The Computer Journal Advance Access published January 27, 2012

doi:10.1093/comjnl/bxr128

Scheduling Workflow Applications Based on Multi-source Parallel Data Retrieval in Distributed Computing Networks

SURAJ PANDEY[1],* AND RAJKUMAR BUYYA[2]

[1]*CSIRO, ICT Centre, Marsfield, Australia*
[2]*Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computer Science and Information Systems, The University of Melbourne, Parkville, Australia*
**Corresponding author: suraj.pandey@csiro.au*

Many scientific experiments are carried out in collaboration with researchers around the world to use existing infrastructures and conduct experiments at massive scale. Data produced by such experiments are thus replicated and cached at multiple geographic locations. This gives rise to new challenges when selecting distributed data and compute resources so that the execution of applications is time- and cost-efficient. Existing heuristic techniques select 'best' data source for retrieving data to a compute resource and subsequently process task-resource assignment. However, this approach of scheduling, which is based only on single source data retrieval, may not give time-efficient schedules when: (i) tasks are interdependent on data, (ii) the average size of data processed by most tasks is large and (iii) data transfer time exceeds task computation time by at least one order of magnitude. In order to address these characteristics of data-intensive applications, we propose to leverage the presence of replicated data sources, retrieve data in parallel from multiple locations and thus achieve time-efficient schedules. In this article, we propose two multi-source data-retrieval-based scheduling heuristic that assigns interdependent tasks to compute resources based on both data retrieval time and task-computation time. We carry out experiments using real applications and deploy them on emulated as well as real environments. With a combination of data retrieval and task-resource mapping technique, we show that our heuristic produces time-efficient schedules that are better than existing heuristic-based techniques for scheduling application workflows.

Keywords: workflow scheduling; parallel data retrieval; data-intensive workflow applications; grid computing; cloud computing

Received 28 July 2011; revised 19 October 2011
Handling editor: Mostafa Bassiouni

1. INTRODUCTION

A growing emphasis on collaborative research has led to an increased use of distributed computing environments and the replication of experimental data for large-scale scientific applications such as the Compact Muon Solenoid (CMS) experiment for the Large Hadron Collider (LHC) at CERN, the Laser Interferometer Gravitational-wave Observatory (LIGO) data-analysis pipeline and so forth. It is important that these experiments make full use of the distributed infrastructure and the replicated data, and optimize the execution time as they carry out repeated experiments. A well-known method behind efficient execution is to model the application as a workflow and schedule the tasks in the workflow over a set of available compute resources [1–4].

Selecting compute and storage nodes based on their location in the network is a basic building block when using distributed computing environments [5]. A workflow scheduler should be able to select the data sources and parallelize the transfer of data to a compute host to optimize the overall transfer time. Similarly, the selection of the compute host, in relation to the selected set of data hosts, should be such that the execution time is minimized. We focus on these two aspects—data host and compute host selection while scheduling data-intensive scientific application workflows. We begin by describing two applications that have motivated us to further optimize the data and compute host selection while scheduling workflow applications.

Case 1: The CMS experiment still produces more than five petabytes data per year when running at peak performance. It has a large number of 'Tier-2' analysis centers where physics analyses are conducted. However, Tier-2 centers rely upon Tier-1s for access to large datasets and secure storage of the new data they produce. Tier-2 sites are responsible for the transfer, buffering and short-term caching of relevant samples from Tier-1, and transfer of produced data to Tier-1 for storage [1]. They are required to import 5 TB/day of data from Tier-1 and other data replicated at Tier-2, and export 1 TB/day (This is based on $\sim 10^8$ simulated events per year per Tier-2, multiplied by the event size and divided by the number of working days). According to James Letts, 'The ability to move and analyze data are essential to any experiment, and so far the data transfer system in CMS seems to be up to the challenge'.

Case 2: The LIGO Scientific Collaboration (LSC), currently made up of almost 700 scientists from over 60 institutions and 11 countries worldwide, is a group of scientists seeking to detect gravitational waves and use them to explore the fundamental physics of gravity. The LIGO Data Grid (LDG) has laboratory centers (Caltech, MIT, LHO and LLO) and LSC institutions (UWM, and three sites in the EU managed by the GEO-600 collaboration) offering computational services, data storage and grid computing services. The LDG uses the LIGO Data Replicator (LDR) to store and distribute data. *Input data are common to the majority of analysis pipelines, and so is distributed to all LDG centers in advance of job scheduling* [2]. The analysis of data from gravitational-wave detectors are represented as workflows. Using middleware technologies, such as Pegasus and Condor DAGMan for management, the LDG continues to manage complex workflows for its growing number of users.

Rationale: In Case 1, Tier-2 members download data from Tier-1 and also cache these data for short-term usage. In the three hypothetical 'use cases' presented in [1], scientists are continuously sharing the cached data for repeated experiments and analysis. Therefore, the presence of these replicated/cached data could be used for minimizing the transfer time, when compared with getting them from Tier-1 directly every time. This justifies the need for multi-source data transfer mechanisms. In addition, the results obtained after analysis are transferred back to Tier-1, which would then be downloaded by users from Tier-2. This back-and-forth transfer of data could also be minimized by caching/transferring the output results to specific locations, where users are active.

Similarly, in Case 2, as input data are replicated at all LDG centers, complex workflows could make use of these multiple data sources while transferring data. The intrinsic characteristic that input data are common to the majority of analysis pipelines justifies the need for replication before application execution. This in turn benefits any heuristic using multi-source retrieval techniques. Similar to Case 1, the results obtained from Case 2 could also be managed/replicated at selected sites so that scientists can retrieve data within a short period of time from these sites.

Different approaches such as the replica selection in the Globus Data Grid [6], Giggle framework [7] and combinations of these methods are used to resolve replicas in data-intensive applications. However, these replica selection services primarily select one 'best' replica per task that gives the MTT to a compute host. But for applications that have tasks with data-dependencies and multiple input files per task, selecting one 'best' replica may not always give the optimal transfer time [8, 9].

Storage and distribution services provided by storage service providers such as Nirvanix Storage Delivery Network[1] and Cloud Storage [Amazon Simple Storage Services (S3)[2]] are enabling users and scientists to store and access content from globally distributed edge servers. These content distribution networks can be used by data-intensive applications for storage and distribution. Users can then retrieve data from these multiple data hosts or edge servers (in contrast to a single 'best' storage resource) in parallel, to minimize the total transfer time. As data are transferred in segments, the transfer process is carried out in parallel when using multiple data sources. This is termed 'multi-source parallel-data-retrieval (MSPDR)' in this article. In addition to selecting data hosts, we also need to choose compute resources to transfer data and execute application tasks.

Realizing the limited efficacy of selecting one best replica, Zhou *et al.* [9] proposed several replica retrieval methods to reduce access latency, improve data locality and increase robustness, scalability and performance for distributed applications. They showed through experiments that probe-based data retrieval (described in more detail in Section 4.4) was best suited for large volumes of data in Data Grid environments. These techniques were proposed and studied in the context of Data Grids and were limited to replica retrieval.

Scheduling of workflow applications based on the multi-source parallel data-retrieval techniques has not been given much attention in the past, due to the complexity involved in selecting data resources, managing data retrievals and selecting compute resources for executing workflow tasks. In contrast to existing methods where one 'best' data sources are selected prior to the selection of compute resources, this article proposes novel algorithms that select both data and compute resources with respect to multiple data locations and computing capability of resources while using MSPDR for data transfers.

In this article, we present two scheduling heuristic that leverage multi-source parallel data-retrieval techniques.

[1]http://www.nirvanix.com.
[2]http://aws.amazon.com.

We experiment with existing (i) probe-based [9], (ii) greedy [9] and (iii) random site selection-based data retrieval techniques for retrieving data from selected data-hosts while scheduling tasks in a workflow. We also propose a tree-based approach for selecting multiple data sources during the scheduling process. We then study the effect of using MSPDR-based heuristic on the makespan (i.e. the length of the schedule) of representative data-intensive application workflows. Finally, we compare the makespan obtained by using MSPDR-heuristic against single 'best' data source selection-based heuristic.

In summary, the contributions of this paper are:

(i) A static and a dynamic scheduling heuristic that leverage multi-source parallel data-retrieval technique.
(ii) A Steiner tree-based resource selection technique for dynamic scheduling.
(iii) A thorough experimental study of the proposed scheduling algorithms on both emulated and real execution platforms using real application workflows.

2. RELATED WORK

In this section, we survey past work on replica selection, data retrieval and workflow scheduling algorithms.

Replica selection and retrieval: Vazhkudai *et al.* [6] presented the design and implementation of a high-level replica selection service in the Globus data Grid. Chervenak *et al.* [7] defined a replica location service that maintains and provides access to information about the physical locations of copies. Hu *et al.* [10] proposed the Instance-Based-Learning (IBL) algorithm for replica selection where only limited data sources are available. Their results show that IBL performs well for data-intensive Grid applications. Zhou *et al.* [9] analyzed various algorithms for replica retrieval and concluded that probe-based retrieval is the best approach, providing twice the transfer rate of the 'best' replica server. Feng *et al.* [8] proposed rFTP that improves the data transfer rate and reliability on Grids by utilizing multiple replica sources concurrently. Their *NWS Dynamic* algorithm depends on Network Weather Service (NWS) [11] deployment at all participating Grid nodes, and *NoObserve* or *SelfObserve* does not use NWS. In these works, the replica selection system seeks one 'best' replica among all available replicas. Retrieving data from the best source may result in poor performance and degraded reliability [8, 9].

When data are partitioned and distributed at various locations without full replication, a set of data-hosts that complete the required data files should be found. The selection of the optimal set of data-hosts in the presence of a large number of replicated files for a single job is computationally intensive. Venugopal and Buyya [12] selected the data-hosts by using one of the solutions to the Set-Coverage problem [13]. In this paper, we assume that data are fully replicated and hence set-coverage is guaranteed by every data source.

The Hadoop Distributed File System (HDFS) distributes data to all the nodes of a Hadoop cluster as it is being loaded into the system [14]. Large data files get split in chunks, which are managed by different nodes in the cluster. In addition, each chunk is replicated across several machines. HDFS provides the capability for retrieving data from multiple sources. However, the Hadoop framework schedules computations (MapReduce processes) in proximity to the location of the data, where the computation happens on a subset of the data. Most data (the subset) are read from the local disk, avoiding network transfers. Moving computation to the data may not be feasible for a workflow application, where data and computing nodes are geographically distributed.

Content Delivery Networks (CDN) maintain content cashing points of presence in data centers strategically located near primary Internet Exchange Points so that end users can download the data with minimal delay.[3] Seeking the best edge-server for data may reduce the transfer time; however, the best edge-server's location may not always have the desired computing capabilities to analyze the downloaded data. Moreover, there are work in CDN domains that show the benefits of using multiple edge-servers [15, 16]. Rodriguez and Biersack [17] proposed a dynamic-parallel access approach to replicated content from multiple servers or caches in CDN. They showed that users experience significant speed-ups and very consistent response times when using multiple parallel transfers. Wu and Chien *et al.* [18] reiterate the need for fetching data from multiple sites concurrently in CDN. To facilitate multipoint-to-point data transfers, they proposed Group Transport Protocol (GTP) and a receiver- based rate allocation scheme and showed that GTP outperforms other point-to-point protocols for multiple-to-point transmission.

Large data could be disseminated to the computing infrastructure and scientific communities using various data movement technologies underpinned by distribution scenarios. Woollard *et al.* [19] proposed the use of *distribution workflows* for specifying the properties of data distribution (e.g. the total volume of data transfer, delivery intervals, number of users receiving the data, etc.). They described the stages of the distribution workflow (data access, data sub-setting, movement technology selection and distribution) and used the Object Oriented Data Technology data distribution framework [20] to provide the services for data sub-setting and delivery of information across heterogeneous organizational structures. In addition, they constructed a decision-making framework that autonomously decides the appropriate data movement technology for a distribution scenario, or a class of scenarios [21]. Their method of specifying data movement parameters, constructing a workflow and dynamically selecting movement technologies fits well in the context of this paper. Similar work on mixed data delivery was proposed

[3] http://www.akamai.com/, http://aws.amazon.com/cloudfront/, http://www.gogrid.com/cloud-hosting/content-delivery-network.php.

by Alinel *et al.* [22] as part of their dissemination-based information systems toolkit. These works implicitly assume that the computing resources are pre-selected prior to constructing the data distribution workflow, which is the same as scheduling workflow tasks independent of data. However, it would be interesting to study the construction of the distribution workflow at run-time, based on the currently executing workflow statistics, such as execution time, locality of resources, etc.

