on Speedup value is as follows: BW-Aware-LHIPWF with Speedup of 1.37, BW-Aware-original-Heft with Speedup of 1.28, BW-Aware-Heft with Speedup of 1.05, Heft with Speedup of 0.71, and Random with Speedup of 0.53. For the large workflow with small data communication, all the workflow scheduling algorithms achieve better Speedup value, that is, greater than one, due to small CCR. The ranking of the algorithms, with the best performer first, based on Speedup value is as follows: BW-Aware-LHIPWF with Speedup of 2.74, Heft with Speedup of 2.58, BW-Aware-Heft with Speedup of 2.47, BW-Aware-original-Heft with Speedup of 2.38, and Random with Speedup of 1.94.

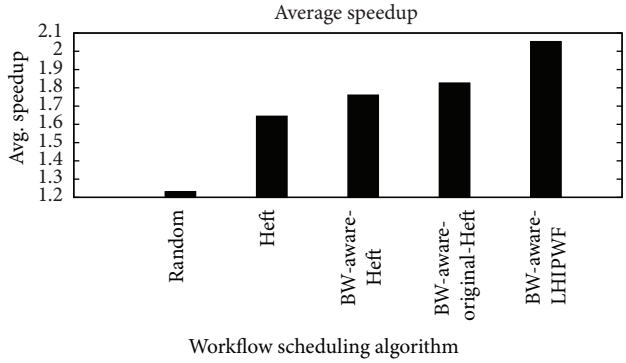

Figure 15: Comparison of Average Speedup of the two workflows of 41 tasks achieved by workflow scheduling algorithms.

To compare workflow scheduling algorithms for combined effect of large data communication and small data communication, the corresponding results of Speedup are averaged. Figure 15 presents Average Speedup, of the two large workflows, achieved by the considered workflow scheduling algorithms. The ranking of the algorithms, with the best performer first, based on Speedup value is as follows: BW-Aware-LHIPWF with Speedup of 2.05, BW-Aware-original-Heft with Speedup of 1.83, BW-Aware-Heft with Speedup of 1.76, Heft with Speedup of 1.64, and Random with Speedup of 1.23.

To summarize, bandwidth-aware original Heft algorithm and bandwidth-aware implementation of Heft of Pegasus WMS perform better than bandwidth-unaware workflow scheduling algorithms: Heft and Random. Furthermore, our proposed algorithm, LHIPWF, provided the best performance among all the other considered workflow scheduling algorithms for the three considered workflow applications, having diverse characteristics; therefore, LHIPWF can become possible candidate for scheduling of workflow applications. Thus, from the experimental results, it can be concluded that by integrating bandwidth information in Pegasus WMS, its scheduling capability increases and workflow applications finish their execution efficiently.

6. Discussion

The results of the experiments indicated that, by integrating the bandwidth information in the scheduling decision, workflow applications could finish their executions efficiently. During performing the scheduling, the bandwidth-unaware workflow scheduling algorithm treated all the sites of equal bandwidth, though all the considered sites have varying bandwidth values. The Heft algorithm of Pegasus WMS assumes the bandwidth value of 5 MBPS for each Grid site and does planning of jobs accordingly. For example, the `ddu_ca` Grid site has lowest bandwidth of 128 Kbits; however, the Heft of Pegasus WMS assumes that this site has the bandwidth of 5 MBPS and does the wrong estimation of EFT for each job. The bandwidth-unaware Heft algorithm of Pegasus WMS could not execute workflows efficiently due to wrong decision taken with assumed same bandwidth value. Bandwidth-aware workflow scheduling algorithms use the measured bandwidth and do planning accordingly and can achieve better performance for workflows having large number of tasks.

We would like to mention the observation made about concurrent data communications among jobs because the concurrent data communications could affect execution time of a workflow. During run of a workflow, two types of jobs: (i) data communication through network and (ii) execution of job on computing node can occur concurrently without affecting each other. However, if more than one data transfer job is happening on a particular Grid site, then the total available bandwidth gets shared among such multiple data transfer jobs. Such behavior was observed for the workflow of 41 tasks with large data communication due to having a large number of parallel tasks for which data communication jobs targeted to a single site happened concurrently. However, despite this limitation, it was observed that the bandwidth-aware workflow scheduling algorithms performed better than bandwidth-unaware workflow scheduling algorithms.

