# Scheduling and Management of Data Intensive Application Workflows in Grid and Cloud Computing Environments

by

**Suraj Pandey**

Submitted in total fulfilment of the requirements
of the degree of

**Doctor of Philosophy**

Department of Computer Science and Software Engineering
The University of Melbourne, Australia

December 2010

**Scheduling and Management of Data Intensive Application Workflows in Grid and Cloud Computing Environments**

Suraj Pandey

*Supervisor: Professor Rajkumar Buyya*

---

**Abstract**

Large-scale scientific experiments are being conducted in collaboration with teams that are dispersed globally. Each team shares its data and utilizes distributed resources for conducting experiments. As a result, scientific data are replicated and cached at distributed locations around the world. These data are part of application workflows, which are designed for reducing the complexity of executing and managing on distributed computing environments. In order to execute these workflows in time and cost efficient manner, a workflow management system must take into account the presence of multiple data sources in addition to distributed compute resources provided by platforms such as Grids and Clouds.

Therefore, this thesis builds upon an existing workflow architecture and proposes enhanced scheduling algorithms, specifically designed for managing data intensive applications. It begins with a comprehensive survey of scheduling techniques that formed the core of Grid systems in the past. It proposes an architecture that incorporates data management components and examines its practical feasibility by executing several real world applications such as Functional Magnetic Resonance Imaging (fMRI), Evolutionary Multi-objective Optimization algorithms, and so forth, using distributed Grid and Cloud resources. It then proposes several heuristics based algorithms that take into account time and cost incurred for transferring data from multiple sources while scheduling tasks. All the heuristic proposed are based on multi-source-parallel-data-retrieval technique in contrast to retrieving data from a single best resource, as done in the past. In addition to non-linear modeling approach, the thesis explores iterative techniques, such as particle-swarm optimization, to obtain schedules quicker.

In summary, this thesis makes several contributions towards the scheduling and management of data intensive application workflows. The major contributions are: (i) enhanced the abstract workflow architecture by including components that handle multi-source parallel data transfers; (ii) deployed several real-world application workflows using the proposed architecture and tested the feasibility of the design on real testbeds; (iii) proposed a non-linear model for scheduling workflows with an objective to minimize both execution time and execution cost; (iv) proposed static and dynamic workflow scheduling heuristic that leverages the presence of multiple data sources to minimize total execution time; (v) designed and implemented a particle-swarm-optimization based heuristic that provides feasible solutions to the workflow scheduling problem with good convergence; (vi) implemented a prototype workflow management system that consists of a portal as user-interface, a workflow engine that implements all the proposed scheduling heuristic and the real-world application workflows, and plugins to communicate with Grid and Cloud resources.

# Declaration

This is to certify that

(i) the thesis comprises only my original work towards the PhD except where indicated in the Preface,

(ii) due acknowledgement has been made in the text to all other material used,

(iii) the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies, appendices, and footnotes.

Signature_____

Date_____

# Acknowledgments

I thank my supervisor, Professor Rajkumar Buyya, whose encouragement, guidance and support from the initial to the final level of my PhD degree enabled me to develop an understanding of the subject. His constant pursuit for perfection, guidance and valuable suggestions have guided me at every step of my career. Particularly, I value his respect towards professionalism and the desire to excel higher and higher. Apart from research expertise, he has made me competent in organizing and managing research activities such as project/grant proposals, international conferences, and software demonstrations. Upon his supervision, I was also able to participate and compete in International software competitions and win awards at several occasions. I also thank him for giving me an opportunity to pursue a doctorate degree in the The Cloud Computing and Distributed Systems (CLOUDS) Laboratory and work with the excellent research group as a team member.

I would also like to thank all past and current members of the CLOUDS laboratory, at the University of Melbourne. In particular, I thank Christian Vecchiola, Linlin Wu, William Voorsluys, Mustafizur Rahman, Yuanhua He, Jinghua Fang, Marcos dias de Assunção, Jia Yu, Marco A. S. Netto, Chee Shin Yeo, Mukaddim Pathan, Srikumar Venugopal, Rajiv Ranjan, James Broberg, Xingchen Chu, Alex Stivala, Sungjin Choi, Kyong Hoon Kim, Saurabh Garg, Anton Beloglazov, Mohsen Amini, Chao Jin, Anthony Sulistio, Rodrigo Calheiros, Nithiapidary Muthuvelu, Dileban Karunamoorthy, Xiao Feng Wang, Alexandre di Costanzo, Amir Vahid, Adam Barker, Javadi Bahman, and Charity Laplap for their friendship and intellectual support during my PhD.

I also thank Professor Rao Kotagiri (University of Melbourne, Australia), Siddeswara Mayura Guru (CSIRO, Australia), James E. Dobson (Dartmouth College, USA), Carlos A. Varela (Rensselaer Polytechnic Institute, USA), Kenjiro Taura (University of Tokyo, Japan), Howard J. Siegel (Colorado State University, USA), Jemal H. Abawajy (Deakin University, Australia), Ivona Brandic (Vienna University of Technology, Vienna), Kenneth Chiu (SUNY Binghamton, Indiana University, USA), and Ahsan Khandoker (University of Melbourne, Australia) for their comments and suggestions on my research. I also thank Manish Parashar (Rutgers, The State University of New Jersey, USA) and Shantenu Zha (Louisiana State University, USA) for commenting and suggesting improvements during software demonstrations at CCGrid conferences.

Several experiments I present in this thesis were conducted in Grid'5000, a large-scale computing facility in France. I thank the excellent support from the Grid'5000 team. I am also grateful to several research labs that provided access to their compute infrastructure for our experiments, which are: Amazon EC2, 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.

I acknowledge the University of Melbourne and the Australian Government for providing me with scholarships to pursue my doctorate studies. I also thank the Department of Computer Science and Software Engineering (CSSE), Melbourne School of Engineering, IEEE Technical Committee for Scalable Computing (TCSC), The Victorian Life Sciences Computation Initiative (VLSCI), and CLOUDS Lab for supporting my travel to attend international conferences. My special thanks goes to Amazon for providing me an education grant, in terms of Amazon Web-Service credits, for carrying out cloud based software demonstrations within Australia. This work is partially supported from grants received

from the Australian Research Council (ARC) and Australian Department of Innovation, Industry, Science and Research. I also extend my gratitude to the staff members and friends from the Computer Science & Software Engineering Department for their help and support, especially Pinoo Bharucha, Madalain Marta Dolic, and Rosanna Parissi. I also thank the anonymous reviewers who evaluated the content of this thesis.

Finally, I thank my family for their love and support at all times. I especially thank my life partner, Mrs. Lianying Hao, who accepted my long hours at research with open heart and mind and encouraged me at every step of my career and life. I also acknowledge moral and emotional support I received from my grandmother, mother, father, and brother. All my friends and family have been a source of inspiration in my life.

*Suraj Pandey*
*Melbourne, Australia*
*December 2010*

# Contents

# List of Figures

# List of Tables

# Listings

# Nomenclature

| | |
|---|---|
| AMPL | A Modeling Language for Mathematical Programming |
| BRS | Best Resource Selection |
| CMS | Compact Muon Solenoid Experiment |
| D-HEFT | Dynamic Heterogeneous Earliest Finish Time |
| DAG | Directed Acyclic Graph |
| EDMH | Enhanced Dynamic Mapping Heuristic |
| EMO | Evolutionary Multi-objective Optimization |
| ESMH | Enhanced Static Mapping Heuristic |
| fMRI | Functional Magnetic Resonance Imaging |
| HBMCT | Hybrid Balanced Minimum Completion Time |
| HEFT | Heterogeneous Earliest Finish Time |
| HPC | High Performance Computing |
| IR | Image Registration |
| LIGO | Laser Interferometer Gravitational-Wave Observatory |
| NLP | Non-linear Programming |
| PSO | Particle Swarm Optimization |
| QoS | Quality of Service |
| SLA | Service Level Agreement |
| VM | Virtual Machine |
| WE | Workflow Engine |
| WfMS | Workflow Management System |
| xWFL | XML based Workflow Language |

# 1 Introduction

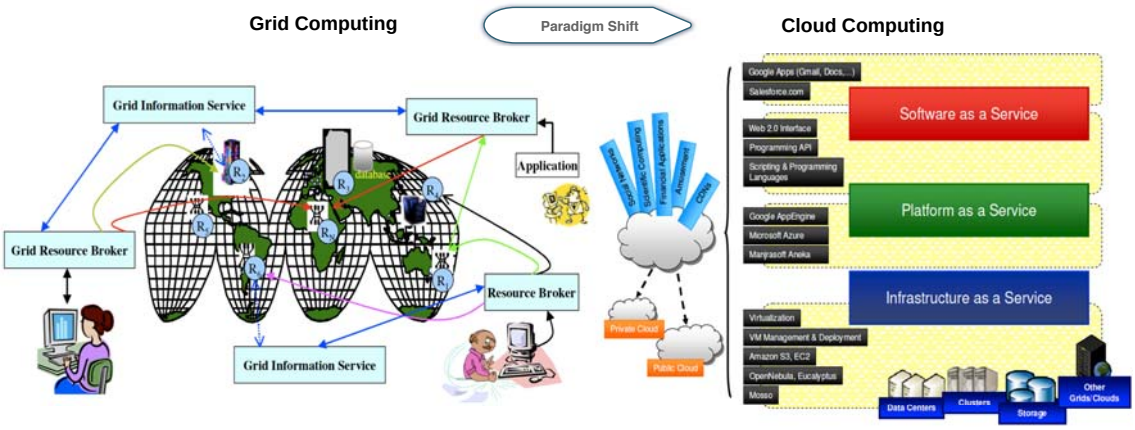
This chapter introduces a high-level description of the work presented in this thesis. It starts off by elaborating the concepts behind data intensive applications and workflow management systems in the context of Grid and Cloud computing. It then presents motivational applications that are very relevant to the problem statement and environment described subsequently. The chapter also describes the core contributions of this thesis. It finally concludes by giving an overview of its chapters, supported by publications.

## 1.1 Background

The field of distributed computing has seen technologies rapidly grow from desktop computing, through Grid computing, and now to Cloud computing. All these technologies focus on delivering computing power to a large number of end-users in a reliable, efficient and scalable manner. More or less, the trend has been to deliver the computing power as a utility, much like how water and electricity is delivered to households these days. This concept of utility computing can be backtracked to 1969, when Leonard Kleinrock, one of the chief scientists of the original Advanced Research Projects Agency Network (ARPANET) project, which seeded the Internet, said [74]: *"As of now, computer networks are still in their infancy, but as they grow up and become sophisticated, we will probably see the spread of, 'computer utilities', which, like present electric and telephone utilities, will service individual homes and offices across the country"*. This vision of computing utilities, based on a service provisioning model, has seen a massive transformation of the entire computing industry in the 21st century, where computing services are available on demand.

### 1.1.1 Grid and Cloud computing Platforms

Grid computing vision started by aggregating distributed resources that were existing under different administrative domains such that the end-users could transparently access them for conducting large scientific data analysis. Some of the production Grids,




- Based on ability to negotiate resource-sharing arrangements
- Coordinates independent resources
- Uses open standards and interfaces
- Allows for heterogeneity of computers
- Resources distributed across large geographical boundaries
- Loose coupling of computers and storage services
- Physical resource-sharing among large number of users usually limits dynamic provisioning and scalability within a single administrative domain

- Resources often leased from Cloud service providers on a pay-as-you-go (PAYG) basis
- Resources are managed in virtualised data centers
- Service provider uses virtualised resources that may be using services from multiple data centers
- Service oriented
- Dynamically provisioned, elastic and highly scalable

**Figure 1.1:** *Grid [22] and Cloud computing [25] platforms.*

such as TeraGrid [30], have been using Grid computing to facilitate parallel executions of both small and large scientific applications in domains, such as climate modeling, astronomy, computational biology, and so forth. Most Grid services have been realized by using standard Web services-based protocols. These services have been deployed in order to facilitate the discovery, access, allocation, monitoring and accounting of compute and storage resources. These requirements are specifically addressed by the Open Grid Services Architecture (OGSA), which is a distributed interaction and computing architecture based around services, assuring interoperability on heterogeneous systems so that different types of resources can communicate and share information. It describes an architecture for a service-oriented Grid computing environment for business and scientific use. The Globus Toolkit [52], as an open-source toolkit for building computing Grids, has implemented the OGSA, as well as the OGF (Open Grid Forum) defined protocols for resource management (e.g. Grid Resource Allocation & Management Protocol – GRAM), information services (e.g. Monitoring and Discovery Service – MDS), security services (e.g. Grid Security Infrastructure – GSI) and data movement and management (e.g. Global Access to Secondary Storage and GridFTP). A number of technologies have also been developed to work together with the Globus Toolkit as a meta-scheduler (e.g. Nimrod, GridWay), service broker (e.g. Gridbus Broker), Quality of Service (QoS) monitoring (e.g. Network Weather Service – NWS), and so forth.

Cloud computing, on the other hand, delivers infrastructure, platform, and software (applications) as services, which are made available as subscription-based services in a pay-as-you-go model to consumers. In industry, these services are referred to as Infras-

tructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS), respectively.

Cloud computing means different things to different people. As a result, there are several definitions and proposals [145]. Vaquero et al. [145] have propose a definition that is centered around scalability, pay-per-use utility model and virtualization. According to Garner, Cloud computing is a style of computing where service is provided across the Internet using different models and layers of abstraction.

Armbrust et al. [7] observe that: "Cloud computing refers to both the applications delivered as services over the Internet and the hardware and system software in the data centers that provide those services". This definition captures the real essence of this new trend, where both software applications and hardware infrastructures are moved from private environment to third parties data centers and made accessible through the Internet. Buyya et al. [26] define a Cloud as a *type of parallel and distributed system consisting of a collection of interconnected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements*. This definition puts Cloud computing into a market oriented perspective and stresses the economic nature of this phenomenon.

Figure 1.1 provides a high-level distinction between Grid and Cloud computing platforms. Clouds aim to power the next generation data centers by designing them as a network of virtual services (hardware, database, user-interface, application logic) so that users are able to access and deploy applications from anywhere in the world on demand at competitive costs depending on users Quality of Service (QoS) requirements [26]. It offers significant benefit to IT companies by freeing them from the low level tasks of setting up basic hardware (servers) and software infrastructures and thus enabling them to focus on innovation and creating business value for their services.

The key feature, emerging from the definition of Cloud given by Armbrust [7] and Buyya [26], is the ability to deliver both infrastructure and software as services that are consumed on a pay-per-use-basis. Previous trends, such as Grid computing, were limited to a specific class of users, or specific kinds of IT resources that were mostly shared on a volunteer basis. The approach of Cloud computing is global and encompasses the entire computing stack. It provides services to the mass, ranging from the end-users hosting their personal documents on the Internet to enterprises outsourcing their entire IT infrastructure to external data centers. Service Level Agreements (SLAs), which include QoS requirements, are set up between customers and Cloud providers. An SLA specifies the details of the service to be provided in terms of metrics agreed upon by all parties, and penalties for violating the expectations. SLAs act as a warranty for users, who can more comfortably move their business to the Cloud. As a result, enterprises can cut down maintenance and administrative costs by renting their IT infrastructure from Cloud vendors. Similarly, end-users leverage the Cloud not only for accessing their personal

data from everywhere, but also for carrying out activities without buying expensive software and hardware.

### 1.1.2 Data Intensive Applications as Workflows

A data intensive computing environment consists of applications that produce, manipulate, or analyze data in the range of hundreds of megabytes (MB) to petabytes (PB) and beyond [96]. According to Lee et al. [80], the field of data intensive computing constitute "the technologies, the middleware services, and the architectures that are used to build useful high-speed, wide area distributed systems". A data intensive application workflow has comparatively higher data workloads to manage than its computational load. In other words, the requirements of resource interconnection bandwidth for transferring data outweigh the computational requirements for processing tasks. This, as a consequence, demands more time to transfer and store data as compared to execution time for tasks in the workflow. It is common to characterize the distinction between data intensive and compute intensive by defining a threshold for the Computation to Communication Ratio (CCR). Applications with lower values of this ratio are distinctly data intensive in nature.

Standard application components of scientific data intensive applications can be combined to process the data in a structured way in contrast to executing monolithic codes [37]. The application is represented as a workow structure, which consists of tasks, data elements, control sequences and their dependencies. According to [177], scientific workow management systems are engaged and applied to the following aspects of scientific computations: 1) describing complex scientific procedures, 2) automating data derivation processes, 3) high performance computing (HPC) to improve throughput and performance, and 4) provenance management and query.

Many scientific applications in the field of astronomy, gravitational-physics, computational biology, climate modeling, and life-sciences have used workflow technology to carry out large-scale experiments [4; 141; 142; 47; 27]. Some of these applications are described as motivational examples in Section 1.2.

### 1.1.3 Workflow Scheduling and Management

In simple terms, a process of mapping of tasks in a workflow (or an entire workflow) to compute resources for execution (preserving dependencies between tasks) is termed as scheduling of workflows. Most applications can be represented in the form of a Directed Acyclic Graph(DAG), where cycles and conditional dependencies are absent. Once the workflow is instantiated in the form of a DAG, middleware technologies such as Pegasus [40], Gridbus Workflow Management System [166] and so forth, are used to schedule the tasks in the DAG onto the distributed. The objectives of scheduling a workflow can

vary from application to application. Most often, a data intensive application workflow is scheduled to minimize total data-transfer time and/or cost, storage space used, total execution time and/or a combination of these.

In order to schedule workflows, several related technologies are used in the scheduling middleware. A resource broker, which is an intermediate entity that acts as a mediator between resources and end users, performs resource allocation and/or scheduling, and manages execution of applications on behalf of one or multiple users. For instance, the Grid Service Broker [152] developed as part of the Gridbus Project, mediates access to distributed resources by discovering resources, scheduling tasks, monitoring and collating results. A user portal provides a user-friendly environment to the end-users (most often scientists) to compose, submit and monitor workflow applications. A workflow editor forms the graphical user-interface at the portal to facilitate the composition and visualization of workflows. A workflow editor, a management portal, a scheduling middleware together form a Workflow Management System (WfMS).

## 1.2 Motivation and Rationale

Scientific experiments like the Compact Muon Solenoid (CMS) experiment for the Large Hadron Collider (LHC) at CERN[1], the Laser Interferometer Gravitational-Wave Observatory's (LIGO) science[2], the projects at Grid Physics Network[3] produce data in the scale of peta-bytes. These experiments are usually represented using workflows, where tasks are linked according to their data flow and compute dependencies. The workflow is classified as compute intensive when the computational needs of individual task are high. Similarly, the workflow is classified as data intensive when the data requirements (e.g. size of each data file, number of files, data storage etc.) are high.

Data intensive workflows can take advantage of the infrastructure provided by environments like data grids. Data Grids provide services such as low latency transport protocols and data replication mechanisms to distribute data intensive applications that need to access, process and transfer large data sets stored in distributed repositories [150].

Scientific applications are executed on distributed resources with the primary objective of optimizing the total makespan of the application's workflow. Makespan of a workflow is defined as the total time taken for the completion of all the tasks in the workflow. Besides the makespan, cost of execution also forms one of the objective functions of scheduling these scientific applications. The makespan depends on both the communication time involved in staging the input and output files and the computation time to execute them. Scheduling the tasks to optimize only one objective results in sub-optimal result as both

[1] http://lhc.web.cern.ch/lhc/LHC_Experiments.htm
[2] http://www.ligo.caltech.edu/advLIGO/
[3] http://www.griphyn.org/

the objectives affect each other.

Traditional scheduling algorithms focused on pull model. The algorithm required to stage data towards the computation resources. They did not consider the location of data. Ignoring locality of data incurs high bandwidth utilization cost. Locality of data cannot be ignored when size of data is huge. This demands a different model where the selection of both data-host and compute-host should be made rather than selecting data-host first and then compute-host or vice versa.

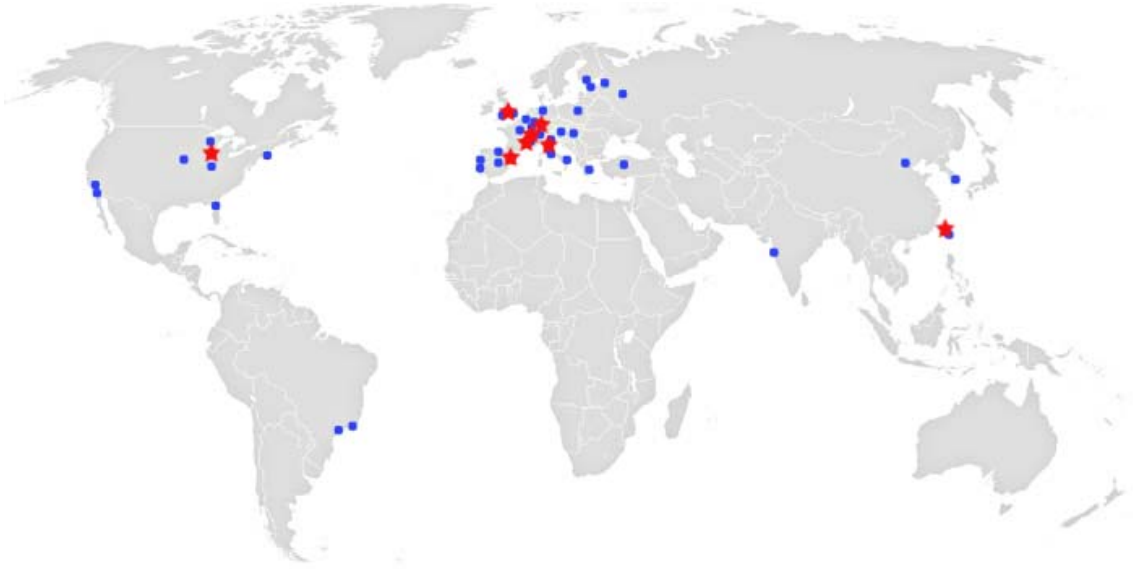


**Figure 1.2:** *Tier-1 (red star) and Tier-2 (blue squares) sites worldwide in CMS (Image courtesy of James Letts, http://cms.web.cern.ch, May 2008)*

**Case 1:** The Compact Muon Solenoid Experiment (CMS) "still produces a huge amount of data that must be analyzed, more than five petabytes per year when running at peak performance[4]". It has large number of "Tier-2" analysis centers where physics analysis are performed, as depicted in Figure 1.2. 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 [11]. 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".

**Case 2:** The Laser Interferometer Gravitational-Wave Observatory (LIGO) is a facility dedicated to the detection of cosmic gravitational waves and the harnessing of these

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

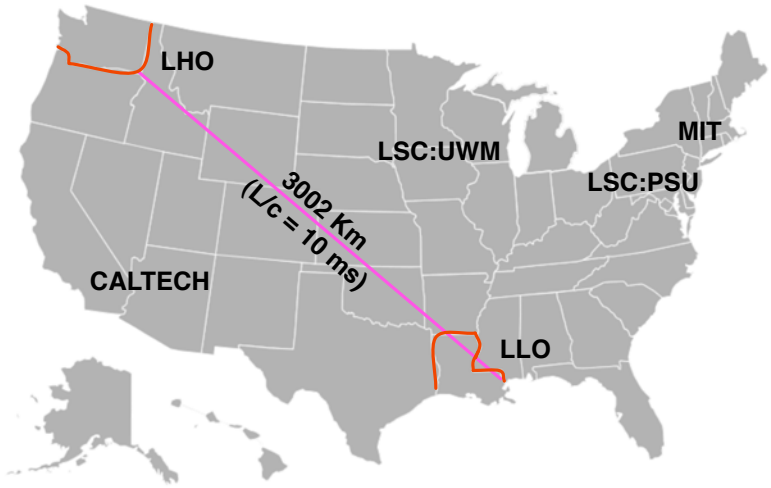



**Figure 1.3:** *LIGO Data Grid (LDG) (courtesy:http://ligo.org.cn/testbeds.shtml)*

waves for scientific research. 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 1.3, 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. With the help of computer-coupled observatories, LIGO has been analyzing data since 2002 in an effort to detect and measure cosmic gravitational waves.

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 task scheduling* [104]. The analysis of data from gravitational-wave detectors are represented as workflows. The workflow analyzes data looking for inspiral signals, which can occur when two compact objects, such as neutron stars or black holes, form binary systems. 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.

The LIGO Lab, the LIGO Scientific Collaboration (LSC), and international partners, are proposing Advanced LIGO to improve the sensitivity by more than a factor of 10. Since the volume of space that the instrument can see grows as the cube of the distance, this means that the event rates will be more than 1,000 times greater. Advanced LIGO will equal the 1 year integrated observation time of initial LIGO in roughly 3 hours. The huge amount of data produced will be made available to all of its sites. The selection of data source and compute-host will need to be done in an efficient manner such that the cost and time of execution are minimized.

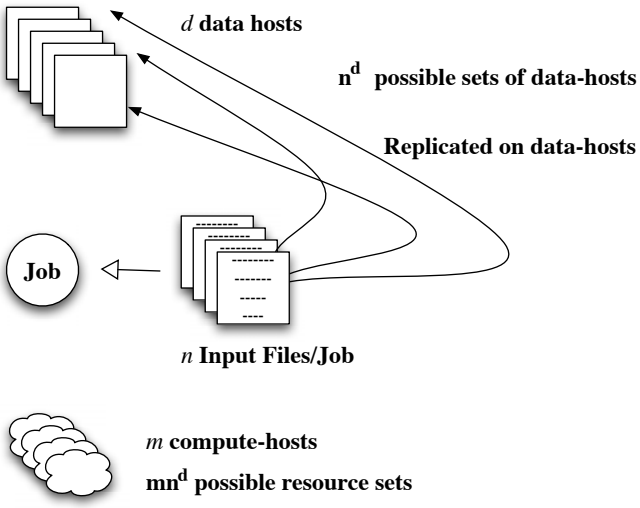
**Rationale:** In Case 1, data is being shared via a central repository with its Tier-2 members and Tier-2 caches these data for short-term usage. In the three hypothetical 'use cases' presented in [11], scientists are continuously sharing the cached data for repeated experiments and analysis. Therefore, 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.

## 1.3 Problem Description

Scientific application workflows listed in Section 1.2 are generally executed using distributed resources, where data required by the application can be retrieved from several data-hosts as there exist replicas of data files. Data has to be staged to a compute resource before any task associated with the data can be executed at that resource. At the end of execution or during the execution, output data is produced that may also be of similar sizes to the input data. These intermediate data should be stored for subsequent tasks requiring them. The sites where the output data are stored could be potential sources of data depending on the policy of retaining or deleting the output data. The total number of data-hosts thus increases as the intermediate output files are stored. This adds complexity to the selection of data hosts.

The computation requirements of these tasks cannot be totally ignored. After the set of candidate data-hosts is found, the tasks have to be assigned to compute-hosts for execution. The mapping of the tasks to compute hosts depends on the objective function. Scheduling of the tasks in the workflow primarily focuses on some of the objective functions or combination of them: minimizing the total makespan, minimizing the overall cost (economic cost) of execution and data transfer, executing within the deadline and allocated budget. The mapping of the workflow tasks to minimize one of the objective functions is a complex sub-problem of the general task scheduling problem. The problem becomes complex with the addition of replicated data sets with tasks requiring more than




Example

| $n$ | $d$ | $m$ | Total |
|---|---|---|---|
| 5 | 10 | 100 | 1000000 |
| 2 | 10 | 100 | 2000 |
| 1 | 10 | 100 | 1000 |
| 1 | 1 | 100 | 100 |

**FIGURE 1.4:** *Datahost and Compute host selection problem for a single task.*

a single file. Figure 1.4 introduces the complexity of finding resource and data-host match for a single task.

In Figure 1.4, a 'task' requires $n$ input files. These $n$ input files are replicated on $d$ data hosts. This gives us $n^d$ possible sets of data hosts. If we have $m$ compute hosts, the total possible resource set combination would be $mn^d$. A table alongside the figure tabulates the total number of combinations possible for varying number of files with fixed number of data hosts and compute hosts.

When the files are replicated and a single task requires more than a single file, the number of comparisons needed to come to the best solution (a combination that gives a set of datahosts and compute hosts) increases. When dependent tasks are present, the problem of finding the combination of compute resources and data-host set becomes a non-trivial problem. This basic problem becomes highly complex when there are constraints involved, such as heterogeneous resources, network costs, storage constraints, etc.

Before defining the problem statement, this chapter first presents a workflow model and a resource model.

### 1.3.1 A Workflow Model

A workflow is represented by a directed acyclic graph $G = (V, E)$, where $V = \{t_1, t_2, ..., t_n\}$, and $E$ represent the vertices and edges of the graph, respectively. Each vertex represents a task $t$ and there are $n$ tasks in the workflow. The edges maintain execution precedence constraints. Having a directed edge from $t_x$ to $t_y$, $x, y \in \mathbb{N}$ means that $t_y$ cannot start to

execute until $t_x$ is completed. The components are described as follows:

1. A set of tasks $T = \{t^1, t^2, \ldots, t^n\}$
2. A set of files $F = \{f_1, f_2, \ldots, f_n\}$
3. A set of compute and data resources $R = \{r_1, r_2, \ldots, r_n\}$

A task $t^x$ requires a set of files $F_x = \{f_1, f_2, \ldots, f_n\}$ to be staged in for execution. Each file $f_x$ required by a task $t^x$ is hosted by multiple data-hosts. Partial segments of each file $d_x \subset f_x$ that need to be transferred for a task $t^x_r$ assigned to a resource $r$ is denoted by a set: $data\_set = \{\{d_x \rightarrow r\}^{t^x_r} \; \forall d_x \subset f_x, r \in R, |\{d_x\}| \leq |R|\}$. The index $x$ establishes a relationship between the set of files that the task needs and the task itself, in the set containing numerous tasks and files.

### 1.3.2 Assumptions

Before defining the problem of scheduling and management of data intensive workflows, and presenting solutions, this thesis makes the following assumptions:

1. This thesis does not consider conditional and cycles in the workflow structure. This leads to an assumption that the cycles and conditions in the workflow can be represented in the form of a *DAG* for the scheduling system to execute them, whenever necessary.
2. This thesis assumes that data required by workflow applications are replicated at multiple sites distributed geographically around the world. This would then enable the data retrieval components to download data in parallel from these distributed sources.
3. 

These assumptions are practical and are based on the motivational examples presented in Section 1.2.

### 1.3.3 Problem Definition

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 [101].