Multi-source parallel data transfers can be much more efficient than single source transfers. It can reduce access times by transferring data from several replicas in parallel according to Yang *et al.* [23] and Feng and Humphrey [8]. GridFTP and rFTP [8] are existing tools that support these types of transfers.

Kosar and Balman [24] described the insufficiency of tradition distributed computing systems in efficiently (and reliably) accessing and transferring widely distributed data. Traditional systems emphasize on computations alone and embed data transfers in the process, causing delays in computation and data movements. They proposed a data-aware scheduler, Stork [25], that implements techniques for scheduling and optimization of data placement jobs. The data placement jobs include data transfer, storage allocation, data removal, etc. They also reported using multiple connections during data transfers between two hosts. They showed that transfer speed improved slightly when using multiple streams and issuing multiple transfers, when compared with using single-connection on wide area networks. In addition, their results show a positive impact on the transfer rate for increased concurrency than increased parallelism. Relating their findings to the work presented in this paper, we use multiple connections from multiple sources, which introduces concurrency in data transfers. The more the number of data sources, the higher the concurrency, but not necessarily the data transfer rate. In order to maximize the data transfer rate, while maintaining higher concurrency, we propose a novel method for selecting data sources.

Static and dynamic workflow scheduling: We focus on list-based scheduling heuristic for computing static schedules (offline); and task partitioning and iterative rescheduling for dynamic schedules (online).

Topcuouglu *et al.* [26] designed the HEFT algorithm based on list scheduling. HEFT is a static scheduling algorithm that attempts to schedule tasks on heterogeneous resources to get minimum execution time. It assigns ranks to the tasks according to estimated communication and computation costs and preserves the job execution precedence. However, the communication and task computation values are average estimates.

Sakellariou and Zhou [27] investigated the performance of the HEFT algorithm produced by different approximation methods. They concluded that the mean value method is not the most efficient choice, and the performance could differ significantly from one application to another. They also proposed a hybrid heuristic that uses a standard list scheduling approach to rank the nodes of the Directed Acyclic Graph (DAG) and then uses this ranking to form groups of tasks that can be scheduled independently. Their Balanced Minimum Completion Time (BMCT) is for scheduling independent tasks formed by using the hybrid heuristic. BMCT algorithm tries to minimize the execution time in the initial allocation, and again tries to minimize the overall time by swapping tasks between machines. We take the Hybrid and BMCT algorithm (HBMCT) for comparing with our static scheduling heuristic.

Deelman *et al.* [3] partitioned a workflow into multiple sub-workflows and allocated resources to tasks of one sub-workflow at a time based on real-time information. Shankar and Dewitt [28] proposed a planner that uses a *file_location* table to determine the locations of cached or replicated files for scheduling data-intensive workflows using the Pegasus framework.

The iterative re-computing technique keeps applying the scheduling algorithm on the yet-to-be-scheduled tasks of a workflow in execution. Sakellariou and Zhou [29] proposed a low-cost Selective Rescheduling (SR) policy by recomputing a schedule only when the delay of a carefully selected task impacts the schedule of the entire workflow. Several works have explored static and dynamic scheduling strategies for workflow execution that focused on: user quality of service and location affinity [30–32], iterative calls of static algorithms [33–35], dynamic programming [36] and so forth. Lopez *et al.* [34] explored several static and dynamic approaches for scheduling workflows and concluded that list-based heuristic significantly outperforms non-list-based heuristic. Yu *et al.* [37] described the strategies involved in scheduling workflows for Grid computing environments.

Many past works on scheduling workflows focus primarily on compute-intensive workflows. Most of these scheduling algorithms could make use of multi-source data retrieval techniques while also scheduling tasks on compute resources. However, the challenge is to use a retrieval technique while scheduling workflow applications. To the best of our knowledge, this problem has not been explored in detail in the past. This paper addressed this challenge using heuristic-based approaches.

3. PROBLEM STATEMENT

We now describe the problem of data host selection and tasks to resource mapping in the presence of a large number of replicated files for workflow applications [38].

DEFINITION 1 DTSP(D, R, T, F, G, L, M). *Given a set of data-hosts D, a set of compute resources R, a set of tasks T, a set of files F (both input and output files of T), and a graph G that represents the data flow dependencies between tasks T, the Data-Task Scheduling Problem (DTSP) is a problem of finding assignments of tasks to compute-hosts* [task_schedule = $\{t_r\}$, $t \in T$, $r \in R$], *and the partial data set* (PD^{t_r}) *to be transferred from selected data-hosts to assigned compute hosts* [data_set = $\{\{\mathrm{PD}_{d_i \to r}\}^{t_r}\ \forall d_i \in D,\ r \in R, i \leq |D|\}$] *for each task, such that: total execution time (and cost) at* r *and data transfer time (and cost) incurred by data transfers* $\{\mathrm{PD}_{d_i \to r}\}$ *are minimized for all tasks* T.

The pre-conditions are:

(i) Data files are replicated across multiple data hosts.
(ii) Each task processes more than one input data files.
(iii) Total time and cost are bounded by L and M, respectively, where L signifies deadline and M denotes maximum money (real currency) that can be spent on executing all the tasks in T

We assume (listed as pre-conditions above) that replicas of data files are present at multiple locations and can be accessed in parallel. Data-intensive applications are known to have higher values for the Communication to Computation Ratio (CCR), which is mainly due to the size and number of data files to be processed. Thus, we assume each task in a workflow to have more than one input files. The third assumption simply put constrains on the time and cost that could be spent on executing a workflow.

Figure 1 shows an example scenario for the problem given in Definition 1, where tasks are mapped to resources and partial data are retrieved from multiple data hosts. In the figure, tasks $t1$ and $t2$ are assigned to resources $r1$ and $r3$, respectively. In this static mapping, Fig. 1 shows partial data being retrieved in parallel from all data hosts $d1, d2, d3$ to the respective compute hosts where the tasks are assigned: $\{\mathrm{PD}_{d_1 \to r1}, \mathrm{PD}_{d_2 \to r1}, \mathrm{PD}_{d_3 \to r1}\}^{t_1 \to r1}$ and $\{\mathrm{PD}_{d_1 \to r3}, \mathrm{PD}_{d_2 \to r3}, \mathrm{PD}_{d_3 \to r3}\}^{t_2 \to r3}$. The objective now is to minimize the transfer time of these partial data to assigned computing resources by transferring the right amount of data from each data source for each task in the workflow.

Definition 1 states a generic problem that includes two objectives for optimization: time and cost. In this article, we propose heuristic-based scheduling techniques for the problem by limiting the number of objectives to only one: minimize the makespan. Hence, execution and data transfer cost (economic cost, measured in terms of physical currency) are assumed *zero*, i.e. $M = \$0$. Using non-zero value for *cost* as an additional parameter when carrying out multi-objective optimizations will be part of our future work.

4. SCHEDULING HEURISTIC

In this section, we begin by describing a static scheduling heuristic (also known as offline scheduling), assuming that the scheduler has advance information of the environment. For dynamic environments, we propose a Steiner Tree-based resource selection method. Using this tree, we then describe a dynamic scheduling heuristic (also known as online scheduling), where decisions are made at run-time.

4.1. Static scheduling heuristic

We propose an Enhanced Static Mapping Heuristic (ESMH), assuming that the scheduling system has advance information of tasks, compute and storage resources and network statistics at the time of scheduling, prior to execution. The following information are known:

(i) number of tasks to be scheduled;

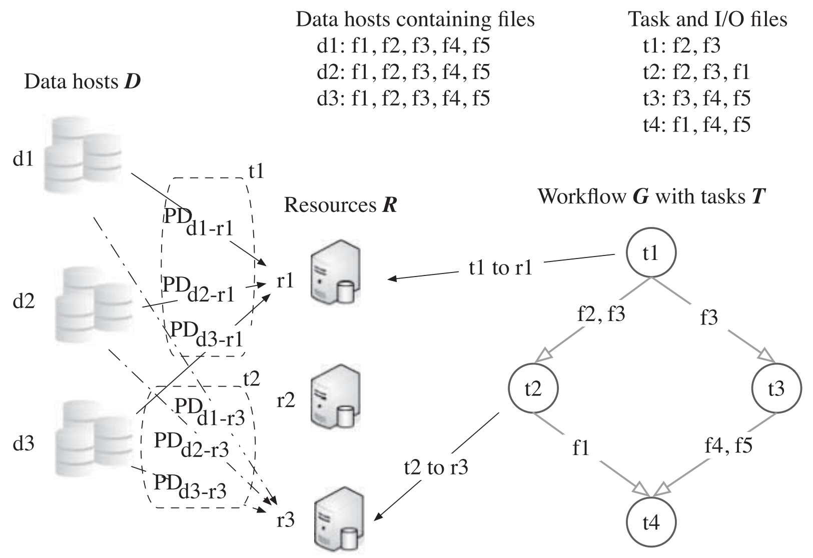


FIGURE 1. A scenario that shows partial data retrievals and task to resource mappings for a workflow in a distributed computing environment.

Algorithm 1. Enhanced static mapping heuristic (ESMH).

```
Data: Tasks $\{t_i\}$ with input files $\{f_1, \ldots, f_n\}_{ti}$,
Data: A data-resource matrix $\{M: m_{ij} = tr(f_i \rightarrow r_j)\}$
Data: Average computation time: $avg\_comp$ of $t_i$ on each $r_j$
Data: $f_k \in F$ can be fully downloaded from more than one location (replicated)
begin
1   for each task starting from the root do
2       Form resource set $\{R\}$ that has $min(tr(\{f_i\}))$ for each file $f_i$ required by task $t_i$
3       Mark $r_k \in \{R\}$ that has min. computation time for $t_i$
4       $EFT_{min\_avg} = min\_avg\_comp(t_i, r_k) + tr(\{f\}_{ti}, r_k)$
5       for each resource $r_i \in \{R\}$ do
6           $EFT_{relative} = avg\_comp(t_i, r_i) + tr(\{f\}_{ti}, r_i)$
7           if $[EFT_{relative}] < [EFT_{min\_avg}]$ then
8               Map $t_i$ to $r_i$; break;
9       if $t_i$ not assigned then
10          Mark $r_m \notin \{R\}$ that has minimum $tr$ time for the largest file required by task $t_i$
11          $EFT_{file} = min\_avg\_comp(t_i, r_m) + tr(\{f\}_{ti}, r_m)$
12          if $[EFT_{file}] \leq [EFT_{min\_avg}]$ then
13              Map $t_i$ to $r_m$
            else
14              Map $t_i$ to $r_k$ /* last option */
15      Update resource availability information based on task-resource mapping
end
```

(ii) estimated execution time of every task on a set of dedicated resources;
(iii) maximum execution time of a task (used for task preemptions in a priority queue-based resource management system);
(iv) size of data handled by each task;
(v) earliest start time (EST) for an unscheduled task on any given resource;
(vi) resource characteristics: CPU MHz, memory;
(vii) average network bandwidth available between resources at the time of scheduling (based on prediction).

Data-resource matrix: Every task $t_k \in T$ processes a set of input files to produce output files, all in the set $\{f_1, \ldots, f_n\}_{tk}$ $\forall f_i \in F$.[4] A data-resource matrix M for a task t_k stores the average time required for each file f_i, or a set of files $\{f_i\}$, to be transferred to a resource (r_j) at a location ij (i corresponds to a file, j corresponds to a resource in the Matrix M) and this value is $m_{ij} = \text{tr}(f_i \rightarrow r_j)$. The m_{ij} values must be calculated/estimated in advance by using partial file transfer mechanisms, e.g. probe, random or greedy. While scheduling application workflows, we use probe-based retrieval to estimate the value of m_{ij} as this method performed better than greedy and random methods (see Section 5.4.4 for experimental results).

[4]It is assumed, the input and output files for each task are known.

In static mapping heuristic (e.g. HEFT, HBMCT, which are described in Section 2), it is a common practice to compute or estimate these transfer times using access logs, prediction models or real executions. This matrix is similar to a meta-data catalog that provides transfer times of files between resources. Our approach is different than the meta-data catalogs as data are transferred from multiple locations in parallel using either probe- or greedy-based retrieval technique.

