Scheduling Data Intensive Workflow Applications based on Multi-Source Parallel Data Retrieval in Distributed Computing Networks

Suraj Pandey, Rajkumar Buyya*

The Cloud Computing and Distributed Systems (CLOUDS) Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia

Abstract

Many large-scale scientific experiments are carried out in collaboration with researchers and laboratories located around the world so that they can leverage expertise and high-tech infrastructures present at those locations and collectively perform experiments quicker. Data produced by these experiments are thus replicated and gets cached at multiple geographic locations. This necessitates new techniques for selection of both data and compute resources so that executions of applications are time and cost efficient when using distributed resources. Existing heuristics based techniques select 'best' data source for retrieving data to a compute resource and then carry out task-resource assignment. But, this approach of scheduling, which is based only on single source data retrieval, may not give time (and cost) efficient schedules when: 1) tasks are interdependent on data (workflow), 2) average size of data processed by every task is large, and 3) data transfer time exceeds task computation time by at least an order of magnitude. To achieve time efficient schedules, we leverage the presence of replicated data sources to retrieve data in parallel from multiple sources by incorporating the technique in our scheduling heuristic. In this paper, we proposed multi-source data retrieval based scheduling heuristic that assign interdependent tasks to compute resources based on both multi-source parallel data retrieval time and task-computation time. We carried out scheduling experiments by modeling applications from life sciences and astronomy domains and deploying them on both emulated and real testbed environments. Hence, with a combination of data retrieval and task-resource mapping technique, we showed that our heuristic can achieve 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, e-Science Applications, Grid Computing, Cloud Computing

*Corresponding author
Email addresses: spandey@csse.unimelb.edu.au (Suraj Pandey), raj@csse.unimelb.edu.au (Rajkumar Buyya)

1. Introduction

Large scale scientific experiments such as the Compact Muon Solenoid (CMS) experiment for the Large Hadron Collider (LHC) at CERN and the Laser Interferometer Gravitational-Wave Observatory's (LIGO) science runs are producing huge amount of data. Scientists around the world are collaborating to leverage the expertise of each other and also utilize the largely distributed IT infrastructure for the analysis of these data. As they carry out repeated experiments, data get replicated and are cached at multiple locations around the world. We describe each of these applications in turn in relation to quantity of data produced/analyzed and locality of data.

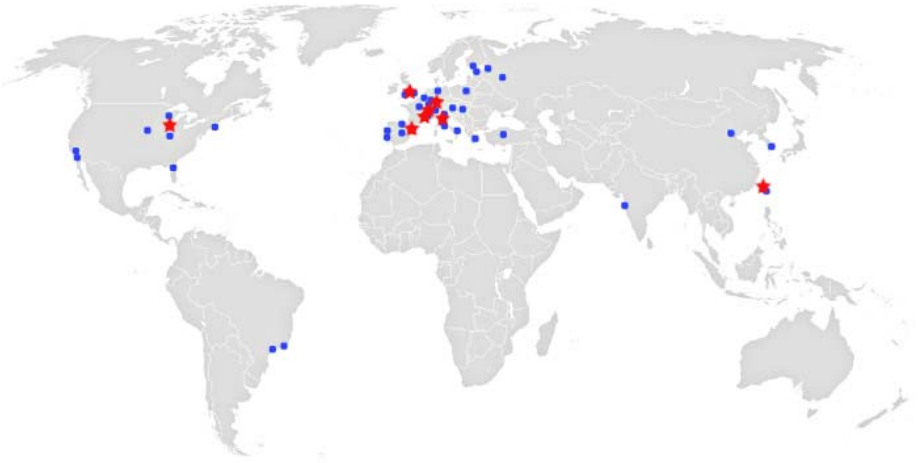


Figure 1: Tier-1 (red star) and Tier-2 (blue squares) sites worldwide in CMS (*Image courtesy of James Letts, http://cms.web.cern.ch, May 2008*)

Case 1: The Compact Muon Solenoid Experiment (CMS) "still produces more than five petabytes per year when running at peak performance[1]". It has large number of "Tier-2" analysis centers where physics analysis are performed, as depicted in Figure 1. 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's, and transfer of produced data to Tier-1's for storage [1]. They are required to import 5 TB/day of data from Tier-1 and other data replicated at T2, 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 is essential to any experiment, and so far the data transfer system in CMS seems to be up to the challenge".

[1] http://cms.web.cern.ch/cms/Detector/Computing/index.html

Figure 2: LIGO Data Grid (LDG) (*courtesy:http://ligo.org.cn/testbeds.shtml*)

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), depicted in Figure 2, has laboratory centers (Caltech, MIT, LHO and LLO) and LSC institutions (UWM, and 3 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 is 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 workflow planning for grids (Pegasus) and Condor DAGMan for management, the LDG continues to manage complex workflows for its growing number of users.

Rationale: In Case 1, data is being shared via a central repository with its Tier-2 members and Tier-2 caches 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, as compared to getting them from Tier-1 directly every time: **the need for multi-source data transfer**. 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: **the need for output data management**.

Similarly, in Case 2, as input data is replicated at all LDG centers, complex workflows could make use of these multiple data sources while transferring data. The intrinsic characteristic that input data is common to 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 short period of time from these sites.

When scheduling and managing the executions of these applications, an application scheduler should be able to select these data sources and parallelize the transfer of data to a compute host to optimize the 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 thus focus on these two aspects – data host and compute host selection while scheduling data intensive scientific application workflows.

Different approaches such as the replica selection in the Globus Data Grid [3], Giggle framework [4] 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 minimum transfer time 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 [5, 6].

Storage and distribution services provided by storage service providers such as Nirvanix Storage Delivery Network (http://www.nirvanix.com), Cloud Storage [Amazon Simple Storage Services (S3), (http://www.amazon.com)], are enabling users and scientists to store and access content from edge servers distributed globally. These content distribution network 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 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 as 'multi-source parallel-data-retrieval (MSPDR)' in this article. In addition to selecting data hosts, we also need to choose a resource where the data is transferred for execution of application tasks.

In this article, we present two scheduling heuristic that leverage multi-source parallel data-retrieval techniques. We experiment with existing (a) probe-based[6], (b) greedy [6], and (c) 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 heuristics on the makespan (i.e., the length of the schedule – data transfer and execution time) of representative data intensive application workflows. Finally, we compare the makespan obtained by using MSPDR-heuristics against single 'best' data source based heuristics.

2. Problem Statement

We now describe the problem of data host selection and tasks to resource mapping in the presence of large number of replicated files for workflow applications [7].

Def 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), 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 $= \{\{PD_{d_i \to r}\}^{t_r} \ \forall d_i \in D, r \in R, i \leq |D|\}$] for each task,* ***simultaneously****, such that: total execution time (and cost) at r and data transfer time (and cost) incurred by data transfers $\{PD_{d_i \to r}\}$ for all the tasks are minimized.*

The pre-conditions are:

1. Data files are replicated across multiple data hosts
2. Each task requires more than one data file
3. 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

In this article, we propose heuristics based scheduling techniques for the problem stated in DEF 1 by limiting the number of objectives to only one: **minimize the makespan**. Hence, execution and data transfer cost are *zero*.