**Def1:** *DTSP*$(D, R, T, F, G, L, M)$ *Given a set of 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* $= \{t^x_r\}, t^x \in T, r \in R$*], and the partial data set (data_set* $= \{t^x_r\}$*)*

*to be transferred from selected data-hosts to assigned compute hosts data_set* $= \{\{d_x \rightarrow r\}^{t_r^x} \; \forall d_x \subset f_x, r \in R, |\{d_x\}| \leq |R|\}$ *for each task,* ***simultaneously****, such that: total execution time (and cost) at r and data transfer time (and cost) incurred by data transfers* $\{d_x \rightarrow r\}$ *for all the tasks are minimized.*

This thesis assumes the following pre-conditions that are associated with $Def1$:

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 $T$ of a workflow graph $G$

To simply the problem for understanding, it can be broken down to two stages. First, a set of data-hosts that hosts the required files for the tasks in the workflow should be found. The selection of the optimal set of data-hosts in the presence of large number of replicated files for a single task is computationally intensive [150]. Venugopal et al. [150] selected the data-hosts by using one of the solutions to the Set-Coverage problem [9]. This selection procedure is compute intensive.

Second, the mapping of tasks to compute-resources is an *NP-complete* problem in the general form. The problem is *NP-complete* even in two simple cases: (1) scheduling tasks with uniform weights to an arbitrary number of processors and (2) scheduling tasks with weights equal to one or two units to two processors [144]. There are only three special cases for which there exist optimal polynomial-time algorithms. These cases are (1) scheduling tree-structured task graphs with uniform computation costs on an arbitrary number of processors [64]; (2) scheduling arbitrary task graphs with uniform computation costs on two processors [92]; and (3) scheduling an interval-ordered task graph [50]. However, these solutions assume communication between tasks to take zero time.

The mapping of tasks in a workflow to the resources in our case is significantly different than the general mapping problem. Given the replicated files and numerous data-hosts, proper selection of data-hosts and compute-hosts should be made for every task in the workflow such that the dependent tasks will benefit from selection set of its parents. The best case is when each task gets the optimal data-host set and compute-host for execution so that the makespan and cost are optimal. The naive case is when the resource set is chosen irrespective of the dependencies for each task.

Figure 1.5 shows a simple workflow with compute-hosts $C$ and data-hosts $Dh$, assigned to each task (the separation of data-hosts and compute-hosts are for clarity only). Lets first consider the data-transfers occurring due to the selection of compute-host and data-host for task $a$ and task $c$. Since the tasks are mapped to two different compute-hosts $C_1$ and $C_2$, the output files from task $a$ need to be transferred from $C_1$ to $C_2$ with cost $dt_{c12}$. Task $a$ has

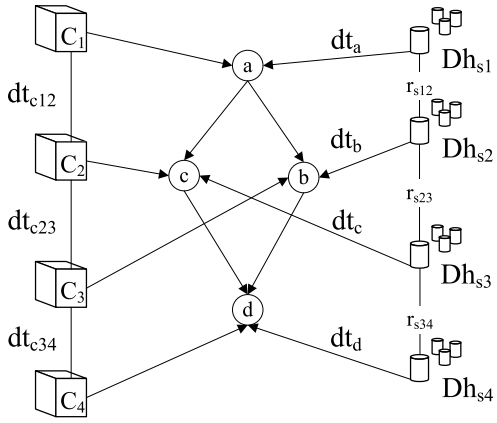



**Figure 1.5:** *Data-host and Compute-host Selection Problem.*

cost $dt_a$ and task $c$ has cost $dt_c$ for transferring data from $Dh_{s1}$ and $Dh_{s3}$, respectively. Now the optimal solution would select a combination of $C_1$, $C_2$, $Dh_{s1}$ and $Dh_{s3}$ to minimize the data transfer cost $\sum (dt_{c12} + dt_a + dt_c)$ and execution time $\sum (C_1 + C_2)_t$. There are several ways to select the data-hosts $Dh_{s1}$ and $Dh_{s3}$ and compute-hosts $C_1$ and $C_2$. To understand the problem statement, we describe two techniques. The first method is by considering the proximity of data-hosts in terms of network distance with the set of compute-hosts. The second method is by trying to maximize the co-relation between two set of data-hosts, depicted as $r_{s12}$ in the figure, for some set of tasks that require same set of files.

The first method always searches the entire set of data-hosts and compute-hosts to find one combination that most closely satisfies the objective function. But it does not take into account that tasks might be sharing the same set of files. Moreover, previous iterations might have already found the data-host and compute-host set combination that can be applied to subsequent set of tasks. For example tasks $a$ and $c$ might be requiring same set of files, so the compute intensive selection of *candidate* data-host sets and compute-host can be avoided for the second task. The second way would almost always select the same set of data-hosts by trying to maximize their co-relation. This also restricts the compute-host set to within the proximity of the co-related data-host set. In the latter case, when the number of tasks increases, both the compute-host and data-host become overloaded, as most of the tasks are mapped to these limited resources. This increasing waiting time of all tasks mapped to these resources. Hence a proper selection algorithm should distribute the load and satisfy the objective function.

### 1.3.4 Challenges

The problem of scheduling workflow, as defined in $Def1$ in Section 1.3.3, faces the following key challenges, which this thesis addresses as contributions in Section 1.4:

- **Scheduling Policy.** A workflow's makespan and overall cost depends on the selection of data sources and mapping of tasks to resources. Selection of 'best' data sources before irrespective of compute sources, as is done in existing scheduling algorithms, does not give time and cost efficient schedules when the size of data is comparatively larger than the computation time of tasks. Significant bandwidth is required to stage-in and stage-out these data prior to the execution of the tasks in a workflow. Similarly, if the data is to be re-used, the scheduling policy must select nearer (in terms of network distance) compute resources as selecting them affects the time/cost of transferring output data, and hence the overall execution time/cost. If the compute host and data-host are closer in terms of network distance, the transfer time is significantly reduced. The questions that needs to be answered are:

  1. How to select data hosts and compute hosts such that the task-resource mappings give a schedule that optimizes overall time and/or cost ?
  2. How to calculate data-hosts and compute-hosts proximity in terms of network distance dynamically?
  3. How to estimate job execution time and data-transfer times ?
  4. How to adjust task-resource mappings to fit the execution environment at run-time?
  5. How to make use of output (or temporary) data? Where to store execution data?

- **User QoS.** Users' Quality of Service (QoS) such as budget (cost payable for using the services) and deadline (time taken for application execution), should also be taken into account while scheduling workflows. The challenging questions are:

  1. How to incorporate user QoS into scheduling policy so that either or both could be optimized?
  2. As optimizing both budget and deadline could be computationally challenging, which heuristics based algorithm to use to reduce the algorithm computation time?

- **Fault Tolerance.** If a task fails, the delay in execution affects the starting time of all other dependent tasks in the workflow unless an efficient fault tolerance mechanism is in place. Generally, for data intensive application, most of the time is spent on communicating data between tasks and resources. As a result, there is a high chance of failures occurring during this phase. When any task fails to execute, based on the scheduling policy, the tasks is either migrated to another host or re-inserted

to be executed on the same host. If data transfer occurs due to rescheduling, the migrated tasks may fail again at the new site. Before staging the task to the site, an informed decision of its past execution, current state of resources available and reliability issues must be taken into account. The questions are:

1. Which transfer mechanism to use to prevent data transfer failures?
2. How can data be transferred such that migration of workflow tasks to remote hosts can be prevented?
3. How to manage the data provenance?

## 1.4 Contributions

This thesis makes several contributions towards management and scheduling of data intensive application workflows on distributed resources. Driven by the motivation, the problem statement, and the challenges described in previous sections of this Chapter, the major contributions are:

1. **A comprehensive survey on scheduling and management techniques for data intensive applications.**

   This thesis provides a comprehensive survey of scheduling and management techniques that have been proposed in the past for data intensive application workflows. The survey covers most of the work in the field of Grid computing and also adds recent work on Cloud computing. These work are classified into groups and subgroups in relation to their proposed techniques, which in turn helps us identify their strength and weaknesses. The classification of techniques helps researchers and scientists working on workflow scheduling to correctly related scheduling algorithms and techniques to their problem environment. It also aids them in choosing past work for comparing against their techniques.

2. **A Workflow management system for data intensive applications.**

   This thesis presents a workflow management system design that facilitates the creation, scheduling, execution, and monitoring of data intensive application workflows on Grid and Cloud environments. Specifically, it describes the implementation of a workflow editor, workflow management portal, workflow engine, and resource plug-in that all form the major components of the management system. The implemented prototype system is then is used for executing two real-world applications on Grid and Cloud platforms. The thesis also presents the results obtained after the executions of these application to demonstrate its capability to handle large applications.

3. **Workflow scheduling algorithms for data intensive applications.**

   Before proposing scheduling algorithms, this thesis begins by providing two motivational applications that are currently used in the real-world. It describes the challenges that these applications face while in practice. It then formally defines the workflow scheduling problem. Then it proceeds towards implementing the workflow management system to address these challenges.

   The thesis proposes several workflow scheduling algorithms that are implemented in the workflow management system. The scheduling algorithms are based on multi-source data retrieval technique. It first describes a non-linear programming (NLP) model based scheduling algorithm that produces near optimal results, which would then be an ideal case for comparison against heuristics based approaches. It shows that the NLP approach is computationally expensive for large data sizes. The thesis then proposes heuristics based algorithms that are practically feasible when used for scheduling data intensive applications. It proposes a static and a dynamic scheduling heuristic that uses multi-source parallel data retrieval based scheduling approach and compares it with existing static and dynamic algorithms. These algorithms are used for minimizing the total execution time of workflow applications. Observing that the dynamic version of the scheduling heuristic performs best for data intensive application, the thesis then proposes a particle-swarm optimization (PSO) based heuristic. The PSO based scheduling heuristic is designed to achieve minimum cost of execution for data intensive application workflows. Due to its fast convergence property, PSO based heuristic is shown to reduce significant cost for sample application executed on Cloud platforms. The thesis compares the results obtained from these algorithms with existing 'single-source' retrieval based heuristics in terms of total makespan and execution cost.

4. **Real-world application workflows as case studies.**

   To demonstrate the proposed heuristics are generic and hence applicable for a range of applications workflows, the thesis models, implements and executes real world data intensive application workflows. They are: i) Image Registration procedure in functional magnetic resonance imaging (fMRI) studies, ii) data mining process for distributed intrusion detection, iii) parallel version of Evolutionary Multi-Objective Optimization (EMO), and iv) workflow structures similar to LIGO and Montage applications. The thesis uses these applications when evaluating the proposed algorithms on distributed resources.

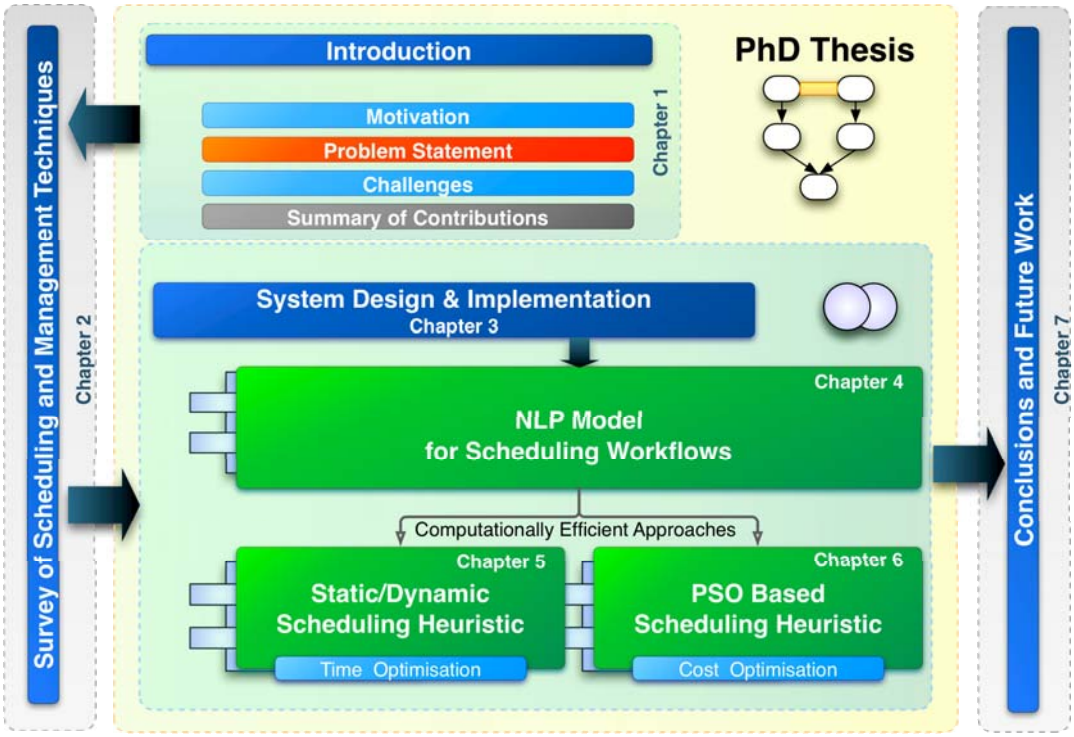



**Figure 1.6:** *Thesis chapter organization and contribution.*

## 1.5 Thesis Overview

Each chapter in this thesis has been derived from various publications during my PhD candidature. Figure 1.6 depicts a pictorial representation of the organization of thesis, which is described below in detail:

- **Chapter 2: A Survey of Workflow Management Techniques for Data Intensive Applications**

  This chapter surveys past work on management techniques of data intensive workflows. It classifies these techniques into groups and subgroups in accordance to their task and data management techniques.

  The chapter is derived from the following publication:

  - **Suraj Pandey**, Rajkumar Buyya. Scheduling and Management Techniques for Data-Intensive Application Workflows, Data Intensive Distributed Computing: Challenges and Solutions for Large-scale Information Management, T. Kosar (ed), IGI Global, USA, 2011.

- **Chapter 3: A Workflow Management System Design**

This chapter presents a workflow management system design and its describes its components in detail. It also presents two real-world case studies that are executed on real platforms using the proposed system.

The chapter is derived from the following publications:

- **Suraj Pandey**, William Voorsluys, Mustafizur Rahman, Rajkumar Buyya, James Dobson, Kenneth Chiu, A Grid Workflow Environment for Brain Imaging Analysis on Distributed Systems. Concurrency and Computation: Practice and Experience, Volume 21, Number 16, Pages: 2118-2139, ISSN: 1532-0626, Wiley Press, New York, USA, November 2009.
- **Suraj Pandey**, William Voorsluys, Mustafizur Rahman, Rajkumar Buyya, James Dobson, Kenneth Chiu, Brain Image Registration Analysis Workflow for fMRI Studies on Global Grids. In Proceedings of the 23rd IEEE International Conference on Advanced Information Networking and Applications (AINA 09), Bradford, UK, May 2009.
- Christian Vecchiola, **Suraj Pandey**, and Rajkumar Buyya, High-Performance Cloud Computing: A View of Scientific Applications, Proceedings of the 10th International Symposium on Pervasive Systems, Algorithms and Networks (I-SPAN 2009, IEEE CS Press, USA), Kaohsiung, Taiwan, December 14-16, 2009.

- **Chapter 4: A Non-Linear Model for Optimisation of Workflow Scheduling**

  This chapter describes the formulation of a non-linear programming model to minimize the data retrieval and execution cost of data intensive workflows in Clouds. The model retrieves data from Cloud storage resources such that the amount of data transferred is inversely proportional to the communication cost. It also presents an example of an intrusion detection application workflow and experiments on real platforms.

  The chapter is derived from the following publication:

  - **Suraj Pandey**, Kapil Kumar Gupta, Adam Barker and Rajkumar Buyya, Minimizing Execution Cost when using Globally Distributed Cloud Services, Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA 2010), Perth, Australia, April 20-23, 2010.

- **Chapter 5: Static and Dynamic Heuristics-Based Scheduling Algorithms**

  This chapter presents heuristics based scheduling algorithms that assigns inter-dependent tasks to compute resources based on both multi-source parallel data retrieval time and task-computation time. In addition to proposing both static and dynamic algorithms, it also describes the application of Steiner tree for resource selection.

The chapter is derived from the following publications:

- **Suraj Pandey** and Rajkumar Buyya, Scheduling of Scientific Workflows on Data Grids, TCSC Doctoral Symposium, Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2008, IEEE CS Press, Los Alamitos, CA, USA), May 19-22, 2008, Lyon, France.
- **Suraj Pandey**, Scheduling Data Intensive Applications based on Multi-Source Parallel Data Retrievals, Doctoral Research Showcase, ACM/IEEE Supercomputing Conference 2010 (SC10), November 13-19,2010 New Orleans, LA, USA.

- **Chapter 6 Particle Swarm Optimization Based Scheduling Heuristic**

  This chapter presents a Particle Swarm Optimization (PSO) based heuristic to schedule workflow applications on Cloud resources. It demonstrates the advantages of using PSO, such as faster convergence and lower computation time, for obtaining better or similar solutions than existing algorithms.

  The chapter is derived from the following publication:

  - **Suraj Pandey**, Linlin Wu, Siddeswara Guru, and Rajkumar Buyya, A Particle Swarm Optimization (PSO)-based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments, Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA 2010), Perth, Australia, April 20-23, 2010. **Best Paper Award**

- **Chapter 7 Conclusion and Future Directions**

  This chapter concludes the thesis by summarizing the contributions. It also provides insights on future work that could be carried out based on the work presented in this thesis.

# 2

# A Survey of Workflow Management Techniques for Data Intensive Applications

This chapter presents a comprehensive survey of algorithms, techniques and frameworks used for scheduling and management of data intensive application workflows. Many complex scientific experiments are expressed in the form of workflows for structured, repeatable, controlled, scalable and automated executions. This chapter focuses on the survey of management techniques for the type of workflows that have tasks processing huge amount of data, usually in the range from hundreds of megabytes to petabytes. Scientists are already using distributed systems that schedule these workflows onto globally distributed resources for optimizing various objectives: minimize total makespan of the workflow, minimize cost and usage of network bandwidth, minimize cost of computation and storage, meet the deadline of the application, and so forth. This chapter classifies the techniques and presents a comprehensive survey. A survey of workflow management techniques is useful for understanding the working of the Grid and Cloud systems, providing insights on performance optimization of scientific applications dealing with data intensive workloads.

## 2.1 Introduction

Scientists and researchers around the world have been conducting simulations and experiments as a part of medium to ultra large-scale studies in high-energy physics, biomedicine, climate modeling, astronomy and so forth. They are always seeking cutting-edge technologies to transfer, store and process the data in a more systematic and controlled manner as the data requirements of these applications range from megabytes to petabytes. Thus, to help them manage the complexity of execution, transfer and storage of results of these large-scale applications, the use of a workflow management systems (WfMS) is in wide practice [167].

Scheduling and management of computational tasks of a workflow were the main focus of WfMS in the past. With the emergence of globally distributed computing resources

and increasing output data from scientific experiments, scientists began to realize the necessity of handling data in conjunction with computational tasks. Scientific workflows were then modeled taking into account the flow of data. However, even with a plethora of techniques and systems, many challenges remain in the area of data management related to workflow creation, execution, and result management [38; 58].

Some challenges for managing data intensive application workflows are:

- High throughput data transfer mechanisms
- Massive, cheap, green and low latency storage solutions and their interfaces
- Composition of scientific applications as workflows
- Multi-core technology and workflow management systems
- Standards for interoperability between workflow systems
- Globally distributed data and computation resources

This chapter classifies and surveys techniques that have been used for managing and scheduling data intensive application workflows to meet the challenges listed above. The classification is based on techniques that take into account data, storage, platform and application characteristics. It sub-divides each general heading into more specific techniques. Then, it lists and describes several work under each sub-heading. Most systems use a combination of existing techniques to achieve the objectives of an application workflow.

The chapter starts by presenting previous studies that focused more on systems side of Grid workflows and Data Grids along with their taxonomy. It then describes an abstract model of a WfMS and its component responsible for data and computation management. The rest of the chapter presents the survey. The chapter finally concludes by identifying research issues in management of data intensive application workflows.

## 2.2 History of Workflow Management Systems

### 2.2.1 Taxonomy of Related Technologies

Over the last few years, we can find much work being done on data intensive environments and workflow management systems. We list taxonomies for Data Grid Systems and Workflow management Systems that present the grounds for our survey.

Venugopal, Buyya, & Ramamohanarao [151] proposed a comprehensive taxonomy of Data Grids for distributed data sharing, management and processing. They characterize, classify and describe various aspects of architecture, data transportation, data replication and resource allocation, and scheduling for Data Grids systems. They list the similarities

and differences between Data Grids and other distributed data intensive paradigms such as content delivery networks, peer-to-peer networks, and distributed databases.

Yu & Buyya [167] proposed taxonomy of workflow management systems for Grid computing. They characterize and classify different approaches for building and executing workflows on Grids. They present a survey of representative Grid workow systems highlighting their features and pointing out the differences. Their taxonomy focuses on workflow design, workflow scheduling, fault management, and data movement.

Bahsi, Ceyhan and Kosar [8] presented a survey and analysis on conditional workflow management. They studied workflow management systems and their support for conditional structures such as if, switch, and while. With case studies on existing WfMS, they listed the differences in implementation of common conditional structures. They show that the same structure is implemented in completely different ways by different WfMS. A system or a user can define explicit conditions in the structure of a workflow to manage the data flow across resources and between tasks for data intensive application workflows.

Yu, Buyya, & Ramamohanarao [169] listed and described several existing workflow scheduling algorithms developed and deployed in various Grid environments. They categorized the scheduling algorithms as either best effort based or Quality of Service (QoS) constraint based scheduling. Under best-effort scheduling, they presented several heuristics and meta-heuristics based algorithms, which intend to optimize workflow execution times on community Grids. Under QoS constraint based scheduling algorithms, they examined algorithms, which intend to solve performance optimization problems based on two QoS constraints, deadline and budged. They also list some of the techniques we have explicitly described for data intensive workflows in this chapter.

Kwok & Ahmad [78] surveyed different static scheduling techniques for scheduling application Directed Acyclic Graphs (DAGs) onto homogeneous platforms. In their model, tasks are scheduled onto multiprocessor systems. The model also assumes that communication is achieved solely by message passing between processing elements. They proposed taxonomy that classified the scheduling algorithms based on their functionality. Their survey also provides examples for each algorithm along with the overview of the software tools for scheduling and mapping.

### 2.2.2 Abstract Model of a Workflow Management System

An abstract model of a workflow management system consists of components that deliver the functional characteristics to the system. It consists of user interface components, middleware components, and resource plug-ins that complete the bidirectional flow from a end-user to the distributed resources.

Figure 2.1 shows the architecture of a Grid workflow system based on the workflow reference model [62] proposed by Workflow Management Coalition (WfMC) (www.wfmc.org)

in 1994. We have extended it to include components that manage data in addition to tasks.

Yu & Buyya [167] described the abstract model in detail, but without the data-centric components. The build time and run time borders separate the functionality of the design to define and execute tasks, respectively. At the core of the run time, we propose components to actively process both data and tasks equally, different from the model presented by Yu et al. [167], where data component was not managed by the scheduler as it did for tasks.

The scheduler, which forms the core of the engine, handles data flow schedules on top of task schedules. For example, if a workflow is modeled such that data transfer tasks are separate from computation tasks, the scheduler may apply a different scheduling policy to the data transfer tasks. Similarly, when there is no distinction between these tasks, the scheduler may prioritize data transfers between certain tasks over computation depending on the structure of the workflow, scheduling objectives, and so forth.

We propose to add a data provenance (also referred to as lineage and pedigree) manager component to the architecture. It keeps the record of data entities associated with the tasks in a workflow. The scheduler may interact with this component for determining specific data flow paths between tasks and distributed resources. For example, when a workflow is executed a number of times, previously produced data may exist that could be reused. In such cases, intermediate data transfer may not be scheduled for some tasks. Similarly, the scheduler may take reference of provenance data to create/delete data transfer and data cleanup tasks for storage aware scheduling. Simmhan, Plale, & Gannon [131] have surveyed and described systems using provenance for data intensive environments in greater detail.

We envision each component in the core architecture to handle data as a first class citizen as also proposed by Kosar & Livny [77]. Data movement component, in particular, should be smart enough to overlap data transfer tasks with computation so that wait-times for data-availability is minimized. Data-transfer tasks could be prioritized for different tasks. Similarly, fault tolerance policies should be capable of handling frequent failures of data transfer tasks. Scheduling steps heavily depend on the capability of data movement and fault tolerance components for data intensive applications, as the repercussions of failure of data transfer tasks can affect the performance of the entire workflow. Different from generic WfMS models, a higher and more sophisticated coordination mechanism is required between these components for handling data intensive application workflows.

New models for IT service delivery (e.g. Clouds Computing) are emerging. Workflow systems should be capable of interacting with these types of service oriented architectures so that it can better utilize the storage and compute facilities provided by them for optimized data delivery, storage and distributed access. Access and security policies may differ from existing Grid policies when resources are from centralized data centers.

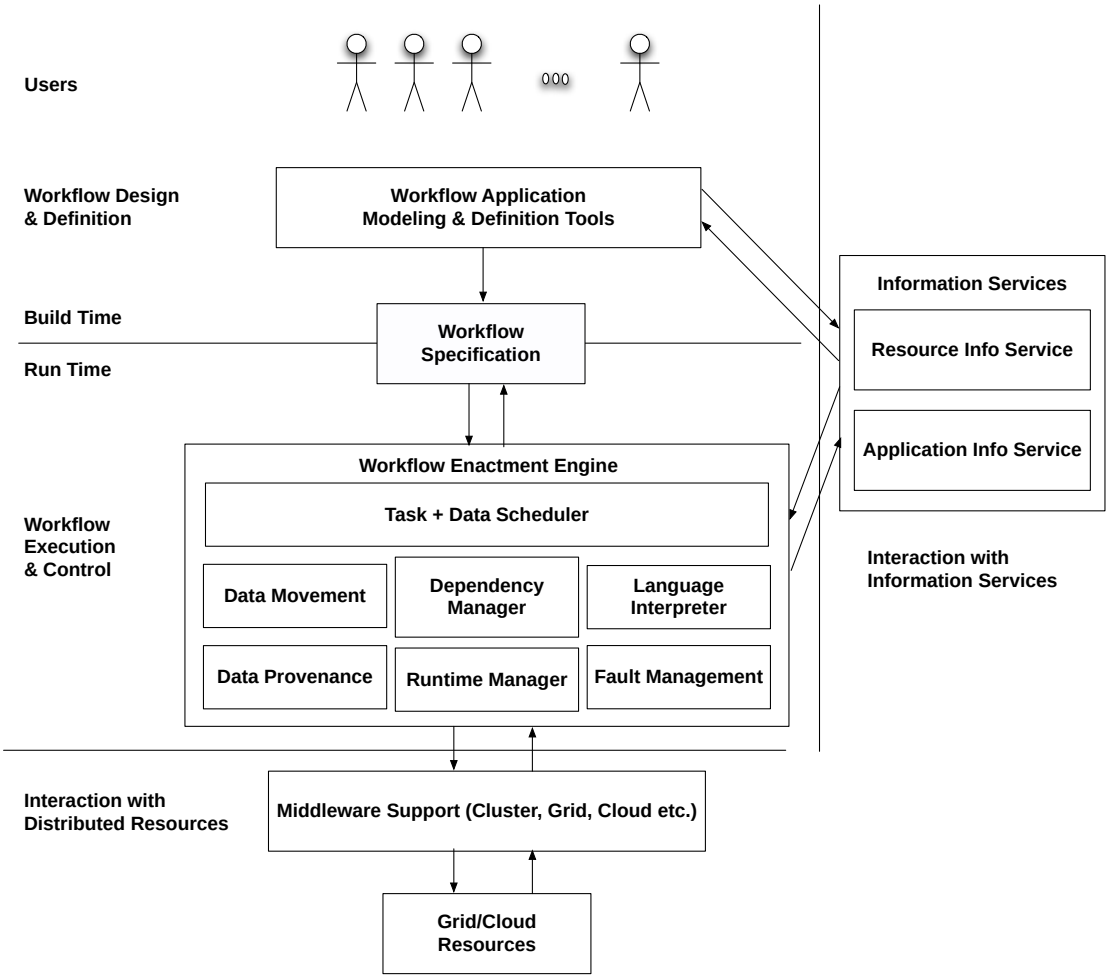



**FIGURE 2.1:** *An abstract model of a Workflow Management System*

## 2.3 Classification of Management Techniques

This section characterizes and classifies key concepts and techniques used for scheduling and management of data intensive application workflows. As shown in Figure 2.2, the techniques can be classified into seven major categories: (a) data locality, (b) data transfer, (b) data-footprint, (c) granularity, (d) model, (e) platform, and (f) miscellaneous technologies. This section describes each of these categories and their branches citing previous work done in the field.

### 2.3.1 Data Locality

In data intensive computing environments, the amount of data involved is comparatively large for communication networks, such as the Internet, to handle in specified amount of time with limited budget. Transferring of data between computing nodes takes significant amount of time depending on the size of data and network capacity between

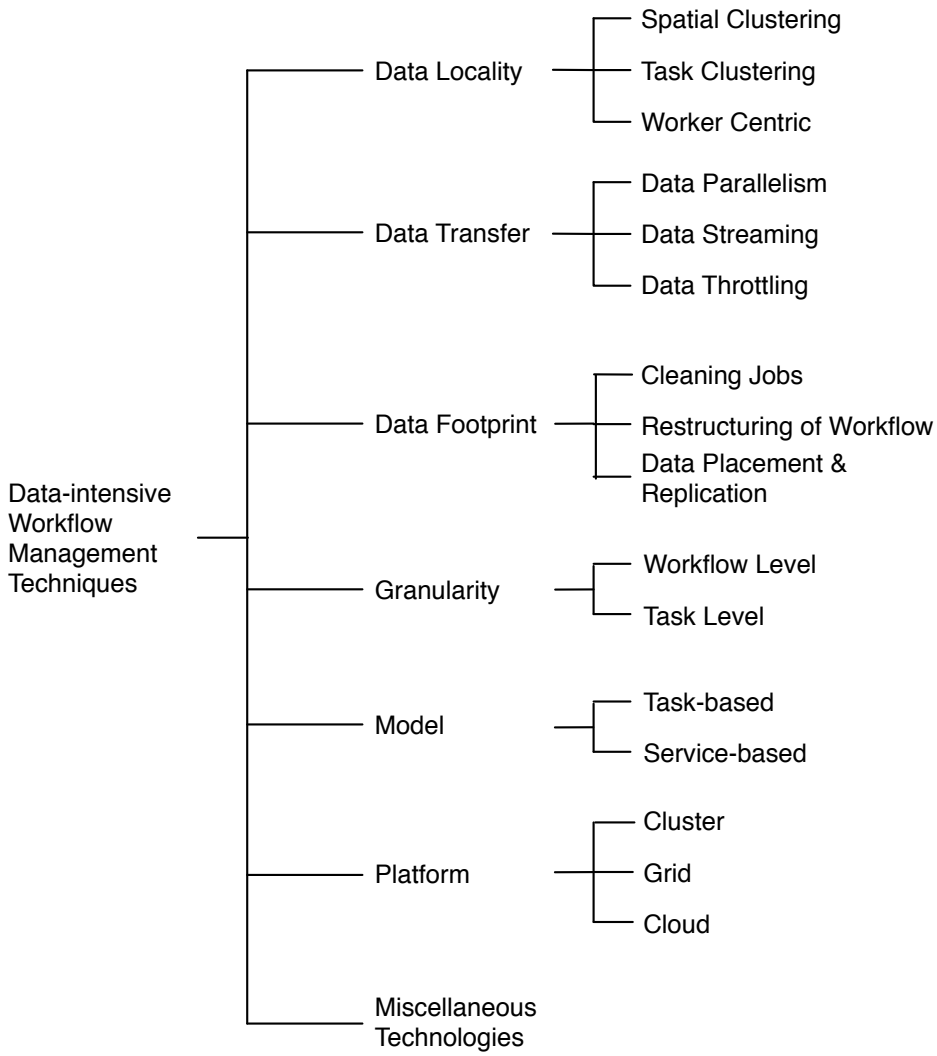



**Figure 2.2:** *Classification of management techniques for data intensive application workflows.*

participating nodes. Hence, most scheduling techniques target on optimizing data transfers by exploiting the locality of data. These techniques can be classified into: (a) Spatial clustering, (b) Task clustering, and (c) worker centric.