Resource selection: We select compute resources based on the Earliest Finish Time (EFT) value we calculate for a task on a resource. This EFT value is calculated by adding the estimated computation time of a task on a resource $min\_avg\_comp(t_i, r_k)$ $\forall t_i \in T; r_k \in R$ and the estimated transfer time of total data to the resource $\text{tr}(\{f\}_{ti}, r_k)$ (applicable to steps 4, 6 and 11 in Algorithm 1). To select the resource that has the minimum EFT for a task, we rely on external information, usually obtained by using an Estimated Time to Compute (ETC) matrix, user supplied information, task profiling, analytical benchmarking, as mentioned by Briceno *et al.* [39].

Heuristic: Algorithm 1 lists the ESMH. We start task-resource mapping by selecting tasks in a workflow on a level-by-level basis.

As the workflow is represented as a DAG, it has many topological orderings. We select tasks such that each node comes before all nodes to which it has outbound edges.

We create a topological order over the directed graph, with arbitrary tie-breaking in case of partial order. The node that gets selected first, after the tie-breaking if any, is termed as the *root* node in Algorithm 1.

For each task, we first form a set of compute resources $\{R\}$ by selecting only those resources that have Minimum Transfer Time (MTT) $\min(\mathrm{tr}(\{f_i\}))$ for each input file f_i of a task t_i (step 2). The transfer time value $\mathrm{tr}(\{f_i\}, \mathrm{resource}) = m_{ij} = m[\mathrm{file}, \mathrm{resource}]$ (used in steps 4, 6, and 11 in Algorithm 1) is obtained from the *data-resource* matrix for all the resources. We then form a compute resource set $\{R\}$ that contains resources having MTT for each input file for all the tasks in the workflow. Among these resources, we mark a resource $r_k \in \{R\}$ that has minimum computation time for the task t_i (step 3). Next, we estimate the EFT value of the task ($\mathrm{EFT}_{\mathrm{min\_avg}}$) (step 4) by adding the average of the minimum Execution Time (ET) given by a resource for the task and the transfer time of all input files required by the task to that resource. This EFT value is an average value since we take the average of all the minimum computation time (min_avg_comp) of tasks on that resource. We assume the minimum computation time of every task varies on a resource as the size of input files vary.

Next, we calculate the EFT value of the task t_i for all the resources in the resource set $\{R\}$. This EFT given by each resource is termed as $\mathrm{EFT}_{\mathrm{relative}}$ (step 6). $\mathrm{EFT}_{\mathrm{relative}}$ is different than $\mathrm{EFT}_{\mathrm{min\_avg}}$ as the EFT value is relatively dependent (convolution relationship) on data transfer and on average computation, whereas $\mathrm{EFT}_{\mathrm{min\_avg}}$ is dominated by the *minimum* value of execution time given by a resource. We then compare the $\mathrm{EFT}_{\mathrm{min\_avg}}$ value against the $\mathrm{EFT}_{\mathrm{relative}}$ (step 7). If we find a resource that has the $\mathrm{EFT}_{\mathrm{relative}}$ value lesser than the $\mathrm{EFT}_{\mathrm{min\_avg}}$, then we assign the task to the resource that has smaller $\mathrm{EFT}_{\mathrm{relative}}$ (step 8).

If none of the resources in the set $\{R\}$ have EFT values lower than the $\mathrm{EFT}_{\mathrm{min\_avg}}$ (step 9), we compute the EFT based on the file size. We choose a resource $r_m \notin \{R\}$ such that it has MTT value for the largest input file of the task t_i (step 10). We then compute the $\mathrm{EFT}_{\mathrm{file}}$ based on this resource r_m (step 11). The task is then assigned to the resource that has minimum EFT value (either r_m or r_k) (step 12–14). We could search for optimal $\mathrm{EFT}_{\mathrm{relative}}$, but it would be computationally not feasible for large resource set $\{R\}$ (discussed further in Section 4.1.2).

Rationale: The formation of a bounded compute resource set $\{R\}$ ensures that only limited, but reasonable candidate resources are selected from a pool of a large number of resources (step 2). The resources in this set are selected based on the transfer time of input files of each task. As we are considering data-intensive workflows, we focus on minimizing data transfer time (i.e. $\mathrm{EFT}_{\mathrm{relative}}$, $\mathrm{EFT}_{\mathrm{file}}$) over task computation time (i.e. $\mathrm{EFT}_{\mathrm{min\_avg}}$) (step 4). Thus, the heuristic maps tasks to resources based on the size of data. If all the input files of a task have higher values of transfer time than its averaged minimum computation time, the task is assigned to the resource that has minimum $\mathrm{EFT}_{\mathrm{relative}}$ value (step 5–step 8). If only some of the input files of a task have higher values of transfer time than computing time, the task is assigned to the resource that has minimum $\mathrm{EFT}_{\mathrm{file}}$ value (step 9–step 13). If the averaged minimum computation time of a task outweighs the transfer time of input files, the task is assigned to the resource that gives minimum $\mathrm{EFT}_{\mathrm{min\_avg}}$ value (step 14). Hence, in a workflow all the tasks get an equal share of resources depending on their data and computation requirements.

ESMH is different than HEFT, HBMCT and existing static heuristic as: (a) it evaluates task schedules based on multi-source file transfer times to a resource (b) it manages task to resource mapping based on both data-transfer and computation requirements, (c) it uses only selected resources in contrast to all available resources and (d) it balances tasks to resource mappings based on both transfer time and computation time. We illustrate this with an example as follows.

4.1.1. *Examples of workflow*

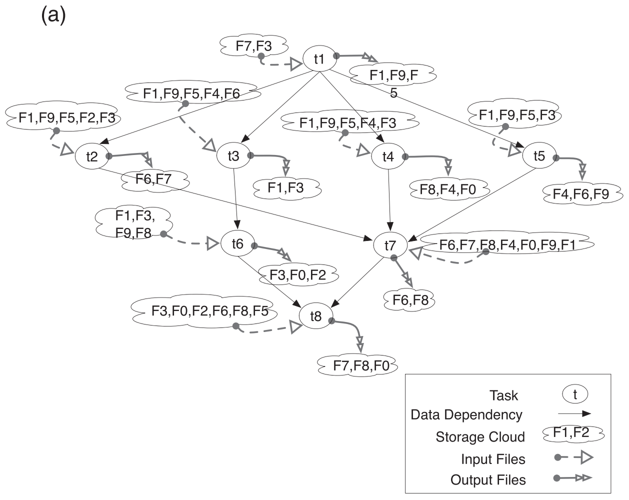
Figure 2 shows an example of a workflow with input and output files and data dependencies between tasks. The table in Fig. 2c shows the schedule length produced by using ESMH listed in Algorithm 1. In this example, ESMH uses values from pre-computed matrices. These matrices are: (a) one that stores values of average computation time of each task on each resource (Fig. 2a and b); (b) one that stores values of transfer time of each data-file from distributed data-centers to each resource (Fig. 2c) based on probe-based multi-source parallel retrieval technique. As ESMH is an offline heuristic, these matrices are computed before the heuristic operates.

In this example, we have three resources $\{R1, R2, R3\}$ and eight tasks $t1$ to $t8$. Each task operates on several input files to produce output files. We take the example of mapping of task $t3$. The schedule given by each resource *R1*, *R2* and *R3* is $(12 + 3917)$, $(4 + 2571)$ and $(21 + 3184)$ seconds, respectively. As resource *R2*'s available time is after task $t1$ finishes execution, the minimum EFT is given by resource *R3*, hence the mapping.

4.1.2. *Optimality of static heuristic*

The mapping of tasks to compute resources is an *NP-complete* problem in the general form [40]. The problem is *NP-complete* even in two simple cases: (i) scheduling jobs with uniform weights to an arbitrary number of processors and (ii) scheduling jobs with weights equal to one or two units to two processors [41]. Hence, there exists heuristics-based approaches for scheduling workflow applications, and ESMH is one of them.

One computationally expensive way to generate the optimal schedule (for offline scheduling) is by evaluating the mapping of each task on all resources and selecting only the time-efficient mappings. Instead of making all such possible comparisons, the ESMH listed in Algorithm 1 reduces the number of comparisons by selecting the candidate resources that could provide minimum computing and transfer time for a given



An example workflow.

(b)

Tasks	R1	R2	R3
t1	8	23	40
t2	8	41	43
t3	12	4	21
t4	29	39	49
t5	35	8	22
t6	2	27	16
t7	26	43	29
t8	3	4	12

Task execution time on resources.

(c)

Files	R1	R2	R3
F0	10	341	1438
F1	376	1351	1344
F2	401	45	108
F3	375	243	942
F4	1005	158	1022
F5	845	379	105
F6	256	240	12
F7	475	1315	639
F8	1441	1259	727
F9	1435	443	701

Multi-source file transfer time based on probe-based parallel retrieval

(d)

Tasks	R1	R2	R3	start	finish
t1	-	680	-	0	680
t2	384	-	-	680	1064
t3	-	-	4224	680	4904
t4	4266	-	-	1064	5330
t5	510	-	-	5330	5840
t6	8698	-	-	5850	14538
t7	-	12145	-	5840	17985
t8	-	20833	-	17985	38818

Makespan = 38818

A schedule generated by using ESMH

FIGURE 2. An example workflow, matrices with estimated values and a schedule generated by using ESMH.

task. This results in the time complexity of the algorithm to be $O((n+e) \times m)$ in the worst case, where n is the number of tasks, e is the number of edges and m is the number of compute resources. The time complexity of HEFT algorithm is equal to $O(n^2 \times m)$ [42].

HEFT produces better schedules when the CCR is significantly small. However, when CCR is large, the communication cost becomes dominant and HEFT performs worse, even for values of CCR as low as 0.1 [43]. The performance gain is more notable in ESMH for larger CCRs.

The static heuristic we have proposed would be most appropriate if all our assumptions listed in Section 4.1 could be realized in practice. However, in a distributed environment where resources are shared among a large number of users, the estimates recorded in the matrices will suffer from large deviations from the values at the time of task execution. Hence the static estimates of computation and communication time cannot be relied upon for time-efficient scheduling of applications that have long-running tasks with large data-sets to process. This limitation prevents us from forming the initial resource set $\{R\}$ in Algorithm 1. To circumvent this limitation, we propose a Steiner-Tree-based resource selection and apply it for online scheduling, as described in the following section.

4.2. A Steiner tree

Definition. *A Steiner Tree problem can be defined as: Given a weighted undirected graph $G = (V, E)$, and a set S subset of V, find the Minimum-Cost tree that spans the nodes in S.*

$G = (V, E)$ denotes a finite graph with a set V of vertices and the set E of edges. A weight w defines a number $w(e) \in R_0^+$ associated with each edge e, i.e. $w : E \to R_0^+$. In particular, the weight $d : E \to R_0^+$, and $c : E \to R_0^+$ represent the delay and the cost of the link, respectively. A *path* is a finite sequence of non-repeated nodes $p = (v_0, v_1, \ldots, v_i)$, such that, $0 < i \leq k, k = |V|$, there exists a link from v_i to $v_{(i+1)} \in E$. A link $e \in p$ means that path p passes through link e. The delay and cost of a path p are thus given by: $d(p) = \sum_{e \in p} d(e)$, and $c(p) = \sum_{e \in p} c(e)$.

A spanning tree T of a graph G whose length is the shortest among all spanning trees is called a minimum spanning tree for G. A Steiner-Minimal-Tree (SMT) for a set of points is a minimum spanning tree, where a finite set of additional vertices V_A is introduced into the space in order to achieve a minimal solution for the length of the path p. An illustration of these two trees is depicted in Fig. 3.

In order to approximate the length of the path p, we assume the delay and cost of a path are in a metric space and can be combined with the distance function ρ. Let (X, ρ) be a metric space. That means, X is a nonempty set of points and $\rho : X^2 \mapsto \Re$ is a real-valued function, called a metric, satisfying the triangle inequality.

$G = (V, E)$ is embedded in (X, ρ) in such a way that V is a set of points in X and E is a set of unordered pairs vv' of points $v, v' \in V$. For each edge vv' a length is given by $\rho(v, v')$. Hence, we define the length of the graph G in (X, ρ) as the total length of G:

$$L(G) = L(X, \rho)(G) = \sum_{vv' \in E} \rho(v, v'). \tag{1}$$

Thus, the SMT is a tree that connects vertices in V with additional vertices V_A to lower the path length ρ. Even though the Internet can be regarded as a non-metric space, where the network flow is constantly changing in time and the triangle inequality may not hold, the assumption that ρ is in metric space highly simplifies the problem of constructing the tree and hence selecting vertices when compared with when using ρ in a non-metric space.