3. Scheduling Heuristic

In this section, we first describe a static scheduling heuristic (also known as offline scheduling) assuming the scheduler has advance information of the environment. For dynamic environments, where the estimates are not used, 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 the scheduler makes scheduling decisions at run-time.

3.1. Static Scheduling Heuristic

We propose an Enhanced Static Mapping Heuristics (ESMH) assuming the scheduling system has advance information of tasks, compute and storage resources and network statistics at the time of scheduling and prior to execution, as listed below:

- Number of tasks to be scheduled
- Estimated execution time of every task on a set of dedicated resources
- Maximum execution time of a task (used for task preemptions in a priority queue based resource management system)
- Size of data handled by each task
- Earliest start time for an unscheduled task on any given resource

- Resource characteristics: CPU MHz, memory, cache
- 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 to produce output files, all in the set $\{f_1, ..., f_n\}_{tk}$ $\forall f_i \in F$. A data-resource matrix 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 m_{ij}. The m_{ij} values must be calculated/estimated in advance by using partial file transfer mechanisms, e.g. probe, random or greedy.

In static mapping heuristics (e.g. HEFT, HBMCT), 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 is 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 $avg\_comp(t_i, R_k)$ $\forall t_i \in T; R_k \in R$ and the estimated transfer time of total data to the resource $tr(\{f\}_{t_i}, R_k)$. 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 Siegel et al. [8].

Heuristics: Algorithm 1 lists the ESMH. We start task-resource mapping by selecting tasks in a workflow on a level-by-level basis. The DAG representation of a workflow is divided into levels to form a tree using breadth-first-search (BFS). The BFS begins at the root node and explores all the neighboring nodes. Then, for each of those neighboring nodes, we explore their unexplored neighbors and so on, until we have explored all the tasks in the DAG. Each search step defines a new level until we reach the leaf nodes.

For each task, we first form a set of compute resources $\{R\}$ by selecting only those resources that have Minimum Transfer Time (MTT) $min(tr(\{f_i\}))$ for each input file f_i of a task t_i. The transfer time value $tr(\{f_i\}, resource) = m_{ij} = m[file, resource]$ 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. Next, we estimate the Earliest Finish Time (EFT) value of the task (EFT_{min_avg}) by adding the average of the minimum Execution Time (ET) given 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.

Algorithm 1: Enhanced Static Mapping Heuristics (ESMH)

Data: Tasks $\{t_i\}$ with input files $\{f_1, ..., f_n\}_{ti}$,
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
  for *each task starting from the root* **do**
    Form resource set $\{R\}$ that has $min(tr(\{f_i\}))$ for each file f_i required by task t_i
    Mark $R_k \in \{R\}$ that has min. computation time for t_i
    $EFT_{min\_avg} = min\_avg\_comp(t_i, R_k) + tr(\{f\}_{ti}, R_k)$
    for *each resource* $r_i \in \{R\}$ **do**
      $EFT_{relative} = avg\_comp(t_i, r_i) + tr(\{f\}_{ti}, r_i)$
      if $[EFT_{relative}] < [EFT_{min\_avg}]$ **then**
        Map t_i to r_i; break;
    if t_i *not assigned* **then**
      Mark $R_m \notin \{R\}$ that has minimum tr time for the largest file required by task t_i
      $EFT_{file} = min\_avg\_comp(t_i, R_m) + tr(\{f\}_{ti}, R_m)$
      if $[EFT_{file}] \leq [EFT_{min\_avg}]$ **then**
        Map t_i to R_m
      else
        Map t_i to R_k /* last option */
    Update resource availability information based on task-resource mapping
end

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 $EFT_{relative}$. $EFT_{relative}$ is different than EFT_{min_avg} as the EFT value is relatively dependent (convolution relationship) on data transfer and on average computation, unlike EFT_{min_avg} which is dominated by the *minimum* value of execution time given by a resource. We then compare the EFT_{min_avg} value against the $EFT_{relative}$. If we find a resource that has the $EFT_{relative}$ value lesser than the EFT_{min_avg}, then we assign the task to the resource that gives this lesser $EFT_{relative}$.

If none of the resources in the set $\{R\}$ have EFT value lower than EFT_{min_avg}, we compute the EFT based on the file size. We choose a resource R_m such that it has minimum transfer time value for the largest input file of the task t_i. We then compute the EFT_{file} based on this resource R_m. The task is then assigned to the resource that has minimum EFT value (either R_m or R_k). We could search for optimal $EFT_{relative}$, but it would be computationally not feasible for large

resource set $\{R\}$.

Rationale: The formation of a bounded compute resource set $\{R\}$ ensures that only limited, but right candidate resources are selected from a pool of large number of resources. 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. $EFT_{relative}, EFT_{file}$) over task computation time (i.e. EFT_{min_avg}). 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 $EFT_{relative}$ value. 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 EFT_{file} value. 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 EFT_{min_avg} value. Hence, in a workflow, all the tasks get equal share of resources depending on their data and computation requirements.

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

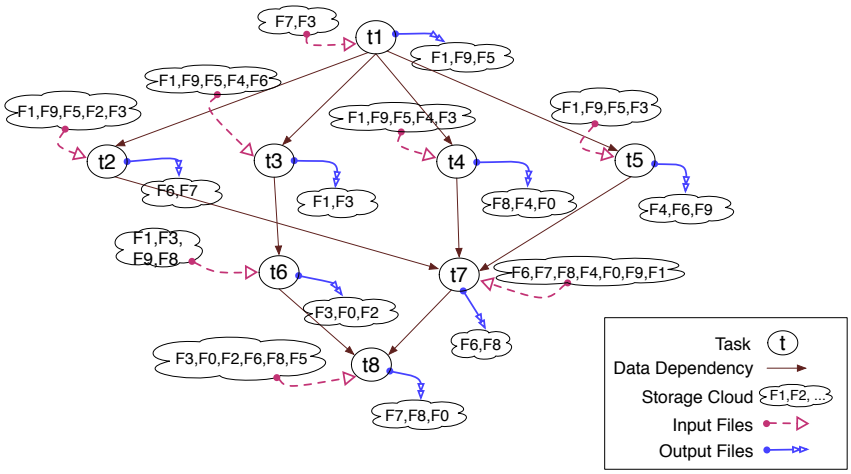
3.1.1. Example Workflow

Figure 3 shows an example of a workflow with input and output files and data dependencies between tasks. The table in Figure 3c 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 (Figure 3a) and b) one that stores values of average transfer time of each data-file from distributed data-centers to each resource (Figure 3c) 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.

3.1.2. Limitations of Static Heuristic

The static heuristic we have proposed would be most appropriate if all of our assumptions listed in sub-section 3.1 could be realized in practice. However, in distributed environment where resources are shared among a large number of users, the estimates recorded in the matrices, as described above, will have large deviations compared to the values at the time of task execution. Hence the



a) Task execution time on resources

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

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

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

c) A schedule generated by using ESMH

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					

Figure 3: An example workflow, matrices with estimated values, and a schedule generated by using ESMH.