We would like to discuss two important aspects: (i) scheduling efficiency of the proposed bandwidth-aware workflow scheduling and (ii) applicability regarding use of the proposed approach for experimentations of real scientific workflows. The major time consuming part in the implementation of the proposed bandwidth-aware workflow scheduling is delay incurred due to use of NWS. Existing workflow scheduling of Pegasus WMS creates the concrete workflow in a few seconds, which varies depending upon the number of tasks and resources. However, when using our proposed approach additional time of 30 to 45 seconds is used in preparing the concrete workflow. Though the proposed approach takes some extra time in planning of the workflow, the benefit achieved in makespan reduction is much larger than the extra time spent in collecting the information provided by NWS. This extra overhead of accessing the bandwidth information in Pegasus WMS could be affected by two things: (i) mechanism used to gather bandwidth information and (ii) the number of computing nodes involved. We create a separate process for each possible pair of Grid sites. Some improvement in collecting the bandwidth information can be achieved by using C API of NWS. But use of C API of NWS requires integrating Java language with C language, as Pegasus WMS uses Java language. We do not see much problem with the number of computing nodes involved, which is related to the scalability of the approach, as our proposed approach requires to run NWS on each Grid site, not on each computing node. For example, if there are 5 Grid sites each containing 200 computing nodes making total of 1000 computing nodes, then we do not need to run NWS on the 1000 nodes. Our proposed approach requires running NWS on per Grid site basis; the responsible node for running NWS would be, generally, either Head node or dedicated storage server on each Grid site.

The second important aspect, applicability of the approach, involves getting information on input data size, output data size, and expected runtime of a job. Pegasus WMS does not provide information on input file size, output file size, and runtime of job. In our test workflows, we supply

this information in abstract workflow. In our work we do not address this issue of estimating data size and computation time, as this issue is outside the scope of our work. However, we briefly mention here how Pegasus WMS can be extended for it. It is possible to keep information of historical data of runs of jobs and use prediction techniques to get estimate of data size and computation time. Information on computation times of past runs can be extracted from kickstart [30] records. Interested readers are directed to the work in [33], which addresses estimating computation times, and the work in [34], which surveys various performance prediction systems.

7. Conclusion

To enable bandwidth-aware workflow scheduling, we developed Java access to NWS, integrated it in Pegasus WMS, and developed bandwidth-aware Heft site-selector algorithms to demonstrate the effectiveness of the proposed work. We also proposed, implemented, and evaluated a new workflow scheduling algorithm called *Level based Highest Input and Processing Weight First* (LHIPWF). From the results of various experiments performed on our Grid testbed, it was observed that bandwidth-aware Heft scheduling algorithms provided better performance than bandwidth-unaware Heft of Pegasus WMS. Furthermore, our proposed algorithm, LHIPWF, was found to perform better than other bandwidth-aware algorithms and bandwidth-unaware algorithms. From experiment results, it was found that the Speedup of large workflow was significant as compared to that of small workflow. Moreover, it was also observed that, with large intermediate data communication, the Speedup value decreases due to more time spent in performing large data communication. To summarize, the integration of bandwidth information enhances scheduling capability of Pegasus WMS and performance of workflow applications.

This paragraph provides further directions for the research work. For planning based or static workflow scheduling, such as what was done by Pegasus WMS, the tasks of workflow need to be tagged with possible size of input-output files and runtime of executables. Such information can be estimated or predicted from data of historic runs. Pegasus WMS supports job clustering/grouping based workflow scheduling algorithms. The use of bandwidth information can be exploited for making cluster or group of jobs based on bandwidth information and input/output data sizes.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The authors would like to thank Karan Vahi and Pegasus WMS team for providing answers to their various questions related to Pegasus WMS. The answers of raised questions, available tutorials on Pegasus WMS, and JavaDoc API have helped a lot in quickly digesting the internal working of Pegasus WMS.

References

[1] I. Foster and C. Kesselman, Eds., *The Grid 2: Blueprint for a New Computing Infrastructure*, The Elsevier Series in Grid Computing, Elsevier, 2nd edition, 2003.

[2] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: enabling scalable virtual organizations," *International Journal of High Performance Computing Applications*, vol. 15, no. 3, pp. 200–222, 2001.

[3] LIGO Laser Interferometer Gravitational Wave Observatory, http://www.ligo.caltech.edu.

[4] "Montage An Astrono mical Image Mosaic Engine," 2014, http://montage.ipac.caltech.edu/docs/gridtools.html.

[5] Blast Workflow, http://exon.niaid.nih.gov/cas/manual/blast-workflow.html.

[6] P. Blaha, K. Schwarz, G. Madsen, D. Kvasnicka, and J. Luitz, *"Wien2k": An Augmented Plane Wave Plus Local Orbitals Program for Calculating Crystal Properties*, Vienna University of Technology, Vienna, Austria, 2001.

[7] E. Deelman, G. Singh, M. Su et al., "Pegasus: a framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, no. 3, pp. 219–237, 2005.