We are motivated to use ρ in metric space as relating geographic coordinates to round-trip-times (RTT) works well in practice [44]. Freedman *et al.* [44] showed a strong linear correlation between geographic distance and RTT across all PlanetLab hosts assuming geographic distance is a good indicator of network distance. In contrast, if ρ was taken in non-metric (i.e. real network delays), we would have to probe each and every resource continuously and exhaustively to establish the network distance between resources. This not only increases the amount of probings per request but also makes the results susceptible to changing network dynamics. As opposed to static geographic locations, exhaustive probing would change the interconnection of resources in the Steiner tree and affect the structure of the tree in two ways: (a) if the resources are distributed across large physical distances (e.g. Fig. 4c), the tree would have minimal changes in terms of the interconnection of nodes, and (b) if the resources are physically located relatively close to each other (e.g. Fig. 4a and b), the nodes would reconnect according to the updated network distance. Thus, taking ρ in metric space reduces the amount of probings and also keeps the structure of the tree close to the real world network topology. The following section describes the validation of the Steiner-tree-based network topology on three real networks.

4.2.1. *Forming a Steiner tree*

The SMT problem is NP-hard; so polynomial-time heuristic are desired [45]. The Bounded Shortest Multicast Algorithm (BSMA) is a very well-known delay-constrained minimum-cost multicast routing algorithm that shows excellent performance in terms of generated tree cost, but suffers from high time complexity [46]. In this paper, we use the *incremental optimization heuristic* developed by Dreyer and Overton [45]. Even though it does not give an optimal solution, we get a feasible solution at any point in time.[5]

The time complexity of constructing SMT with minimal length for a finite set of points N in the metric space (X, ρ) depends on $n = |N|$ and, the time taken to compute $\rho(x, y)$ for any point $(x, y) \in V$ of the space. The definition of the distance function in terms of the delay and cost of a path p is

$$\rho(v, v') = w_1 d(p) + w_2 c(p). \tag{2}$$

The weights w_1, w_2 are considered as a measure of the significance of each objective in the distance function of Equation (2). We could have obtained a Pareto optimal solution by choosing the right combination of w_1 and w_2, which minimizes the distance. But, we are interested in reducing the time complexity of the overall process. Hence, we leave the values of these weights ($delay$ and $cost$) to the user to select at runtime. Moreover, even a random choice (but within acceptable bounds) of delay values (keeping the cost a constant in this article) helps achieve the objective of our problem when compared with using a single source.

[5]http://www.nirarebakun.com/graph/emsteinercli.html.

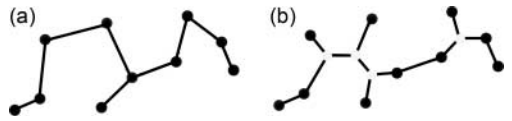


FIGURE 3. Two ways of connecting vertices that minimizes total path length: (**a**) Minimum spanning tree; (**b**) Steiner tree.

4.2.2. *Why a Steiner tree?*

Our main objective is to connect multiple data sources to a compute resource for transferring data in parallel so that we could minimize the total transfer time. While using parallel transfers could speed up the process, we also need to optimize the total network bandwidth (path of transfers). We could construct a minimum spanning tree to connect all the data sources together. But, the resulting tree is not the *smallest* tree. We demonstrate this in Fig. 3a by connecting arbitrary vertices using a minimum spanning tree.

The Steiner tree is distinguished from the minimum spanning tree in that we are permitted to construct or select intermediate connection points to reduce the cost of the tree. In Fig. 3b, the vertices are connected using the minimum Steiner tree approach, which has lower path length than the minimum spanning tree. It can be shown that the optimal group minimum spanning tree is *at most twice* as long as the optimal group Steiner minimal tree [47].

One another reason to use Steiner tree is for finding out the candidate vertices that have maximum connectivity with its neighboring vertices. For instance, the intermediate connection points (four added vertices) in Fig. 3b are all candidate nodes from where computing resources could be selected for task executions. The neighboring nodes could then be used as multiple data sources for transferring data to the intermediate computing node. The minimum spanning tree approach does not identify the candidate nodes by itself.

4.3. Steiner-tree-based resource selection and multi-source data retrieval

The problem of selecting multiple data sources can be handled by forming a SMT. In our formulation of the Steiner tree problem, the vertices V represent both data D and compute R sources (as noted in Section 3) and the links E represent the network connection between them in the graph G.

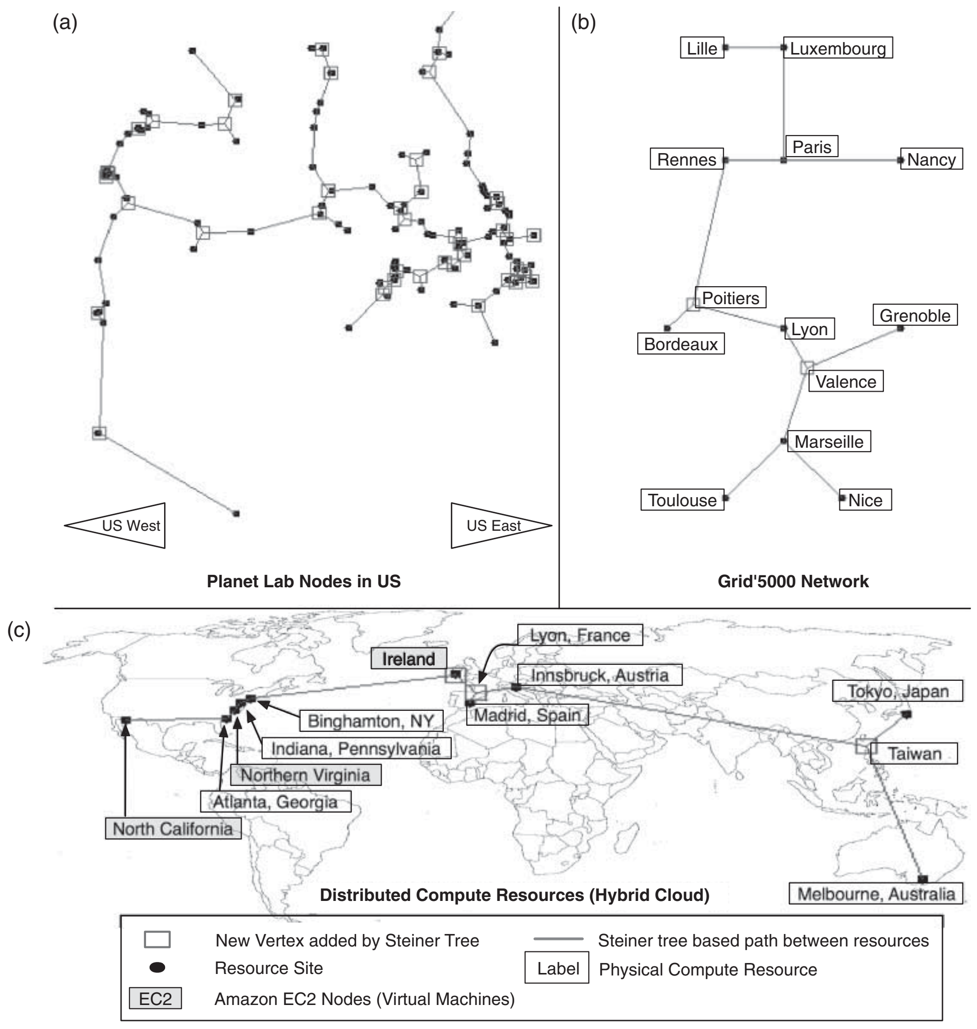


FIGURE 4. A Steiner tree constructed for planet lab nodes, Grid‘5000 network and distributed resources available for our experiment.

The additional vertices V_A represent nodes around V that are not data or compute sources, but would minimize the path length if the communication network connects through them.

In order to validate our implementation of the Steiner-tree-based resource selection method, we constructed the tree on three independent networks: Planet Lab nodes in the US, the Grid‘5000 network and compute resources from research labs around the world. The research labs that shared their resource for our experiment are: Binghamton University, Victorian Partnership for Advanced Computing, InTrigger at University of Tokyo, Georgia State University, DPS group at University of Innsbruck and Complutense University of Madrid. The distribution of these resources matches the resources used in the CMS experiment. These trees are shown in Fig. 4. The trees are SMTs constructed out of vertices scattered in an Euclidean plane—the coordinates of these points on the plane are the latitudes and longitudes of the cities where resources are present. A tree connects the vertices with lines such that the distance between the points in the plane is minimal. If there are vertices that fall on the minimal path, the tree routes through them without the need of additional vertices, as in the Planet Lab network (boxes with a marked dot). Additional vertices are added (boxes without a marked dot) where nodes do not fall in the path of the tree, as in the other two networks. The trees drawn using the Euclidean plane are close enough to the real world network connections between the resources (e.g. Grid‘5000), thus validating our implementation of Steiner trees. In addition to validation tests, we use the distributed compute resources depicted in Fig. 4c when conducting real experiments (see Section 5).

In Fig. 4, there are vertices (existing and added) that have in-degree (in_d: the number of edges coming into a vertex in a graph) of two and three. A higher value of in_d signifies the number of connections a vertex can make for parallel data retrieval. For e.g. if a vertex $v \in V$ has $\text{in}_d = 3$ with vertices $v_1, v_2, v_3 \in V$, a resource located at/nearby v can retrieve data from these three

Algorithm 2. Enhanced dynamic mapping heuristic (EDMH).

```
Data: A Tasks $\{t_i\}$ with input files $\{f_1, \ldots, f_n\}_{ti}$
Data: Earliest Start Time (EST) for a task on each $r_j$
Data: $f_k$ can be sourced from more than one location (replicated)
begin
1     Construct a Steiner tree using all available resources
2     Get $N$ compute resources $\{R\}_N$ with highest value of $in_d$
3     for each task $t_i$ starting from the root do
4         Set minimum Start Time $(minST) = \infty$
5         for each resource $r_i \in \{R\}_N$ do
6             Probe each connected neighbour $r_i^{neighbor}$ of $r_i$ for calculating instantaneous bandwidth (max of $in_d$ probes)
7             Split input files based on probe values: $\{f^{split-1}, \ldots, f^{split-in_d}\}_{ti} \in \{f\}_{ti}$
8             Estimate total transfer time $tr(\{f\}_{ti}, r_i)$ for transferring split files $\{f^{split}\}_{ti}$ from each $r_i^{neighbor}$ to $r_i$
9             $StartTime(ST) = EST(t_i, r_i) + tr(\{f\}_{ti}, r_i)$
10            if $(ST \leq minST)$ then
11                $minST = ST, minCR = r_i$
12        Assign $t_i$ to the $minCR \Rightarrow$ the $r_i$ that gives minimum $ST$
13        Wait for $polling\_time$
14        Update the $ready\_list$
15        Distribute output data of task $t_i$ to resources that host the files required by successors of $t_i$
end
```

data sources with minimal path length:

$$L(X, \rho) = \sum_{i=1, v_i v \in E}^{i=3} \rho(v_i, v).$$

In the Planet Lab network, there are several nodes (dots with square) that have an in-degree of three. In the case of the Grid‘5000 network, Paris and Marseille possess compute nodes that are each connected to three other sites around them. If Poitiers and Valence were to host compute nodes, these sites would also have an in-degree of three. Thus, we first construct a Steiner tree on a network and select resources that have the highest value of in-degree in_d. This selection procedure is then used for dynamic mapping heuristic.

4.4. Dynamic mapping heuristic

In this section, we describe the Enhanced Dynamic Mapping Heuristic (EDMH) using the Steiner tree-based resource selection as described in Section 4.2. The EDMH is listed in Algorithm 2.

EDMH is an online heuristic where tasks are assigned to resources based on resource load and network latency values available at runtime. Unlike static heuristic (or offline heuristic), EDMH does not estimate or use the average computation time of tasks, but instead relies on the EST provided/forecast by resources. In addition, EDMH selects an initial pool of resources based on the Steiner Tree-based selection method.

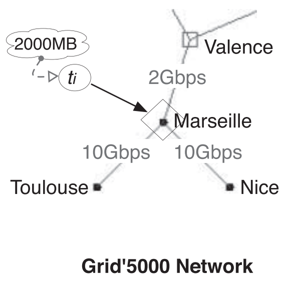
Pre-Scheduling: We construct a Steiner tree using all the available resources. The tree helps us identify resources that are well connected and thus have high in_d. These resources form a set of candidate resources $\{R\}_N$, which are later chosen for executing tasks. We construct the tree using a metric space (X, ρ)[6].