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.

3.2. 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, $w : E \rightarrow R_0^+$. In particular, the weight $d : E \rightarrow R_0^+$, and $c : E \rightarrow R_0^+$ represent the delay and the cost of the link, respectively. A *path* is a finite sequence of non-repeated nodes $p = (v_0, v_1, \cdots, v_i)$, 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 with length, which is the shortest among all spanning trees, is called a minimum spanning tree for G. An 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.

In order to approximate the length of the path p, we assume the delay and cost of a path are in metric space and can be combined with the a 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:

1. $\rho(x, y) \geq 0$ for any x, y in X; whereby equality holds if and only if $x = y$;
2. $\rho(x, y) = \rho(y, x)$ for any x, y in X; and
3. $\rho(x, y) \leq \rho(x, z) + \rho(z, y)$ for any x, y, z in X (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 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 as compared to when using ρ in a non-metric space.

3.2.1. Forming a Steiner Tree

The SMT problem is NP-hard, so polynomial-time heuristics are desired [9]. 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 [10]. In this paper, we use the *incremental optimization heuristic* developed by Dreyer et al. [9]. Even though it does not give an optimal solution, we get a feasible solution at any point in time [2].

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:

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

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

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) help achieve the objective of our problem as compared to using a single source.

3.3. 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 2) and the links E represent the network connection between them in the graph G. 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 US under our slice, the Grid'5000 network, and a distributed compute resources similar to CMS resources in Figure 1. These trees are shown in Figure 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 connect the vertices with lines such that the distance between the points in the plane is minimal. If there are vertices which 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 Figure 4(c) when conducting real experiments, as described further in Section 4.

In Figure 4, there are vertices (existing and added) which have in-degree (in_d: the number of edges coming into a vertex in a graph) of two and three. 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 $in_d = 3$ with vertices $v_1, v_2, v_3 \in V$, a resource located at/nearby v can retrieve data from these three data sources with minimal path length:

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

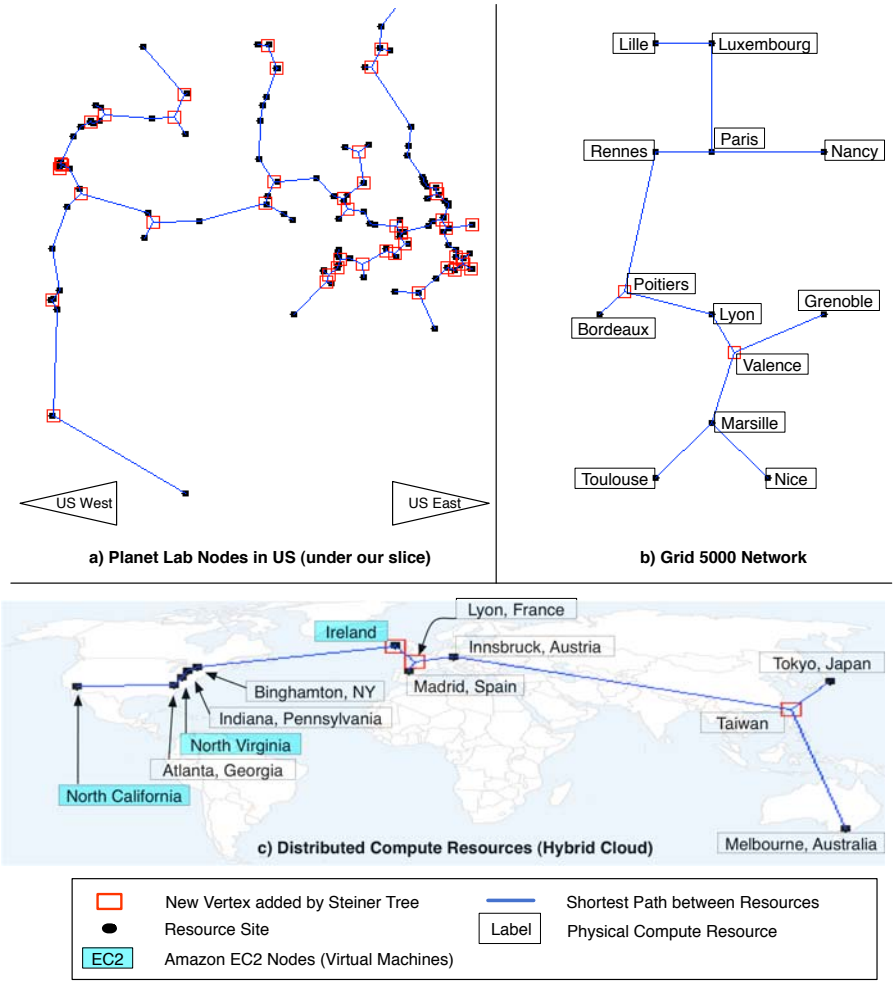


Figure 4: A Steiner Tree constructed for Planet Lab nodes, Grid'5000 network and distributed resources available for our experiment

In the Planet Lab network, there are several nodes (dots with square) that have an in-degree of three. In the case of Grid'5000 network, Paris, Marsille possess compute nodes that are each connected to 3 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 will then be used for dynamic mapping heuristics.

Algorithm 2: Enhanced Dynamic Mapping Heuristics (EDMH)

Data: A Tasks $\{t_i\}$ with input files $\{f_1, ..., f_n\}_{ti}$
Data: Earliest Start Time (EST) for a task on each r_j
Data: f_k can be sourced from more than one location (replicated)

```
1  begin
2      Construct a Steiner tree using all available resources
3      Get N compute resources {R}_N with highest value of in_d
4      for each task t_i starting from the root do
5          Set minimum Start Time (minST) = ∞
6          for each resource r_i ∈ {R}_N do
7              Probe each connected neighbor r_i^neighbor of r_i for calculating
               instantaneous bandwidth (max of in_d probes)
8              Split input files based on probe values:
               {f^split-1, ..., f^split-in_d}_ti ∈ {f}_ti
9              Estimate total transfer time tr({f}_ti, r_i) for transferring split
               files {f^split}_ti from each r_i^neighbor to r_i
10             StartTime(ST) = EST(t_i, r_i) + tr({f}_ti, r_i)
11             if (ST ≤ minST) then
12                 minST = ST, minCR = r_i
13         Assign t_i to the minCR ⇒ the r_i that gives minimum ST
14         Wait for polling_time
15         Update the ready task list
16         Distribute output data of task t_i to resources that host the files
           required by successors of t_i
17 end
```

3.4. Dynamic Mapping Heuristic

In this section, we describe the Enhanced Dynamic Mapping Heuristic (EDMH) using the Steiner tree based resource selection as described in Sub-section 3.2. 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 heuristics (or offline heuristics), EDMH does not estimate or use average computation time of tasks, instead relies on the Earliest Start Time provided/forecast by resources. In addition, EDMH selects 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)[3].

Scheduling: EDMH is a list-based scheduling heuristic. We maintain a *ready-list*, 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 Start Time (ST) is set to a high value (e.g. ∞). 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 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 *probe-based* approach to estimate the instantaneous bandwidth between connected resources and hence estimate approximate time it would take to download input data from multiple resources for every task.