### Spatial Clustering

Spatial clustering creates a task workflow based on the spatial relationship of files in the input data set. In spatial clustering, clusters of jobs are created based on spatial proximity, each job then assigned to a cluster, each cluster to a grid site and during the execution of the workflow, all jobs scheduled belonging to the cluster to the same site (Meyer, Annis, Wilde, Mattoso, & Foster [94]). It improves data reuse and reduces total number of file transfers by clustering together tasks with high input-set overlap. These clustered tasks

are scheduled to the resource with the maximum overlap of input data. This reduction benefits the Grid as a whole by reducing traffic between the sites. It also benefits the application by improving its performance.

Meyer et al. [94] presented a generalized approach to planning spatial workflow schedules for Grid execution based on the spatial proximity of files and the spatial range of jobs. They proposed SPCL (for "spatial clustering") algorithm that takes advantage of data locality through the use of dynamic replication and schedules jobs in a manner that reduces the number of replicas created and the number of file transfers performed when executing a workflow. They evaluated their solution to the problem using the file access pattern of an astronomy application that performs coaddition of images from the Sloan Digital Sky Survey (SDSS)[1].

Brandic, Pllana & Benkner [16] developed QoS-aware Grid Workflow Language (QoWL), by extending the Business Process Execution Language (BPEL) that allows users to define preferences regarding the execution location affinity for activities with specific security and legal constraints. Use of QoS parameters that direct the WfMS to restrict the movement of sensitive and proprietary data to only agreed domains is very important for certain kinds of applications. A set of QoS-aware service-oriented components is provided for workflow planning to support automatic constraint-based service negotiation and workflow optimization.

#### Task Clustering

With task clustering, small tasks are grouped together as one executable unit such that the overhead of data movement can be eliminated. Task clustering groups tasks so that the intermediate files produced by each task in the group remains in the same computing node the grouped task was submitted to. Other tasks in the same group can now access the file locally. This scheme reduces the need to transfer the intermediate output files in case the tasks in the group were scheduled to different computing nodes. Clustering also eliminates the overhead of running small tasks.

Singh, Kesselman, & Deelman [133] explored approaches for restructuring of workflows so that the dependencies in the workflow graph can be reduced. They group independent jobs at the same level into clusters. Their task clustering does not imply that the tasks in a group is scheduled to one processor or executed sequentially. They show workflow performance using clustering with centralized (single submit host) and distributed (multiple submit hosts) job submission. In the centralized submission, the whole workflow is submitted and executed using a single submit host. In order to increase the dispatch rate of jobs for execution, their distributed job submission strategy has a central manager, multiple submit hosts and worker nodes. The workflow is restructured with

[1] https://www.darkenergysurvey.org

multiple clusters at each level. The number of clusters at each level is equal to the number of submit hosts in the pool. The schedulers on the submit hosts then try to find suitable nodes for the submitted jobs.

Pandey et al. [102] used task clustering to schedule data intensive tasks for a medical application workflow. They clustered tasks based on their execution time, data transfer and level. If tasks were having high deviation and value of average execution time, they were executed without clustering. Tasks with lower deviation and value of execution time were clustered together. They showed that clustering tasks for data intensive application workflows has better makespan than scheduling the workflow without clustering, mainly attributed to the decrease in file transfers between tasks in the same cluster.

#### Worker Centric

Worker centric approaches exploit locality of interest present in data intensive environments. Ko, Morales, & Gupta [75] presented an algorithm where one global scheduler, upon receiving a request from a worker (computation node), calculates the weight of each unscheduled task and chooses the best task to assign to the requesting worker. The weight calculation procedure takes into account the set of files already present at the worker site and additional files required by the worker for the task. This scheme exploits locality of file access, and thus minimizes both the number of files that need to be transferred as well as prefers workers that accessed the same files in the past. They proposed both deterministic and randomized metric that can be used with worker-centric scheduling and found that metrics considering the number of file transfers generally gave better performance over metrics considering the overlap between a task and a storage. They experiment with traces of Coadd[2].

### 2.3.2 Data Transfer

Researchers have proposed several mechanisms for transferring data so that data transfer time is minimized. These techniques are: (a) data parallelism, (b) data streaming, and (c) data throttling.

#### Data Parallelism

Data Parallelism denotes that a service is able to process several data fragments simultaneously with minimal performance loss. This capability involves the processing of independent data on different computing resources. Glatard, Montagnat, Lingrand, & Pennec [59] designed and implemented a workflow engine named MOTEUR. They propose algorithms that combine well-defined data composition strategies and fully parallel

[2]https://www.darkenergysurvey.org

execution. They adopted the Simple Concept Unified Flow Language (SCUFL) as the workflow description language for conveniently describing data flows. In their system, tasks and data are scheduled such that most data sets are processed by independent computing resources, but the precedence constraints are preserved. They evaluated the system using a medical imaging application run on the EGEE (Enabling Grids for E-Science EU IST project[3]) grid.

**Data Streaming**

In data streaming, real-time data generated through simulation or experiment is delivered in an asynchronous, high-throughput, low-latency and robust way to data analysis and storage machines. Bhat et al. [13; 14] investigated data streaming for executing scientific workflows on the Grid. They proposed the design, implementation and experimental evaluation of an application level self-managing data streaming service that enables efficient data transport to support Grid-based scientific workflows. The system provides adaptive buffer management mechanisms and proactive QoS management strategies based on model-based online control and user-defined policies. They showed that online data streaming can have significant impact on the performance and robustness of the data intensive application workflow applications in Grids. They used a fusion simulation workflow consisting of long-running coupled simulations to evaluate the data streaming service and its self-managing behaviors.

Bhat, Parashar, & Klasky [13; 14] investigated reactive management strategies for in-transit data manipulation for data intensive scientific and engineering workflows. Their framework for in-transit manipulation consists of processing nodes in the data path between the source and the destination. Each node is capable of processing, buffering and forwarding the data. Each node processes the data depending on its capabilities and the amount of processing still remaining. The data is dynamically buffered as it flows through the node. Eventually the processed data is forwarded until it reaches the sink. The choice between forwarding and further processing is dependent upon the network congestion. They used application level online controllers for high throughput data streaming.

Korkhov et al. [76] & Afsarmanes et al. [1] proposed Grid-based Virtual Laboratory AMsterdam (VLAM-G), a data-driven WfMS. Their system uses Globus services (Globus Project[4]) to allow data streams to be established efficiently and transparently between remote processes composing a scientific workflow. The execution engine initiates 'point-to-point' data streams between workflow components allowing intermediate data to flow along the workflow pipeline, without requiring local storage. They use unidirectional, typed streams to ensure that proper connection can be established. Control and monitoring communication is not transmitted on such typed streams. They model the system

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

such that all the resources needed for data stream driven distributed processing have to be made available (e.g. by advance reservation) simultaneously in contrast to the scenario where Grid resources join and leave the system at anytime.

#### Data Throttling

Data throttling is a process of describing and controlling when and at what rate data is transferred in contrast to moving data from one location to another as early as possible. In scientific workflows with data intensive workload, individual tasks may have to wait for large amounts of data to be delivered or produced by other tasks. Instead of transferring the data immediately to a task, it can be delayed or transferred using lower capacity links so that the resources can be dedicated to serve other critical tasks.

Park & Humphrey [105] identified the limitation of current systems in that there is no control available regarding the arrival time and rate of data transfer between nodes. They designed and implemented new capabilities for higher efficiency and balance in Grid workflows by creating a data-throttling framework that regulates the rate of data transfers between the workflow tasks via a specially created QoS-enabled GridFTP server. Their workflow planner constructs a schedule that both specifies when/where individual tasks are executed, as well as when and at what rate data is to be transferred. The planner allows a workflow programmer/engine to specify the requirements on the data movement delay. This delay helps to keep a balance between execution time of workflow branches by eliminating unnecessary bandwidth usage, resulting in more efficient execution.

DAGMan (Directed Acyclic Graph MANager)[5] is a workflow engine under the Pegasus [40] WfMS. It supports job and data throttling using parameters. Pegasus uses DAGMan to run the executable workflow. In DAGMan a "prescript" and a "postscript" step, associated with each workflow job, are responsible for transferring input files and deleting output files, respectively. It controls the number of prescripts that can be concurrently (across all jobs) started using the MAXPRE parameter. This serves as a convenient workflow-wide throttle on the data transfer load that the workflow manager can impose on the Grid from the submit host.

### 2.3.3 Data Footprint

Workflow systems adopt several mechanisms to track and utilize the data footprint of the application. These mechanisms can be classified into: (a) cleaning jobs, (b) restructuring of workflow, (c) data placement & replication.

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

### Cleaning Jobs

Cleaning jobs are introduced in the workflow to remove the data from the resources once its no longer needed. When applications require large amount of data storage, tasks in the workflow can only be scheduled to those compute resources that can provide temporary storage large enough to hold the input and output files the tasks need. Scheduling decisions should take into consideration the storage capability of the compute resource for all tasks with data intensive workloads.

Singh et al. [134] presented two algorithms for reducing the data footprint of workflow type applications. The first algorithm adds a cleanup job for a data file when that file is no longer required by other tasks in the workflow or when it has already been transferred to permanent storage. Given the possibility of data being replicated on multiple resources, the cleanup jobs are made on a per resource basis. The algorithm is applied after the executable workflow has been created, but before it is executed. The second algorithm is an improvement in terms of the number of cleanup jobs and dependencies it adds to the workflow. As the workflow engine has to spend considerable amount of time managing job execution for every added job or dependency, the authors design the algorithm to reduce the number of cleanup tasks at the possible cost of workflow footprint. This is achieved by adding at most one cleanup node per computational workflow task in contrast to one cleanup job for every file required or produced by tasks mapped to the resource as done in the first algorithm. It reduces data footprint but as a consequence the workflow execution time increases as a result of the increased number of workflow levels.

Ramakrishnan et al. [116] proposed an algorithm for scheduling data intensive application workflows onto storage-constrained resources. Their algorithm first takes into account disk space availability in resources and then prioritizes resources depending on performance. The algorithm starts by identifying all resources that can accommodate the data files needed for a task to be scheduled. If no resource is available that satisfies the space requirement of any ready task, the algorithm halts. It then tries to allocate the task to the resource that can achieve the earliest finish time (data transfer time and execution time) for the task. Finally, it cleans up any unnecessary data file remaining in the resource.

### Restructuring of Workflows

The structure of the workflow defines the data footprint. Restructuring of workflows is a transformation of the workflow structure such that it influences the way input/output data is placed, deleted, transferred, or replicated during the execution of the workflow. Task clustering and workflow partitioning are common ways to restructure workflows. Tasks can be clustered and dependencies re-defined in such a way that data transfer is minimized, data re-use is maximized, storage resources and compute resources have

well-balanced load and so forth. Singh et al. (2007) defined workflow restructuring as the ordering or sequencing of the execution of the tasks within the workflow. They restructure the workflow primarily to reduce the data footprint of the workflow. They introduce dependencies between stage-in tasks and the previous-level computational tasks. This prevented multiple data transfers from occurring at the same time as soon as tasks become ready.

Pegasus [40] has the capability to map and schedule only portions of the entire workflow at a given time, using partitioning techniques. Deelman et al. [40] demonstrate the technique using level-based partitioning of the workflow. The levels refer to the depth of the tasks in the workflow. In their Just-in-time planning algorithm [15], Pegasus waits (using DAGMan) to map the dependent workflow until the preceding workflow finishes its execution. Original dependencies are maintained even after partitioning. They also investigate partition-level failure recovery. When resources fail during execution, the entire task is retried and new partitions are not submitted to that resource.

Duan, Prodan, & Fahringer [43] proposed an algorithm for partitioning a scheduled workflow for distributed coordination among several slave enactment engine services. They incorporated the algorithm in the ASKALON [42] distributed workflow Enactment Engine. Their purpose of workflow partitioning was to minimize the communication between the master and the slave engines that coordinates the individual partitions of the entire workflow. The partitioning algorithm is based on a graph transformation theory. Partitioning reduced the number of workflow activities and, therefore, the job submission and management latencies and eliminated the data dependencies within partitions. However, the algorithm was used for compute intensive scientific workflows with large numbers of small sized data dependencies. In contrast to Pegasus, which partitions the workflow before the scheduling phase, they partition the workflow after scheduling. This results in reduced overheads for job submissions and aggregated file transfers.

**Data Placement & Replication**

Data placement techniques try to strategically manage placement of data before or during the execution of a workflow. Data placement schedulers can either be coupled or decoupled from task schedulers. Replication of data onto distributed resources is a common way to increase the availability of data. Replication also occurs when scientists download and share the data for experimental purposes, in contrast to explicit replications done by workflow systems. In data intensive applications, replication may or may not be feasible. Schedulers make the decision of data placement and replication based on the objectives to be optimized. If data analysis workloads have locality of reference, then it is feasible to cache and replicate data at each individual compute node, as high initial data movement costs can be offset by many subsequent data operations performed on cached data [115].

Kosar et al. [77] presented Stork, a scheduler for data placement activities in the Grid. They propose to make data placement activities a first class citizen in the Grid. In Stork, data placement is a full-fledged job and decoupled from computational jobs. Users describe the data placement job explicitly in the classads. DAGMan, a workflow scheduler for Condor, uses Stork for managing these data placement jobs. It manages the dependencies between Condor and stork jobs as defined by the dependencies in a DAG [35]. Under Stork, data placement jobs are categorized into three types. Transfer jobs are for transferring a complete or partial file from one physical location to another. Allocate jobs are used for allocating storage space at the destination site, allocating network bandwidth, or establishing a light-path on the route from source to destination. Release jobs are used for releasing the corresponding resource, which was allocated before.

Chervenak et al. [33] studied the relationship between data placement services and workflow management systems for data intensive applications. They propose an asynchronous mode of data placement in which data placement operations are performed as data sets become available and according to the policies of the virtual organization and not according to the directives of the WfMS. The WfMS can however assist the placement services on placement of data based on information collected during task executions and data collection. Their approach is proactive as it examines current workflow needs to make data placement decisions rather than depending on the popularity of data in the past. They evaluated the benefits of pre-staging data using the data replication service versus using the native data stage-in mechanisms of the Pegasus WfMS. Using the Montage astronomy example, they conclude that as the size of data sets increases, pre-staging data increases the performance of the overall analysis.

Shankar & DeWitt [129] presented architecture for Condor in which the input, output and executable files of jobs are cached on the local disks of machines in a cluster. Caching can reduce the amount of pipeline and batch I/O that is transferred across the network. This in turn significantly reduces the response time for workflows with data intensive workloads. With caching enabled, data intensive applications can reuse the files and also are able to compare old and new versions of the file. They presented a planning algorithm that takes into account the location of cached data together with data dependencies between jobs in a workflow. Their planning algorithm produces a schedule by comparing the time saved by running jobs in parallel with the time taken for transferring data when dependent jobs are scheduled on different machines. By executing the BLAST[6] application workflow they showed that storing files on the disks of compute nodes significantly improves the performance of data intensive application workflows.

Ranganathan & Foster [120; 118; 119] conducted extensive studies for identifying dynamic replication strategies, asynchronous data placement, and job and data scheduling algorithms for Data Grids. Their replication process at each site periodically generates

[6] http://blast.ncbi.nlm.nih.gov.gov/

new replicas for popular datasets. For dataset placement scheduler they define three algorithms: *Data-DoNothing-* no active replication takes place, *DataRandom-* popular datasets are replicated to a random site on the Grid, *DataLeastLoaded-* popular datasets are replicated to a least loaded neighboring site. They proposed to decouple data movement from computation scheduling, which is also known as asynchronous data placement. This provides opportunity for optimizing both data placement and scheduling decisions, also simplifying the design and implementation of the Data Grid system. They concluded through simulations on independent jobs that scheduling jobs to locations that contain the data they need and asynchronously replicating popular data sets to remote sites achieves better performance than coupled systems.

### 2.3.4 Granularity

Workflow schedulers can make scheduling decisions based on either: (a) task level, or (b) workflow level.

#### Task level

Task level schedulers map individual tasks to compute resources. The decision of resource selection and data movement is based on the characteristics of each task and its dependencies with other tasks.

#### Workflow level

Workflow level schedulers map the entire workflow rather than a set of available tasks to compute resources. Workflow compute and storage requirements guide the scheduler to make a decision on resource selection and data movement.

Blythe et al. [15] compared several task-based and workflow-based approaches to resource allocation for workflow applications. In their workflow-based approach, the entire workflow is mapped a priori to the resources to minimize the makespan of the whole workflow. The mapping is changed according to the changing environment, if necessary. The mapping of the jobs does not imply scheduling all the jobs ahead of time. They use a local search algorithm for workflow allocation based on generalized GRASP procedure (greedy randomized adaptive search) [49]. The final schedule is chosen after an iterative and greedy comparison between alternative schedules. On each iteration, task is mapped to resource based on the minimum margin of increase to the current makespan of the workflow if the task was allocated to that resource. This approach is based on the min-min [18] heuristic. They noticed that during large file transfers, resources spent significant time waiting for all the files to arrive before it could start executing the scheduled job. They proposed a weighted min-min heuristic that takes into account the idle times of all the resources if a job was scheduled to a resource. Based on the weighted sum of

the idle times and estimated completion time, a job is mapped to the resource that gives the minimum weighted sum. The step is repeated until all the jobs have been mapped. Due to the pre-mapping, the workflow-based approach could pre-position the data to the known destination by transferring a large file immediately after it is created. In the task-based approach, transfers could not begin until the job is scheduled which happened only after its parent was scheduled. They also simulated the impact of inaccurate estimates of transfer times for data intensive application workflows. They show that the performance of task-based approach degrades rapidly with increasing uncertainty in comparison to workflow-based approach. Based on these facts, they conclude that workflow-based approaches perform better for data intensive applications than task-based approaches.

### 2.3.5 Model

Workflow scheduling model depends on the way the tasks and data are composed and handled. They can be classified into two categories: (a) task-based, and (b) service-based.

#### Task Based

Tasks based approaches mention data dependencies explicitly. The workflows are generally complex in structure. Optimizations used by most systems are simple in nature. The WfMS has greater control over the data flow as it can define data placement, cleanup and transfer tasks separately from the workflow tasks. DAGMan, Pegasus, GridAnt [6], GrADS [12], and GridFlow [28] are some of the workflow systems that support task based approaches. These have been described individually in preceding sections.

#### Service Based

Service based approaches, also referred to as meta computing, wrap application codes into standard interfaces. Such services are hidden from the users and only invocation interface is known. Various interfaces such as Web Services [5] or gridRPC [97] have been standardized [59]. In this model, the application is described separately from the data. Data is declared as parameters to the service. In this approach, workflows are generally simple in structure. In contrast to task based approaches, workflow systems use complex optimizations. This model is useful when an application workflow is to be repeatedly executed over a large number of varying data sets. Instead of replicating the task for each data set, service based model has the ability to define different data composition strategies over the input data of a service. Kepler system [89], the Taverna workbench [98], and the Triana workflow manager [140], are some of the service based workflow systems.

The myGrid project[7] has developed a comprehensive loosely coupled suite of middleware components specifically to support data intensive in-silico experiments represented

[7] http://www.mygrid.org.uk/

as workflows, using distributed resources. The main tool is the Taverna Workbench [98]. Taverna allows for the automation of experimental methods through the integration of various services, including WSDL-based single operation web services, into workflows. It uses FreeFluo[8] as a workflow enactment engine that facilitates intermediate data transfers and service invocations. Workflows are represented using the Simple Conceptual Unified Flow Language (SCUFL). A workflow graph consists of processors, each of which transforms a set of data inputs into a set of data outputs. Using SCUFL, implicit iteration over incoming data sets can be carried out based on user specified strategy. Users can use the Thread property to specify the number of concurrent instances that can send parallel requests to the iteration processor for handling simultaneous processing. This can help reducing the service wait time as workflow engine can send data while the service is still working on previously sent data.

Kepler [89] provides support for web service-based workflows. Using an extension of PTOLEMY II [21], it uses an actor-oriented design approach for composing and executing scientific application workflows. Computational components are termed as actors, which are linked together to form a workflow. A director represents the interaction between these components. It specifies and mediates all inter-actor communication, separating workflow orchestration, and scheduling from individual actor execution. Two of the directors (namely, Synchronous Data Flow (SDF) and Process Networks (PN)) work primarily by controlling the sequencing of actors according to the data availability, to preserve the order of execution of the workflow. The WebService actor provides a simple plug-in mechanism to execute any WSDL defined web service. An instantiation of the actor acts as a proxy for the web service being executed and links to other actors through its ports. Using this component, any application that can be deployed as a remote service, can be used as a Kepler component [67].

Kalyanam et al. [70] proposed a web service-enabled distributed data-driven workflow system on top of the TeraGrid[9] infrastructure. The workflow system is based on an existing data management architecture that provides easy access to scientific data collections via the TeraGrid network. It leverages JOpera [108], an open-source workflow engine that integrates web services into a processing pipeline. Users can construct data-driven workflows using local or TeraGrid data and computation resources. Their system helps automate the operations such as data discovery, movement, filtering, computationally intensive data processing, and so forth, by organizing them as a pipeline so that researchers can execute applications with minimal user interaction.

Brandic, Pllana & Benkner [17] presented a service-oriented environment, named as Amadeus, for QoS-aware Grid workflows. For data intensive application workflows, QoS parameters may be defined for data-transfer time, reliability, storage requirements,

[8]http://freefluo.sourceforge.net/
[9]http://www.teragrid.org

cost, and so forth. It allows users to specify QoS constraints at workflow composition, planning, and execution stages. Various QoS-aware service components are provided for workflow planning to support automatic constraint-based service negotiation and workflow optimization.

### 2.3.6 Platform

Data intensive application workflows could be executed in different resource configuration and environments (e.g. Cluster, Data Grids, Clouds etc.) depending on the requirements of the application. Clusters are generally composed of homogeneous processors and are under a single domain. For data intensive applications, clusters provide a viable platform for low cost and enhanced performance. When the data produced and stored are local and not globally shared, cluster based platforms is more feasible than Grids or Clouds.

Data Grids are globally distributed resources designed for data intensive computing. Data is generated and/or used in research labs distributed globally, giving rise to sharing and re-use. Data grids are feasible for large-scale experiments that are a result of worldwide collaboration of resources and scientists.

Clouds are an emerging model for centralized but highly available and powerful infrastructure. Large-scale storage and computation is provided by data centers. Computing power is achieved by using virtualization technology. Data intensive applications can highly benefit from using services provided by Clouds as compared to Data Grids and Clusters when factors such as scalability, cost, performance, and reliability are important.

In the past, scientific workflows were generally executed on a shared infrastructure such as TeraGrid, Open Science Grid[10], and dedicated clusters. In such systems, the file system is usually shared for easy data movement. However, this can be a bottleneck for data intensive operations [177].

Deelman et al. [39] presented a simulation-based study of costs involved when executing scientific application workflows using Cloud services. They studied the cost performance trade-off of different execution and resource provisioning plans, and storage and communication fees of Amazon S3 in the context of an astronomy application called Montage. They showed that for a data intensive application with a small computational granularity, the storage costs were insignificant as compared to the CPU costs. They concluded that cloud computing is a cost-effective solution for data intensive applications.

Broberg, Buyya, & Tari [20] introduced MetaCDN, which uses 'Storage Cloud' resources to deliver content to content creators at low cost but with high performance in terms of throughput and response time. Data could be delivered to tasks in a workflow using tools provided by CDN.

[10]http://www.opensciencegrid.org

In our work with data intensive application workflows, we studied the performance characteristics of a brain Image Registration workflow (IR) (Pandey et al., 2009). We executed the application on an experimental Grid platform, Grid'5000 [29], and profiled each task execution and data flow. We were able to decrease the makespan of the workflow significantly by using Grid resources. We also used partial data retrieval technique to retrieve data from distributed storage resources while scheduling data intensive application workflows (see Chapter 5). We proposed static and dynamic heuristics that incorporated the retrieval techniques. We experimented with two synthetic and one real data intensive application workflow (IR workflow). Executions were done using Virtual Machines (VM) connected through a simulated network environment. Experimental results showed that retrieving data from multiple sources significantly improves the time taken to download data to the execution sites. Cumulative effect thus decreased the total makespan of all the workflows.

Ramakrishnan & Reed [117] studied the impact of varying resource availability on application performance. They applied performance analysis (i.e., a measure of the system's performance in the event of failures) at two levels - computational resources and the network, to obtain the application workflow's overall execution time, given the failure level of resources. They used these values to estimate task completion times during each iteration of the workflow-scheduling algorithm. Their HYBRID approach, which takes resource failure and repair into account, performs better than the approach that does not take failures into account, when the failure-to-repair rates increase. Through simulation results, they concluded that the joint analysis of performance and reliability can improve dynamic workflow scheduling and fault tolerance strategies required for Grid and cloud environments.

Juve & Deelman et al. [69] studied the performance and cost of using different storage systems that could be used as data hosts during execution of scientific workflows on Clouds. The storage systems they used for execution three workflow applications were: Amazon S3, NFS [128], GlusterFS[11], and PVFS [61]. They show the impact of type of storage system used on the application's runtime and cost. Specifically, they observed that Amazon S3 provided improved performance due to the caching of data at the client side, but performed poorly when the number of small files were large. In addition, Amazon would charge per transaction for accessing files, which meant the total cost would increase if Amazon alone was used as a storage server. In terms of performance benefit, they observed that addition of more Cloud resources for use in workflow executions did improve the performance, but only to a certain extent. In order to minimize total cost as well as make use of Cloud resources, they proposed to provision a single virtual cluster instead of large number of resources all at once, so that multiple workflows could be executed in succession.

[11] http://www.gluster.org

### 2.3.7 Miscellaneous

In this section, we list some technologies that have been used for enhancing the performance of data intensive application workflows.

#### Semantic Technology

The myGrid project[12] exploits semantic web technology to support data intensive bioinformatics experiments in a grid environment. The semantic description of services in RDF and OWL is used for service discovery and matchmaking. Kepler [89] is a data-driven workflow system (as described under the sub heading "Service Based" workflows), which allows semantic annotations of data and actors, and can support semantic transformation of data.

#### Database Technology

GridDB [84] is a grid middleware based on a data centric model for representing workflows and their data. It uses database to store memory and process tables that store the inputs and outputs of a program that has completed, and process state of executing programs, respectively. It provides functional data modeling language (FDM) for expressing the relationship between programs and their inputs and outputs.

Shankar, Kini, DeWitt, & Naughton [130] have pointed out the advantages of tightly coupling workflow management systems with data-manipulation for data intensive scientific programs. They also presented a language for modeling workflows that is tightly integrated with SQL. Data products from workflows are defined in relational format. They use SQL for invocation and querying of programs.

## 2.4 Research Issues

Most workflow systems in the past focused on performance of tasks rather than data management. The reason might have been due to cluster management systems and shared storage space. But with globally distributed resources, it is a functional requirement that these systems take into account the data flow management along with compute tasks. Composition of workflows that enables distributed coordinated execution of globally distributed scientific applications thus remains a challenge.

Requirements of data intensive applications can be specified using QoS parameters at all levels of a WfMS. To meet QoS requirements of users executing data intensive scientific application workflows, we need to:

[12] http://www.mygrid.org.uk/

- Define an architectural framework and principles for the development of QoS-based workflow management systems,
- Develop QoS-based data and tasks scheduling algorithms for scheduling e-Research workflow applications,
- Develop efficient data transfer/retrieval mechanisms that enable parallel data transfers for data intensive applications
- Integrate data aware scheduling algorithms, workflow management tools and distributed computing and storage resources seamlessly

With the advent of virtualization technologies, Cloud storage systems, content delivery networks (CDN) and so forth, it is likely that big scientific projects will start using services provided by third parties for storing and processing application data. As companies such as Amazon, IBM, and Google are proposing innovative ways to use their huge data centers for commercial use as Cloud services, data intensive applications may leverage their utilities and not depend on conventional, error-prone, costly and unreliable solutions [26]. However, due to higher usage and access costs of these commercial services, small-scale scientific projects may still need to rethink of deploying their application on Clouds.

## 2.5 Conclusions

In this chapter, we classified and surveyed techniques for managing and scheduling data intensive application workflows. Under each classification, there were several specific techniques that workflow systems use for executing data intensive application workflows on globally distributed resources. We listed and described each such work in detail. We found that most systems focused on minimizing data transfers and optimally structuring model of execution to subdue the effect of large data requirements of most scientific data intensive applications. We also found that many systems used a combination of techniques we listed to achieve higher scalability, fault tolerance, lower costs and increase performance. A single technique alone would not suffice to minimize the effect on performance and cost for increasing data processing requirements of scientific applications. Due to the lack of standardization and interoperability, many of the systems were developed in isolation. As a result, techniques for managing data for data intensive workflows were mixed and duplicated. Nevertheless, scientific community has been able to successfully achieve the goals of all scientific projects with promising results till date. However, as the volume of data grows, new computing paradigms emerge, the scientific community faces growing challenges in managing data within user specified budgets and deadlines.

This chapter thus helps identify capabilities that are existing in current workflow management systems and that should be added to enable them to manage large scale data intensive applications in distributed computing environments such as Grids and Clouds. The subsequent chapters address these issues by presenting WfMS design, algorithms and application case studies, all supported by real world scientific applications.