Scheduling: EDMH is a list-based scheduling heuristic, listed in Algorithm 2. We maintain a *ready_list* (step 16), where tasks are added as they become available for scheduling. In dependent-task scheduling, child tasks become ‘ready’ only when their parents have successfully completed execution. The *ready_list* is filled by the scheduling loop, starting from the root task, as tasks are scheduled and get completed.

Initially, every ready task’s minimum Start Time (ST) is set to a high value (step 6). This time will be set to a wall-clock time later as tasks are assigned to available resources. Before any task can start execution, we assume that data required by the task must be available at the assigned resource. The downloading of a task’s input data to a resource depends on the total time it takes for multi-source parallel retrieval of data. To determine this time, we use a *probe-based* approach to estimate the instantaneous bandwidth between connected resources (step 8) and hence estimate the approximate time it would take to download input data from multiple resources for every task (step 10).

The instantaneous bandwidth is different from the maximum bandwidth (or maximum throughput) of a network access.

[6]Using a non-metric space also validates our heuristic as long as we can define the distance function $\rho(v, v')$ and the path length $L(G)$.



Total Instantaneous BW = 22 Gbps
Proportional Share of BW: 10/22, 10/22, 2/22 (45.5%, 45.5%, 9%)
Total Input File Size for a task t_i : 2000 MB
Consider Task t_i is at site: Marsille

fsplit-toulouse = 45.5% of 2000 MB = 910 MB
fsplit-nice = 45.5% of 2000 MB = 910 MB
fsplit-valence = 9.0% of 2000 MB = 180 MB

Minimum Transfer Time (for the largest segment 910 MB): 0.71 secs

Calculation of Partial Downloads

FIGURE 5. An example showing the partitioning of input files in proportion to the available instantaneous bandwidth.

The latter is calculated as *TCP Window Size/Round-trip time*, which is more representative in wide-area networks and gives the gross bit rate that is transferred physically. The instantaneous bandwidth however is the application level throughput that excludes protocol overhead and retransmitting data packets, and is generally lower than both the maximum throughput and network access connection speed. As the receiving end will have limitation on the maximum throughput/bandwidth it can use while downloading/uploading data (due to multiple transfers, hardware, etc.), we used the instantaneous bandwidth instead of maximum throughput for approximating the transfer time. This is simply an approximation and depends on the size of the file transferred, the time taken to transfer and the load on the network, all of which vary in time.

As each resource $r_i \in \{R\}_N$ is connected with multiple neighboring nodes $\{r_i^{\text{neighbor}}\}$ in the network (the Steiner tree helps in the identification of these connections based on the in_d), we probe these neighboring nodes to determine instantaneous bandwidth available (step 8). Based on this value, we split each input file among the n_d resources connected to r_i in proportion as: $\{f^{\text{split}-1}, \ldots, f^{\text{split}-\text{in}_d}\}_{\text{ti}} \in \{f\}_{\text{ti}}$ (step 9). This split will enable parallel download of the respective segments from each of the connected neighbors to the resource where a task is scheduled. For example, in Fig. 5, if a task ti were to be scheduled at Marseille, input files of total size 2000 MB were split into three segments (as $n_d = 3$ for Marseille): fsplit-toulouse, fsplit-nice and fsplit-valence. These segments are then downloaded to Marseille in parallel from the three neighboring sites. In this example, entire input files can be downloaded from all the three sites. The MTT of 0.71 seconds depends on the size of the largest segment (910 MB in the example). While this segment is being downloaded, we assume the smaller segments will have finished downloading, so that we can approximate the value of ST. We take this minimum time as the total transfer time $tr(\{f\}_{\text{ti}}, r_i)$ and add it to the EST of the task $\text{EST}(t_i, r_i)$ to obtain the value of ST (step 11). The transfer time function $\text{tr}(\{f\}_{ti}, r_i)$ is analogous to the path length $L(G)$ of the metric Steiner tree. Similarly, the value of instantaneous bandwidth resembles the distance function $\rho(v, v')$ (see Section 4.2).

We assign a task t_i to the resource that projects the minimum ST based on our estimations (step 14). The scheduler then waits for a duration defined by the delay: $polling\_time$ (step 15). In online scheduling, $polling$ is necessary as the scheduler needs to update the status of completed/failed tasks so that, after this delay, it can update and schedule ready tasks.

We partition and distribute the output files produced by each task to those sites that host files needed by the immediate successors of the task producing the output (step 17). This distribution step ensures that these output files can be downloaded from multiple sites that are also hosting the input files of the child tasks. This distribution should ensure that neighboring hosts of the candidate resources $\{R\}_N$ have all the segments to complete the entire file when downloaded at a resource, using any replication algorithm [48].

While scheduling workflows there exists more than one task that can be scheduled independently of one another. Since our workflows are data-intensive in nature, the transfer time are dominant when compared with the computation time. This gives rise to the possibility that the majority of tasks are assigned to a single or only few compute resources. This occurs only when tasks have more than one input file in common and these files are only available from limited number of resources. In such cases, grouping these tasks to form a batch task and submitting to a resource reduces data-transfer time.

5. PERFORMANCE EVALUATION

We have evaluated proposed heuristic techniques by two methods: (i9) using emulation, where the network was virtualized, and (ii) real environment. First we describe the performance metric, application workflows and data locality, which are common to both the experiments.

5.1. Performance metric

We used *average makespan* as a metric to compare the performance of heuristic-based approaches on the network topology and workload distribution for both emulation and real environment. *Average makespan* for every heuristic is computed by taking an average of all the makespan produced by the heuristic for an application workflow, under a setting. During

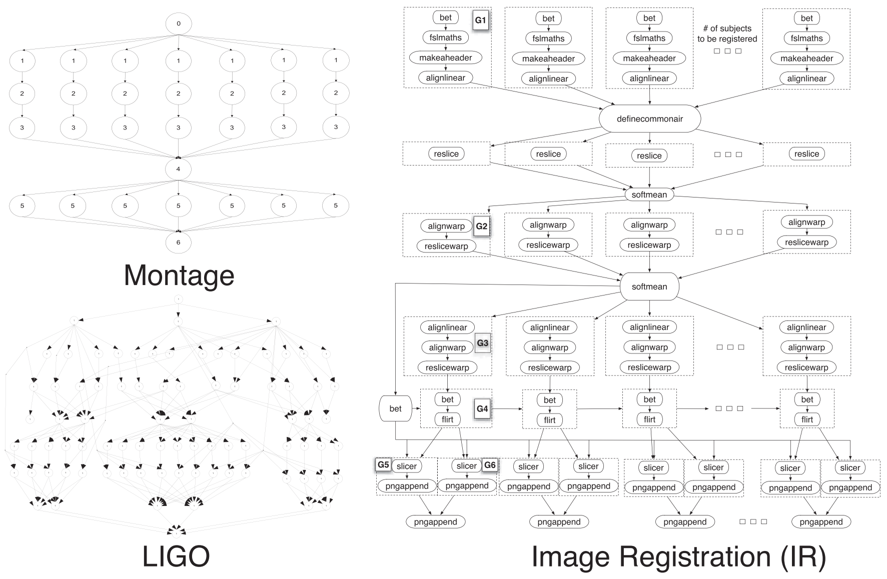


FIGURE 6. Application Workflow types: Balanced (Montage), complex (LIGO) and hybrid (IR) workflows.

the scheduling process, the value of makespan estimated (for static heuristic) and obtained (for dynamic heuristic) differ for every experiment per application workflow due to the loss model (based on a normal distribution) attached to each link between stub domains in the emulation settings, which in turn changes the data transfer estimations. Similarly, when experimenting on real platforms, the change in network loads would change the actual makespan of the workflow applications. Thus, averaging the makespan obtained from experimental runs provides a mean value that is closer to the actual value. The smaller the value of the average makespan, the better the heuristic performs. We provide standard errors as a means for comparing our algorithms, as indicated in Figs 9 and 13.

5.2. Application workflows

We used three types of workflows: balanced, complex and hybrid, as depicted in Fig. 6. Figure 6-Montage depicts a workflow similar to the Montage Workflow [49]. In this type of workflow structure, tasks are symmetrically distributed in the graph and can be easily partitioned/separated into levels according to their dependencies. Figure 6-LIGO represents a complex workflow similar to a subset of the workflow used to analyze data from the second LIGO science run for binary inspirals [50]. In this type of workflow, tasks cannot be partitioned into levels easily as they have links (data dependencies) to/from tasks across levels. Figure 6-IR represents a real application workflow used for Image Registration (IR) for fMRI applications [51], which has both balanced and complex structures. Group of pipelined tasks form a balanced structure that can be easily partitioned into levels, whereas some parts are complex.

The makespan of a workflow is highly dependent on the structure of the workflow (data-dependencies between tasks). Random selection of data-sources at any level may increase the data transfer time, delaying the ST of child tasks, which in turn increases the makespan.

We experimentally recorded the makespan for all the three types of workflows to determine: (a) the relationship between makespan and the workflow structures, and (b) the effect of multi-source retrieval technique-based scheduling on the makespan of all workflow structures. We also checked the effect of using multi-source data retrieval technique on both static and dynamic scheduling approaches.

5.3. Data locality

For experimenting with greedy retrieval technique, we segmented each file and distributed them uniformly to the number of resources used. The maximum and minimum file segment size for our experiment varied between 0.5 and 500 Mb. We manually configured at least 30% of the resources to have all the segments of 50% of the files, for each workflow type. For experimenting with the probe-based retrieval, we replicated all the files in all the resources.

5.4. Emulation-based evaluation

Here, we list the intrinsic components of our emulation setup: the network topology, compute and storage resources, and the design of the emulation platform.

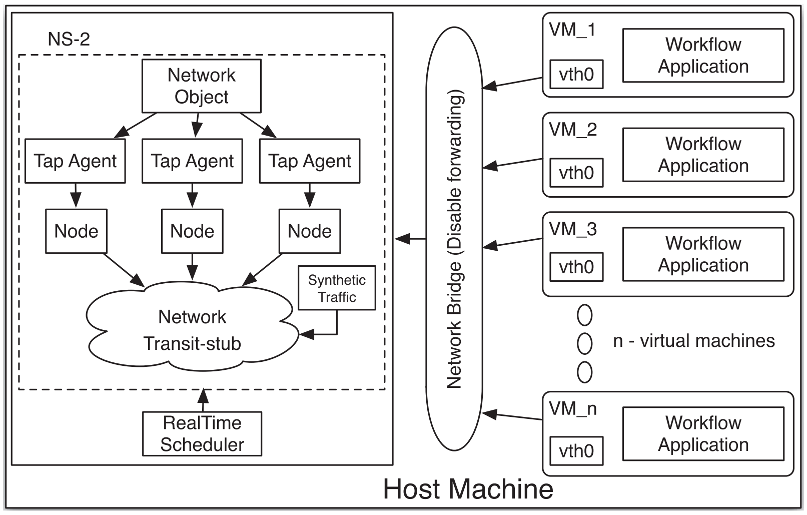


FIGURE 7. Experiment design using NS-2-based emulation.

5.4.1. *Emulation setup*

We used NS-2[7] as the emulation tool. For simulating the network connecting resources, we constructed a dense network topology by interconnecting *ns* nodes. As an emulator, we injected workflow execution traffic into the simulator and emitted packets on to live network using NS-2's real-time scheduler. Figure 7 depicts the architecture of our emulation platform. The virtual network connects a set of User-mode Linux (UML [52]) VMs, all running on a single physical machine. VMs are connected via an Ethernet bridge in the host machine using a virtual interface. The network bridge is configured such that it blocks all the packets forwarded through it, and passes the traffic through the network defined in NS-2. We mapped each VM to a *ns* node, by using the correspondence between the Ethernet addresses of the VMs and the network layer addresses of the NS-2 nodes [53].

Quetier *et al.* [54] compared four virtual machine (VM) technologies (Vserver, Xen, UML and VMware) in the context of large-scale emulation. We used UML-based virtualization mainly for its low CPU and network overheads and ease of integration with NS-2. The 20 virtual nodes running on a single machine had minimal impact on the performance of the applications. We used our Workflow Engine (WFE) [51] as a workflow execution and scheduling engine for executing the workflows. All the scheduling heuristic are implemented in the WFE. We allocated one VM for running the WFE and another for hosting the *NWS* nameserver and memory. We used NWS for monitoring the network's bandwidth and latency.

5.4.2. *Network topology*

We used the GT-ITM internetwork topology generator to generate random graphs.[8] GT-ITM is a topology modeling package that produces graphs that look like wide-area Internet. We used the Transit-Stub network model, where hierarchical graphs are generated by composing interconnected transit and stub domains (see [55] for more details).