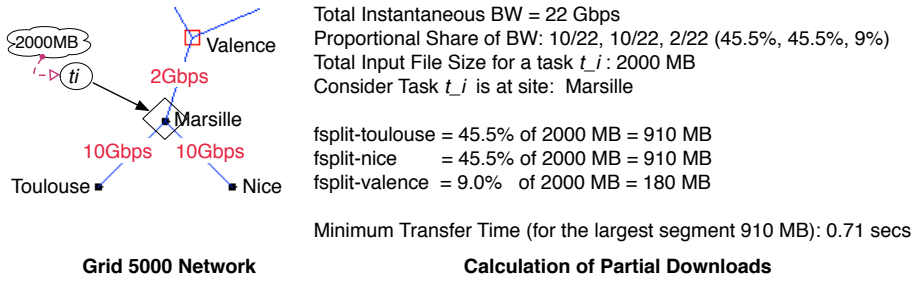


Figure 5: An example showing the partitioning of input files in proportion to the available instantaneous bandwidth.

As each resource $r_i \in \{R\}_N$ is connected with multiple neighboring nodes $\{r_i^{neighbor}\}$ in the network (the Steiner tree helps identify these connections based on the in_d), we probe these neighboring nodes to determine instantaneous bandwidth available. Based on this value, we split each input file among the n_d resources connected to r_i in proportion as: $\{f^{split-1}, \cdots, f^{split-in_d}\}_{ti} \in \{f\}_{ti}$. 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 Figure 5, if a task *ti* were to be scheduled at Marsille, input files of total size 2000MB is split into three segments (as $n_d = 3$ for Marsille): fsplit-toulouse, fsplit-nice, and fsplit-valence. These segments will then be downloaded to Marsille in parallel from the three neighboring sites. In this example, entire input files can be downloaded from all the three sites. The minimum transfer time of 0.71 seconds depends on the size of the largest segment (910MB in the example). While this segment is being downloaded, we assume the smaller segments

[3]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 $L(G)$

will have finished downloading, so that we can approximate the value of ST. We take this minimum time as the total transfer time $tr(\{f\}_{ti}, r_i)$ and add it to the Earliest Start Time (EST) of the task $EST(t_i, r_i)$ to obtain the value of ST. The transfer time function $tr(\{f\}_{ti}, r_i)$ 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 3.2).

We assign a task t_i to the resource ($minCR$) that projects the minimum start time ($minST$) based on our estimations. The scheduler then waits for a duration defined by the delay: $polling\_time$. 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. This distribution step ensures that these output files can be downloaded from multiple sites which 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 [11].

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 as compared to the computation time. This gives rise to the possibility that 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 selected few resources. In such cases, grouping these tasks to form a batch task and submitting to a resource reduces data-transfer time.

4. Performance Evaluation

We have evaluated proposed heuristic by two methods: 1) using emulation, where the network was virtualized, and 2) real environment. First we describe the performance metric, application workflows and data locality, which are common to both the experiments.

4.1. Performance Metric

We used *average makespan* as a metric to compare the performance of the 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. The smaller the value of the average makespan, the better the heuristic performs in terms of executing the application in time scale.

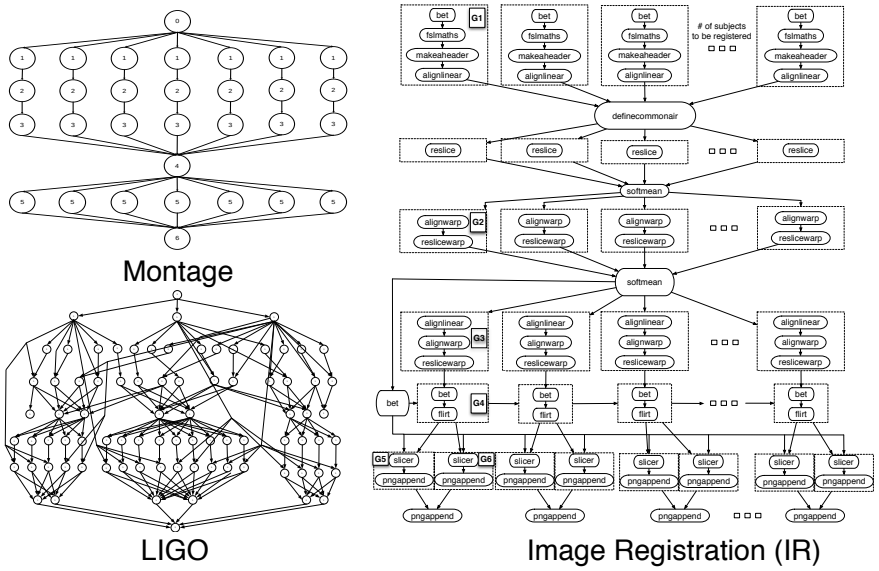


Figure 6: Application Workflow types: Balanced (Montage), complex (LIGO) and hybrid (IR) workflows

4.2. Application Workflows

We used three types of workflows: pipelined, complex and hybrid, as depicted in Figure 6. Figure 6-Montage depicts a workflow similar to the Montage Workflow [12]. 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 [13]. In this type of workflows, tasks cannot be partitioned into levels easily as they have linked (data dependencies) to/from tasks across levels. Figure 6-IR represents a real application workflow used for Image Registration (IR) for fMRI applications [14]. This type of workflows have both balanced and complex structures: group of pipelined tasks form a balanced structure that can be easily partitioned into levels (upper half of the workflow), whereas some parts are complex (lower half of the workflow). In all these workflows, each task requires at least two files as input and produces at least one file as output (similar to the example workflow depicted in Figure 3).

In contrast to parallel tasks, where there are no data dependencies, 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 will increase the data transfer time, delaying the start time of child tasks which in-turn increases the makespan.

We experimentally recorded the makespan for all the three types of workflows to determine: a) if the makespan and the workflow structures were related, and

b) the effect of multi-source retrieval technique based scheduling on decreasing the makespan of all workflow structures. We also checked if static scheduling or dynamic scheduling approach produced better makespan when using multi-source data retrieval technique.

4.3. Data Locality

For experimenting 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 0.5Mb to 500Mb. We tracked progress of file downloads by segment number. 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 the probe-based retrieval, we distributed all the files without segmenting to all the resources. This is because the size of a file to be downloaded depended on the value calculated by probing (as described in Sub-Section 3.4).

4.4. 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. Then, we present the results.

4.4.1. Emulation Setup

We used NS-2 (http://www.isi.edu/nsnam/ns) 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 is connecting a set of User-mode Linux (UML [15]) Virtual Machines (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 using NS-2's *network objects* and *tap agents*, by using the correspondence between the Ethernet addresses of the VMs and the network layer addresses of the NS-2 nodes [16].