[8] T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, C. Seragiotto Jr., and H. Truong, "ASKALON: a tool set for cluster and Grid computing," *Concurrency Computation Practice and Experience*, vol. 17, no. 2–4, pp. 143–169, 2005.

[9] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludäscher, and S. Mock, "Kepler: an extensible system for design and execution of scientific workflows," in *Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM '04)*, pp. 423–424, Santorini Island, Greece, June 2004.

[10] G. von Laszewski and M. Hategan, "Workflow concepts of the Java CoG Kit," *Journal of Grid Computing*, vol. 3, no. 3-4, pp. 239–258, 2005.

[11] D. Hull, K. Wolstencroft, R. Stevens et al., "Taverna: a tool for building and running workflows of services," *Nucleic Acids Research*, vol. 34, supplement 2, pp. W729–W732, 2006.

[12] I. Taylor, M. Shields, and I. Wang, "Resource management for the triana peer-to-peer services," in *Grid Resource Management*, vol. 64 of *International Series in Operations Research & Management Science*, pp. 451–462, Springer, New York, NY, USA, 2004.

[13] I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields, Eds., *Workflows for e-Science: Scientific Workflows for Grids*, Springer, 2007.

[14] M. L. Pinedo, *Scheduling: Theory, Algorithms and Systems*, Springer, New York, NY. USA, 3rd edition, 2008.

[15] F. Dong and S. G. Akl, *Scheduling Algorithms for Grid Computing: State of the Art and Open Problems*, School of Computing, Queens University, Kingston, Canada, 2006.

[16] J. Yu, R. Buyya, and K. Ramamohanarao, "Workflow scheduling algorithms for grid computing," in *Metaheuristics for Scheduling in Distributed Computing Environments*, pp. 173–214, Springer, 2008.

[17] T. Kosar and M. Balman, "A new paradigm: data-aware scheduling in grid computing," *Future Generation Computer Systems*, vol. 25, no. 4, pp. 406–413, 2009.

[18] C.-T. Yang, S.-Y. Chen, and T.-T. Chen, "A grid resource broker with network bandwidth-aware job scheduling for computational grids," in *Advances in Grid and Pervasive Computing*, vol. 4459 of *Lecture Notes in Computer Science*, pp. 1–12, Springer, Berlin, Germany, 2007.

[19] M. L. Massie, B. N. Chun, and D. E. Culler, "The ganglia distributed monitoring system: design, implementation, and experience," *Parallel Computing*, vol. 30, no. 7, pp. 817–840, 2004.

[20] R. Wolski, N. T. Spring, and J. Hayes, "Network weather service: a distributed resource performance forecasting service for metacomputing," *Future Generation Computer Systems*, vol. 15, no. 5, pp. 757–768, 1999.

[21] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Task scheduling algorithms for heterogeneous processors," in *Proceedings of the 8th Heterogeneous Computing Workshop (HCW '99)*, pp. 3–14, San Juan, Puerto Rico, USA, 1999.

[22] R. Wolski, "Dynamically forecasting network performance using the network weather service," *Cluster Computing*, vol. 1, no. 1, pp. 119–132, 1998.

[23] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, NY, USA, 1990.

[24] NetBeans IDE, http://www.netbeans.org/.

[25] "Installing GT 5.2.3," http://www.globus.org/toolkit/docs/5.2/5.2.3/admin/install/.

[26] NTP: The Network Time Protocol, http://www.ntp.org/.

[27] J. Frey, "Condor dagman: handling inter-job dependencies," Tech. Rep., Department of Computer Science, University of Wisconsin, 2002.

[28] Computing with HTCondor TM, http://research.cs.wisc.edu/htcondor/.

[29] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "Condor-g: a computation management agent for multi-institutional grids," *Cluster Computing*, vol. 5, no. 3, pp. 237–246, 2002.

[30] J. S. Vockler, G. Mehta, Y. Zhao, E. Deelman, and M. Wilde, "Kickstarting remote applications," in *Proceedings of the International Workshop on Grid Computing Environments*, 2007.

[31] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M. Su, and K. Vahi, "Characterization of scientific workflows," in *Proceedings of the 3rd Workshop on Workflows in Support of Large-Scale Science (WORKS '08)*, 10, p. 1, IEEE, November 2008.

[32] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[33] S. Krishnaswamy, S. W. Loke, and A. Zaslavsky, "Estimating computation times of data-intensive applications," *IEEE Distributed Systems Online*, vol. 5, no. 4, 2004.

[34] S. Seneviratne, D. C. Levy, and R. Buyya, "A taxonomy of performance prediction systems in the parallel and distributed computing grids," CoRR, http://arxiv.org/abs/1307.2380.