We attached our VM to one node in the hierarchical network, such that the node was in a stub domain. We fixed the number of *ns* nodes to 100, with average node degree at 3.5 and 50% asymmetry in the network links.

For the IR workflow depicted in Fig. 6, we recorded estimates of execution time and data transfer time of each task on compute resources provided by Grid'5000. We refer the reader to our paper that focuses on IR experiment [51] on Grid'5000. However, for Montage and LIGO workflows, we used random execution times and data sizes. The execution times of each task on every machine were randomly generated from a uniform distribution in the interval [10, 50] seconds. To maintain higher values of CCR, we chose each file size in the interval [1, 1000] MB. These random values, once computed, remained fixed throughout the experiment.

5.4.3. *Storage and compute resources*

Since we are not concerned about sub-millisecond response times, which are common in the case of transactional processing systems and not in scientific workflows, we have assumed our data centers to have characteristics similar to those of an Internet-based storage service provider. In our experiment, the data passes through the Internet and suffers delays as any other normal traffic. This delay is incorporated in the network topology modeled using NS-2 using the synthetic traffic and loss model.

We created synthetic non-real traffic in NS-2 using UDP constant bit rate (CBR) and exponential traffic generators. The real-time traffic from the VMs passed through the simulated network and suffered from mixing with non-real time traffic in

[7] http://www.isi.edu/nsnam/ns.
[8] http://www.cc.gatech.edu/projects/gtitm.

the links to create congestion. All links had a delay of 10 ms. In order to produce losses, we attached a loss model based on a normal distribution to each link between stub domains. However, we did not load the VMs with additional workload, besides the executing workflows.

For our emulation, we used UML-based VMs for both compute resources and storage servers. The total number of VMs running at one instance was limited to 50, half the size used in Quetier's experiment [54]. We assigned 20 of these VMs as storage resources and the remaining as compute resources. The number of storage nodes remained fixed while the number of compute nodes running was changed based on the type of workflow (Montage, LIGO or IR) being executed. Figure 8 summarizes the parameters associated with the emulation.

5.4.4. *Experimental results*

Comparison of retrieval techniques: Data can be retrieved from multiple sources using: greedy, probe or random retrieval techniques. To choose between one of these technologies, we compared the data transfer time (excluding the processing time) of files of different sizes using each technique, as depicted in Fig. 9. The average data transfer times obtained using these techniques were close to the results listed by Zhou *et al.* [9]. As the size of files increased, the random method gave the worst and the greedy gave the best transfer times on our emulated network topology. However, the greedy method suffered from high processing time.

To compare the processing times of these techniques, we computed the ratio of the processing time over transfer time, depicted as a percentage in Fig. 10 (upper half). By processing time, we mean the total time spent for (a) numerous repeated connections to hosts due to the large number of segments per file, (b) overheads in maintaining the transfer threads for each segment, (c) repeated retrievals of data segments due to intermittent failures (significant factor) and (d) time taken to combine segments to form a single data file. Our results showed that the overall time taken when using probe-based retrieval technique was less than the greedy-based retrieval even though the latter gave a better transfer time. Also, the probe method gave lower transfer-time than the random/single-source-based retrieval, in addition to the lower overheads than the greedy-based retrieval. We obtained the total transfer time (T) and overheads (processing time (P)) for the retrieval methods on all the three types of workflows, as depicted in Fig. 10.

As the workflow structure becomes complex (from WF type 1 to WF type 3), both the transfer time and the processing time increased for greedy and random-based retrievals, as depicted in Fig. 10 (upper half). However, probe-based retrieval had minimum overheads when compared with the other two methods but gave higher transfer time (T), which in turn made its ratio P/T lower. This experiment demonstrated that both transfer time and overheads needed consideration before choosing a retrieval method for complex workflows. Thus, using probe-based data retrieval for complex workflows (with large number of files) was better in terms of time and complexity than using greedy/random/single-source retrievals. Hence, we

	Montage Workflow	LIGO Workflow	IR Workflow
Workflow	Balanced	Complex	Hybrid
Number of tasks	200	200	164 (20 Subjects, grouped tasks)
Execution Time/Task	[10, 50] sec	[10, 50] sec	[1, 2656] sec
File Size/Task	[1, 1000] MB	[1, 1000] MB	[0.8, 80] MB
Storage Resources	20	20	20
Compute Resources	< 30	< 30	< 30
Number of Data hosts containing 100% of file	6 (greedy) 20 (probe, random)	6 (greedy) 20 (probe, random)	6 (greedy) 20 (probe, random)
Synthetic Traffic	UDP + Exponential	UDP + Exponential	UDP + Exponential
Network Loss Model	Normally distributed	Normally distributed	Normally distributed

FIGURE 8. A table summarizing parameters used in the emulation.

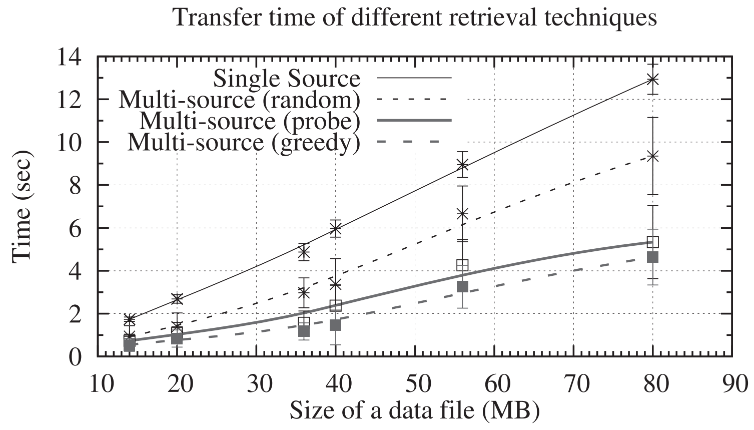


FIGURE 9. Comparison of transfer time and error margins between single-source and multi-source data retrieval techniques using data files of different sizes (files are from IR Workflow experiment only) (Bézier curve fitting used).

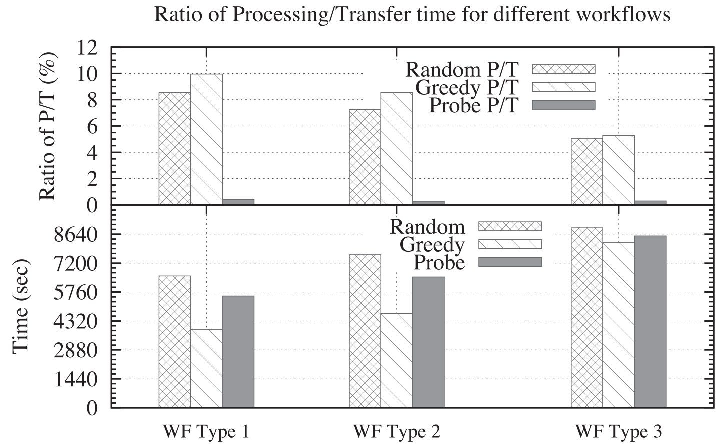


FIGURE 10. Average data transfer and segment-processing time of all files using random, greedy, and probe-based retrieval techniques for three workflows (WF 1, WF 2 and WF 3 correspond to Montage, LIGO and IR workflows in Fig. 6, respectively).

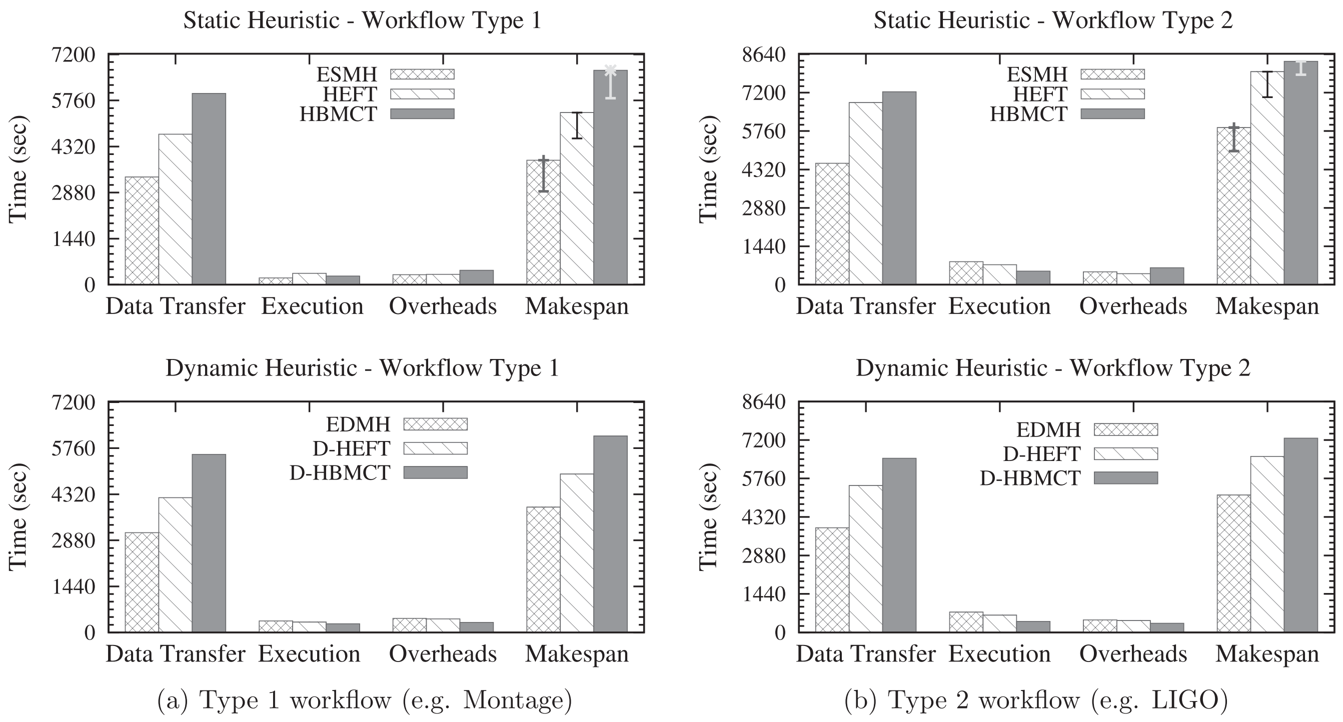


FIGURE 11. Makespan of Montage and LIGO (Type 1 and 2) workflows when using static and dynamic heuristic.

used probe-based data retrieval technique in our heuristic for its advantage over greedy and random techniques.

Comparison of heuristic approaches: We compared the static and dynamic approaches in turn.

Static approaches: We executed the workflows on our emulated platform, based on the static mappings given by our heuristic, to obtain the actual makespans. The makespans for real execution had higher values than their corresponding static estimates, as the estimated transfer time was lower than the actual transfer time on the emulated network. As the network was subjected to synthetic non-real traffic load (CBR and exponential traffic generators) during the executions of the workflows, the total data transfer time varied considerably from their estimates at the time of scheduling. We depict the static estimates of the makespan generated by all the static heuristic as the lower bound of the vertical lines in Figs 11 and 12. Each makespan is the addition of data transfer time, task execution time and overheads. For all the three types of workflow, ESMH estimated minimum makespan (lower value of the vertical lines). When executed on our emulated environment, the actual

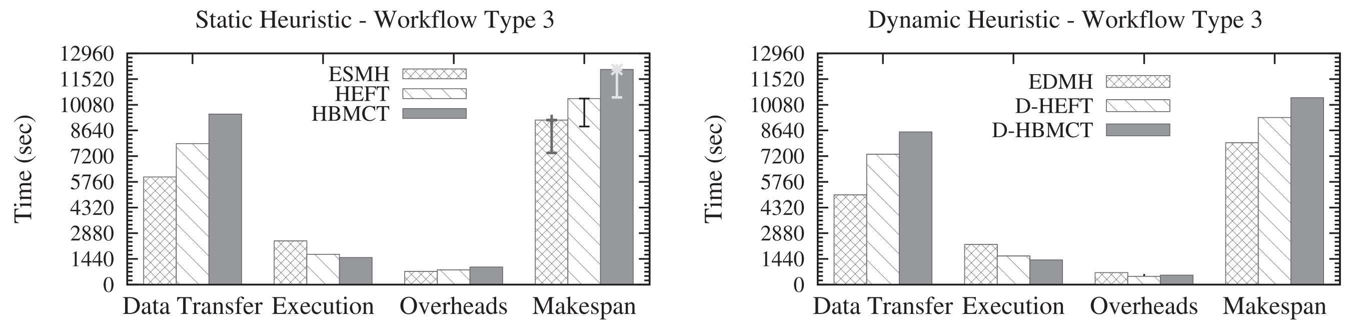


FIGURE 12. Makespan of IR workflow (type three) using static and dynamic heuristic.

makespan recorded for ESMH was lower than HEFT and HBMCT algorithms.