Cappello et al. [17] have compared 4 virtual machine 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, which was application even when 100 VMs were running on a single host. Hence, the 20 virtual nodes running on a single machine had minimal impact on the performance of the applications. We used our Workflow Engine (WFE) [14, 18] as a workflow execution and scheduling engine for executing the workflows depicted in Figure 6. All the scheduling heuristic are implemented in WFE. We dedicated 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.

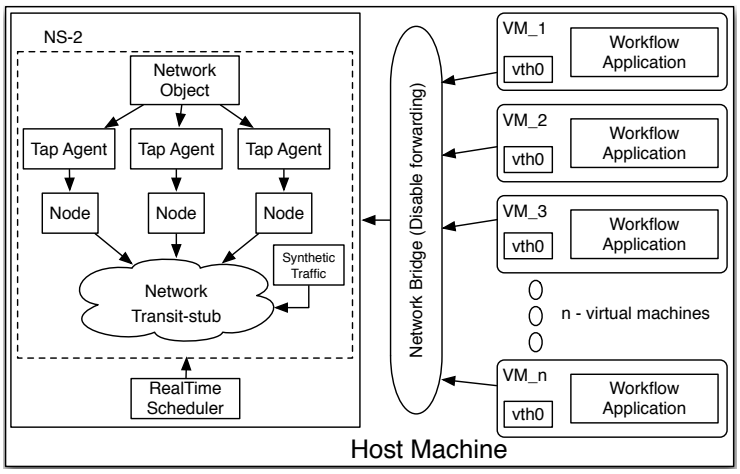


Figure 7: Experiment design using NS-2 based emulation

4.4.2. Network Topology

We used the GT-ITM internetwork topology generator to generate random graphs (http://www.cc.gatech.edu/projects/gtitm). 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 [19] 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 Figure 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 the our paper that focuses on IR experiment [14] on Grid'5000. However, for Montage and LIGO workflows, we used random execution time and data size. 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 communication-to-computation ratio, we chose each file size in the interval [1, 1000] Mb. Each task for the two workflows were dummy computation that remained in execution until its assigned execution time expired. The files associated with each task were generated by writing blocks of random characters until the file was of required size. These random values, once computed, remained fixed throughout the experiment.

4.4.3. Storage and Compute Resources

Modeling of storage resources is a challenge. However, emerging technologies like Cloud storage systems, as mentioned in Section 5, are addressing storage services from a higher level. Since we are not concerned about sub-millisecond

	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

response times, usually in the case of transactional processing systems and not in scientific workflows, we have assumed our data centers to have characteristics similar to that 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 the links to create congestion. All links had a delay of 10ms. 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. All VMs used the network defined in NS-2. The total number of VMs running at one instance was limited to 50, half the size as experimented by Cappello [17]. We assigned 20 of these VMs as storage resources and remaining as compute resources. The number of storage nodes remained fixed while the number of compute nodes running was changed based on the workflow (Montage, LIGO or IR) being executed. Figure 8 summarizes all the parameters used for our emulation.

4.4.4. Experimental Results

Comparision 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

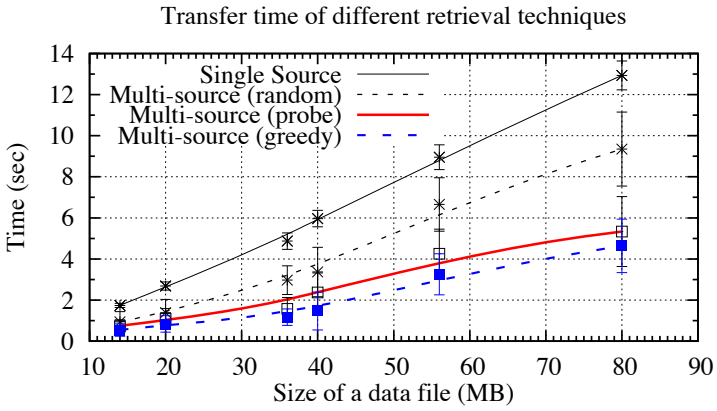


Figure 9: Comparison of transfer time and error margins between single-source and multi-source data retrieval techniques using data files of different sizes (files are from IR Workflow experiment only) (Bézier curve fitting used)

depicted in Figure 9. The average data transfer times obtained using these techniques were close to the results listed by Zhou et al.[6]. As the size of files were increased, random method gave the worst and the greedy gave the best transfer times on our emulated network topology. However, greedy method of retrieval 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 Figure 10 (upper half). By processing time, we mean the total time spent for (a) numerous repeated connections to hosts due to 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 better transfer time. Also, the probe method gave lower transfer-time than random/single-source based retrieval, in addition to the lower overheads than greedy-based retrieval. We obtained the the total transfer time (T) and overheads (processing time (P)) for the retrieval methods on all the three types of workflows, as depicted in Figure 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 Figure 10 (upper half). However, probe-based retrieval had minimum overheads as compared to 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)

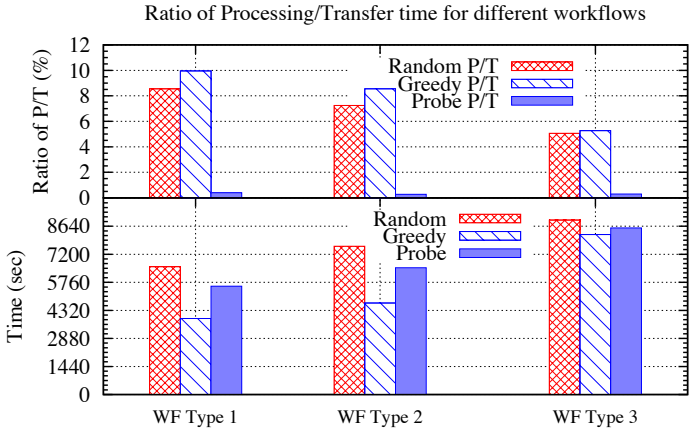


Figure 10: Average data transfer and segment-processing times of all files using random, greedy and probe-based retrieval techniques for three workflow (WF 1, WF 2 and WF 3 corresponding to Montage, LIGO and IR workflows in Figure 6)

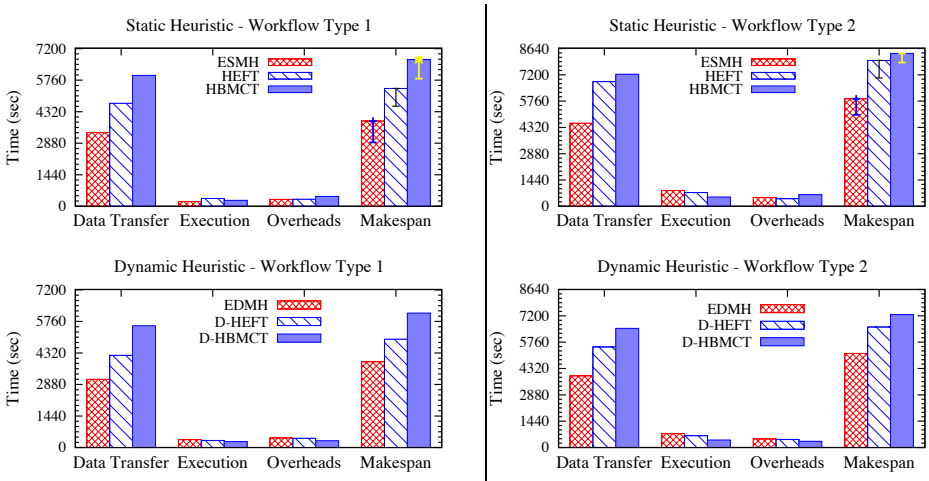
was better in terms of time and complexity than using greedy/random/single-source retrievals. Hence, we used probe-based data retrieval technique in our heuristic for its advantage over greedy and random techniques.