# 3

# A Workflow Management System for Data Intensive Applications

This chapter presents a workflow management system design. The first half of the chapter describes the fundamental components of the system: the workflow management portal, the workflow editor, the workflow engine and the workflow monitor. It then describes the integration of the management tools with distributed computing resources and platforms. The second half of the chapter studies two real-world applications. It presents application characteristics and their performance analysis on both Grid and Cloud resources.

## 3.1 Introduction

Scientific applications, in domains such as high-energy physics and life sciences, are typically modeled as workflows and consist of tasks, data, control sequences and data dependencies. The workflow models the underlying processes as a series of coordinated steps that simplifies the complexity of execution and management of applications. Processing and managing large amounts of data and tasks modeled by these workflows require the use of a distributed collection of computation and storage facilities. Workflow management systems are responsible for managing and executing these workflows.

According to Zhao et al. [115], scientific workflow management systems are engaged and applied to the following aspects of scientific computations: (a) describing complex scientific procedures (using GUI tools, workflow specific languages), (b) automating data derivation processes (data transfer components), (c) high performance computing (HPC) to improve throughput and performance (distributed resources and their coordination), and (d) provenance management and query (persistence components).

This chapter presents a workflow management system [102] that consists of components that are responsible for handling tasks, data and resources taking into account users' QoS requirements. Its design is depicted in Figure 3.1. The architecture consists of three major sections: (a) the user interface, (b) the core and, (c) plug-ins. The user-interface

allows end-users to work with workflow composition, workflow execution planning, submission, and monitoring. These features are delivered through a web portal (or through a stand-alone application that is installed at the user's end). Workflow composition is done using an XML based Workflow Language (xWFL). Users define task properties and link them based on their data dependencies. Multiple tasks can be constructed using copy-paste functions that are a part of standard graphical interfaces. The components within the core are responsible for managing the execution of workflows. They facilitate in the translation of high-level workflow descriptions (defined at the user-interface using XML) to task and data objects. These objects are then used by the execution subsystem. The scheduling component applies user-selected scheduling policies and plans to workflows at various stages in their execution. The tasks and data dispatchers interact with the resource interface plug-ins to continuously submit and monitor tasks in the workflow. These components form the core part of the workflow engine. The Plug-ins support workflow executions on different environments and platforms. Our system has plug-ins for querying task and data characteristics (e.g. querying Meta-data Services, reading from trace files), transferring data to and from resources (e.g. transfer protocol implementations, and storage and replication services), monitoring the execution status of tasks and applications (e.g. real-time monitoring, logs of execution, and the scheduled retrieval of task status). The resources are at the bottom layer of the architecture and include Clusters, global Grids and Clouds. The WfMS has plug-in components for interacting with various resource management systems present at the front end of distributed resources. Currently, the WfMS supports Aneka, PBS, Globus and fork-based middleware.

The recent progress in virtualization technologies and the rapid growth of Cloud computing services have opened a new paradigm in distributed computing for utilizing existing (and often cheaper) resource pools for on-demand and scalable scientific computing. In order to leverage the benefits of Cloud services, our workflow management system has tools to communicate and use this new paradigm. At the plug-in layer of the design depicted in Figure 3.1, the resource managers may communicate with the market maker, scalable application manager and InterCloud services for global resource management [149].

## 3.2 A Workflow Management System Design

This thesis extends the workflow engine [166], an abstract model of which is given in Figure 2.1, by implementing components and algorithms to support data intensive workflow applications on both Grid and Cloud computing environments. In this section, we first describe each component of the WfMS, then relate the components to a typical life cycle of an application workflow.

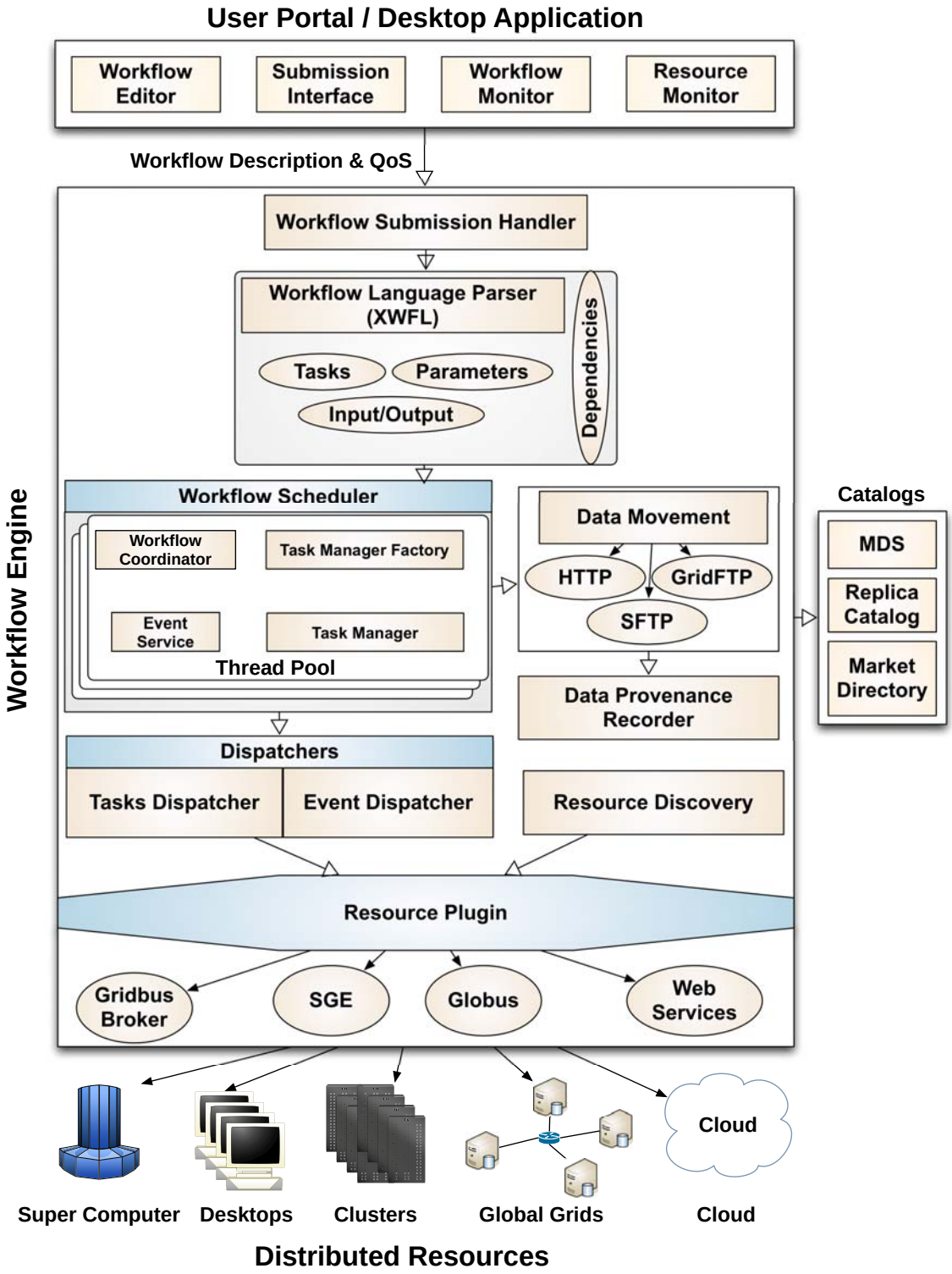
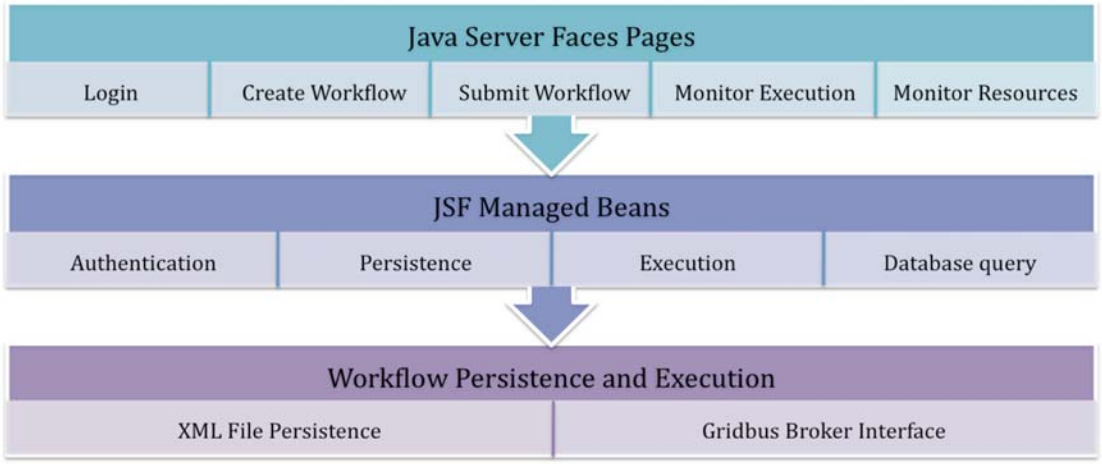


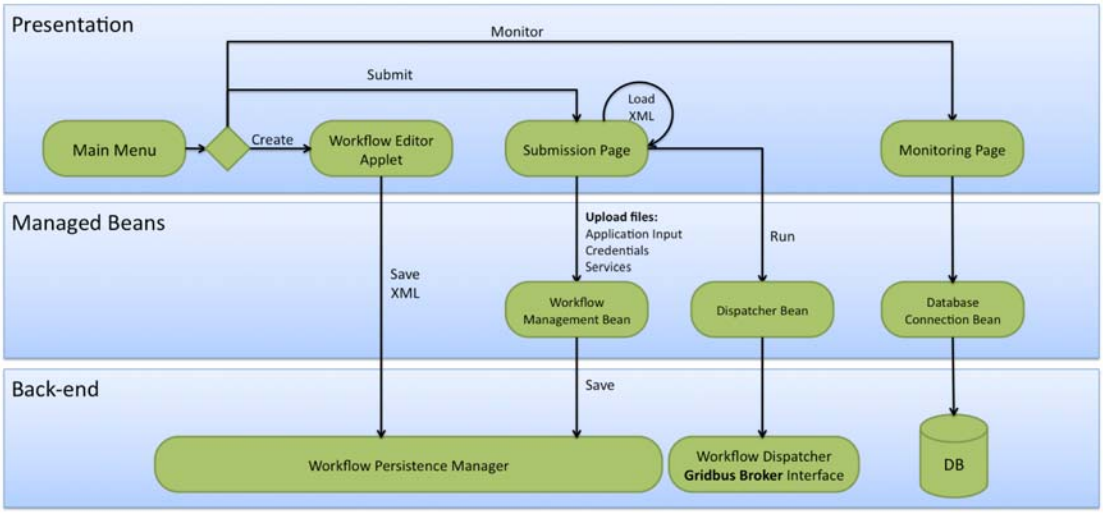

**Figure 3.1:** *Workflow Management System Design.*

### 3.2.1 Components of WfMS

Figure 3.1 depicts the components of WfMS that are grouped into three layers. At the top layer, there are user interface components. The middle layer groups the middleware logic behind scheduling and management, and the bottom layer depicts the resources.




(a)




(b)

**Figure 3.2:** *A workflow management portal. (a) Layered architecture of the workflow portal. (b) Activities the portal supports.*

#### Web-based Portal

Our WfMS provides a Web Portal as a primary interface for managing workflows. Following are the major functionalities that the portal supports:

1. A workflow editor, which enables users to compose new workflows and modify existing ones.

2. A submission page, through which users can upload all necessary input files to run a workflow including the workflow description file, credentials, and services files (Figure 3.3 (a)) to the system.

3. A monitoring and output visualization page, which allows users to monitor multiple workflow executions in progress. The most common monitoring activity consists of keeping track the status of each task through the workflow monitor, which provides a real-time updated graphical representation of workflow tasks. The application's output is usually presented in the form of images (Figure 3.3 (d), 3.12 (d)), where possible.

4. A resource information page, which shows the characteristics of all available distributed resources.

5. An application specific page, which in the current implementation provides generation of two workflow description files for two applications: Image Registration (see Section 3.3.1) and Evolutionary Multi-objective Optimisation (see Section 3.3.2).

Although the current version of the Portal implements support for direct creation of few applications, our design is generic to support any workflow application. Apart from few parts of the output visualization page and the application specific page that generates application specific output, the portal infrastructure is generic enough to allow the porting of almost any workflow application into the portal.

Figure 3.2(a) shows a layered architecture of the WfMS portal. In the top layer, a set of Java Server Faces (JSF) pages enable actions such as creating, submitting and monitoring a workflow execution. In the middle layer, a set of session beans manage users' requests, which are in turn forwarded to the back-end (bottom) layer. The infrastructure layer handles persistence of workflow description and input files and their submission for execution via the resource plug-in (Gridbus Broker interface shown in the Figure). Figure 3.2(b) depicts possible user activities and their flow through the layered portal architecture. A typical activity consists of workflow design by means of the workflow editor, which generates a workflow description XML file. Subsequently, the description file is loaded on the submission page, which requests the user to upload all input files referenced in the description file. Once all files are uploaded, the submission page allows the user to submit the workflow and subsequently monitor its progress.

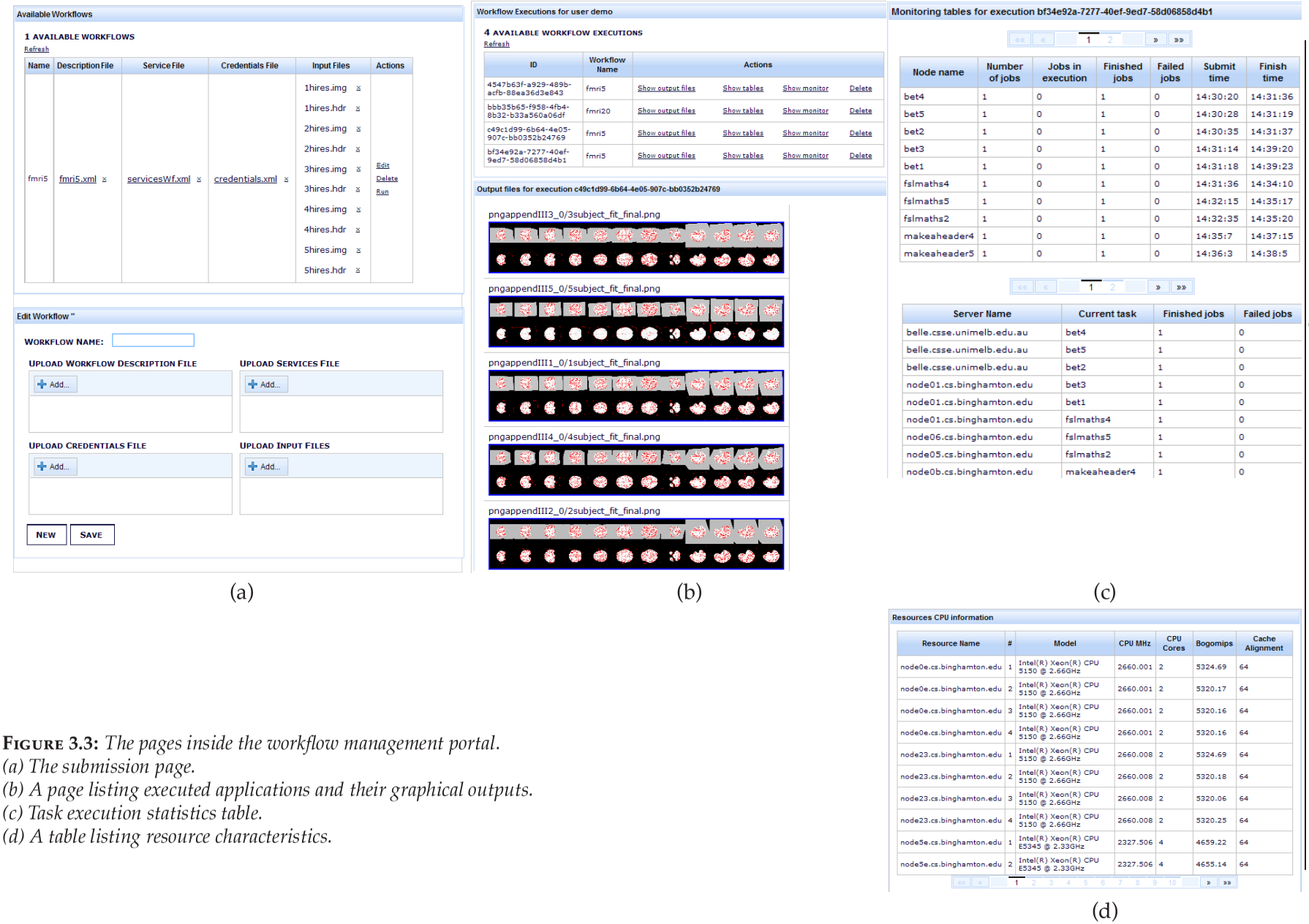



**FIGURE 3.3:** *The pages inside the workflow management portal.*
*(a) The submission page.*
*(b) A page listing executed applications and their graphical outputs.*
*(c) Task execution statistics table.*
*(d) A table listing resource characteristics.*

The following sub-sections describe the workflow editor, the workflow monitor, and the workflow engine.

### Workflow Editor

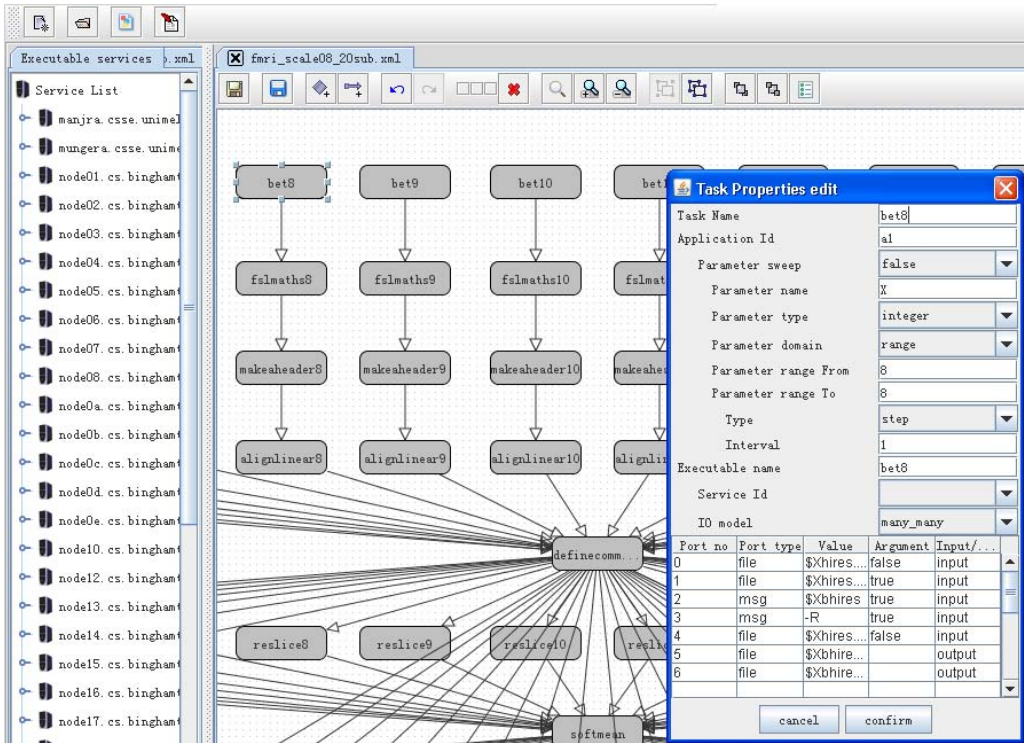
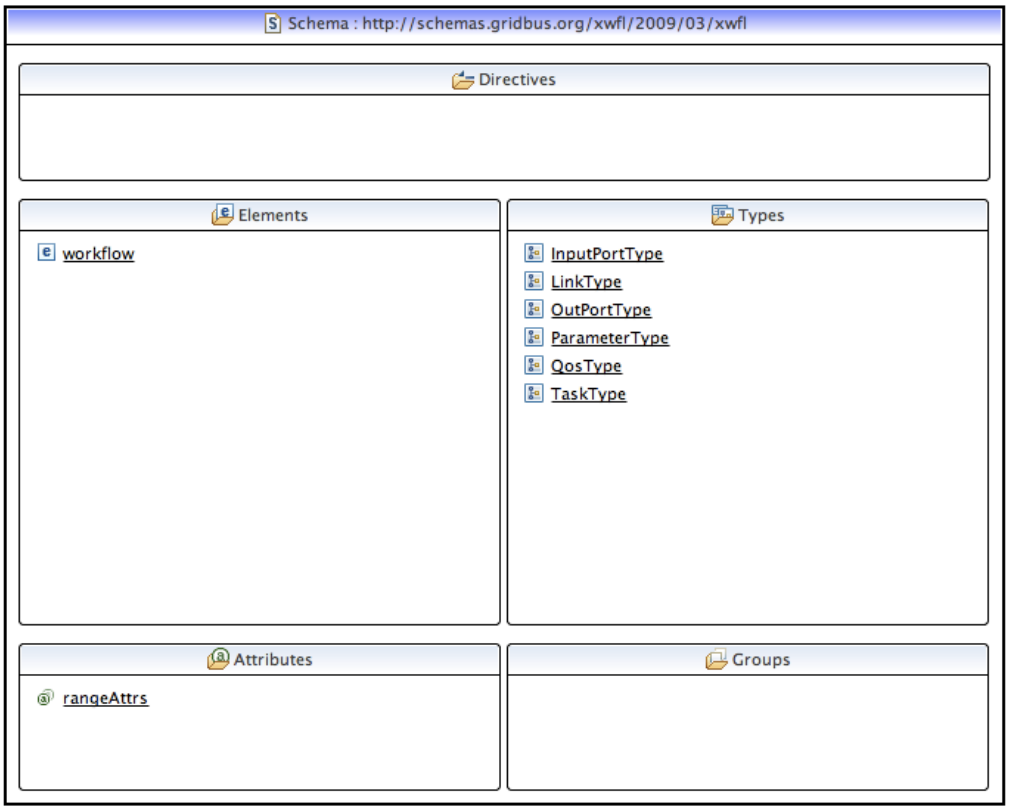


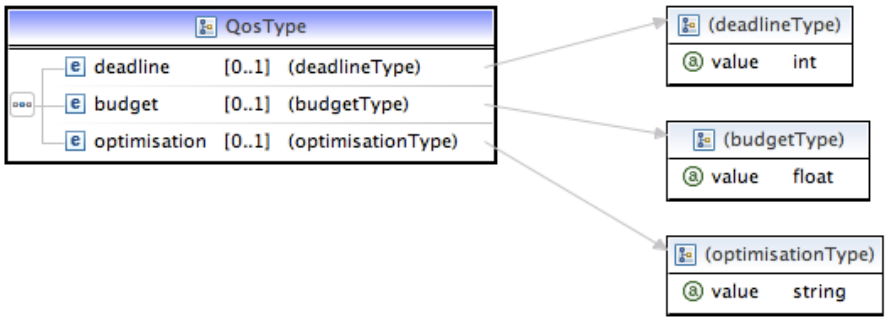

**FIGURE 3.4:** *The workflow editor showing the Image Registration application workflow and its parameters.*

The workflow editor provides a Graphical User Interface (GUI) that allows users to create new workflow and modify existing ones utilizing the drag and drop facilities of the interface. Figure 3.4 depicts the editor interface that displays an application workflow and its parameters. In the editor, workflows are created graphically as a Directed Acyclic Graph (DAG). Each node represents the computational activities of a particular task in the workflow and a link is used to specify the data flow between two tasks. The workflows are based on an XML-based workflow language (xWFL). Using the editor, users can design and create workflows for complex scientific procedures following the specifications of the workflow language and the nature of application, starting from an XML then porting it to the editor for visualization. Primitive users can reuse XML-based workflow files and visualize it or edit it using the editor.

The first version of the xWFL was proposed by Yu et al. [165] as part of her PhD thesis, as shown in Figure 3.5(a). This thesis enhances the language by adding QoS parameters such as: deadline and budget, as depicted in Figure 3.5(b) The deadline is specified in seconds and the budget using real values. It also adds an option for optimization type




(a)




(b)

**FIGURE 3.5:** *(a)The extended version of the xWFL schema. (b) The schema for QoS definition.*

while scheduling tasks, so that users can choose between time and cost or let the system decide when none is specified. Our current implementation facilitates the user to specify the QoS parameters for each task separately. The scheduling system grants preferences (based on the scheduling policy used) to the values provided by the user than the values generated by the algorithm at run-time. This separation becomes effective when user themselves identify the critical tasks in the workflow before submission for execution.

Figure 3.6 shows the schema of task and link definition using xWFL. < *task* > represents a node in the workflow, which is a set of instructions to be executed in a resource.

```
<?xml version="1.0" encoding="UTF-8"?>
<workflow xmlns="http://schemas.cloudbus.org/xwfl/2009/03/xwfl"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://schemas.cloudbus.org/xwfl/2009/03/xwfl">
    <tasks>
        <task name="bet1" paramsweep="false">
            <paras>
                <para name="X" type="integer" domain="range">
                    <range from="1" to="1" type="step" interval="1" />
                </para>
            </paras>
            <executable>
                <name value="bet1" I_OModel="many_many" />
                <service serviceID="a1" />
                <input>
                    <port number="0" type="file" value="$Xhires.img" arg="false" />
                    <port number="1" type="file" value="$Xhires.hdr" arg="true" />
                    <port number="2" type="msg" value="$Xbhires" arg="true" />
                    <port number="3" type="msg" value="-R" arg="true" />
                    <port number="4" type="file" value="$Xhires.hdr" arg="false" />
                </input>
                <output>
                    <port number="5" type="file" value="$Xbhires.hdr" />
                    <port number="6" type="file" value="$Xbhires.img" />
                </output>
            </executable>
        </task>
        <task name="fslmaths1" paramsweep="false">
            <paras>
                <para name="X" type="integer" domain="range">
                    <range from="1" to="1" type="step" interval="1" />
                </para>
            </paras>
            <executable>
                <name value="fslmaths1" I_OModel="many_many" />
                <service serviceID="a2" />
                <input>
                    <port number="0" type="msg" value="$Xbhires" arg="true" />
                    <port number="1" type="msg" value="$Xbhires" arg="true" />
                    <port number="2" type="msg" value="-odt" arg="true" />
                    <port number="3" type="msg" value="short" arg="true" />
                    <port number="4" type="file" value="$Xbhires.hdr" arg="false" />
                    <port number="5" type="file" value="$Xbhires.img" arg="false" />
                </input>
                <output>
                    <port number="6" type="file" value="$Xbhires.hdr" />
                    <port number="7" type="file" value="$Xbhires.img" />
                </output>
            </executable>
        </task>
    </tasks>
```

(a)

```
    <links>
        <link>
            <from task="bet1" port="5" />
            <to task="fslmaths1" port="4" />
        </link>
        <link>
            <from task="bet1" port="6" />
            <to task="fslmaths1" port="5" />
        </link>
        <link>
            <from task="fslmaths1" port="6" />
            <to task="makeaheader1" port="0" />
        </link>
        <link>
            <from task="fslmaths1" port="7" />
            <to task="makeaheader1" port="8" />
        </link>
        <link>
            <from task="makeaheader1" port="9" />
            <to task="alignlinear1" port="5" />
        </link>
        <link>
            <from task="makeaheader1" port="10" />
            <to task="alignlinear1" port="6" />
        </link>
        <link>
            <from task="alignlinear1" port="7" />
            <to task="definecommonair" port="6" />
        </link>
        <link>
            <from task="alignlinear1" port="8" />
            <to task="definecommonair" port="7" />
        </link>
        <link>
            <from task="alignlinear1" port="9" />
            <to task="definecommonair" port="8" />
        </link>
    </links>
</workflow>
```

(b)

**Figure 3.6:** *Defining task and links in an IR workflow using xWFL. (a) A task definition. (b) Definition for links to define data precedence between tasks. The DAG of this workflow description is given in Appendix A.*

The element < *executable* > is used to define the information about the task, corresponding I/O model as well as the input and output data. xWFL supports both abstract and concrete workflows. For defining concrete workflows, the users can specify the location of a particular compute service that provides the required application executable using < *service* > element. For defining abstract workflows, users can use a service category to direct the engine to identify providers dynamically at run-time. The < *input* > and < *output* > elements are used to define the input and output data of a task, respectively. Each data is associated with a unique port number of that task. These port numbers are used for linking the output of one task to the input of their children tasks. Input values can be both file and message. The element < *links* > is a set of definitions that define the dependencies between tasks in a workflow. Each < *link* > defines a source task using the *from* tag and the destination task using the *to* tag. If there are multiple file dependencies, the port number is used to distinguish each file from another.

The workflow editor provides the following advantages to the users:

*Portal-based editor*: As the workflow editor is embedded as an online tool in the

workflow portal, users do not have to download or install any client side components. Users can create, edit and save their workflows in the server and can access them from anywhere and whenever required. Moreover, the interface is compatible with all Internet browsers that support Java, which enables users to access the interface even using mobile devices.

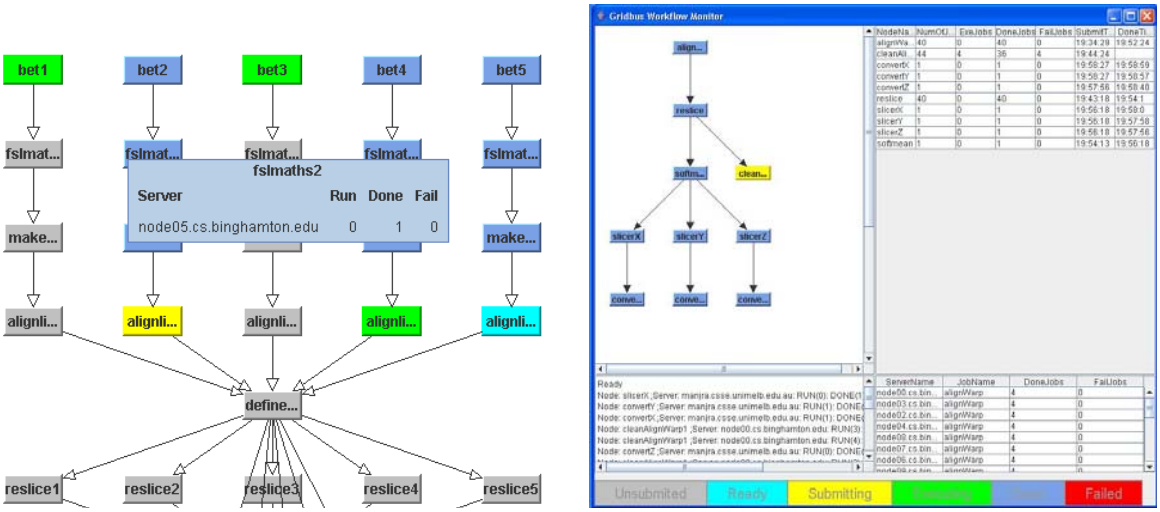
*Hiding complexity*: The underlying structure of workflows is represented using XML, which is parsed by the workflow engine. However, representing workflows in XML description demands advanced skill from workflow designers which involves thorough understating of the workflow language e.g. xWFL, and expertise of managing XML files from various namespaces. The workflow editor hides these complexities from scientists and researchers by providing a GUI. Users only need to draw boxes for tasks and lines for connecting them, and specify their properties for every task and links. The editor compiles the graphical representation and generates the XML description of the workflow in the background. Advanced users can use both the graphical interface and the XML code to manipulate the workflow definition.