Dynamic approaches: In Figs 11 and 12, we also depict the makespans generated by each dynamic mapping heuristic. EDMH confirms its superiority in generating minimum makespan when compared with the 'dynamic' versions of the HEFT and HBMCT algorithms for all the three types of workflows. However, dynamic heuristic show mixed results when compared with the makespans given by the corresponding static heuristic.

For symmetric workflows (Fig. 6-Montage), the makespans generated by static heuristic are similar to that generated by their corresponding dynamic mapping heuristic. This is mainly due to the structure of the workflow: in Fig. 6-Montage there are only two tasks that download output files from more than one parent, while other tasks download the files from their immediate parent. As a result, both the scheduling heuristic try to schedule the pipelined tasks to the same resource to avoid file transfers between resources. This resulted in transfer time being similar for both the static and dynamic heuristic as depicted by the data transfer time components of the makespan in Fig. 11 for workflow type 1 (Montage).

However, for both complex and hybrid workflows (Fig. 6-LIGO,IR), the makespans generated by dynamic mapping heuristic were at least 5% less than that generated by their corresponding static heuristic. This is entirely due to the reduction in total transfer time when using a dynamic scheduling approach. As the static approach estimated the bandwidth between resources at the scheduling time (not at run-time) for all the tasks, it was lower than the actual makespan recorded after execution. The dynamic approach scheduled each task at runtime by probing bandwidth right before dispatching the task to resources for execution. This difference can be easily seen when comparing the data transfer time component of makespan in Figs 11 and 12 for workflow type 2 (LIGO) and type 3 (IR).

Dynamic heuristic performed better even when we considered resource load to be fairly constant when compared with changing network bandwidth. In cases where resource usage changes randomly, static scheduling approaches may perform even worse than dynamic approaches.

In Figs 11 and 12, in addition to the overall makespan, we also compared the individual components of the schedule, namely the data transfer time, task execution time and the processing time of the scheduling approaches. With the use of the multi-source data retrieval technique, both ESMH and EDMH achieved minimum data transfer times for all the types of workflows when compared with other heuristic. However, both ESMH and EDMH performed poorly when scheduling tasks to resources based on task execution time alone. HEFT and HBMCT performed better than our heuristic in terms of execution time, with HBMCT giving better results on average. As HBMCT tries to schedule independent tasks to optimize minimum completion time, it has better estimates for task executions. The average scheduling overhead of HBMCT was higher than all other static heuristic for all the three types of workflows. ESMH and HEFT were comparable in scheduling overhead. In the case of dynamic heuristic, EDMH gave better makespans than both D-HEFT and D-HBMCT even though it suffered from higher scheduling overheads. Clearly, dynamic approaches produced better makespans for all the three types of workflows; IR workflow benefiting the most in terms of total data-transfer time.

However, when the data transfer time was added into the makespan, ESMH and EDMH produced lower makespans than all the other static and dynamic approaches HEFT, HBMCT, and D-HEFT, D-HBMCT, respectively.

5.5. Real deployment-based evaluation

In this section, we present the results obtained using a real testbed depicted in Fig. 4c. In order to determine the feasibility of proposed heuristic in practical environments, we experimented with the IR workflow, depicted in Fig. 6-IR, using the real testbed. This application used data and scripts that were provided to us by the Dartmouth Brain Imaging Center.

5.5.1. *Experiment setup*

We deployed an experimental test-bed consisting of compute resources from worldwide research labs and Amazon EC2, as depicted in Fig. 4c. These resources were a combination of real

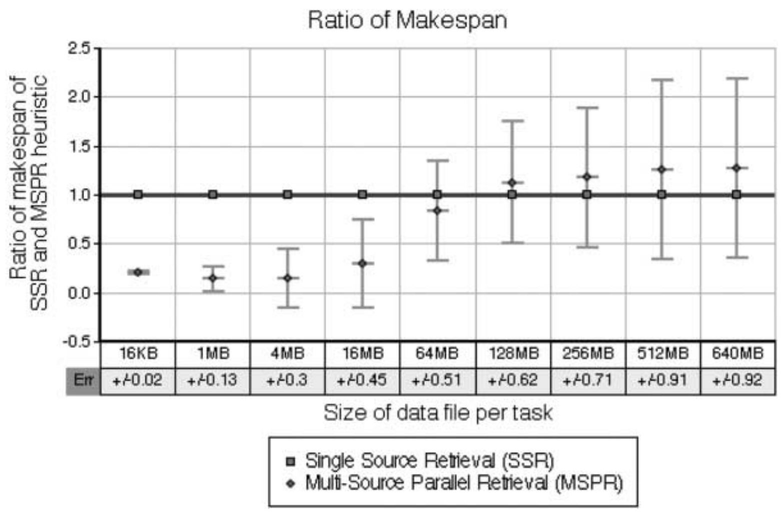


FIGURE 13. Determining the benefit of using multi-source parallel data retrieval by comparing the ratio of makespan for IR workflow (WF type three) in real environments.

compute nodes and VMs (Amazon EC2 nodes), similar to a hybrid Cloud. Figure 4 labels each resource by the name of the city where it is located. We chose to distribute these resources worldwide so that we could study the effect of locality of data on the total transfer time when using multi-source parallel retrievals.

We reserved two nodes at each location; 24 compute nodes in total, all running Linux. Each physical node had at least a dual-core 2 GHz CPU, 1 GB memory and 20 GB free disk space. Each Amazon VM was a large instance with 4 EC2 Compute units (2 virtual cores with 2 EC2 compute units each), 7.5 GB memory and 850 GB local storage. We used the IR application workflow with data distribution described in Section 5.3.

The nodes were all connected via the Internet. To approximate the network topology, we used each node's location (latitudes and longitudes) and constructed the Steiner tree (Fig. 4c). Using the Steiner tree, we could identify the in-degrees of each node: nodes at Lyon and Taiwan had in-degree of three; nodes at Atlanta, North Virginia, Indiana, Binghamton, Ireland and Innsbruck had in-degrees of two. Thus, these nodes were the candidate resource set $\{R\}_N$ in EDMH (Algorithm 2) with value of $\text{in}_d \geq 3$, $\text{in}_d \geq 2$, respectively.

5.5.2. *Experimental results*

We executed the IR workflow consisting of 20 subjects on the reserved compute resources using the EDMH. Typically, when the input file size is 16 MB per task, the total size of data handled by a 20-subject IR workflow exceeds 12 GB [51]. We varied the input file size for each task from 16 KB to 640 MB and iterated the experiment for eight times for each input size (Fig. 13) using D-HEFT and EDMH, in turns. We then calculated the ratio of makespan given by dynamic heuristic when using single source (D-HEFT) and multi-source parallel data retrieval (EDMH) techniques, as given by Equation (3):

$$\text{Makespan Ratio} = \frac{(\text{Makespan}_{\text{D-HEFT}})}{(\text{Makespan}_{\text{EDMH}})}. \quad (3)$$

Figure 13 plots the average values of the ratio from Equation (3) for varying file sizes for the IR workflow. It also plots the standard error[9] about the mean values. Based on eight measurements, the ratio was 0.25 ± 0.02 and 1.25 ± 0.92 for 16 KB and 640 MB of data per task, respectively. Greater than unity values of the ratio clearly showed that the makespan given by EDMH was smaller than that given by D-HEFT. However, the ratio is greater than unity only for file sizes 128 MB and above, clearly indicating the relationship of retrieval technique to data sizes.

The results obtained in Fig. 13 is in conformity with that obtained in the emulation. When the size of data was small (16 KB $\ll$ 128 MB), the overheads of using multi-source parallel retrievals resulted in higher values of makespan for EDMH. However, when size of data was increased ($\geq$128 MB), the EDMH started producing better makespan than D-HEFT. This was because the transfer time for a large amount of data was significantly higher than the overheads and EDMH reduced the transfer time more than D-HEFT.

The change in bandwidth between the resources and intermittent failures caused higher than expected deviation in the real experiment results when compared with the emulated version. This resulted in higher values of standard error, as reported in Fig. 13. Also, the break-even point (the intersection of the two lines) in Fig. 13 occurred for file sizes much higher in value ($\geq$128 MB) than the results we obtained in our emulation.

[9]The standard error is calculated by dividing the standard deviation by the square root of number of measurements.

This can be attributed to the largely distributed settings of our experimental platform. However, both the experiments showed that multi-source retrieval technique reduces the total data transfer time, and hence makespan, for data-intensive workflow applications.

6. CONCLUSIONS AND FUTURE WORK

In this article, we proposed two workflow scheduling heuristic that leverage multi-source parallel data-retrieval techniques. We showed that probe-based data retrieval from many resources (multi-source) produces better transfer times and hence better makespans for data-intensive workflows than selecting one 'best' storage resource for both static and dynamic scheduling methods. In static scheduling heuristic, we used a probe-based approach to select candidate sources, whereas in dynamic scheduling, we applied Steiner-tree-based multiple resource selection technique to enable multi-source parallel retrievals. We compared the makespans produced by our heuristic against that produced by both static and dynamic versions of the HEFT and HBMCT algorithms for three different types of workflows in an emulated network environment. To determine the feasibility of our approach, we also carried out experiments using a real execution platform. The results obtained from both emulation and real experiments consistently showed that makespan of workflows decreases significantly when using multi-source parallel data retrieval techniques while scheduling workflows. From our experimental results, we also conclude that, on average, EDMH produces more time-efficient makespan than the HEFT, HBMCT, D-HEFT and D-HBMCT algorithms for data-intensive workflows.

As part of our future work, we would like to constrain the resources and network bandwidth based on pricing (similar to the pricing model of CDN, SDN and storage Clouds) and propose a multi-objective (time and cost) scheduling techniques.

ACKNOWLEDGEMENTS

We are also thankful to: Binghamton University, Victorian Partnership for Advanced Computing, University of Tokyo, Georgia State University, University of Innsbruck and Complutense University of Madrid, for providing access to their computing resources. We thank James Dobson for providing us data and scripts for the fMRI IR workflow. We thank Surya Nepal, William Voorsluys and Rodrigo Calheiros for their comments on improving the article.

FUNDING

This research is partially supported by research grants from the Australian Research Council (ARC) and the Department of Innovation, Industry, Science and Research. Some experiments were carried out using the Grid'5000 experimental test bed. We thank Amazon for providing us educational grant for using EC2 and S3 resources.

REFERENCES

[1] CERN-LHCC-2005-023 (2005) CMS Computing: Technical Design Report. CMS Collaboration. Geneva, Switzerland.

[2] LIGO-T0900389-v1 (2009) The LSC-Virgo White Paper on Gravitational Wave Data Analysis. The LSC-Virgo Data Analysis Working Groups.

[3] Deelman, E. *et al.* (2005) Pegasus: a framework for mapping complex scientific workflows onto distributed systems. *Sci. Program.*, **13**, 219–237.

[4] Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E.A., Tao, J. and Zhao, Y. (2006) Scientific workflow management and the kepler system: research articles. *Concurrency Comput. Pract. Exp.*, **18**, 1039–1065.

[5] Wong, B., Slivkins, A. and Sirer, E.G. (2005) Meridian: A Lightweight Network Location Service Without Virtual Coordinates. *Proc. 2005 Conf. Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM 2005)*, Philadelphia, PA, USA, August 21–26, pp. 85–96. ACM, New York, NY, USA.

[6] Vazhkudai, S., Tuecke, S. and Foster, I. (2001) Replica Selection in the Globus Data Grid. *Proc. 1st Int. Symp. Cluster Computing and the Grid (CCGrid 2001)*, Brisbane, Australia, May 15–18, pp. 106–113. IEEE Computer Society Press.

[7] Chervenak, A. *et al.* (2002) Giggle: a Framework for Constructing Scalable Replica Location Services. *Proc. 2002 ACM/IEEE Conf. Supercomputing*, Baltimore, MD, USA, November 16–22, pp. 1–17. IEEE Computer Society Press.

[8] Feng, J. and Humphrey, M. (2004) Eliminating Replica Selection—Using Multiple Replicas to Accelerate Data Transfer on Grids. *Proc. 10th Int. Conf. Parallel and Distributed Systems (ICPADS 2004)*, Newport Beach, CA, USA, pp. 359–366. IEEE Computer Society Press.