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 than 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 Figures 11 and 12. Each makespan is the addition of data transfer time, task execution time, and overheads. For all the three types of workflows, ESMH estimated minimum makespan (lower value of the vertical lines). When executed on our emulated environment, the actual makespan recorded for ESMH was lower than HEFT and HBMCT algorithms.

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

For symmetric workflows (Figure 6-Montage), the makespans generated by



(a) Type 1 workflow (e.g. Montage) (b) Type 2 workflow (e.g. LIGO)

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

static heuristic are similar to that generated by their corresponding dynamic mapping heuristic. This is mainly due to the structure of the workflow: in Figure 6-Montage there are only two tasks that download output files from more than one parent, while other tasks can 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 similar transfer time for both the static and dynamic heuristic as depicted by the data transfer time components of the makespan in Figures 11 for workflow type 1 (Montage).

However, for both complex and hybrid workflows (Figure 6-LIGO, IR), makespans generated by dynamic mapping heuristics 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 dynamic scheduling approach. As static approach estimated the bandwidth between resources at the scheduling time (not at the 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 Figures 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 as compared to changing network bandwidth. In cases when resource usage changes randomly, static scheduling approaches may perform even worse than dynamic approaches.

In Figures 11 and 12, in addition to the overall makespan, we also compared

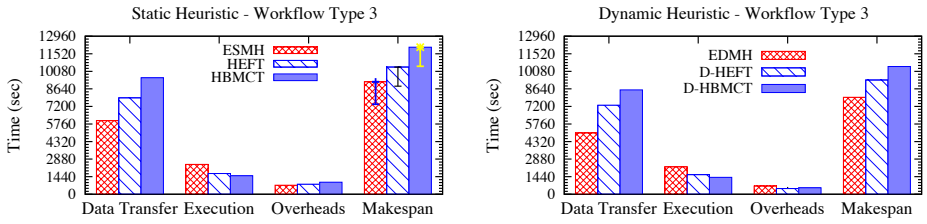


Figure 12: Makespan of IR workflow (type three) using static and dynamic heuristic.

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 as compared to 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 heuristics, 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.

4.5. Real Experiment based Evaluation

In this section, we present the results obtained using a real testbed depicted in Figure 4(c). This testbed was selected to match the resource distribution shown in the motivation section in Figure 1. In order to determine the feasibility of proposed heuristic in practical environments, we experimented the IR workflow, depicted in Figure 6-Image Registration, using the real testbed. This application used data and scripts provided to us by Dartmouth Brain Imaging Center.

4.5.1. Experiment Setup

We formed an experimental testbed consisting of compute resources from worldwide research labs and Amazon EC2, as depicted in Figure 4(c). These resources were a combination of real compute nodes and virtual machines (VM) (Amazon EC2 nodes), similar to a hybrid Cloud. The Figure 4 labels each resource by the name of the city where it is located. We chose to distribute

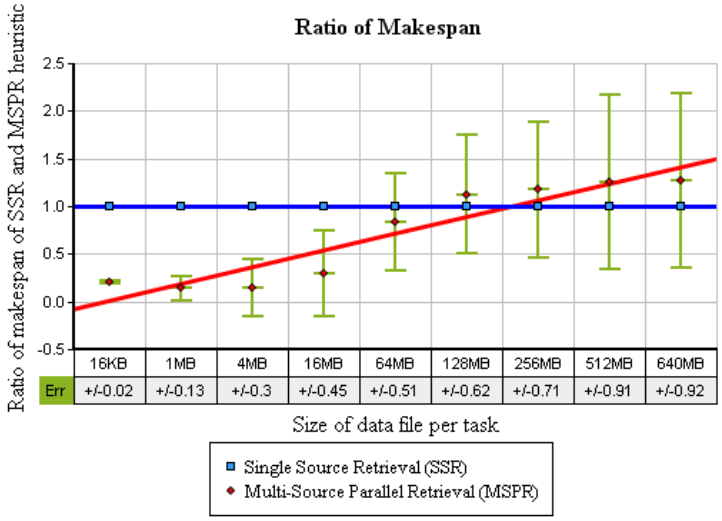


Figure 13: Determining the benefit of using multi-source parallel data retrieval by comparing the ratio of makespan for IR workflow (WF type three) in real environment

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. As we are interested in data intensive applications, the location of the resources is of primary concern to us than their compute power.

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, 1GB memory and 20GB free disk space. Each Amazon VM was a large instance with 4 EC2 Compute units (2 virtual cores with 2 EC2 compute units each), 7.5GB memory and 850GB local storage. We used the IR application workflow with data distribution described in sub-subsection 4.3.

The nodes were all connected via Internet. In order to approximate the network topology, we used each node's location (latitudes and longitudes) and constructed the Steiner tree (Figure 4(c). Using the Steiner tree, we could identify the in-degrees of each node: nodes at Lyon and Taiwan were having 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 $in_d \geq 3, in_d \geq 2$, respectively.

4.5.2. 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 16MB per task, the total size of data handled by a 20-subject IR workflow exceeds 12GB [14]. We varied the input file size for each task from 16KB to 640MB and iterated the experiment for eight times for each input size (Figure 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:

$$MakespanRatio = \frac{(Makespan_{D-HEFT})}{(Makespan_{EDMH})} \tag{3}$$

Figure 13 plots the mean values of the ratio from Equation 3 for varying file sizes for the IR workflow. It also plots the standard error[4] about the mean values. Based on eight measurements, the ratio was 0.25 ± 0.02 and 1.25 ± 0.92 for 16KB and 640MB of data per task, respectively. Positive values of the ratio clearly showed that the makespan given by EDMH was smaller than that given by D-HEFT. However, the ratio became positive only for file sizes 128MB and above, clearly indicating the relationship of retrieval technique to data sizes.

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

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

5. Related Work

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

Replica Selection and Retrieval: Vazhkudai et al. [3] presented design and implementation of high-level replica selection service in the Globus data Grid. Chervenak et al. [4] defined a replica location service (RLS) that maintains and provides access to information about the physical locations of copies. Hu et al. [20] 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. [6] analyzed various algorithms for replica retrieval and concluded that probe-based retrieval

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

is the best approach, providing twice the transfer rate of the 'best' replica server. Feng et al. [5] 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) [21] deployment at all participating Grid nodes, and *NoObserve* or *SelfObserve* does not use NWS. In these work, 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 as noted by Zhou et al. [6] and Feng et al. [5]. Our work leverages these retrieval techniques.