*Ease of use*: The workflow editor provides a drag and drop facility to draw the workflow using boxes and lines. The common edit operations such as cut, copy, paste are applicable for all tasks and links. This means, whenever users need to replicate parts of a complex workflow, they just need to select the desired part, copy and paste it on the same editor window. The editor also supports usual file operations such as create, open and save. Furthermore, users can check the executable services as a list, as shown in Figure 3.4. There is also a functionality to save the graphical representation of the workflow as an image for use in presentation slides.

*Interoperability*: The editor can generate the graphical representation of the workflow, as a DAG, from a XML description file that follows the xWFL schema. Thus, if any user creates a workflow and saves it as XML file, any other user can reuse that by opening it using the workflow editor or other XML editors, and modify according to their needs.

**Workflow Monitor**

The workflow monitor provides a GUI for viewing the status of each task in the workflow. Users can easily view the ready, executing, stage-in, and completed tasks as depicted in Figure 3.7. The monitor is being implemented as a portal based tool as well as a standalone application. Task status is represented using different colors codes: blue represents completed task, green represents task are in execution, yellow represents input files are staged, and cyan represents the task is ready for submission for execution. Users can also view the site of execution of each task, the number of tasks being executed (in case of a parameter sweep type of application) and the failure history of each task by hovering the mouse pointer on top of the task of interest. This information is displayed on a rectangular window with the task name as the title, the server where it is executing, and




(a) (b)

**FIGURE 3.7:** *(a)A portal based workflow monitor showing the status of tasks of the IR workflow application. (b) A workflow monitor standalone application.*

its execution status as shown in Figure 3.7 for the task *fslmaths*2. The workflow structure is made editable for users such that they can drag tasks and group or separate tasks of interest into blocks. This is very useful when there are numerous tasks in the workflow. Users can also use the zoom-in and zoom-out functions to visualize each task or the entire workflow.

The monitor interacts with the workflow engine using an event mechanism by using the tuple space model. In the back-end, a database server stores the states of each task for each application workflow. Whenever any task changes state, the monitoring interface is notified and the status values are displayed at run-time. This enables multiple users to access the monitoring interface using different computers from several locations either using a portal or the standalone application. The monitoring interface does not have support for deletion and insertion of individual tasks at run-time.

### Workflow Engine

Scientific application portals submit task definitions along with their dependencies in the form of the workflow language to the Workflow Engine (WE). Then the WE schedules the tasks in the workflow application through the middleware services and manages the execution of tasks on distributed resources. The key components of the WE are: workflow submission, workflow language parser, resource discovery, dispatchers, data movement and workflow scheduler.

The workflow engine is designed to support an XML-based WorkFlow Language (xWFL). This facilitates user level planning at the submission time. The workflow language parser converts workflow description from XML format to objects such as: Tasks,

Parameters, Data Constraint (workflow dependency), etc., that can be accessed by the workflow scheduler. The resource discovery component queries the information services such as Globus MDS, directory service, market directory, and replica catalogs, to locate suitable resources for execution of the tasks in the workflow by coordinating with middleware technologies such as Gridbus Broker [152]. Our WE uses the Gridbus Broker as one of the middleware technologies for deploying and managing task execution on various other middleware. Gridbus Broker as a middleware mediates access to distributed resources by (a) discovering resources, (b) deploying and monitoring task execution on selected resources, (c) accessing data from local or remote data source during task execution, and (d) collating and presenting results.

The WE has been significantly enhanced to support interaction with web-services hosted by Cloud computing service providers such as Amazon. It also has plug-ins for interacting with Aneka [147]. Figure 3.8 depicts the interaction of WE with Aneka using Simple Object Access Protocol (SOAP). Aneka exposes its task submission interface service using its web services. The WE encapsulates a workflow task inside a SOAP request and submits it to Aneka. Aneka manages these tasks on its own set of resources using its own scheduling algorithm. However, the choosing of the tasks for submission is based on the scheduling algorithm implemented in the he WE.

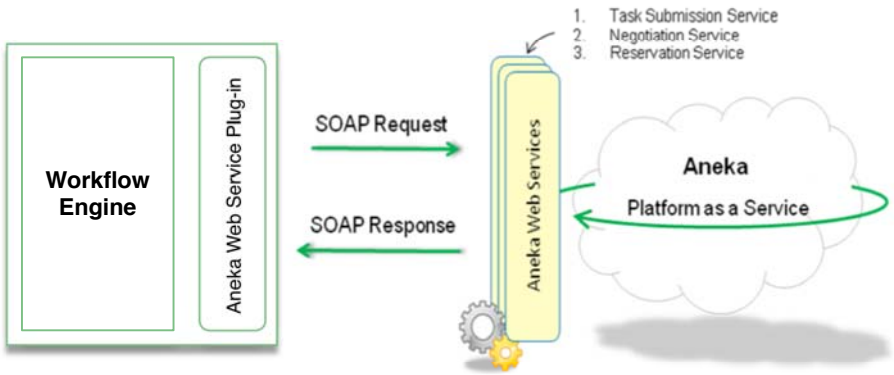



**Figure 3.8:** *The workflow engine interacting with Aneka web services using SOAP request/response.*

The workflow engine is designed to be loosely-coupled and flexible by using a tuple spaces model, event-driven mechanism, and subscription/notification approach in the workflow scheduler, which is managed by the workflow coordinator component. The workflow executor is the central component in WE. With the help from dispatcher component it interacts with the resource discovery component to find suitable compute resources at run time, submits a task to resources, and controls input data transfer between task execution nodes.

The basic functionality of submitting workflow tasks to distributed resources was developed by Jia Yu [165] as part of her PhD thesis. Previously, all workflow tasks were

executed inside a single workflow coordinator. The thread model that existed in previous version of the WE could not handle increasingly large number of workflow tasks. In addition, when tasks required large size data to be staged-in, tasks would time-out and got resubmitted. Fault tolerance of the workflow was limited to only the tasks. This thesis enhances the WE design by making the following changes:

1. Addition of thread pooling for handling large number of tasks when using both single and multiple workflows.

2. Robust failure handling when transferring large sized input/output data files for every task.

3. Implemented multiple transfer protocols when transferring data between distributed resources such as SFTP, GridFTP and HTTP.

4. Mechanism to transfer output files of every task to a storage resource that is different than the compute resource where the task was executed.

The addition of thread pooling enabled queuing of tasks in the workflow as opposed to a suspended thread that existed in the previous design. Queuing of tasks does not consume memory and are thus manageable for large number of tasks. Whereas, creating execution threads for every task and keeping their state in memory proved to be memory inefficient for large number of tasks, which resulted in frequent 'out of memory' errors. The pooled threads limited the number of tasks being submitted for scheduling based on the number of resources available to the WE. The threshold for a thread pool that limited the number of tasks pooled for scheduling, is determined at run-time by the WE. As a result, there exists several pools for each application workflow handled by the WE, as shown by the layers inside the workflow scheduler component in Figure 3.1.

Any task in a data intensive workflow may handle large sized data files at both input and output stages of its execution. In order to prevent premature task failure due to time-outs during data transfers between resources, we implemented a fail-safe mechanism that accounts and keeps track of data transfers. In case of failure, our mechanism re-transfers the segments that failed to transfer in the past. As all the scheduling algorithm implemented in the WE are based on partial data transfers from multiple data sources, record keeping of data segment transfers was implemented by the data provenance recorder. The data movement component of WE enables data transfers between distributed resources by using SFTP, GridFTP or HTTP protocols.

We have a mechanism to specify the location to store the intermediate output data for every task in a workflow. For data intensive applications, data may not be stored in the compute nodes where the task finished execution due to their size. So we store them in storage locations specified by the user. For those applications that do not need

tertiary storage, the compute node that executes the tasks may store the data locally until the workflow finishes execution. Depending on the user's requirements, data can then be either migrated to a centralized server after the execution or deleted from the compute nodes entirely. This configuration is also set by the user before the workflow starts executing. The data provenance recorder handles the transfer and indexing of these data in case they are stored in tertiary storage resources.

In our current implementation, we handle failures by resubmitting the tasks to resources that do not have failure history for those tasks. Task resubmission and task duplication have been one of the commonly used techniques for handling failures.

**Algorithm 1** Just-In-Time Scheduler

```
1: for each root task t_i ∈ T_roots do
2:     Assign t_i to an available compute resource c_k
3: end for
4: repeat
5:     for all t_i ∈ T_nonroots do
6:         Assign ready task t_i to any available compute resource c_k
7:     end for
8:     Dispatch all the mapped tasks
9:     Wait for POLLING_TIME
10:    Update the ready task list
11: until there are unscheduled tasks in the ready list
```

As basic scheduling policy, the WE has a just-in-time scheduler that allows the resource allocation decision to be made at the time of task submission. Algorithm 1 lists the scheduling algorithm. Tasks at the top level (that have no parent) are assigned to first available resources (that have not exceeded their job limit and are accepting tasks for execution). Tasks become ready as a result of their parents finishing execution and producing valid data. A list is formed to store all tasks. This list is updated during the *polling_time*. The scheduler then assigns these ready tasks to resources based on the availability of each resource. The optimum choice for *polling_time* depends on the number of tasks in the application, resource management policies, scheduler overhead etc, which is configurable by the user before executing any workflow application. In addition to the just-in-time scheduler, the WE also implements a random and round-robin scheduling policies as basic policies. All other scheduling policies that are designed for data intensive applications (and are major contributions of this thesis) are described in subsequent chapters.

The next subsection describes a process of constructing and executing an application workflow relating to the set of tools described above.

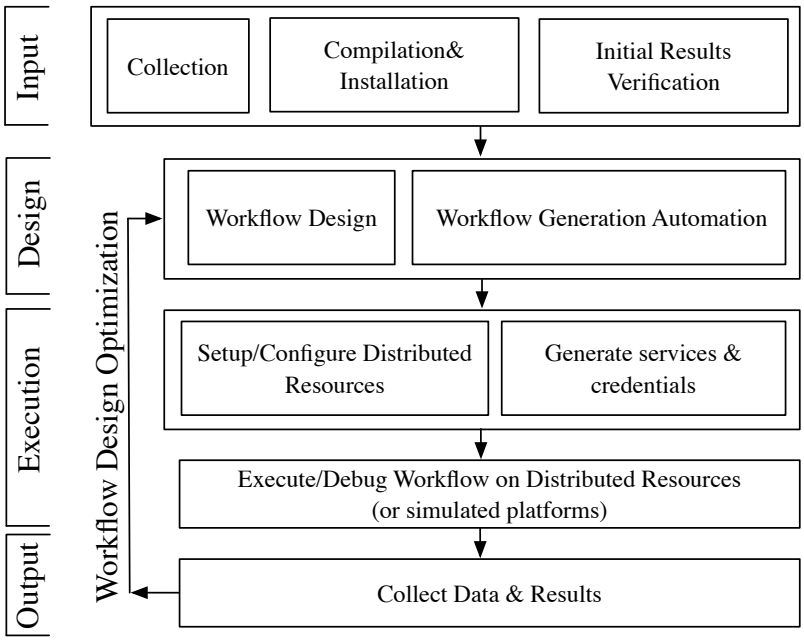



**Figure 3.9:** *A workflow deployment cycle.*

### 3.2.2 Workflow Deployment Cycle

A scientific application needs to be modeled in the form of a workflow before it can take the advantage of coordinated executions on distributed resources. There are a series of steps a workflow designer follows, which we term as the "life cycle" of deployment of the workflow.

The life cycle of deployment of a workflow is depicted in Figure 3.9. In the *input* phase, scientists provide the designer input data, batch scripts for execution, sample output files, and application requirements in a form they have been using at their end. In order to run the application on distributed heterogeneous resources, the executables are required to be compiled and installed (can be submitted at runtime) at both remote and local sites. This step involves linking the bash scripts to related libraries for different platforms and installing dependent utilities. After the application has been installed, the *initial results verification* step involves the testing of conformance of the execution with that of the sample results provided by the scientists. Once the initial results are verified for correctness, the workflow structure and its details are ready for composition in the *designing* phase. The design phase involves constructing the workflow structure by grouping tasks, linking them based on data dependencies and labeling input/output data for data provenance. Usually, after the structure has been finalized, an automated generation tool is used for generating the workflow structure as both an XML document and graphical structure.

In the *Execution* phase of the life cycle, the designer sets up the distributed resources

used during the large scale deployment of the application, very similar to the initial compilation and installation done during the *input* phase. The resource list, its credentials and the services provided by each resource need to be inserted into the system catalog for the workflow engine's use. When experiments are conducted repeatedly over time, resource availability and conditions will have changed. This requires services and credentials list to be generated for each execution with the help of the catalog. Once the resources and catalog are setup, the application is ready for execution on the available resources. Usually debugging and testing is done while the application is being executed, but this depends on the software development process being used at the production site. Depending on the analysis of the results from the *output* phase, the workflow design is further optimized based on the requirements and feedback from the user. These are the basic steps involved in constructing most of the scientific workflows.

The workflow management system depicted in Figure 3.1 facilitates a workflow designer at every stage of the deployment life cycle. After the designer gets hold of the application and its data, the workflow editor helps him to construct the workflow, edit it and automate it as per the requirements. The editor's DAG to XML conversion feature helps in automating the construction of the workflow. The portal based workflow creation, service entries, submission, monitoring and output visualization aids in the design, execution, verification and optimization of the workflow.

### 3.2.3 Integrating Workflow Engine with Grids and Clouds

Figure 3.10 presents a high-level architectural view of a workflow management system utilizing local and Cloud resources to drive the execution of a scientific workflow application.

Local and Grid resources are physical resources that are usually shared among its users and are accessed by using the local resource management system. However, Cloud services vary in the levels of abstraction, and hence the type of service they present to application users. Infrastructure virtualization enables providers such as Amazon to offer virtual hardware for use in compute and data intensive workflow applications. Platform-as-a-Service (PaaS) expose a higher-level development and runtime environment for building and deploying workflow applications on Cloud infrastructures. Such services may also expose domain specific concepts for rapid-application development. Further up in the Cloud stack are Software-as-a-Service providers who offer end-users with standardized software solutions that could be integrated into existing workflows.

User applications could only use Cloud services or use Cloud together with existing Grid/cluster based solutions. Figure 3.10 depicts two scenarios, one where the Aneka platform is used in its entirety to provide the workflow compute and storage resources, and the other where Amazon EC2 is used to supplement a local cluster when there are insufficient resources to meet the QoS requirements of the application. Aneka [147], is a

Platform-as-a-Service Cloud and can be run on a corporate network, a dedicated cluster or hosted entirely on an IaaS Cloud. Given limited resources in local networks, Aneka is capable of transparently provisioning additional resources by acquiring new resources in third party Cloud services such as Amazon EC2 to meet application demands. This relieves the WfMS from the responsibility of managing and allocating resources directly by simply negotiating the required resources with Aneka. Aneka also provides a set of web services for service negotiation, job submission and job monitoring. The WfMS would orchestrate the workflow execution by scheduling jobs in the right sequence to the aneka web services.

The typical flow of events when executing an application workflow on Aneka would begin with the WfMS staging in all required data for each job onto a remote storage resource, such as Amazon S3 or an FTP server. In this case, the data would take the form of a set of files, including the application binaries. This data can be uploaded by the user prior to execution, and stored in storage facilities offered by Cloud services for future use. The WfMS then forwards workflow tasks to Aneka's scheduler via the web service interface. These tasks are subsequently examined for required files and the storage service is instructed to stage them in from the remote storage server, so that they are accessible by the internal network of execution nodes. The execution begins by scheduling tasks to available execution nodes (also known as worker nodes). The workers download any required files for each task they execute from the storage server, execute the application, and upload all output files as a result of the execution back to the storage server. These files are then staged out to the remote storage server so that they are accessible by other tasks in the workflow managed by the WfMS. This process continues until the workflow application is complete.

The second scenario describes a situation in which the WfMS has greater control over the compute resources and provisioning policies for executing workflow applications. Based on user specified QoS requirements, the WfMS schedules workflow tasks to resources that are located at the local cluster and in the Cloud. Typical parameters that drive the scheduling decisions in such a scenario include deadline (time) and budget (cost) [103; 168]. For instance, a policy for scheduling an application workflow at minimum execution cost would utilize local resources and then augment them with cheaper Cloud resources if needed, than using high-end but more expensive Cloud resources from start. On the contrary, a policy that scheduled workflows to achieve minimum execution time would always use high-end cluster and Cloud resources, irrespective of costs. The resource provisioning policy determines the extent of additional resources to be provisioned on the public Clouds. In this second scenario, the WfMS interacts directly with the resources provisioned. When using Aneka, however, all interaction takes place via the web service interface.

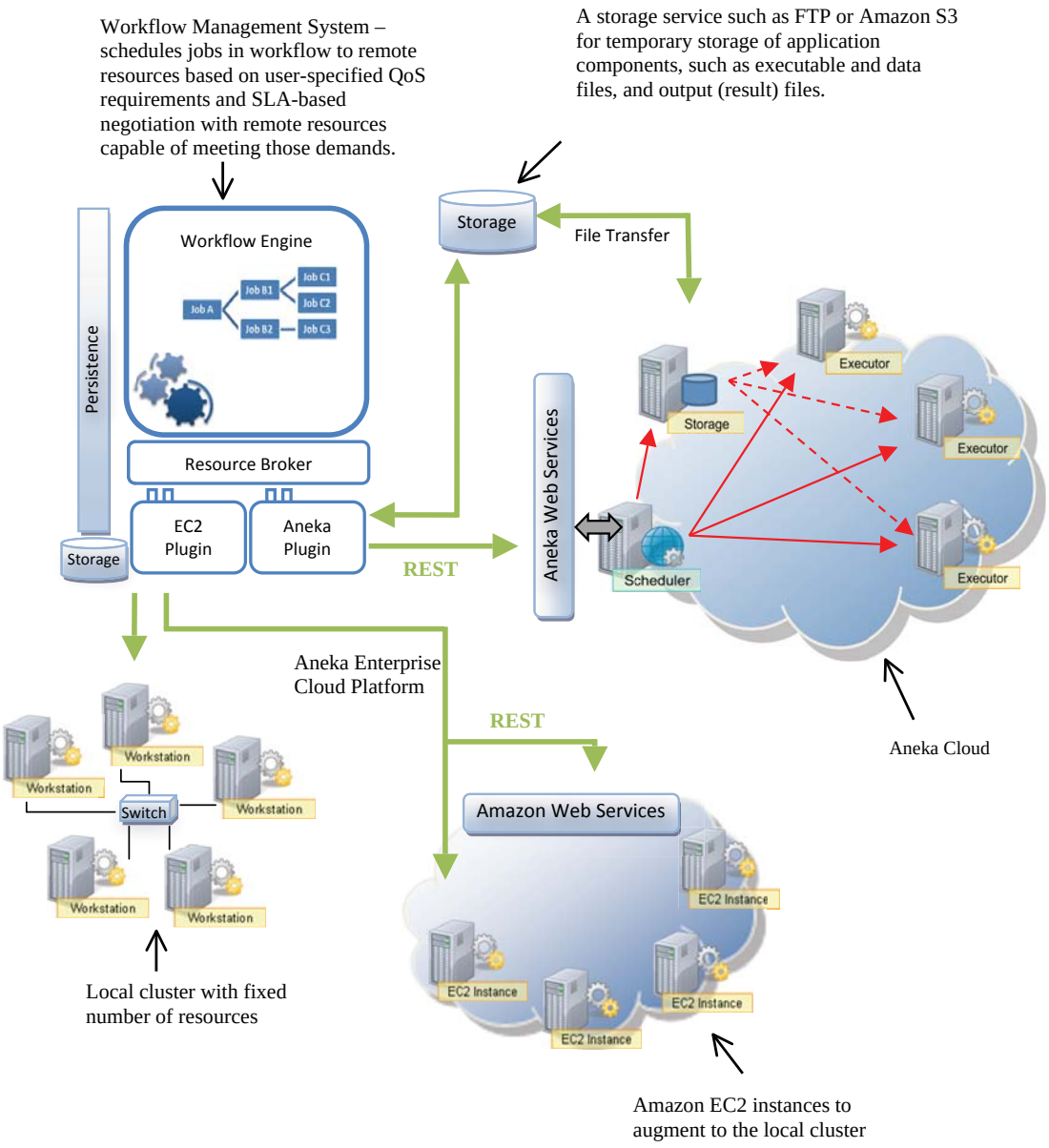



**FIGURE 3.10:** *Integration of workflow engine with Grids and Cloud*

### Tools for Utilizing Cloud Services

As the WfMS is broken down into components to be hosted across multiple Cloud resources, we need a mechanism to access, transfer and store data, enable and monitor executions that can utilize this approach of scalable distribution of components. The Cloud service provider may provide APIs and tools for discovering the VM instances that are associated to a user's account. As various types of instances can be dynamically created, their characteristics such as CPU capacity, amount of memory available, etc., are a

part of the Cloud service provider's specifications. Similarly, for data storage and access, a Cloud may provide data sharing, data movement and access rights management capabilities to user's applications. Cloud measurement tools may be in place to account for the amount of data and computing power used, so that users are charged on the pay-per-use basis. A WfMS now needs to access these tools to discover and characterize the resources available in the Cloud. It also needs to interpret the access rights (e.g. Access Control Lists provided by Amazon), use the data movement APIs and sharing mechanisms between VMs to fully utilize the benefits of moving to Clouds. In other words, traditional catalog services such as the Globus Monitoring and Discovery Service (MDS) [51], Replica Location Services, Storage Resource Brokers, Network Weather Service [155], etc, could be easily replaced by more user-friendly and scalable tools and APIs associated with a Cloud service provider.

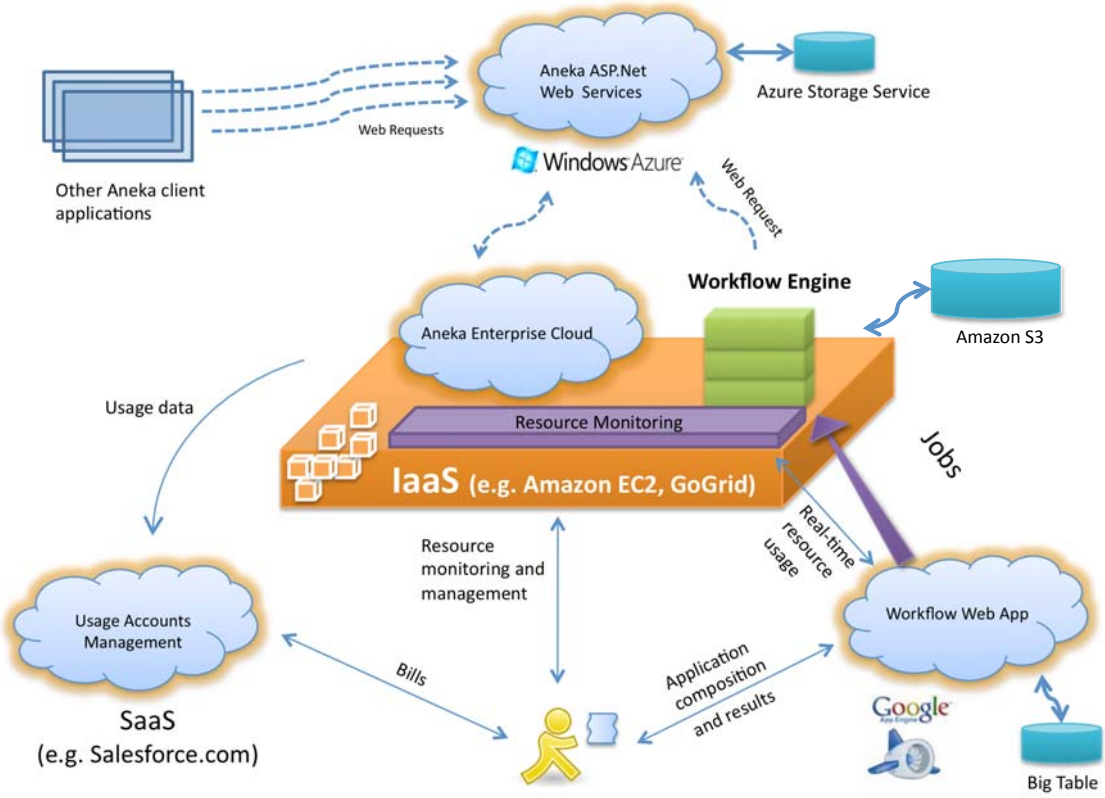



**Figure 3.11:** *The workflow system utilizing various Cloud services.*

The range of tools and services offered by Cloud providers play an important role in integrating WfMSs with Clouds. Such services can facilitate in the deployment, scaling, execution and monitoring of workflow systems. This section discusses some of the tools and services offered by various service providers that can complement and support the WfMS.

The WfMS manages dynamic provisioning of compute and storage resources in the

Cloud with the help of tools and APIs provided by service providers. The provisioning service is required to dynamically scale out/in according to application requirements. For instance, data intensive workflow applications may require large amount of disk space for storage. The WfMS could provision dynamic volumes of large capacity that could be shared across all instances of VMs (similar to snapshots and volumes provided by Amazon). Similarly, for compute intensive tasks in a workflow, the WfMS could provision specific instances that would help accelerate the execution of these compute-intensive tasks.

A persistence mechanism is often important in workflow management systems, for managing meta-data such as available resources, job queues, job status, and user data including large input and output files. Technologies such as Amazon S3, Googles BigTable and the Windows Azure Storage Services can support most storage requirements for workflow systems, while also being scalable, reliable and secure.

The current version of the WfMS implements dynamic provisioning of compute resources in the Amazon Cloud using Amazon's APIs. This provisioning rule depends on several factors: the application being scheduled, the number of VM instances running, user's QoS, and so forth. Hence, it is implemented within the scheduling component of the WE. The persistence of data is handled using Amazon S3, where data can be stored as binary files inside buckets.

Most Cloud service providers also offer services and APIs for tracking resource usage and the costs incurred. This can complement our workflow system that support budget based scheduling by utilizing real time data on the resources used, the duration and the expenditure. This information can be used for both, making scheduling decisions on subsequent jobs and billing the user at the completion of the workflow application .

Cloud services such as Google App Engine and Windows Azure provide platforms for building scalable interactive web applications. This makes it relatively easy to port the graphical components of a workflow management system to such platforms while benefiting from their inherent scalability and reduced administration. For instance, such components deployed on Google App Engine can utilize the same scalable systems that drive Google applications, including technologies such as BigTable [31] and GFS [57].

## 3.3 Case Studies: Executing Real-World Applications using WfMS

Nowadays many scientific experiments such as climate modeling, structural biology and chemistry, neuroscience data analysis, and disaster recovery are conducted through complex and distributed scientific computations that are represented and structured as scientific workflows [58]. Representing these application in the form of a workflow highly simplifies the layout of individual or a group of components of the application as compared to the raw form (usually scripts).

Scientific workflows usually need to process a huge amount of data and are computationally intensive activities. Neuroscience data analysis is one such application that has been a focus of much research in recent years (NIFTI, BIRN). With the use of the additional capacity offered by distributed resources and suitable middleware technologies, we can achieve much shorter execution time, distribute compute and storage load, and add greater flexibility to the execution of these scientific applications than we could ever achieve in a single compute resource.

This section describes two real-world applications that are used as a case study to demonstrate the functionality of WfMS and also test it on real platforms. The applications are: Image Registration (IR) for Functional Magnetic Resonance Imaging (fMRI) and distributed Evolutionary Multi-objective Optimization.

First, we present the processing of IR for fMRI studies. We characterize the application, list its requirements and then transform it to a workflow. We use the WfMS for executing the neuroscience application on Grid'5000 platform and present extensive performance results. We show that the IR application can have 1) significantly improved makespan, 2) distribution of compute and storage load among resources used, and 3) flexibility when executing multiple times on Grid resources. Then we execute the same application using Cloud resources and compare the time and cost of execution on Grid and Cloud resources, in turn.

Second, we present a distributed version of the Evolutionary Multi-objective Optimization algorithm by modeling it as a workflow. Using WfMS as the workflow manager and Aneka as Cloud platform, we parallelize the computation of the workflow using Amazon EC2 resources. We also present experiments where compute resources are dynamically provisioned to handle increasing number of tasks for every added iteration cycle.

### 3.3.1 Image Registration for fMRI Applications

The neuroscience data analysis application, we present in this subsection, has several tasks that need to be structured according to their data dependencies for correct execution. Both the data and computation requirements are very high, depending on the number of subjects analyzed. Given the typically large number of subjects' data being analyzed, it takes significant amount of time for this application to produce results when executed as a sequential process on limited resources. Moreover, scientists may need to re-run the application by varying run-time parameters. Often researchers and users around the globe may share the results produced. To facilitate these requirements such as high compute power, repeated experiments, sharing of data and results, this application may leverage the power of distributed resources presented by platforms such as Grids. By executing this application on distributed resources execution time can be minimized, repeated executions can be performed with little overhead, reliability of execution can be increased,

and resource usage can be distributed. It is a very demanding task for researchers to handle these complex applications directly on Global Grids without proper management systems, interfaces, and utilities. Therefore, user friendly systems are increasingly being developed to enable e-scientists to integrate, structure and orchestrate various local or remote data and service resources to perform scientific experiments to produce interesting scientific discoveries.

A scientific workflow management system is one of the popular approaches that provide an environment for managing scientific experiments, which have data dependent tasks, by hiding the orchestration and integration details inherent while executing workflows on distributed resources. Our WfMS is one such workflow management system that aids users (scientists) by enabling their applications to be represented as a workflow and then execute on the Grid from a higher level of abstraction. The WfMS provides an easy-to-use workflow editor for application composition, an XML-based workflow language for structured representation, and a user-friendly portal with discovery, monitoring, and scheduling components that enables users to select resources, upload files and keep track of the application's progress, as described earlier in this chapter.