[9] Zhou, X., Kim, E., Kim, J.W. and Yeom, H.Y. (2006) Recon: A Fast and Reliable Replica Retrieval Service for the Data Grid. *Proc. 6th IEEE Int. Symp. Cluster Computing and the Grid (CCGrid 2006)*, Singapore, May 16–19, pp. 446–453. IEEE Computer Society Press.

[10] TR-2004-03 (2004) *IBL for Replica Selection in Data-Intensive Grid Applications*. The University of Chicago, Chicago, IL, USA.

[11] Wolski, R., Spring, N.T. and Hayes, J. (1999) The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Gener. Comput. Syst.*, **15**, 757–768.

[12] Venugopal, S. and Buyya, R. (2006) A Set Coverage-Based Mapping Heuristic for Scheduling Distributed Data-Intensive Applications on Global Grids. *Proc. 7th IEEE/ACM Int. Conf. Grid Computing (GRID 2006)*, Barcelona, Spain, September 28–29, pp. 238–245. IEEE Computer Society Press.

[13] Balas, E. and Padberg, M.W. (1972) On the set-covering problem. *Oper. Res.*, **20**, 1152–1161.

[14] Borthakur, D. (2007) *The Hadoop Distributed File System: Architecture and Design*. The Apache Software Foundation.

Free Implementation of the Design of Google File System and MapReduce.

[15] Pallis, G. and Vakali, A. (2006) Insight and perspectives for content delivery networks. *Commun. ACM*, **49**, 101–106.

[16] Park, K. and Pai, V. (2004) Deploying Large File Transfer on an http Content Distribution Network. *Proc. USENIX Workshop on Real, Large Distributed Systems (WORLDS '04)*, San Francisco, CA, USA, December 5. USENIX Association, Berkeley, CA, USA.

[17] Rodriguez, P. and Biersack, E.W. (2002) Dynamic parallel access to replicated content in the internet. *IEEE/ACM Trans. Netw.*, **10**, 455–465.

[18] Wu, R.X. and Chien, A.A. (2004) Gtp: Group Transport Protocol for Lambda-Grids. *Proc. 4th IEEE/ACM Int. Symp. Cluster Computing and the Grid (CCGrid 2004)*, Chicago, IL, USA, April 19–22, pp. 228–238. IEEE Computer Society Press.

[19] Woollard, D., Medvidovic, N., Gil, Y. and Mattmann, C.A. (2008) Scientific software as workflows: from discovery to distribution. *IEEE Softw.*, **25**, 37–43.

[20] Mattmann, C.A., Crichton, D.J., Medvidovic, N. and Hughes, S. (2006) A Software Architecture-Based Framework for Highly Distributed and Data Intensive Scientific Applications. *Proc. 28th Int. Conf. Software Engineering (ICSE 2006)*, Shanghai, China, May 20–28, pp. 721–730. ACM, New York, NY, USA.

[21] Mattmann, C.A., Woollard, D. and Mahjourian, R. (2007) Software Connector Classification and Selection for Data-Intensive Systems. *Proc. 2nd Int. Workshop on Incorporating COTS Software into Software Systems: Tools and Techniques (IWICSS 2007)*, Minneapolis, MN, USA, pp. 4–4. IEEE Computer Society Press.

[22] Altinel, M., Aksoy, D., Baby, T., Franklin, M., Shapiro, W. and Zdonik, S. (1999) Dbis-toolkit: adaptable middleware for large scale data delivery. *SIGMOD Rec.*, **28**, 544–546.

[23] Yang, L., Schopf, J.M., and Foster, I. (2005) Improving Parallel Data Transfer Times Using Predicted Variances in Shared Networks. *Proc. 5th IEEE Int. Symp. Cluster Computing and the Grid—Volume 2 (CCGrid 2005)*, 9–12 May, pp. 734–742. IEEE Computer Society Press.

[24] Kosar, T. and Balman, M. (2009) A new paradigm: data-aware scheduling in grid computing. *Future Gener. Comput. Syst.*, **25**, 406–413.

[25] Kosar, T. and Livny, M. (2004) Stork: Making Data Placement a First Class Citizen in the Grid. *Proc. 24th Int. Conf. Distributed Computing Systems (ICDCS 2004)*, Tokyo, Japan, March 23–26, pp. 342–349. IEEE Computer Society Press.

[26] Topcuouglu, H., Hariri, S. and Wu, M.-y. (2002) Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.*, **13**, 260–274.

[27] Sakellariou, R. and Zhao, H. (2004) A Hybrid Heuristic for Dag Scheduling on Heterogeneous Systems. *Proc. 13th Int. IEEE Heterogeneous Computing Workshop (HCW 2004)*, Santa-Fe, NM, USA, April 26. IEEE Computer Society Press.

[28] Shankar, S. and DeWitt, D.J. (2007) Data Driven Workflow Planning in Cluster Management Systems. *Proc. 16th Int. Symp. High Performance Distributed Computing (HPDC 2007)*, Monterey, CA, USA, June 27–29, pp. 127–136. ACM, New York, NY, USA.

[29] Sakellariou, R. and Zhao, H. (2004) A low-cost rescheduling policy for efficient mapping of workflows on grid systems. *Sci. Program.*, **12**, 253–262.

[30] Brandic, I., Pllana, S. and Benkner, S. (2006) An approach for the high-level specification of qos-aware grid workflows considering location affinity. *Sci. Program.*, **14**, 231–250.

[31] Bhat, V., Parashar, M. and Klasky, S. (2007) Experiments with In-Transit Processing for Data Intensive Grid Workflows. *Proc. 8th IEEE/ACM Int. Conf. Grid Computing (GRID 2007)*, Austin, TX, USA, September 19–21, pp. 193–200. IEEE Computer Society Press.

[32] Barga, R.S., Fay, D., Guo, D., Newhouse, S., Simmhan, Y. and Szalay, A. (2008) Efficient Scheduling of Scientific Workflows in a High Performance Computing Cluster. *Proc. 6th Int. Workshop on Challenges of Large Applications in Distributed Environments (CLADE 2008)*, Boston, MA, USA, June 23, pp. 63–68. ACM, New York, NY, USA.

[33] Prodan, R. and Fahringer, T. (2005) Dynamic Scheduling of Scientific Workflow Applications on the Grid: A Case Study. *Proc. 2005 ACM Symp. Applied Computing (SAC 2005)*, Santa Fe, NM, March 13–17, pp. 687–694. ACM, New York, NY, USA.

[34] Lopez, M.M., Heymann, E. and Senar, M.A. (2006) Analysis of Dynamic Heuristics for Workflow Scheduling on Grid Systems. *Proc. 5th Int. Symp. Parallel and Distributed Computing*, Timisoara, Romania, July 6–9, pp. 199–207. IEEE Computer Society Press.

[35] Yu, Z. and Shi, W. (2007) An Adaptive Rescheduling Strategy for Grid Workflow Applications. *Proc. 21th Int. Parallel and Distributed Processing Symp. (IPDPS 2007)*, Long Beach, CA, USA, March 26–30, pp. 1–8. IEEE Computer Society Press.

[36] Prodan, R. and Wieczorek, M. (2010) Bi-criteria scheduling of scientific grid workflows. *IEEE Trans.Autom. Sci. Eng.*, **7**, 364 –376.

[37] Yu, J., Buyya, R. and Kotagiri, R. (2008) Metaheuristics for Scheduling in Distributed Computing Environments. In *Workflow Scheduling Algorithms for Grid Computing*, pp. 173–214. Springer, Berlin, Germany.

[38] Pandey, S. and Buyya, R. (2008) Scheduling of Scientific Workflows on Data Grids. *Proc. 8th IEEE Int. Symp. Cluster Computing and the Grid (CCGrid 2008)*, Lyon, France, May 19–22, pp. 548–553. IEEE Computer Society Press.

[39] Briceno, L.D., Oltikar, M., Siegel, H.J. and Maciejewski, A.A. (2007) Study of an Iterative Technique to Minimize Completion Times of Non-Makespan Machines. *Proc. 21th Int. Parallel and Distributed Processing Symp. (IPDPS 2007)*, Long Beach, CA, USA, March 26–30, pp. 1–14. IEEE Computer Society Press.

[40] Zhang, F., Cao, J., Hwang, K. and Wu, C. (2011) Ordinal Optimized Scheduling of Scientific Workflows in Elastic Compute Clouds. *Proc. 3rd IEEE Int. Conf. Cloud Computing Technology and Science (CloudCom 2011)*, Athens, Greece, Nov 29–Dec 1. IEEE Computer Society Press.

[41] Ullman, J.D. (1975) Np-complete scheduling problems. *J. Comput. Syst. Sci.*, **10**, 384–393.

[42] Ilavarasan, E. and Thambidurai, P. (2007) Low complexity performance effective task scheduling algorithm for heterogeneous computing environments. *J. Comput. Sci.*, **3**, 94–103.

[43] Shi, Z. and Dongarra, J.J. (2006) Scheduling workflow applications on processors with different capabilities. *Future Gener. Comput. Syst.*, **22**, 665–675.

[44] Freedman, M.J., Lakshminarayanan, K., and Mazières, D. (2006) OASIS: Anycast for Any Service. *Proc. 3rd Conf. Networked Systems Design & Implementation—Volume 3 (NSDI 2006)*, San Jose, CA, USA, May 8–10. USENIX Association, Berkeley, CA, USA.

[45] Dreyer, D.R. and Overton, M.L. (1998) Two heuristics for the euclidean steiner tree problem. *J. Global Optim.*, **13**, 95–106.

[46] Salama, H.F., Reeves, D.S., and Viniotis, Y. (1997) Evaluation of multicast routing algorithms for real-time communication on high-speed networks. *IEEE J. Sel. Areas Commun.*, **15**, 332–345.

[47] Robins, G. and Zelikovsky, A. (2009) Minimum steiner tree construction. In Alpert, C.J., Mehta, D.P. and Sapatnekar, S.S. (eds), *The Handbook of Algorithms for VLSI Physical Design Automation*, CRC Press.

[48] Qiu, L., Padmanabhan, V.N., and Voelker, G.M. (2001) On the Placement of Web Server Replicas. *Proc. 20th Annual Joint Conf. IEEE Computer and Communications Societies (INFOCOM 2001)*, Anchorage, AK, USA, April 22–26, pp. 1587–1596.

[49] Jacob, J., Katz, D., Prince, T., Berriman, G., Good, J. and Laity, A. (2004) The Montage Architecture for Grid-Enabled Science Processing of Large, Distributed Datasets. *Proc. 4th Annual Earth Science Technology Conf. (ESTC 2004)*, Palo Alto, CA, USA, June 22–24. Jet Propulsion Laboratory, National Aeronautics and Space Administration, Pasadena, CA, USA.

[50] Taylor, I.J., Deelman, E., Gannon, D.B. and Shields, M. (2006) *Workflows for e-Science: Scientific Workflows for Grids*. Springer, New York, Inc., Secaucus, NJ, USA.

[51] Pandey, S., Voorsluys, W., Rahman, M., Buyya, R., Dobson, J.E. and Chiu, K. (2009) A grid workflow environment for brain imaging analysis on distributed systems. *Concurrency Comput. Pract. Exp.*, **21**, 2118–2139.

[52] Hoxer, H.-J., Buchacker, K., and Sieh, V. (2002) Implementing a User-Mode Linux with Minimal Changes from Original Kernel. *9th Int. Linux System Technology Conference (Linux-Kongress 2002)*, Cologne, Germany, September 4–6, pp. 72–82.

[53] Mahrenholz, D. and Ivanov, S. (2004) Real-Time Network Emulation with ns-2. *Proc. 8th IEEE Int. Symp. Distributed Simulation and Real-Time Applications (DS-RT 2004)*, Budapest, Hungary, October 21–23, pp. 29–36. IEEE Computer Society Press.

[54] Quetier, B., Neri, V., and Cappello, F. (2006) Selecting a Virtualization System for Grid/p2p Large Scale Emulation. *Proc. Workshop on Experimental Grid Testbeds for the Assessment of Large-Scale Distributed Applications and Tools (EXPGRID 2006)*, Paris, France, June 19–23.

[55] Zegura, E.W., Calvert, K.L., and Bhattacharjee, S. (1996) How to Model an Internetwork. *Proc. 15th Annual Joint Conf. IEEE Computer and Communications Societies (INFOCOM 1996)*, Vol. 2, San Francisco, CA, USA, March 24–28, pp. 594–602.