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 large number of replicated files for a single job is computationally intensive. Venugopal et al. [22] selected the data-hosts by using one of the solutions to the Set-Coverage problem [23]. In this paper, we assume that data are fully replicated and hence set-coverage is guaranteed by every data source.

Some work on transport protocols have focused on receiver based flow contention management. Rodriguez et al. [24] proposed a dynamic-parallel access to replicated content from multiple servers or caches in content delivery networks. They showed that users experience significant speedups and very consistent response times when using multiple parallel transfers. Similarly, Wu et al. [25] proposed Group Transport Protocol (GTP) and a receiver based rate allocation scheme to manage multi-source data transmissions. They also showed that GTP outperforms other point-to-point protocol for multiple-to-point transmission.

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. This has been studied in detail by Yang et al. [26] and Feng and Humphrey [5]. GridFTP and rFTP [5] are existing tools which support these types of transfers.

Static and Dynamic Workflow Scheduling: We focus on list based scheduling heuristics for computing static schedules (offline); and task partitioning and iterative rescheduling for computing dynamic schedules (online).

Topcuouglu et al. [27] designed the HEFT algorithm based on list scheduling. HEFT is a static scheduling algorithm which 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 et al. [28] 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 heuristics that uses standard list scheduling approach to rank the nodes of the Directed Acyclic Graph (DAG) and then uses this ranking to form group of tasks which 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 minimizes 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. [29] 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 Deelman et al. [30] 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.

Iterative re-computing technique keeps applying the scheduling algorithm on the yet-to-be-scheduled tasks of a workflow in execution. Sakellariou and Zhou et al. [31] 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 work have explored static and dynamic scheduling strategies for workflow execution that focused on: user quality of service and location affinity [32, 33, 34], iterative calls of static algorithms [35, 36, 37], dynamic programming [38] and so forth. Lopez et al. [36] explored several static and dynamic approaches for scheduling workflows and concluded that list-based heuristics significantly outperform non-list based heuristics. Yu et al. [39] have described the strategies involved in scheduling workflows for Grid computing environments, in much detail.

Many past work on scheduling workflows focused primarily on compute-intensive workflows. Most of these scheduling algorithms could make use of multi-source data retrieval technique 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 heuristics based approaches.

6. Conclusions

In this paper, we presented two workflow scheduling heuristic that leverages multi-source parallel data-retrieval techniques. We showed that probe-based data retrieval from as 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 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 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 testbed. The results obtained from both emulation and real experiments consistently showed that makespan of workflows can be decreased significantly

when using multi-source parallel data retrieval technique while scheduling workflow. From our experimental results, we also conclude that, on average, Enhanced Dynamic Mapping Heuristic (EDMH) produces time-efficient makespan than 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 heuristic. We would also like to explore the effect of changing data to compute ratio, segment sizes and replica distribution on the makespan produced by ESMH, EDMH and other heuristics based approaches.

Acknowledgments

This work is partially supported through Australian Research Council (ARC) Discovery Project grant, and International Science Linkages (ISL) program of the Department of Innovation, Industry, Science and Research. Some experiments presented in this paper were carried out using the Grid'5000 experimental testbed, an initiative from the French Ministry of Research. We thank Amazon for providing us educational grant for using EC2 and S3 resources. We are also grateful to several research labs that provided access to their compute infrastructure for our experiments: 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. We thank James Dobson from Department of Psychological & Brain Sciences, Dartmouth College, for providing us real data and scripts for fMRI image registration application.

References

[1] G. L. Bayatyan, M. Della Negra, A. Foa, Herve, A. Petrilli, CMS computing : Technical Design Report, Tech. Rep. CERN-LHCC-2005-023, CMS Collaboration (May 2005).

[2] M. A. Papa, The LSC-Virgo white paper on gravitational wave data analysis, Tech. Rep. LIGO-T0900389-v1, The LSC-Virgo Data Analysis Working Groups (August 2009).

[3] S. Vazhkudai, S. Tuecke, I. Foster, Replica Selection in the Globus Data Grid, in: CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid, IEEE, Washington, DC, USA, 2001, pp. 106 – 113.

[4] A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunszt, M. Ripeanu, B. Schwartzkopf, H. Stockinger, K. Stockinger, B. Tierney, Giggle: a framework for constructing scalable replica location services, in: Supercomputing '02: Proceedings of the

2002 ACM/IEEE conference on Supercomputing, IEEE, Los Alamitos, CA, USA, 2002, pp. 1–17.

[5] J. Feng, M. Humphrey, Eliminating Replica Selection - Using Multiple Replicas to Accelerate Data Transfer on Grids, in: ICPADS '04: Proceedings of the Tenth International Conference on Parallel and Distributed Systems, IEEE, Washington, DC, USA, 2004, pp. 359–366.

[6] X. Zhou, E. Kim, J. W. Kim, H. Y. Yeom, ReCon: A Fast and Reliable Replica Retrieval Service for the Data Grid, in: CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid, IEEE, Washington, DC, USA, 2006, pp. 446–453.

[7] S. Pandey, R. Buyya, Scheduling of Scientific Workflows on Data Grids, in: CCGRID '08: Proceedings of the Eighth IEEE International Symposium on Cluster Computing and the Grid, IEEE, Washington, DC, USA, 2008, pp. 548–553.

[8] L. D. Briceno, M. Oltikar, H. J. Siegel, A. A. Maciejewski, Study of an Iterative Technique to Minimize Completion Times of Non-Makespan Machines, in: IPDPS '07: Proceedings of the 21th International Parallel and Distributed Processing Symposium, IEEE, USA, 2007, pp. 1–14.

[9] D. R. Dreyer, M. L. Overton, Two Heuristics for the Euclidean Steiner Tree Problem, Journal of Global Optimization 13 (1) (1998) 95–106.

[10] H. F. Salama, D. S. Reeves, Y. Viniotis, Evaluation of multicast routing algorithms for real-time communicationon high-speed networks, IEEE Journal on Selected Areas in Communications 15 (1997) 332–345.

[11] L. Qiu, V. N. Padmanabhan, G. M. Voelker, On the Placement of Web Server Replicas, in: INFOCOM '01: Proceedings of IEEE INFOCOM, 2001, pp. 1587–1596.

[12] J. Jacob, D. Katz, T. Prince, G. Berriman, J. Good, A. Laity, The Montage Architecture for Grid-Enabled Science Processing of Large, Distributed Datasets, in: ESTC '04: Fourth Annual Earth Science Technology Conference, 2004.

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

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

[15] H.-J. Hoxer, K. Buchacker, V. Sieh, Implementing a User-Mode Linux with Minimal Changes from Original Kernel, in: Linux-Kongress '02: Proceedings of the 9th International Linux System Technology Conference, Cologne, Germany, 2002, pp. 72–82.

[16] D. Mahrenholz, S. Ivanov, Real-Time Network Emulation with ns-2, in: DS-RT '04: Proceedings of the 8th IEEE International Symposium on Distributed Simulation and Real-Time Applications, IEEE, Washington, DC, USA, 2004, pp. 29–36.