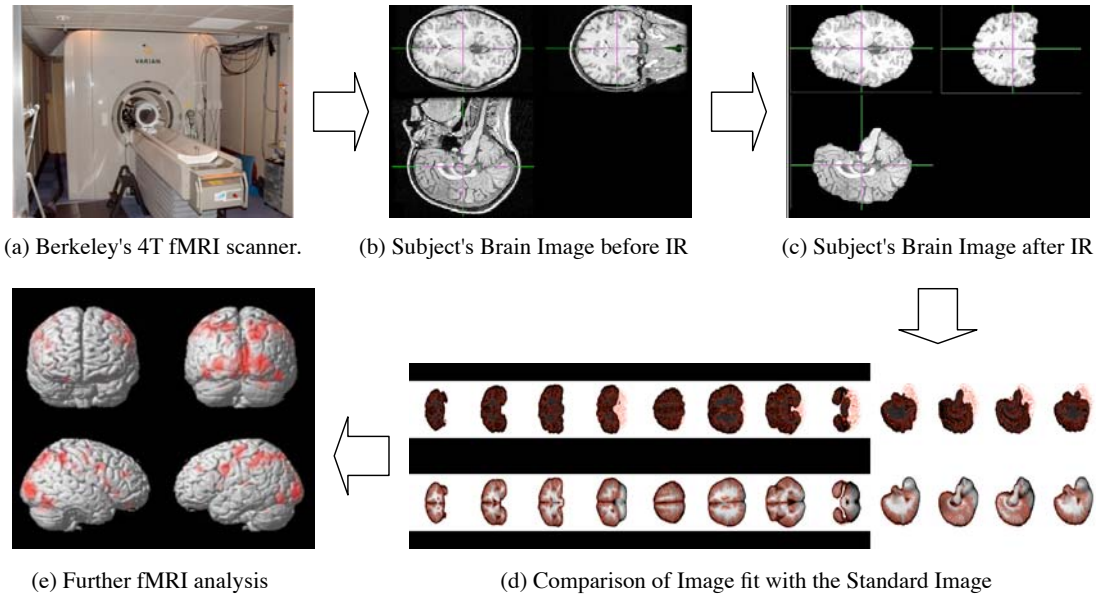
Our main contributions when executing this application are as follows:

1. representation of a brain imaging application in the form of a workflow.
2. characterization of the tasks of brain imaging application workflow in terms of data and computational requirements.
3. design and implement a tool set to manage a complete life cycle of the IR workflow and execute it on distributed resources using the WfMS.
4. a performance evaluation of the workflow execution on the Grid'5000 and Cloud computing platforms.
5. present performance results that could directly assist neuro scientists using brain imaging technology in clinical areas such as epilepsy, stroke, brain trauma and mental health.

**A Scenario and Requirements**

Image registration is a brain imaging technique. We describe the Image Registration (IR) procedure as a Scientific Workflow Application. We construct the IR workflow and describe each of its tasks, and then tabulate the requirements of executing each task in the workflow.

**fMRI and IR:** fMRI attempts to determine which parts of the brain are active in response to some given stimulus. For instance, a person (referred as subject in this thesis),



(a) Berkeley's 4T fMRI scanner. (b) Subject's Brain Image before IR (c) Subject's Brain Image after IR

(e) Further fMRI analysis (d) Comparison of Image fit with the Standard Image

**FIGURE 3.12:** *Image Registration process under fMRI studies.*

in the Magnetic Resonance (MR) scanner, would be asked to perform some tasks, e.g., finger-tap at regular intervals. As the subject performs the task, researchers effectively take 3-D MR images of his brain. The goal is to identify those parts of the brain responsible for processing the information the stimulus provides. This enables doctors and medical scientists to carry on further analysis of the brain or spinal cord of humans or other animals across multiple datasets acquired with different protocols and MR scanners.

IR is ubiquitous in fMRI analysis, especially in the case of multi-subject studies. IR is the process of estimating an optimal transformation between two images, also known as "Spatial Normalization" in functional neuroimaging [110]. When registering images we are determining a geometric transformation, which aligns one image to fit another. The aim is to establish a one-to-one continuous correspondence between the brain images of different individuals. The transformation will reduce the anatomical variability between high-resolution anatomical brain images from different subjects. This enables analysts to compute a single activation map representing the entire group of subjects or to compare the brain activation between two different groups of subjects.

The IR procedure and its relation to fMRI is depicted in Figure 3.12. The scanner acquires high-resolution images of each subject's brain. Due to subject movements, the images can be oriented in different positions at the time of scanning. One such image of a subject before registration is shown in Figure 3.12 (b). The registration process ensures that all the images of different subjects are normalized against a standard image and in a common 3D space. The normalized image of the subject is shown in Figure 3.12 (c). After normalization, the subject's normalized image is compared with the *atlas* (reference

image) for the quality of fit. This comparison is shown in Figure 3.12 (d). The workflow studied in this section, first produces the *atlas*, then produces the comparison image (Figure 3.12 (d)) as output for each subject.

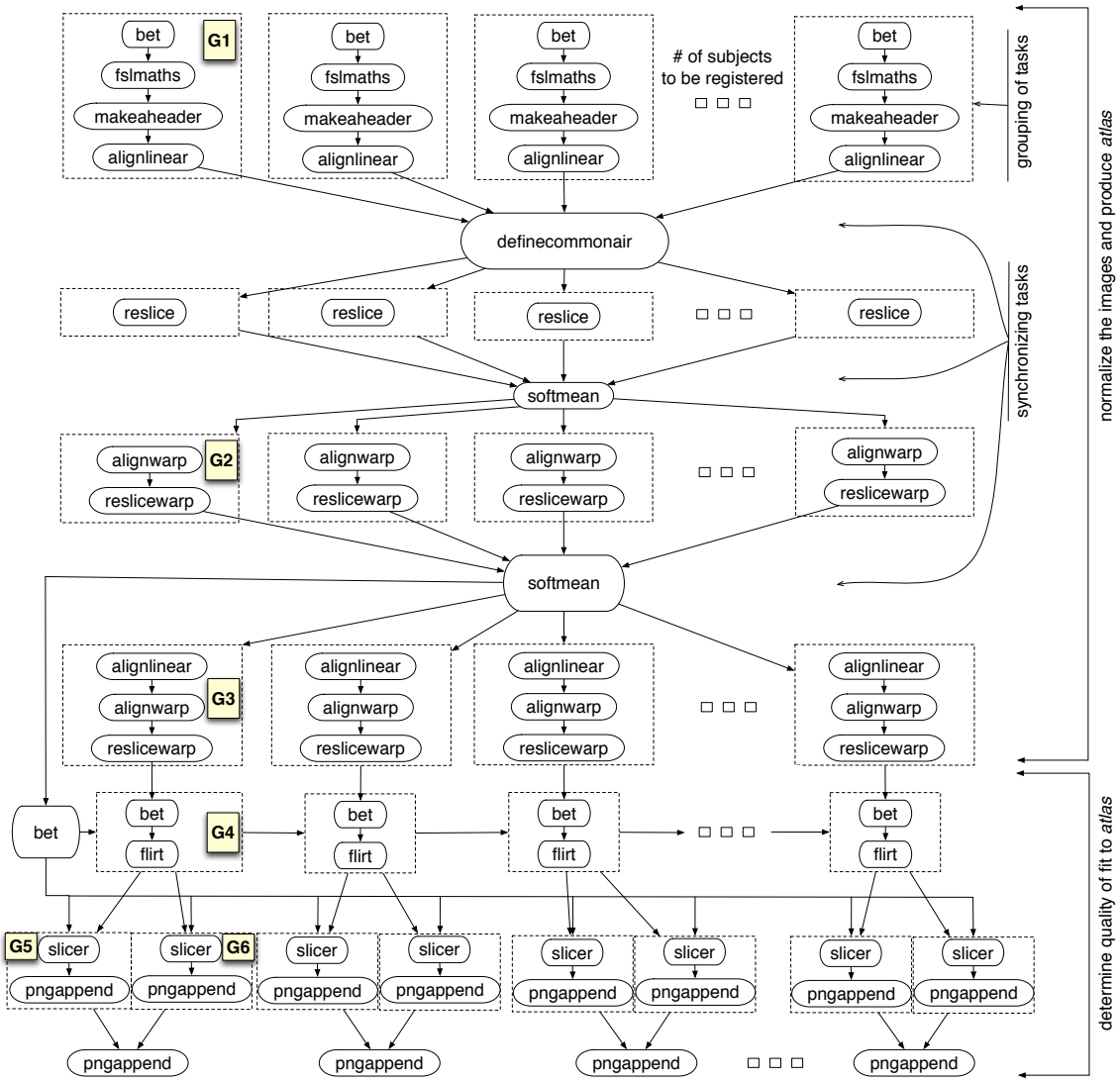



**Figure 3.13:** *Image Registration Workflow.*

**Application Description:** The IR procedure, expressed as a scientific workflow is shown in Figure 3.13. The tasks are linked according to their data dependencies. Individual tasks that form the workflow are described below[1] [135].

BET: (Brain Extraction Tool) deletes non-brain tissue from an image of the whole head and extracts brain's image.

[1]http://bishopw.loni.ucla.edu/AIR5/index.html

FSLMATHS: allows mathematical manipulation of images.

MAKEAHEADER: generates a header (.hdr) file based on the parameters (type, x-y-z dimensions and size).

ALIGNLINEAR: is a general linear intra modality registration tool. Any image can be aligned to a representative with a transformation model using alignlinear. It generates *.air* files that can be used to reslice the specified reslice data set to match the specified standard data set. We use affine 12-parameter full-afine model.

DEFINECOMMONAIR: defines new *.air* files with a new standard file that defines the "average" position, size and shape of the various reslice files.

RESLICE: takes a *.air* file and uses the information that it contains to load the corresponding image file and generate a new, realigned file.

SOFTMEAN: averages together a series of files.

ALIGN_WARP: compares the reference image to determine how the new image should be warped, i.e. the position and shape of the image adjusted, to match the reference image. It is a nonlinear registration tool that generates a *.warp* file that can be used to reslice the specified reslice data set to match the specified standard data set.

RESLICE_WARP: takes a *.warp* file and uses the information that it contains to load the corresponding image file and generate a new, realigned file.

FLIRT: performs affine registration. It produces an output volume, where the transform is applied to the input volume to align it with the reference volume (*atlas* created in previous steps).

SLICER: takes 3D image and produces 2D pictures of slices.

PNGAPPEND: processes addition/subtraction of *.png* images.

**Application Requirements:** According to Zhao et.al [176], in a typical year the Dartmouth Brain Imaging Center about 60 researchers pre-process and analyze data from about 20 concurrent studies. The raw fMRI data for a typical study would consist of three subject groups with 20 subjects per group, five experimental runs per subject, and 300 volume images per run, yielding 90,000 volumes and over 60 GB of data. Intensive analysis begins once the images are processed. IR forms a part of the image pre-processing step using only the high-resolution data, which represents a minor portion of the entire workflow's execution time.

Table 3.1 summarizes the characteristics of each task in the IR workflow. It lists each task, its input files and sizes, its average computation time ($\bar{w}_i$) on a single machine,

**Table 3.1:** *Characteristics of tasks for a single subject's IR.*
note: X = hires, '–' = not applicable (depends on # of subjects),
taskname(*) = same task but different execution instance, N = subject index ($N \in \mathbb{Z}$)

| # | Task Name | Input Files | Source Tasks | Size of i/p (MB) | $\bar{w}_i$(sec) | $\sigma$ |
|---|---|---|---|---|---|---|
| 1 | bet(1) | X.{hdr,img} | Staging server | 16 | 45 | 11.91 |
| 2 | fslmaths | bX.{hdr,img} | bet(1) | 16 | 1 | 0.42 |
| 3 | makeaheader | bX.{hdr,img} | fslmaths | 16 | $\ll 1$ | – |
| 4 | alignlinear(1) | bX.{hdr,img} | fslmaths | 16 | 2 | 0.47 |
| 5 | definecommonair | X.air, bX.{hdr,img} | alignlinear(1) | 16 | 94 | – |
| 6 | reslice | X.air.aircommon, bX.{hdr,img} | definecommonair | 16 | 5 | 0.5 |
| 7 | softmean(1) | X-reslice.{hdr,img} | reslice | 20 | 140 | – |
| 8 | alignwarp(1) | atlas-linear.{hdr,img}, X-reslice.{hdr,img} | softmean(1) | 40 | 971 | 620.17 |
| 9 | reslicewarp(1) | atlas-linear.{hdr,img}, X-reslice.{hdr,img,warp} | alignwarp(1) | 40 | 9 | 1.88 |
| 10 | softmean(2) | X-reslice-warp.{hdr,img} | reslicewarp(1) | 20 | 111 | – |
| 11 | bet(2) | atlas.{hdr,img} | softmean(2) | 20 | 11 | 1.5 |
| 12 | alignlinear(2) | bX.{hdr,img}, atlas.{hdr,img} | definecommonair, softmean(2) | 36 | 23 | 10.25 |
| 13 | alignwarp(2) | X.air, atlas.{hdr,img}, bX.{hdr,img} | alignlinear(2) | 36 | 2656 | 1501 |
| 14 | reslicewarp(2) | bX.{hdr,img,warp} | alignwarp(2) | 16 | 9 | 1.88 |
| 15 | bet(3) | nX.{hdr,img} | reslicewarp(2) | 16 | 15 | 1.3 |
| 16 | flirt | batlas.{hdr,img}, nbX.{hdr,img} | bet(2), bet(3) | 56 | 6 | 0.44 |
| 17 | slicer(1) | batlas.{hdr,img}, N-fit.{hdr,img} | bet(2), flirt | 80 | 8 | 0.44 |
| 18 | pngappend(1) | {sla,slb,...,slk,sll}.png | slicer(1) | 0.3 | 4 | 0.51 |
| 19 | slicer(2) | batlas.{hdr,img}, N-fit.{hdr,img} | bet(2), flirt | 80 | 8 | 0.44 |
| 20 | pngappend(2) | {sla,slb,...,slk,sll}.png | slicer(2) | 0.3 | 4 | 0.28 |
| 21 | pngappend(3) | {N-fit1, N-fit2}.png | pngappend(1), pngappend(2) | 0.8 | 4 | 0.28 |
| 22 | OUTPUT | **N-fit.png** | pngappend(3) | (o/p size) 0.8 | | |
| | **Average data volume and computation time:** | | | **558.2 MB** | **~69min** | |

and standard deviation ($\sigma$) computed over 40 subjects on a set of 10 random machines in Grid'5000 [29]. The random machines chosen did not vary much in their processing powers. The high values of standard deviation ($\sigma$) for certain tasks can be explained by examining the nature of the operation of that task. In this application, tasks having longer

execution time have higher values of deviation. The execution time also depends on the orientation of the image to be aligned.

A complete execution of the workflow of 40 subjects on a single CPU with single core (without local load) takes two and a half days to complete. The total storage space needed for the complete execution of 40 subjects exceeds 20GB on a single machine when the intermediate files are retained. Moreover, the computation time and storage requirements limit the number of subjects that can be used for execution at one time on a single machine.

When the application is executed on distributed resources with no resource scarcity the application should take as much time to execute all the subjects as a single machine would take to execute a single subject workflow without local load. However, the real execution time is higher than the ideal case (~69 minutes) for 1 subject as shown in Table 3.1, due to the significant amount of time taken to transfer the intermediate files from one resource to another. We can decrease the transfer time by allowing multiple tasks to run on a single machine (grouping of tasks). Also, the synchronizing tasks take longer to execute when there are more subjects. The coordination time taken by the middleware also adds to the overall increase in total execution time.

**Application to End-users and Challenges:** In an effort to improve the quality of image registration and to provide clean, high-resolution images for 3D display, researchers have been collecting multiple T1-weighted structural MRI volumes. Recent projects have used up to four of these volumes per subject. The resulting average volume can then be combined with the larger subject population in order to produce the probabilistic atlas. In additional to these anatomically derived processes researchers have begun experimenting with the use of methods that will use functional information to register data collected in a time-series. The introduction of new protocols and the acquisition of multiple high-resolution volumes have increased both the time to acquire and pre-process a typical study. In order to facilitate these new methods imaging centers will need to provide additional data storage and compute capacity. These centers will most likely need to create a shared database of subjects, along with the increased variety of imaging modalities collected, and to expose to individual investigators the methods used to calculate average structural volumes.

Typical fMRI experiments have between twenty and thirty subjects with some having over fifty subjects. A number are removed from the analysis, often due to excessive head movement that image registration algorithms are unable to correct for. While the image registration application described in this chapter makes use of only a single high-resolution volume per subject, the addition of several more volumes would be trivial. Doing such would enable both a cleaner volume for the subject as well as a tighter fit to the atlas. Processes such as these are often repeated many times with spot checks at

critical points, such as during the first average. In the case of a poor fit to the atlas, or an outlier distorting the overall registration, modifications can be made to the workflow. These modifications might include to the rejection of a subject or a change in application parameters.

### Experimental Evaluation

The IR application together with WfMS was demonstrated at the First IEEE International Scalable Computing Challenge (SCALE 2008) in conjunction with CCGrid 2008, May 19-22, 2008, using resources provided by SUNY at Binghamton and the University of Melbourne. The application was also demonstrated at the Fourth IEEE International Conference on e-Science, December 10-12, 2008. The results presented in this section are from the executions of the application on Grid'5000 [29]. We now describe the experiment setup, results obtained, and observations.

**Table 3.2:** *Grid'5000 sites used; # cores (n), # tasks (t) executed and average computation time ($\bar{c}$) (in seconds) taken on each site for each experimental group.*

| Site | 10Sub | | | 10Sub (G) | | | 20Sub | | | 20Sub (G) | | | 40Sub | | | 40Sub (G) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Name | #n | #t | $\bar{C}$ | #n | #t | $\bar{C}$ | #n | #t | $\bar{C}$ | #n | #t | $\bar{C}$ | #n | #t | $\bar{C}$ | #n | #t | $\bar{C}$ |
| bordeaux | 32 | 19 | 189 | 16 | 10 | 83 | 20 | 58 | 141 | 0 | 0 | 0 | 20 | 114 | 235 | 62 | 76 | 306 |
| lille | 16 | 22 | 267 | 12 | 14 | 586 | 64 | 76 | 187 | 16 | 45 | 383 | 20 | 121 | 282 | 44 | 105 | 297 |
| lyon | 6 | 12 | 17 | 6 | 8 | 443 | 24 | 43 | 226 | 8 | 22 | 672 | 6 | 62 | 226 | 6 | 18 | 626 |
| nancy | 10 | 31 | 120 | 0 | 0 | 0 | 14 | 70 | 126 | 0 | 0 | 0 | 10 | 88 | 131 | 0 | 0 | 0 |
| orsay | 10 | 36 | 26 | 8 | 16 | 337 | 0 | 0 | 0 | 4 | 10 | 54 | 10 | 79 | 289 | 20 | 83 | 431 |
| rennes | 10 | 13 | 38 | 0 | 0 | 0 | 14 | 57 | 97 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| sophia | 12 | 24 | 121 | 40 | 24 | 137 | 0 | 0 | 0 | 28 | 58 | 174 | 20 | 135 | 178 | 28 | 121 | 468 |
| toulouse | 20 | 27 | 219 | 12 | 12 | 639 | 20 | 60 | 249 | 20 | 29 | 586 | 20 | 125 | 374 | 0 | 0 | 0 |
| TOTAL | 116 | 184 | | 94 | 84 | | 156 | 364 | | 76 | 164 | | 106 | 724 | | 160 | 403 | |

### Experiment Setup

**Workflow Configuration:** We executed the IR workflow using 40, 20, 10, and 2 subjects. By varying the number of subjects used, we calculated the makespan of the workflow, total storage space required for execution, and parallelism that can be achieved. We grouped the tasks when there was more than a single sequential task between two synchronizing tasks, as depicted in Figure 3.13. Grouping tasks implicitly demands more than one task to be executed at the same site where it is submitted, unlike the ungrouped version where all tasks would be distributed.

**Resource Configuration:** We used the resources provided by Grid'5000 as depicted in Figure 3.16. The Grid'5000 project provides a highly reconfigurable, controllable, and monitorable experimental Grid platform gathering 9 sites geographically distributed in France featuring a total of 5000 processors [29]. Table 3.2 lists the Grid'5000 sites used

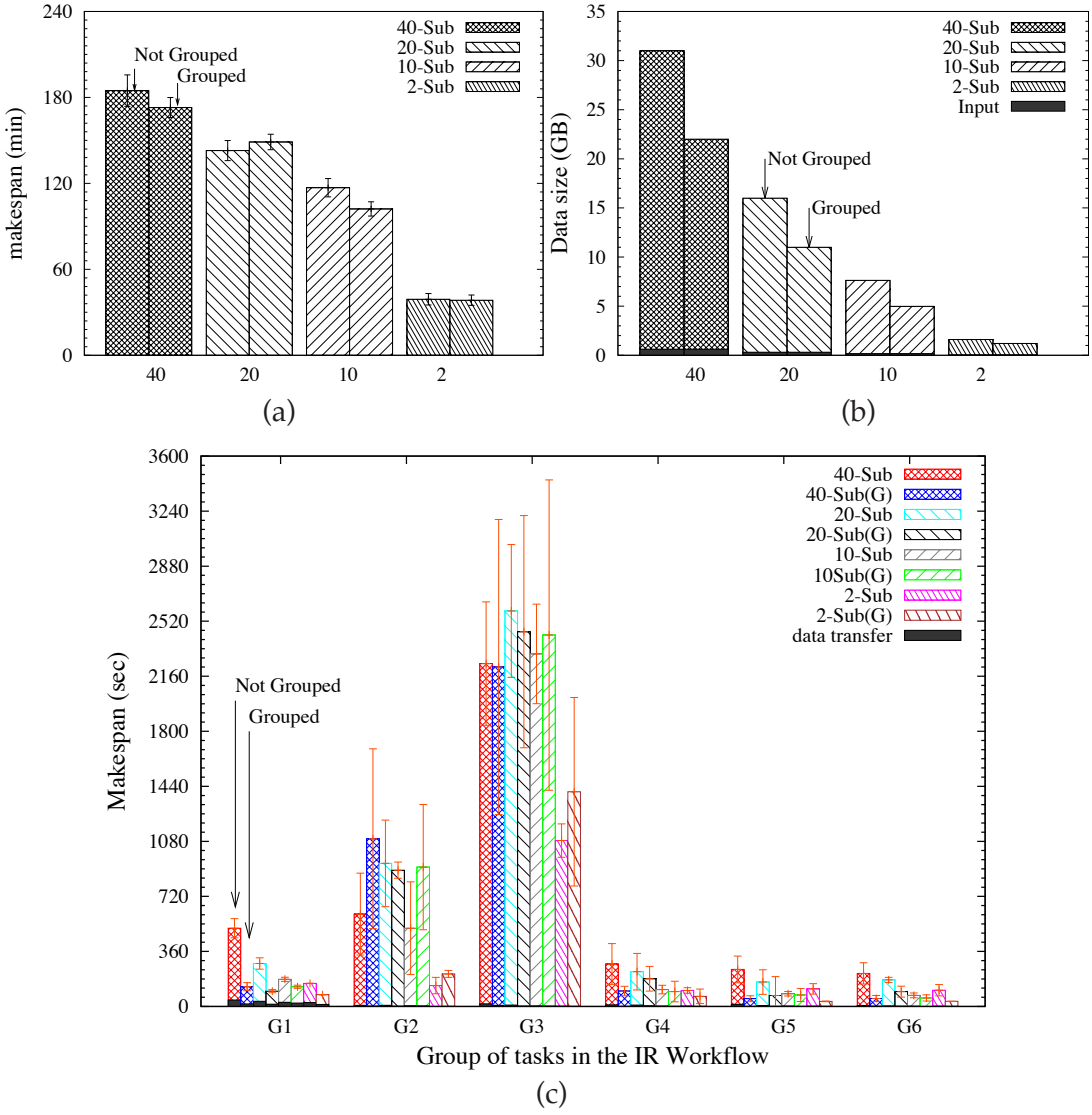



**FIGURE 3.14:** *(a) Comparison of makespan of workflow according to the number of subjects used. (b) Data size according to the number of subjects used. (c) Comparison of makespan between grouped and ungrouped tasks (see Figure 3.13 for grouping of tasks).*

for the experiment. The resources were reserved for the duration of the experiment. The reservation ensured that the Grid resources were dedicated to our experiment. We used resources with the 'x86_64' CPU architecture with AMD Opteron Processors-246, 248, 250, 252, and 275. We used 8 out of the 9 sites (excluding Grenoble). The distributed resources across 8 sites have varying network interconnection bandwidth, number of cores per CPU, CPU frequency, memory, and storage space available [29].

The characteristics of Grid'5000 resources does not vary across 9 sites so much to abruptly affect our application performance. Also, Grid'5000 mandates the users to reserve the necessary nodes before execution. This scenario led us to use a basic level

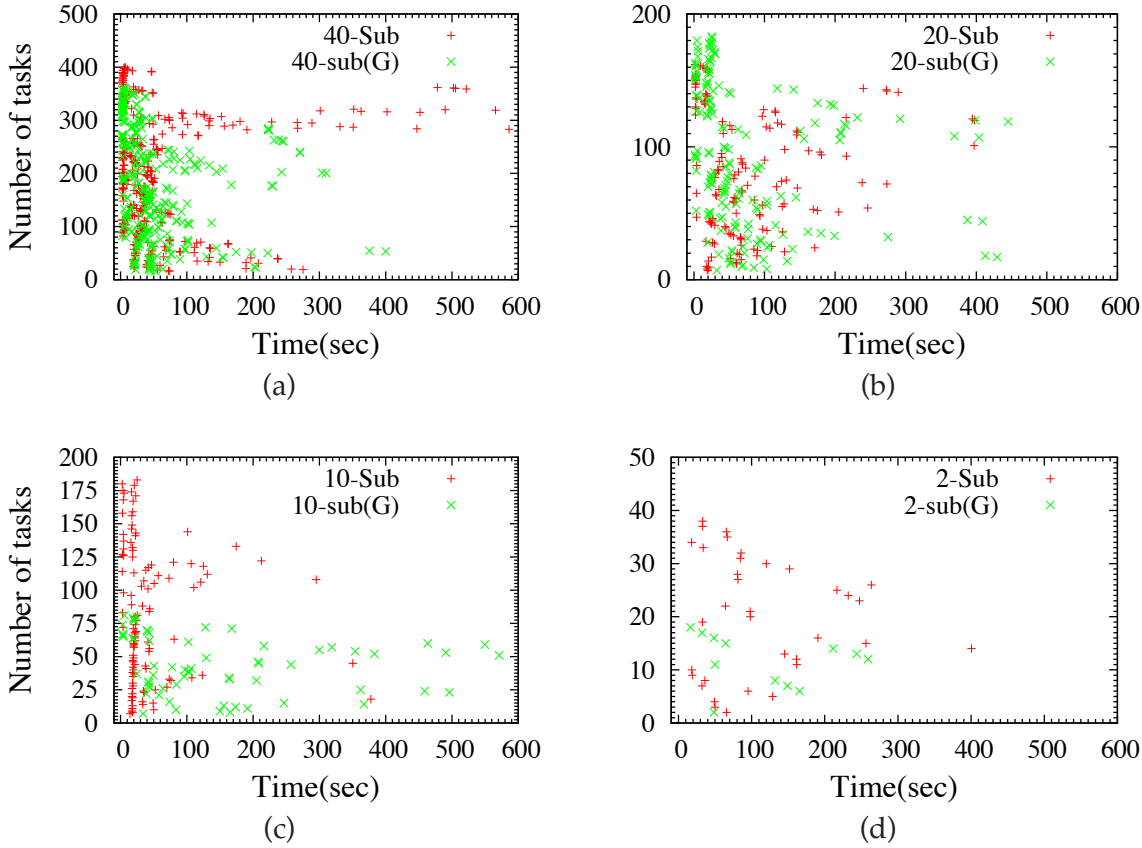



**Figure 3.15:** *Number of tasks executed in time: Parallelism that was achieved in the system. (a) Number of tasks executed in time for 40 subjects. (b) Number of tasks executed in time for 20 subjects. (c) Number of tasks executed in time for 10 subjects. (d) Number of tasks executed in time for 2 subjects.*

scheduling algorithm as listed in Algorithm 1.

**Performance Metrics:** As a measure of performance, we used average makespan as the primary. Makespan of each workflow is measured by taking the difference between the submission time of the first submitted task and the output arrival time of the last exit task to be executed on the system. Makespan also includes the staging-in of the input files to the entry tasks and the staging-out of the results from the exit tasks.

### Results and Observations

Table 3.2 lists the number of cores used at each site, the number of tasks submitted to the site and the average computation time used at the site for each experiment group. Figure 3.14 depicts the total makespan for different subjects, comparison of makespan between grouped and un-grouped tasks of the workflow and the size of data produced during the execution. Figure 3.15 depicts parallelism of tasks executed in time by the

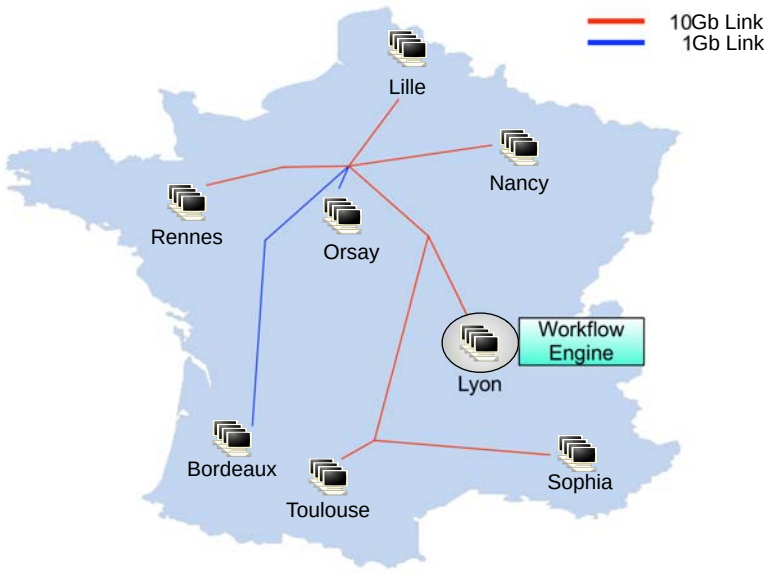



**FIGURE 3.16:** *Grid'5000 Network [29].*

workflow engine for 40, 20, 10 and 2 subjects.

Execution of the IR workflow on the Grid showed significant advantages over a single machine. The total makespan of the workflow decreased considerably from 2.5 days in a single machine to approximately 3 hours on the Grid. The storage requirements were distributed among the resources used. As the number of subjects used was increased, the makespan increased slightly. This can be attributed to the increase in execution time of synchronizing tasks and the coordination time required by the system for additional tasks. The main point to be noted is that as the number of subjects was increased, the average makespan remained within similar bounds and did not increase exponentially, as can be seen for 40, 20, and 10 subjects in Figure 3.14 (a). By inference for more than 40 subjects the makespan should not increase by more than double the difference between the 40 subject and 20 subject makespan.

Grouping of tasks reduced the transmission of data between individual tasks as they were executed on the same machine the group was submitted to. Also, no coordination was required for the individual tasks in the group. This contributed to the reduced makespan in the case of grouped tasks. Figure 3.14(c) shows that the grouping of tasks that have higher value of standard deviation of execution did not yield an optimized makespan. Not grouping tasks with higher execution time and a higher standard deviation value gave lower makespan than the grouped version (center of the graph) of that set of tasks. Tasks with lower execution time and lower standard deviation value had lower makespan value when grouped than when not grouped.

The size of data produced during the execution of the workflow increased when the

number of subjects was increased. The input data size (16MB per subject) was low in comparison to the total data produced during the execution as shown in Figure 3.14(b).

Due to the use of highly available resources, almost all the workflow's ready tasks were executed in parallel, as depicted in Figure 3.14. The graph shows the plot of tasks that finished execution versus time. At a certain interval in the beginning of execution most of the tasks finished execution at the same time showing the parallelism of execution of tasks. Most of the grouped tasks finished execution at the beginning of the execution interval unlike ungrouped tasks. This early execution behavior helped reduce the makespan of the whole workflow as the grouped tasks executed more than one task at a time through a bash script, which is seen as a single task by the resource. In the case of not grouped tasks, each task needed to be mapped onto a resource and as the resource approached its maximum job limit, no more tasks could be submitted to it. This is also the reason that fewer grouped tasks were executed on the system than ungrouped tasks after 100 seconds.

We used a just-in-time scheduling algorithm to schedule tasks in the workflow, as listed in Algorithm 1. As the tasks became ready the scheduler was able to find the available resource and then submitted the task to it for execution. Failure of tasks was handled on a per task basis. When tasks failed, they were re-submitted to another resource, which did not have a failure history for those tasks. Although some tasks failed to execute and were rescheduled, their total count was very small. Tasks can fail due to many reasons. In our experiment, failures occurred when a task finishes execution without throwing any errors but the data produced by the task is not complete. In such cases, the child tasks that depend on that data always fail. We can resolve this fault by resubmitting the immediate parent task and all its child tasks for execution.

The workflow was executed multiple times by changing the parameters of the workflow with the help of the portal. This feature provided flexibility while executing grouped and not grouped versions of the workflow for each of the 40,20,10 and 2 subjects. Without this feature, orchestrating the whole experiment would have taken a longer amount of time than executing the application on a single machine.

**Moving from Grid to Cloud Environment**

The Cloud computing paradigm is emerging and is being adopted at a rapid rate. Gartner ranks it at the top of the hype cycle for the year 2010. As the technology is being adopted by practitioners industry wide, there are numerous challenges to overcome. However, these challenges could be addressed via a realistic vision of the Cloud computing models as proposed by several software and service giants such as Google, Amazon and Microsoft. They own large data centers for providing a variety of Cloud services to customers. These independent and disparate initiatives would eventually lead to an interconnection model where users can choose a combination of services from different providers in their

applications.

Our system design provides an entity responsible for brokerage of resources across different Cloud providers, termed as the Market Maker [149]. These Inter-Cloud environments would then facilitate executions of workflow applications at distributed data centers. Large scientific experiments would then be able to use Inter-Cloud resources, brokered through the Market Maker.

The essence of using Cloud services is to be able to dynamically scale the applications running on top of it. Automating resource provisioning and VM instance management in Clouds based on multi-objectives (cost, time and other QoS parameters) can help achieve this goal. The automation process should be transparent to the end users who would just be interested in running workflow applications under their time and budget constraints. Users would specify either flexible or tight deadline for the cost they pay for using Cloud services. It becomes the responsibility of the workflow engine running in the Cloud to dynamically scale the application to satisfy multiple users request.

In order to facilitate fair but competitive use of Cloud resources for workflow applications, a service negotiation module must be in place. This entity would negotiate with multiple service providers to match users' requirements to a service provider's capabilities. Once a match is found, required resources can then be allocated to the user application. The Cloud market directory service maintains a catalog of services from various Cloud service providers, as depicted in Figure 3.1.

Data and their communication play a vital role in any data intensive workflow application. When running such applications on Clouds, storage and transfer costs needs to be taken into account in addition to the execution cost. The right choice of compute location and storage service provider would result in minimizing the total cost billed to a user. A Cloud Market Maker could handle these task and communication issues at the time of negotiation between various Cloud service providers.

In view of the paradigm shift towards Cloud computing, it is interesting to compare the performance analysis of scientific applications on Grids and Cloud platforms. This section presents the experimental results obtained by executing the IR application using Amazon Cloud resources and compares with the results obtained in the previous section, which was primarily based on Grid resources.

The experiment results presented in this section are a part of the experiment that was conducted during the Second IEEE International Scalable Computing Challenge (SCALE 2009) held at CCGrid 2009 conference in Shanghai, China. The software demo was one of the two winners of the competition.

The system deployed to run the experiments was completely hosted within the Amazon Cloud infrastructure. The execution of the workflow was managed by the Workflow Engine that handled the execution of tasks in the workflow depicted in Figure 3.13. The experiment has been repeated with 2, 10 and 20 subjects and executed on the Amazon

Cloud by using EC2 as a provider of computing resources and S3 for the storage of input data. The results of the execution have been compared with the execution of the same workflow in Grid'5000, in which each compute node in the network served as both storage and compute resource. The metrics used to compare the results of the two executions is the makespan (difference between the submission time of the first submitted task and the output arrival time of the last exit task to be executed on the system) and execution cost of the workflow. The execution cost in Grid5000 is assumed to be zero.

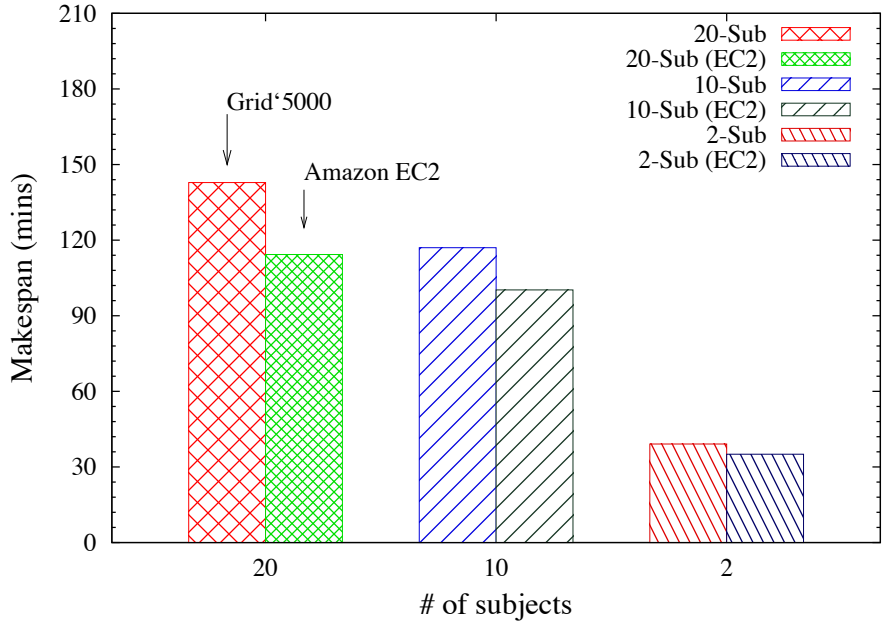



**Figure 3.17:** *Makespan comparison between EC2 and Grid'5000 setups.*

Figure 3.17 compares the makespan of the workflow when the number of subjects used is varied. We observe that for large number of subjects, the makespan decreases when using EC2. For 2 subjects, the change in the makespan is not significant. This difference in makespan is mainly due to the shortening of the data-transfer time between the virtual nodes in EC2 as compared to the transfer between multiple physical sites in Grid'5000. For a large workflow (20 subjects) individual file transfer time gets cumulated, resulting in a significant difference in total makespan when compared to the results of Grid5000.

Figure 3.18 compares the change in makespan versus the EC2 usage cost. The data transfer and the storage cost during execution were very minimal as the compute nodes were part of a Cloud datacenter. As the number of execution nodes is increased from 2 to 20 EC2 nodes, the makespan decreases significantly from 391 minutes to 107 minutes for a workflow analyzing 20 subjects. The cost of usage of Cloud nodes rose from 5.2*to*14.28. However, the ratio between the cost and the number of EC2 nodes used shows that: the total cost of computation of a large workflow (20 subjects) using 20 EC2 nodes would

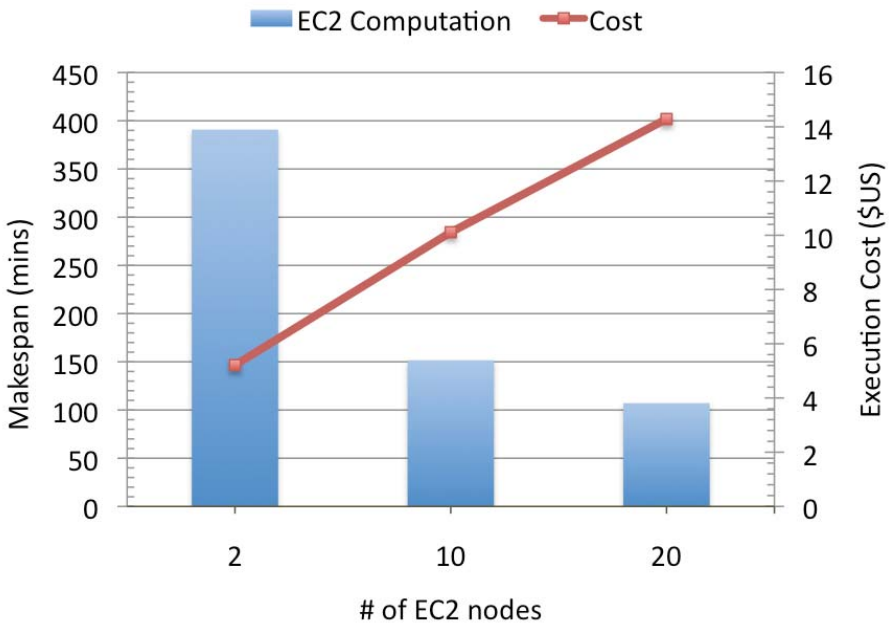



**Figure 3.18:** *Comparison of execution time and cost when using Cloud resources.*

be $0.714/machine, as opposed to $2.6 when executing the same workflow using only 2 EC2 nodes. The average cost of usage per machine decreases as the number of resources provisioned increases from 2 to 20. Consequently, the overall application execution cost increased by not more than three times with a decrease in execution time by similar factor.

From our experiments, we conclude that large high performance applications can benefit from on-demand access and scalability of compute and storage resources provided by public Clouds. Hence, the increase in cost is subdued by the significant reduction in application execution time by making use of abundance of Cloud resources, which can be provisioned on demand. The experiments performed showed that the effective use of Cloud resources was important and a trade-off between cost and performance have to be carefully evaluated.

### 3.3.2 Evolutionary Multi-Objective Optimizations

This section presents a scientific application workflow based on an iterative technique for optimizing multiple search objectives, known as Evolutionary Multi-objective Optimization (EMO) [148]. EMO is a technique based on genetic algorithms. Genetic algorithms are search algorithms used for finding optimal solutions in a large space where deterministic or functional approaches are not viable. Genetic algorithms use heuristics to find an optimal solution that is acceptable within a reasonable amount of time. In the presence of many variables and complex heuristic functions, the time consumed in finding even an acceptable solution can be too large. However, when multiple instances are run in parallel

in a distributed setting using different variables, the required time for computation can be drastically reduced.

The following are the objectives for modeling and executing an EMO workflow on Clouds:

1. Design an execution model for EMO, expressed in the form of a workflow, such that multiple distributed resources can be utilized.

2. Parallelize the execution of EMO tasks for reducing the total completion time.

3. Dynamically provision compute resources needed for timely completion of the application when the number of tasks increase.

4. Repeatedly carry out similar experiments as and when required.

5. Manage application execution, handle faults, and store the final results for analysis using the WfMS.

A detailed description of the EMO application, its workflow model and a sample output is given in Appendix A.

**Deployment and Results**

**EMO Application:** In our experiments, we carry out 10 iterations within a branch for 5 different topologies. We merge and split the results of each of these branches 10 times. For this scenario, the workflow constituted of a total of 6010 tasks. We varied the tasks by changing the number of merges from 5 to 10. In doing so, the structure and the characteristics of the tasks in the workflow would remain unchanged. This is necessary for comparing the execution time when the number of task increases from 1600 to 6000 when we alter the number of merges from 5 to 10.

**Compute Resource:** We used 40 Amazon EC2 compute resources for executing the EMO application. 20 resources were instantiated at US-east-1a and 20 were instantiated at US-east-1d. Among these resources, 1 was used for the workflow engine, 1 for Aneka's master node and the rest were worker nodes. The characteristics of these resources are listed in Table 3.3.

The workflow engine, along with a database for persistence, the IBM TSpace [82] based co-ordination server, and the Tomcat web container was instantiated on a medium instance VM.

**Table 3.3:** *Characteristics of Amazon compute resources (EC2) used in our experiment*

| Characteristics | Aneka Master/Worker | Workflow Engine |
|---|---|---|
| Platform | Windows 2000 Server | Linux |
| CPU (type) | 1 EC2 Compute Units[1] (small) | 5 EC2 Compute Units[2] (medium) |
| Memory | 1.7 GB | 1.7 GB |
| Instance Storage | 160 GB | 350GB |
| Instance Location | US-east-1a (19) US-east-1b(20) | US-east-1a |
| Number of Instances | 39 | 1 |
| Price per hour | $US 0.12 | $US 0.17 |

[1]Small Instance (Default) 1.7 GB of memory, 1 EC2 Compute Unit (1 virtual core with 1 EC2 Compute Unit), 160 GB of instance storage, 32-bit platform
[2]High-CPU Medium Instance 1.7 GB of memory, 5 EC2 Compute Units (2 virtual cores with 2.5 EC2 Compute Units each), 350 GB of instance storage, 32-bit platform (Source: Amazon)

### Experimental Results when using Clouds

As the EMO workflow is an iterative approach, increasing the number of iterations would increase the quality of optimization in the results. Analogously, the more the number of tasks completing in the workflow, the more the number of iterations, hence the better is the optimization. As the iterations can be carried out for an arbitrarily large number of times, it is usually a best practice to limit the time for the overall calculation. Thus, in our experiment we set the deadline to be 95 minutes. We then analyze the number of tasks completing within the first 95 minutes in two classes of experiments:

**Experiment 1** : 7 additional EC2 instances were added In this experiment, we started executing the tasks in the EMO workflow initially using 20 EC2 compute resources (1 node for Workflow engine, 1 node for Aneka master, 18 Aneka worker nodes). We instantiate 7 more small instances to increase the total number of resources to 25. They were available for use after 25 minutes of execution. At the end of 95 minutes, a total of 1612 tasks were completed.

**Experiment 2** : 20 additional EC2 instances were added In this experiment, we started executing the tasks in the EMO workflow using 20 EC2 compute resources, similar to Experiment 1. We instantiated 20 more EC2 instances after noticing the linear increase in task completion rate. These instances however were available for use after 40 minutes of execution. At the end of 95 minutes, a total of 3221 tasks were completed.

**Analysis of the results** : In both the experiments, the initial task completion rate increased linearly until we started more instances, as depicted in Figure 3.19 . As the number of resources was increased, the rate of task completing increased drastically. This is due to the submission of queued tasks in Aneka to the newly available resources, which would have remained queued if resources were not added.

As depicted in Figure 3.19, the completion rate curve rises up to a point until when all the queued tasks are submitted. The curve then rises gradually due to the fact that

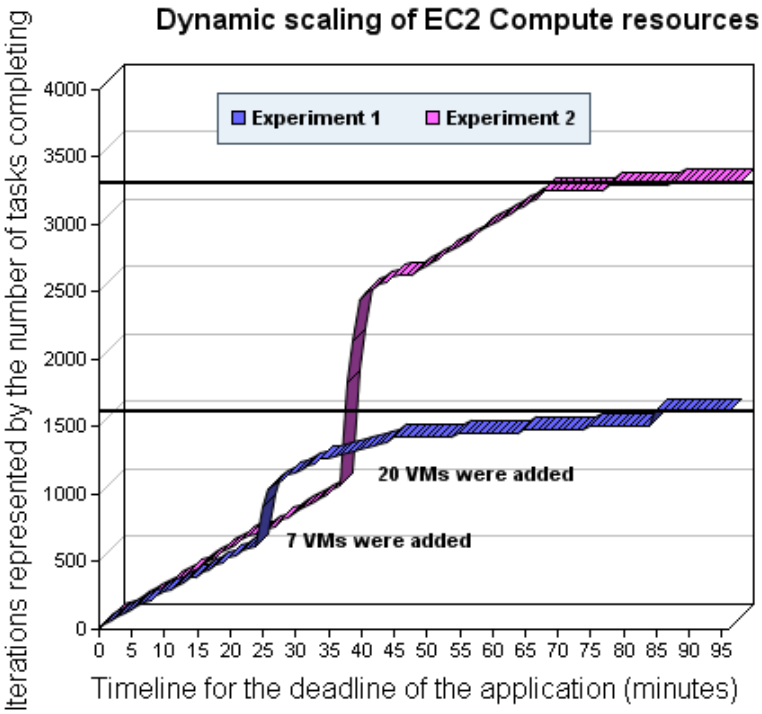



**Figure 3.19:** *Number of tasks completing in time as the number of compute resources provisioned were increased at run-time*

the EMO application is a workflow. Tasks in the workflow get submitted gradually as their parents finish executions. Hence, the completion rate has similar slope as the initial rate, even after increasing the number of resources (from 30 to 45 minutes for Experiment 1; from 45 to 70 minutes for Experiment 2). When more tasks began completing as a result of adding new resources, the workflow engine was able to submit additional tasks for execution. As a result, tasks started competing for resources and hence were being queued by Aneka. Due to this queuing at Aneka's scheduler, the curve flattens after 45 minutes for Experiment 1 and after 70 minutes for Experiment 2, respectively.

The most important benefit of increasing the resources dynamically at run time is the increase in the total number of tasks completing, and hence the quality of final result. This is evident from the two graphs depicted in Figure 3.19. If a total of 25 resources were used, Experiment 1 would complete 1612 tasks by the end of the 95 minutes deadline. Where as, Experiment 2 would complete executing nearly 3300 tasks within the same deadline if 20 additional resources were added. The quality of results would be twice as better for Experiment 2 than for Experiment 1. However, if a user wants to have the same quality of output as in Experiment 1 but in much shorter time, he should increase the number of resources used well before the deadline. A line just above 1600 in Figure 3.19 depicts the cut-off point where the user could terminate all the VM instances and obtain the same quality of results as Experiment 1 would have obtained by running for 95 minutes. It took

45 minutes less time for Experiment 2 to execute the same number of tasks as Experiment 1. This drastic reduction in time was seen even when both the experiments initially started with the same number of resources. In terms of cost of provisioning additional resources, Experiment 2 is cheaper as there are fewer overheads in time spent queuing and managing task submissions, as the tasks would be submitted as soon as they arrive at Anekas master node. If Amazon were to charge EC2 usage cost per minute rather than per hour, Experiment 2 would save 45 minutes of execution time at the cost of 20 more resources.

## 3.4 Related Work

Over the recent past, a considerable body of work has been done on the use of workflow systems for scientific applications. Some of them have investigated workflow technology with respect to our target applications (fMRI and EMO) and workflow management. This section briefly describes these work.

**Workflow Management Systems:** Some of the popular workflow systems for scientific applications include DAGMan (Directed Acyclic Graph MANager), Pegasus [40], Kepler [89], and Taverna workbench [98]. DAGMan is a workflow engine under the Pegasus workflow management system. Pegasus uses DAGMan to run the executable workflow. Kepler provides support for Web Service based workflows. It uses an actor-oriented design approach for composing and executing scientific application workflows. The computational components are called actors, and are linked together to form a workflow. The Taverna workbench enables the automation of experimental methods through the integration of various services, including WSDL-based single operation Web Services, into workflows. Yu et al. [167] provided a comprehensive taxonomy of workflow management systems based on workflow design, workflow scheduling, fault management, and data movement. They characterized and classified different approaches for building and executing workflows on Grids. They also studied existing Grid workflow systems highlighting key features and differences.

Deelman et al. [37] have done considerable work on planning, mapping and data-reuse in the area of workflow scheduling. They propose Pegasus [40], which is a framework that maps complex scientific workflows onto distributed resources such as the Grid. DAGMan, together with Pegasus, schedules tasks to Condor system. With the integration of Chimera [53] and Pegasus based [37] mapping, it can execute complex workflows based on pre-planning.

The Taverna project [98] developed a tool for the composition and enactment of bioinformatics workflows for the life science community. This tool provides a graphical user interface for the composition of workflows. Other well-known projects on workflow

systems include GridFlow [28], Unicore [123], ICENI [55], GridAnt [6] and Triana [139].

In contrast to existing workflow management systems, the design of WfMS presented in this chapter integrates not only local compute resources, but compute and storage resources leased from Cloud service providers. This results in the system being able to schedule applications workflow across multiple hosts taking into account their differing characteristics in computing time and cost. In addition to making the system highly available, the WfMS also provides user-friendly environment for editing, submitting, executing, and monitoring workflow applications. The workflow engine is capable of integrating scheduling algorithms as plug-in components. Using this feature, any new algorithms can be used with the engine to schedule workflow applications in distributed environments.

**Workflow on Clouds:** Scientific workflows are commonly executed on shared infrastructure such as Tera-Grid, Open Science Grid , and dedicated clusters [73]. Existing workflow systems tend to utilize these global Grid resources that are made available through prior agreements and typically at no cost. The notion of leveraging virtualized resources was new and the idea of using resources as a utility [22; 23] was limited to academic papers and was not implemented in practice. With the advent of Cloud computing paradigm, economy based utility computing is gaining widespread adoption in the industry.

Deelman et al. [39] presented a simulation-based study on the costs involved when executing scientific application workflows using Cloud services. They studied the cost performance tradeoffs of different execution and resource provisioning plans, and the storage and communication fees of Amazon S3 in the context of an astronomy application known as Montage [37; 39]. They conclude that Cloud computing is a cost-effective solution for data intensive applications.

The Cloudbus toolkit [25] is one of the initiatives towards providing viable solutions for using Cloud infrastructures. It proposed a wider vision that incorporates an inter-cloud architecture and a market-oriented utility computing model.

**Distributed Execution of Scientific Applications:** Olbarriaga et al. [99] presented the Virtual Laboratory for fMRI (VL-fMRI) project, whose goal is to provide an IT infrastructure to facilitate management, analysis, and sharing of data in neuroimaging research with a focus on functional MRI. The workflow system design presented in this chapter share a common objective to facilitate the data logistics and management in fMRI analysis via workflow automation. Their system could use our workflow management system as a pluggable component.

Neurobase [45] used grid technology for the integration and sharing of heterogeneous sources of information in neuroimaging from both data and computing aspects.

Buyya et al. [24] studied instrumentation and distribution analysis of brain activity data on global grids. They presented the design and development of Magnetoencephalography (MEG) data analysis system. They described the composition of the neuroscience application as parameter-sweep application and its on-demand deployment on Global Grids.

Ellis et al. [44] executed their IR algorithm by registering several couples of T1 MRI images coming from different subjects in 5 minutes on a Grid consisting of 15 2GHz Pentium IV PCs linked through a 1Gigabit/s network. This is an example where the capabilities of Grid have been used to speedup brain imaging applications.

The LONI Pipeline [121] was developed to facilitate ease of workflow construction, validation and execution like many similar workflow environments, primarily used in the context of neuroimaging. This initiative, which was as early as 2003, clearly demonstrated that workflow technology could be used and is viable for neuroimaging applications.

The NIMH Neuroimaging Informatics Technology Initiative (NIFTI[2]) was formed to aid in the development and enhancement of informatics tools for neuroimaging. Likewise, the Biomedical Informatics Research Network (BIRN[3]) is another high profile effort working to develop standards (eg. LONI) among its consortia membership. Such efforts are contributing more towards standards, efficiency, interoperability and integration of tools.

Distributed execution of population based metaheuristics has been investigated by in greater detail by Enrique Alba [2]. In particular, their work on parallel execution models of parallel genetic algorithms [3] using heterogeneous computing concluded that these algorithms benefited from the computational resources offered by modern LANs and by Internet. Specifically, there is a specific class of genetic algorithms that provides better performance with near optimal results only when applied to large populations. These algorithms are those that best benefit from distributed infrastructure because population partitioning techniques can be applied. EMO is one of such application that is suitable for demonstrating the usefulness of distributed computing.

The workflow engine [102], presented in this chapter, scales out workflow applications on Clouds using market-oriented principles. It has been used for demonstrating the scalability of e-Science applications from biomedical, astrophysics, and engineering at the IEEE International Scalable Computing Challenge (SCALE) several times.

## 3.5 Conclusions

This chapter presented a workflow management system design and described its core components. It also presented a comprehensive description of using workflow engine

[2]http://nifti.nimh.nih.gov
[3]http://nbhirn.net

in Grid and Cloud computing environments. By focusing the limitations of existing workflow management systems when handling data intensive applications, it proposed changes that needed to be incorporated in the system design when moving from Grids to new distributed platforms, such as Clouds. This was supported by the inclusion of Cloud tools that could help applications use Cloud services.

This chapter also presented the processing of two compute and data intensive applications: brain imaging application and distributed evolutionary algorithms, as case studies. These applications used the workflow management system as a middleware to access and leverage the power of distributed resources. It modeled both of these applications as a workflow, listed their execution characteristics, and deployed them on Grids and Cloud platforms. Specifically, the IR application was experimented on the Grid'5000 platform. To demonstrate a practical scenario of deploying the workflow engine in Clouds, the EMO application was experimented using Amazon Cloud services. By modeling real-world data and compute intensive application in the form of a workflow, this chapter presented experimental results that demonstrated an order of magnitude improvement in the application run-time. Thousands of tasks completed in a short period of time as resources were provisioned and managed by the proposed WfMS.

In contrast to Grid services, Cloud services charge users based on the resource usage. Due to this fact, although Clouds offer many benefits, they can not and will not replace Grids. Clouds will augment Grids. Users will use Cloud services together with their in-house solutions (cluster/Enterprise Grids) to enhance the performance of their applications as and when needed. In this regard, the proposed WfMS provides a flexible environment to that users can use to access distributed computing platforms such as Grids and Clouds.

The WfMS system design presented in this chapter has been extended substantially from its abstract model given by Yu et al. [165] as part of her PhD thesis. These extensions have been put in place so that the system can integrate both Grid and Cloud resources and execute data intensive applications. The XML based workflow language was extended to support QoS parameters and multiple data files for a single task. The workflow scheduling algorithms implemented were designed to support replicated data files and their retrievals from multiple data sources. The WfMS is also now capable of connecting to Cloud resource providers such as Amazon EC2 and Cloud platform services such as Aneka. The integration of a workflow editor, a monitor into the web-based management portal makes WfMS a complete system. All these characteristics present in one system is a clear distinction of WfMS from existing systems such as Kepler [89], Taverna workbench [98], GridFlow [28], Unicore [123], ICENI [55], GridAnt [6] ,and Triana [139]. Most of the existing systems are either a GUI based toolkit to construct and model a workflow or execution engines supporting Grid and cluster resources. WfMS integrates these two features into one system and adds capabilities that reach emerging distributed paradigms

such as Clouds. As WfMS uses XML to represent its workflow applications, it has the capability of interacting with existing workflow systems for submitting workflow tasks. For e.g. Kepler too uses XML to represent its applications to be submitted to its processing components (actors). WfMS could use a schema compliant to Kepler's actor model and transform its XML workflows to represent a workflow that Kepler could execute. Hence, interoperability and connectivity can be easily achieved between WfMS and existing workflow systems using XML based workflow application definitions.

In conclusion, the experiments performed on the two case studies demonstrated the feasibility of the WfMS in practical environments. Based on the results obtained when using Grid and Cloud services, this chapter concludes that large applications can certainly benefit in terms of decreased run-time; on-demand resource provisioning; and ease of resource management.

# 4

# A Non-Linear Model for Optimisation of Workflow Scheduling

Cloud computing is an emerging technology that allows users to utilize on-demand computation, storage, data and services from around the world. However, Cloud service providers charge users for these services. Specifically, to access data from their globally distributed storage edge servers, providers charge users depending on the user's location and the amount of data transferred. When deploying data intensive applications in a Cloud computing environment, optimizing the cost of transferring data to and from these edge servers is a priority, as data play the dominant role in the application's execution.

In this chapter, we formulate a non-linear programming model to minimize the data retrieval and execution cost of data intensive workflows in Clouds. Our model retrieves data from Cloud storage resources such that the amount of data transferred is inversely proportional to the communication cost. We take an example of an 'intrusion detection' application workflow, where the data logs are made available from globally distributed Cloud storage servers. We construct the application as a workflow and experiment with Cloud based storage and compute resources. We compare the cost of multiple executions of the workflow given by a solution of our non-linear program against that given by Amazon CloudFront's 'nearest' single data source selection. Our results show a savings of three-quarters of total cost using our model.

## 4.1 Introduction

Scientific and commercial applications are leveraging the power of distributed computing and storage resources [39; 174]. These resources are available either as part of general purpose computing infrastructure such as Clusters and Grids, or through commercially hosted services such as Clouds [7]. Clouds have been defined to be a type of parallel and distributed system consisting of inter-connected and virtualized computers. These computers can be dynamically provisioned as per users' requirements [26]. Thus, to achieve

better performance and scalability, applications could be managed using commercial services provided by Clouds, such as Amazon AWS, Google AppEngine, and Microsoft Azure. Some of these cloud service providers also have data distribution services, such as Amazon CloudFront[1]. However, the cost of computing, storage and communication over these resources could be very high for compute intensive and data intensive applications.