[17] B. Quetier, V. Neri, F. Cappello, Selecting A Virtualization System For Grid/P2P Large Scale Emulation, in: EXPGRID '06:Proceedings of the Workshop on Experimental Grid testbeds for the assessment of large-scale distributed applications and tools, Paris, France, 2006.

[18] J. Yu, R. Buyya, A Novel Architecture for Realizing Grid Workflow using Tuple Spaces, in: GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, IEEE, Washington, DC, USA, 2004, pp. 119–128.

[19] E. W. Zegura, K. L. Calvert, S. Bhattacharjee, How to Model an Internetwork, in: INFOCOM '96: Proceedings of the IEEE Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation. Proceedings of IEEE INFOCOM, Vol. 1-3, 1996, pp. 594–602 vol.2.

[20] Y. Hu, J. Schopf, IBL for Replica Selection in Data-Intensive Grid Applications, Tech. Rep. TR-2004-03, The University of Chicago (2004).

[21] R. Wolski, N. T. Spring, J. Hayes, The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing, Future Generation Computing Systems 15 (5-6) (1999) 757–768.

[22] S. Venugopal, R. Buyya, A Set Coverage-based Mapping Heuristic for Scheduling Distributed Data-Intensive Applications on Global Grids, in: GRID '06: Proceedings of the 7th IEEE/ACM International Conference on Grid Computing, IEEE, Washington, DC, USA, 2006, pp. 238–245.

[23] E. Balas, M. W. Padberg, On the set-covering problem, Operations Research 20 (6).

[24] P. Rodriguez, E. W. Biersack, Dynamic parallel access to replicated content in the internet, IEEE/ACM Transactions on Networking 10 (4) (2002) 455–465.

[25] R. X. Wu, A. A. Chien, GTP: group transport protocol for lambda-Grids, in: CCGRID '04: Proceedings of the Fourth IEEE International Symposium on Cluster Computing and the Grid, IEEE, Washington, DC, USA, 2004, pp. 228–238.

[26] L. Yang, J. M. Schopf, I. Foster, Improving parallel data transfer times using predicted variances in shared networks, in: CCGRID '05: Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid - Volume 2, IEEE, Washington, DC, USA, 2005, pp. 734–742.

[27] H. Topcuouglu, S. Hariri, M.-y. Wu, Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing, IEEE Transactions on Parallel and Distributed Systems 13 (3) (2002) 260–274.

[28] R. Sakellariou, H. Zhao, A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems, in: HCW '04:Proceedings of the 13th IEEE Heterogeneous Computing Workshop, Santa-Fe, New Mexico, USA, 2004.

[29] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, D. S. Katz, Pegasus: A framework for mapping complex scientific workflows onto distributed systems, Scientific Programming 13 (3) (2005) 219–237.

[30] S. Shankar, D. J. DeWitt, Data driven workflow planning in cluster management systems, in: HPDC '07: Proceedings of the 16th international symposium on High performance distributed computing, ACM, New York, NY, USA, 2007, pp. 127–136.

[31] R. Sakellariou, H. Zhao, A low-cost rescheduling policy for efficient mapping of workflows on grid systems, Scientific Programming 12 (4) (2004) 253–262.

[32] I. Brandic, S. Pllana, S. Benkner, An approach for the high-level specification of QoS-aware grid workflows considering location affinity, Scientific Programming 14 (3,4) (2006) 231–250.

[33] V. Bhat, M. Parashar, S. Klasky, Experiments with in-transit processing for data intensive grid workflows, in: GRID '07: Proceedings of the 8th IEEE/ACM International Conference on Grid Computing, IEEE, Washington, DC, USA, 2007, pp. 193–200.

[34] R. S. Barga, D. Fay, D. Guo, S. Newhouse, Y. Simmhan, A. Szalay, Efficient scheduling of scientific workflows in a high performance computing cluster, in: CLADE '08: Proceedings of the 6th international workshop on Challenges of large applications in distributed environments, ACM, New York, NY, USA, 2008, pp. 63–68.

[35] R. Prodan, T. Fahringer, Dynamic scheduling of scientific workflow applications on the grid: a case study, in: SAC '05: Proceedings of the 2005 ACM symposium on Applied computing, ACM, New York, NY, USA, 2005, pp. 687–694.

[36] M. M. Lopez, E. Heymann, M. A. Senar, Analysis of Dynamic Heuristics for Workflow Scheduling on Grid Systems, in: ISPDC '06: Proceedings

of the Proceedings of The Fifth International Symposium on Parallel and Distributed Computing, IEEE, Washington, DC, USA, 2006, pp. 199–207.

[37] Z. Yu, W. Shi, An Adaptive Rescheduling Strategy for Grid Workflow Applications, in: IPDPS 07: Proceedings of the 21th International Parallel and Distributed Processing Symposium, IEEE, USA, 2007, pp. 1 –8.

[38] R. Prodan, M. Wieczorek, Bi-Criteria Scheduling of Scientific Grid Workflows, Automation Science and Engineering, IEEE Transactions on 7 (2) (2010) 364 –376.

[39] J. Yu, R. Buyya, K. Ramamohanarao, Metaheuristics for Scheduling in Distributed Computing Environments, Vol. 146/2008, Springer, Berlin, Germany, 2008, Ch. Workflow Scheduling Algorithms for Grid Computing, pp. 173–214.

Biography

Suraj Pandey is a PhD student at the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He received Masters degree from Inha University, Korea in 2007 and Bachelors degree from Tribhuvan University, Nepal in 2003. His research spans many areas of high performance computing, including scheduling, and accelerating life sciences applications in distributed systems. He has been participating in international software demonstrations and has been awarded in several occasions.

Dr. Rajkumar Buyya is Professor of Computer Science and Software Engineering; and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He is also serving as the founding CEO of Manjrasoft Pty Ltd., a spin-off company of the University, commercializing its innovations in Grid and Cloud Computing. He has authored and published over 300 research papers and four text books. The books on emerging topics that Dr. Buyya edited include, High Performance Cluster Computing (Prentice Hall, USA, 1999), Content Delivery Networks (Springer, Germany, 2008), Market-Oriented Grid and Utility Computing (Wiley, USA, 2009), and Cloud Computing (Wiley, USA, 2019). He is one of the highly cited authors in computer science and software engineering worldwide. Software technologies for Grid and Cloud computing developed under Dr. Buyya's leadership have gained rapid acceptance and are in use at several academic institutions and commercial enterprises in 40 countries around the world. Dr. Buyya has led the establishment and development of key community activities, including serving as foundation Chair of the IEEE Technical Committee on Scalable Computing and four IEEE conferences (CCGrid, Cluster, Grid, and e-Science). He has presented over 250 invited talks on his vision on IT Futures and advanced computing technologies at international conferences and institutions in Asia, Australia, Europe, North America, and South America. These contributions and international research leadership of Dr. Buyya are recognized through the award of "2009 IEEE Medal for Excellence in Scalable Computing" from the IEEE Computer Society, USA. Manjrasofts Aneka technology for Cloud Computing developed under his leadership has received 2010 Asia Pacific Frost & Sullivan New Product Innovation Award.