Data mining is an example application domain that comprises of data intensive applications often with large distributed data and compute intensive tasks. Examples of data mining applications are: checking bank account lists with lists of suspected criminals (Watch List Compliance), checking for duplication of customer data in financial marketing, using catalogue data in astrophysical image analysis or detecting the spread of Internet worms using intrusion detection systems. Detecting the spread of Internet worms is an example of a data mining application. This scenario will be revisited in Section 4.2.

The data to be mined may be widely distributed depending on the nature of the application. As the size of these data-sets increases over time, the analysis of distributed data-sets on computing resources by multiple users (repeated executions) has the following challenges:

- A well-designed application workflow: Large number of data-sets and mining tasks make the application complex.
- Minimization of communication and storage costs: Large size and number of distributed data-sets make the application data intensive.
- Minimization of repeated data mining costs: Cost of computing (classification/-knowledge discovery) and transferring of data increases as the number of iterations/data-sets increase.

In this chapter, we address the challenges listed above for data intensive workflows by making the following three contributions:

1. We take Intrusion detection as a data mining application which will be referenced throughout the remainder of this chapter. This application has all the features as listed in the previous paragraph when executing commercially [174]. We design the application as a workflow that simplifies the basic steps of data mining into blocks.
2. We model the cost of execution of an intrusion detection workflow on Cloud resources using a Non-Linear Programming (NLP) model. The NLP-model retrieves data partially from multiple data sources based on the cost of transferring data from those sources to a compute resource, so that the total cost of data-transfer and computation cost on that compute resource is minimized. The solver gives a solution

[1]http://aws.amazon.com/cloudfront/

that chooses data sources and compute sites for multi-source partial data retrievals and computation proportional to the cost of communication and computation, respectively.

3. We then apply the NLP-model on the intrusion detection application to minimize repeated execution costs when using commercial compute and storage resources. As an example, we compare the costs between our model and Amazon CloudFront.

The remainder of the chapter is organized as follows: we present an intrusion detection application and its workflow design in Section 4.2; cost minimization problem using NLP model in Section 4.3; the NLP-model and its use for the intrusion detection application in Section 4.4; experimental setup in Section 4.5 and analysis in Section 4.6; related work in Section 4.7. Finally, we conclude in Section 4.8.

## 4.2 Intrusion Detection Using Data from Distributed Data Sources

First, we describe a use-case for Internet worm detection. Then, we describe the process of intrusion detection in general and present a workflow design for executing data mining steps over distributed intrusion data logs.

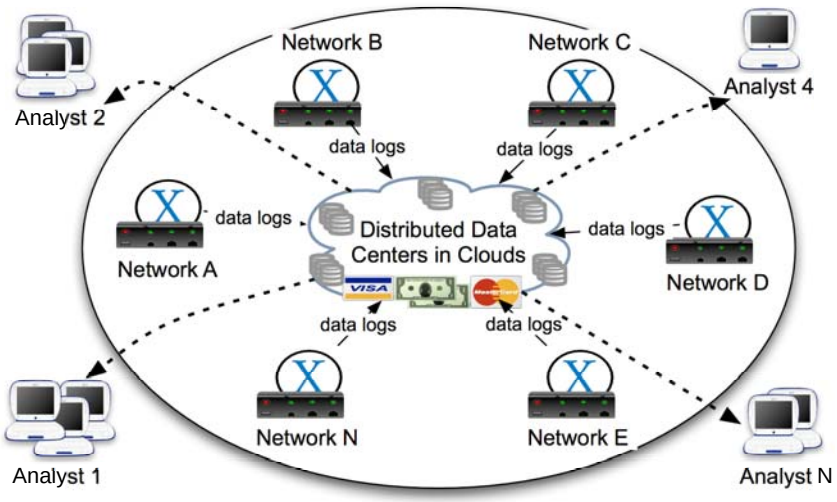



**Figure 4.1:** *Global Intrusion Detection scenario*

### 4.2.1 Intrusion detection scenario

Intrusion detection as defined by the SysAdmin, Audit, Networking, and Security (SANS[2]) institute is the act of detecting actions that attempt to compromise the confidentiality, in-

[2]http://www.sans.org/security-resources/

tegrity or availability of a resource. We take an example of detecting the spread of a malicious worm over the Internet.

In practice, a large number of independent networks spanning throughout the Internet share their network logs to detect such an outbreak. The logs from each individual network are continuously fed to the Amazon Cloud storage (or some other services), which distributes them to globally distributed edge servers.

The aim of the intrusion detection system (or the analyst) is to analyze these combined logs to detect an outbreak of a worm. Such analysts can be located at multiple locations close to some of the data sources but at a large network distance from a majority of the other data sources.

Assuming that every intrusion detection system (or analyst) follows the same data mining process, which we describe later in the chapter, the Naive approach is to separately aggregate the log data from all independent networks for every analyst. It is not hard to visualize the redundancy in the data transfer (for each individual network) and hence the cost associated with such massive amount of data transfers.

Using the distributed edge servers, we can minimize the cost of data transfer to each individual intrusion detection system (analyst). We represent this scenario in Figure 4.1. With an aim to minimize the cost of data transfer, we develop a non linear programming based approach, described later in the chapter, and compare it with the standard nearest source approach adopted by CloudFront and observe that our model achieves a significant savings of three-quarters of the total cost.

### 4.2.2 Intrusion detection process as a workflow

Network monitoring, as a part of intrusion detection, is a common process carried out by network administrators in order to observe the activities in a particular network. As it is a continuous process, the size of data that must be monitored varies with the bandwidth and latency of the network, which can be in several Gigabits per second. This makes the application data intensive. Furthermore, networks are not restricted to a small room or a building and can spread throughout the globe. In such a distributed setting, it becomes critical to optimize the cost of data transfer from distributed sources in order to perform very frequent network monitoring. The situation becomes more challenging when the raw data, which can be used to detect such attacks are globally distributed. Hence, in this chapter, we focus on minimizing the cost of such distributed analysis.

Data mining techniques have become prominent in detecting intrusions [81; 60]. Detecting intrusions can be considered as finding the outliers (or unusual activities) and, hence, data mining can be easily applied to perform this task.

We modeled the intrusion detection process as a workflow as illustrated by Figure 4.3. The figure separates the training, testing, and real-time processes into blocks as Block A, Block B and Block C, respectively. The first step for training is to collect some training

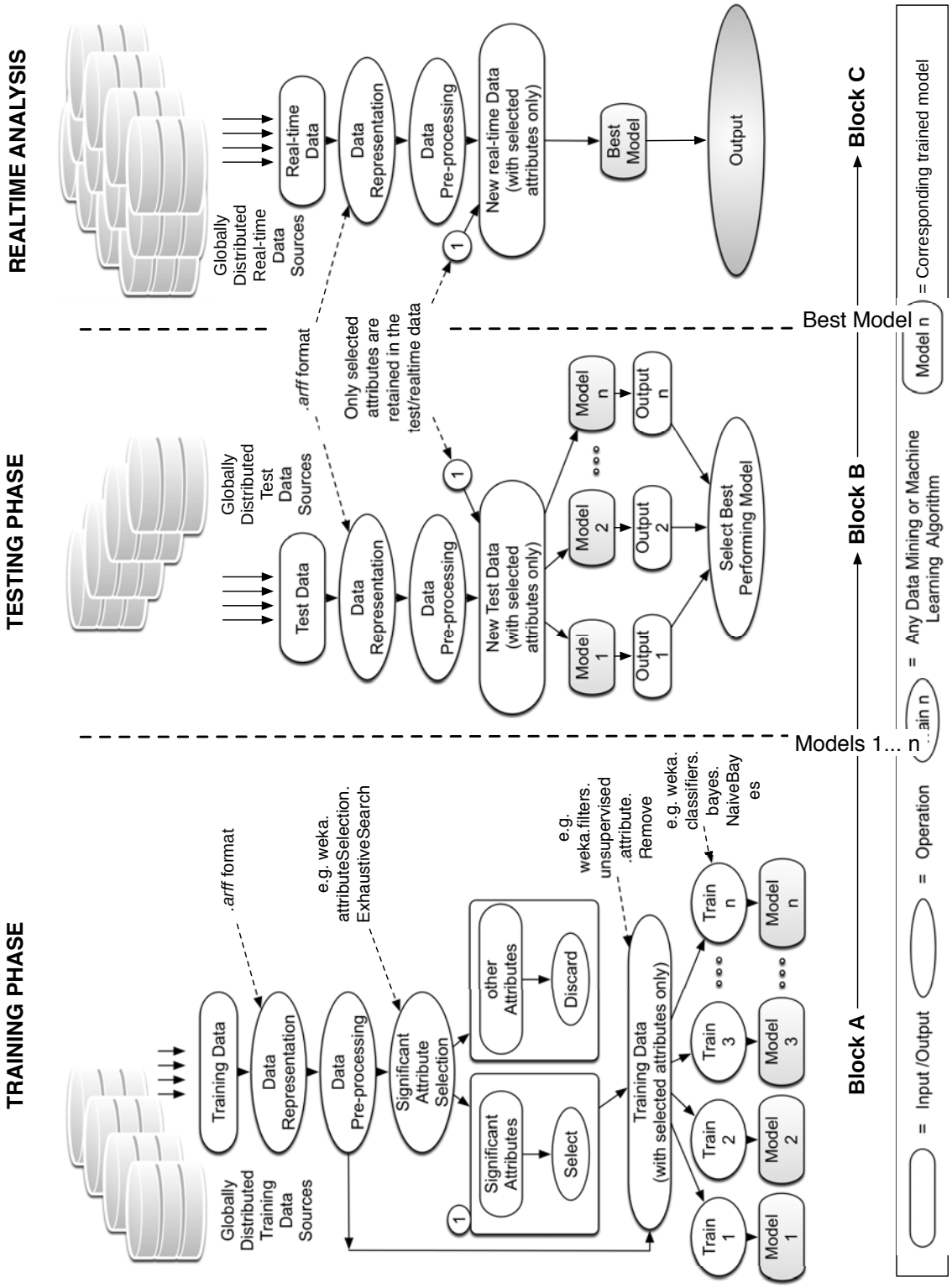



**Figure 4.2:** *Intrusion Detection Process*

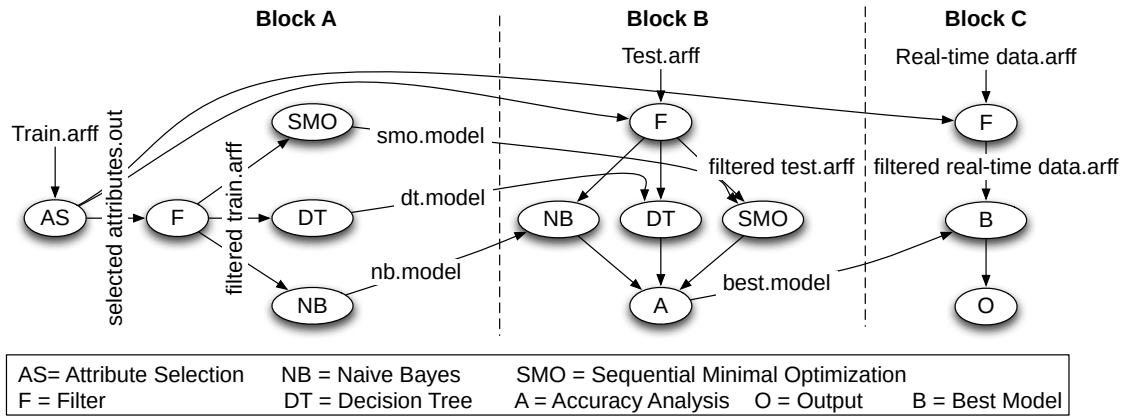



**Figure 4.3:** *Intrusion Detection workflow*

data, which can be the contents of IP packets, web server logs, etc.. Collected data are pre-processed (normalization, adding missing values, etc), represented in a format that is supported by the data mining tool (in our case it is *.arff* format), and pruned to contain a small set of features that are significant to improve the performance and accuracy of the system. The feature selection is the attribute selection (AS) in the figure. Applying these selected features on the training data is the Filter (F) process. Finally, with the reduced training data, we apply different algorithms to train corresponding models. In our experiments, we selected well known methods for data mining and intrusion detection such as Naive Bayes (NB), Decision Trees (DT) and Support Vector Machines (SVM). For SVM, we used Sequential minimal optimization (SMO), an algorithm for solving the optimization problem which arises during the training of support vector machine.

To evaluate the effectiveness of the trained models, we perform the testing on the test data (Block B in the figure). We repeat the same steps as in Block A on the test data, except the AS. We then use the trained model to generate output using the test data. Finally, we select the best performing model based on the accuracy of classification of individual models, denoted as (A) in the figure. This model, which is the most accurate, is then used on the real-time data from distributed data sources (Block C in the figure). Real-time data is filtered (F), then the best model applied (B) and the output (O) is generated.

The advantage of Naive Bayes and Decision Trees is that they are highly efficient and generally result in good classification. Support Vector Machines are high quality systems and have good classification accuracy. We must also remember that our objective is *not to discover the best model for intrusion detection* rather it is to minimize the cost of data transfer and computation when using data from distributed Cloud storage servers for any analysis of intrusion detection in Clouds.

The input data for Block A are the training data, Block B are the testing data and Block C are the real-time data. The output data from each task are labels. At the end of Block A's execution, three models (nb.model, dt.model & smo.model) files are created, which

then becomes input for Block B. Block B's execution generates of the most accurate model (*best.model*) as input for Block C. Block C then applies the model for the real-time data logs to obtain the intrusion classification.

**Table 4.1:** *Accuracy of intrusion detection using SMO*

| | | |
|---|---|---|
| Correctly Classified Instances | 407214 | 99.3115 % |
| Incorrectly Classified Instances | 2823 | 0.6885 % |
| Total Number of Instances | 410037 | |

In Table 4.1, we give the accuracy of the best performing model (SMO). Other methods (Naive Bayes & Decision Trees) had lower accuracy. We see that, using SMO, about 99% of instances are correctly classified. Hence, when this system is deployed in real-time environment, we can expect similar accuracy of classification.

In Table 4.2, we present the classification using the SMO with the help of a confusion matrix which lists the classification per class (number of correctly and incorrectly classified instances for each class of intrusion). For example, in row 3, we see that 3018 instances are correctly classified as probes while 371 probes are incorrectly classified as normal, 53 probes are incorrectly classified as Denial of Service (DoS), 0 probes are incorrectly classified as unauthorized access from a remote machine (R2L) and 0 probes are incorrectly classified as unauthorized access to root (U2R).

## 4.3 Cost Minimization using Non-Linear Programming Model

### 4.3.1 Notations and problem

We denote an application workflow using a Directed Acyclic Graph (DAG) by $G$=($V$, $E$), where $V$=$\{T_1, ..., T_n\}$ is the set of tasks, and $E$ represents the data dependencies between these tasks, that is, $tdata^k = (T_j, T_k) \in E$ is the data produced by $T_j$ and consumed by $T_k$.

We have a set of storage sites $S = \{1, ..., i\}$, a set of compute sites $P = \{1, ..., j\}$, and a set of tasks $T = \{1, ..., k\}$. We assume the 'average' computation time of a task $T_k$ on a compute resource $P_j$ for a certain size of input is known. Then, the cost of computation of a task on a compute host is inversely proportional to the time it takes for computation on that resource. We also assume the cost of unit data access $txcost_{i,j}$ from a storage resource

**Table 4.2:** *Classification of data using SMO model*

| a | b | c | d | e | classified as |
|---|---|---|---|---|---|
| 80255 | 43 | 12 | 85 | 4 | a = normal |
| 2083 | 323128 | 0 | 0 | 0 | b = DoS |
| 371 | 53 | 3018 | 0 | 0 | c = probe |
| 148 | 0 | 0 | 796 | 3 | d = R2L |
| 10 | 0 | 0 | 11 | 17 | e = U2R |

$S_i$ to a compute resource $P_j$ is known. The transfer cost is fixed by the service provider (e.g. Amazon CloudFront) or can be calculated according to the bandwidth between the sites. We assume that these costs are non-negative, symmetric, and satisfy the triangle inequality: that is, $txcost_{i,j} = txcost_{j,i}$ for all $i, j \in N$, and $txcost_{i,j} + txcost_{j,k} \geq txcost_{i,k}$ for all $i, j, k \in N$. These relations can be expressed as:

$$
\begin{aligned}
&ecost \propto 1/\{execution\ time\ or\ capability\ of\ resource\} \\
&txcost \propto bandwidth\ OR = (tx\ cost/unit\ data)/site \\
&total\ cost\ of\ computation: \\
&\quad C \leq ecost * etime + txcost * data + overheads
\end{aligned}
$$

The cost-optimization problem is: *Find a feasible set of 'partial' data-sets* $\{d^k_{i,j}\}$ *that must be transferred from storage host* $S_i$ *to compute host* $P_j$ *for each task* ($T_k \in V$) *such that the total retrieval cost and computation cost of the task on* $P_j$ *is minimal, for all the tasks in the workflow (not violating dependencies)* .

### 4.3.2 Non-linear model

Here, we try to get the minimum cost by formulating a non-linear program for the cost-optimization problem, as depicted in Figure 4.4. The formulation uses two variables $y, d$ and pre-computed values $txcost, ecost, txtime, etime$ as listed below:

- $y$ characterizes where each task is processed. $y^k_j = 1$ iff task $T_k$ is processed on processor $P_j$.
- $d$ characterizes the amount of data to be transferred to a site. e.g. $d^k_{i,j} = 50.2$ denotes 50.2 units of data are to be transferred from $S_i \Rightarrow P_j$ for task $T_k$.
- $txcost$ characterizes the cost of data transfer for a link per data unit. e.g. $txcost_{i,j} = 10$ denotes the cost of data transfer from $S_i \Rightarrow P_j$. It is added to the overall cost iff $d^k_{i,j} > 0$ & $y^k_j = 1$.
- $ecost$ characterizes the cost of computation (usage time) of a processor. e.g. $ecost_j = 1$ denotes the cost of using a processor $P_j$. It is added to the overall cost iff $y^k_j = 1$.
- $txtime$ characterizes the average time for transferring unit data between two sites. e.g. $txtime_{i,j} = 50$ denotes the time for transferring unit data from $S_i \Rightarrow P_j$. It is added to the Execution Time (ET) for every task iff $d^k_{i,j} > 0$ & $y^k_j = 1$.
- $etime$ characterizes the computation time of a task averaged over a set of known and dedicated resources. e.g. $etime^k_j = 20$ denotes the time for executing a task $T_k$ on a processor $P_j$. It is added to ET iff $y^k_j = 1$.

The constraints can be described as follows:

$$
\begin{array}{l}
\textit{Minimize total Cost } (C) \\
C = \sum_{i \in S, j \in P, k \in T} d_{i,j}^{k} * txcost_{i,j} * y_j^k + ecost_j * etime_j^k * y_j^k \\
\\
\textit{Subject to :} \\
\quad (a)\ \forall k \in T, j \in P \qquad y_j^k \geq 0 \\
\quad (b)\ \forall i \in S, j \in P, k \in T \quad d_{i,j}^{k} \geq 0 \\
\quad (c)\ \forall k \in T \qquad tdata^k \geq 0 \\
\quad (d)\ \forall i \in S, j \in P \qquad txcost_{i,j} \geq 0 \\
\quad (e)\ \forall i \in S, j \in P \qquad txtime_{i,j} \geq 0 \\
\quad (f)\ \forall k \in T, j \in P \qquad ecost_j \geq 0 \\
\quad (g)\ \forall k \in T, j \in P \qquad etime_j^k \geq 0 \\
\\
\quad (h)\ \sum_{j \in P} y_j^k = 1 \\
\quad (i)\ \sum_{i \in S, j \in P} y_j^k * d_{i,j}^{k} = tdata^k \\
\quad (j)\ \sum_{i \in S, j \in P, k \in T} y_j^k * d_{i,j}^{k} = \sum_{k \in T} tdata^k \\
\\
\hline \\
\textit{Execution time of task } k\ (ET^k) \\
ET^k = \sum_{i \in S, j \in P, k \in T} (d_{i,j}^{k} * txtime_{i,j} * y_j^k) + etime_j^k * y_j^k
\end{array}
$$

**Figure 4.4:** *NLP model*

- (a) & (h) ensure that each task $k \in T$ is computed only once at processor $j \in P$ when the variable $y_j^k > 0$. For partial values of $y_j^k$, we round up/down to the nearest integer (0 or 1). Tasks are not partitioned or migrated.
- (b) & (c) ensure that partial data transferred and total data required by a task cannot be negative.
- (d), (e), (f) and (g) ensure that cost and time values are all positive.
- (i), (a) & (b) ensure that partial-data are transferred only to the resource where a task is executed. For all such transfers, the sum of data transferred should equal to the data required by the task, which is $tdata^k$.
- (j) ensures that the total data transfer for all the tasks are bounded by the sum of data required by each task. This is important for the solvers to relate (h), (i) & (j),
- (i) & (j) combined ensure that whenever partial-data $d_{i,j}^k$ is transferred to a compute host $P_j$, then a compute host must have been selected at $j$ ($y_j^k = 1$), and that total

data transfer never exceeds the bound $tdata^k$ for each task and in total.

To get an absolute minimum cost, we map the tasks in the workflow onto resources based only on cost optimization (not time). This eliminates the time dependencies between tasks. However, the task to compute-resource mappings and data-source to compute-resource mappings minimizes the cost of execution *but not the makespan*. The execution time of a task ($ET^k$) is calculated based on the cost-minimized mappings given by the solver. The total: $\sum_{k \in T}(ET^k + waiting\_time)$ is the makespan of the workflow with the minimum cost, where the $waiting\_time$ denotes the minimum time a task has to wait before its parents finish execution.

## 4.4 Cost Minimization for The Intrusion Detection Application

In this section, we describe the method we used to solve the non-linear program formulated in Section 4.3. We then describe how we applied the solution for minimizing the total cost of execution to the intrusion detection application workflow.

**NLP-solver:** We wrote a program using the Modeling Language for Mathematical Programming (AMPL) [54] for solving our NLP-model. We used DONLP2[137], a non-linear program solver, to solve the model. The computation time of the solver to reach a solution (for a maximum of 2000 iterations) was less than 2 seconds, which is insignificant as compared to the data-transfer time in our experiments.

**Partial-data retrieval and task-to-resource mapping:** Based on the integer values of $y_j^k$ given by DONLP2, we statically mapped the tasks in the intrusion detection application workflow to each compute resource $P_j$. Data retrievals are also fixed for each ready task from each $S$ based on the value of $d_{i,j}^k$ and $y_j^k = 1$. The steps of mapping and data retrievals are given in Algorithm 4.4. The heuristic computes the values for task mapping $y_j^k$ and $d_{i,j}^k$ for all the tasks in the beginning according to the solution given by a NLP-solver. As all the tasks in the workflow are mapped initially, the *for* loop preserves the dependencies of the tasks by dispatching only the ready tasks to the resources. For dispatched tasks, partial data retrievals to the assigned compute resource occur from chosen resources. All child tasks wait for their parents to complete, after which they appear in the ready list for dispatching. The scheduling cycle completes after all the tasks are dispatched successfully. The output data of each completed task is staged back to the Cloud storage as part of the task's execution. The Cloud storage should ensure that the files are distributed to the edge-servers within certain time bound such that child tasks do not have to wait for availability of data longer than downloading directly from the Cloud's central server.

An example model, data and the program used for obtaining a solution to the NLP model is given in Appendix B.

**Algorithm 2** Scheduling heuristic using the NLP model

1: Compute $y_j^k$ & $d_{i,j}^k$ for all tasks by solving the NLP
2: **repeat**
3: Get all the 'ready' tasks in the workflow
4: **for** each task $t_k \in T_{ready}$ **do**
5: Assign $t_k$ to the compute resource $P$ for which $y_j^k = 1$
6: Fix partial data transfers $d_{i,j}^k$ from $S_i$ to the compute resource $P_j$ for which $y_j^k = 1$
7: **end for**
8: Dispatch all the mapped tasks for execution
9: Wait for *POLLING_TIME*
10: Update the ready task list
11: (Upload output files of completed tasks to the storage central for distribution)
12: **until** there are unscheduled tasks in the ready list

## 4.5 Experimental Setup

In this Section, we describe Intrusion Detection data and tools, the experimental setup, and the results.

### 4.5.1 Intrusion detection application data and tools

**Data:** For our experiments, we used part of the benchmark KDD'99 intrusion data set[3]. This database contains a standard set of data to be audited, which includes a wide variety of intrusions simulated in a military network environment. We use 10 percent of the total training data and 10 percent of the test data (with corrected labels), which are provided separately. Each record in the data set represents a connection between two IP addresses, starting and ending at defined times and protocol. Furthermore, every record is represented by 41 different features. Each record represents a separate connection and is hence considered to be independent of any other record. Training data are either labeled as normal or as one of the 24 different types of attack. These 24 attacks can be grouped into four classes; Probing, Denial of Service (DoS), unauthorized access from a remote machine (R2L) and unauthorized access to root (U2R). Similarly, test data are also labeled as either normal or as one of the attacks belonging to the four attack groups.

To perform data mining we used algorithms implemented in an open source WEKA library [154]. We used three types of probabilistic classification models: Naive Bayes, decision tree and Sequential Minimal Optimization (SMO), from the WEKA library. The commands are depicted in Figure 4.5. The number of log-data analysis for detecting intrusion varies depending on the characteristics of the log data. To reflect all types of scenarios, we perform the real-time log-data analysis for 10 times. We interpolate the cost for 10,000 times execution by multiplying the cost of 10 executions multiplied by 1000.

[3]http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

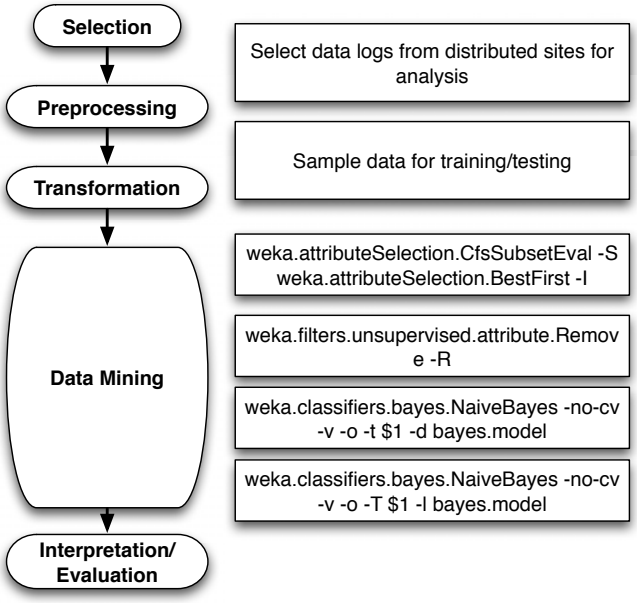



**Figure 4.5:** *Data mining using WEKA library*

The total data used by the intrusion detection workflow (Figure 4.3) is divided into 30MB, 60MB, 90MB and 120MB. This was be achieved by filtering the training, testing and real-time data by random sampling.

All data mining was performed using WEKA classes as described in Figure 4.2. Weka, is one of the most commonly used tool for performing data mining and machine learning tasks. The advantage of using Weka is that it implements a large number of mining algorithms and can be rapidly used to compare different methods. It can be used either as a command line tool or with the GUI to perform standard machine learning tasks such as data preparation, data pre-processing, feature or attribute selection, training and testing and evaluation of results. Hence, we used this tool as described in Figure 4.2.

### 4.5.2 Middleware and tools

We used Workflow Engine [102] for scheduling and managing workflow executions on Cloud resources. The engine together with the application broker are part of the Cloudbus Toolkit [25]. We use the scheduling heuristic listed in Algorithm 4.4. Both, multi-site partial downloads and CloudFront downloads were carried out over HTTP using *JPartialDownloader* tool[4]. HTTP/1.1 range requests allow a client to request portions of a resource.

[4]http://jpd.sourceforge.net/

### 4.5.3 Distributed compute and storage resources

For selecting the nearest storage location of a file relative to a compute resource, we use the functionality of Amazon CloudFront. CloudFront fetches data to a compute resource from the nearest edge-server. The data transfer cost (per GB) from the edge locations is presented in Table 4.3. The data transfer cost (DTx cost) from the CloudFront to the execution sites is based on the edge location through which the content is served. We assume the data transfer cost to and from a storage location to be equal in all our experiments. This simplifies the model for the selection of storage sites for partial data retrievals and data upload. For partial data retrievals, all the resources listed in Table 4.3 also served as storage resources. For our experiments, we ignored the data storage cost on Clouds, which could easily be added to the overall execution cost as a constant (e.g. $0.150 per GB for the first 50 TB / month of storage used[5]).

We used compute resources from US, Europe and Asia as listed in Table 4.3. The execution cost (Ex cost) on each CPU is calculated based on the number of cores (cost is similar to Amazon EC2 instances) available.

## 4.6 Analysis

We now present results obtained by executing the intrusion detection application workflow using globally distributed resources as listed in Table 4.3.

### 4.6.1 Experiment objectives

We conduct the following two classes of experiments:

1. Measure total cost when using commercial Cloud as content distribution and publicly available compute resources for execution ($ecost_j = 0, txcost_{i,j} > 0$).

2. Measure total cost of execution when using commercial Cloud for content storage, distribution and execution ($e_c ost > 0, txcost_{i,j} > 0$)

The first experiment (subsection 4.6.2) measures the cost of data transfer if Cloud resources were used only for data distribution and tasks executed on publicly available compute resources. In this scenario, the compute resources in Table 4.3 served both as storage (mimicking distributed Cloud storage) and compute resources. We use a solution to our model for determining quantity of partial data transfers from the distributed storage such that the transfer cost is minimized. The tasks are mapped to the compute resources such that the partial transfers have minimum cost.

The second experiment (subsection 4.6.2) measures the cost of executing the application on Cloud resources, with non-zero data transfer and computation costs. In this

[5]http://aws.amazon.com/s3/#pricing

scenario, our model gives a solution for minimizing both partial data transfers and computation costs, with tasks mapped to resources accordingly. Here too, the compute-servers in Table 4.3 serve as distributed Cloud storage and compute resources.

We compare the costs obtained from each of the above experiments against the cost incurred when using data-transfers from nearest (with respect to the compute resource where the task is assigned) Cloud storage resource. We measure the total cost incurred for transferring data from nearest location by making compute-resource cost: zero (relating to publicly available resources) and non-zero (relating to commercial Cloud resources), consecutively.

We finally compare the cost savings when using NLP based task+data resource selection against CloudFront's data resource selection.

### 4.6.2 Results

The results obtained are an average of 15 executions. The cost values in Figures 4.6 and 4.7 are for executing a single instance of the intrusion detection workflow. The cost values in Figure 4.8 are the result of executing the workflow 10,000 times (the cost of 10 executions multiplied by 1000).

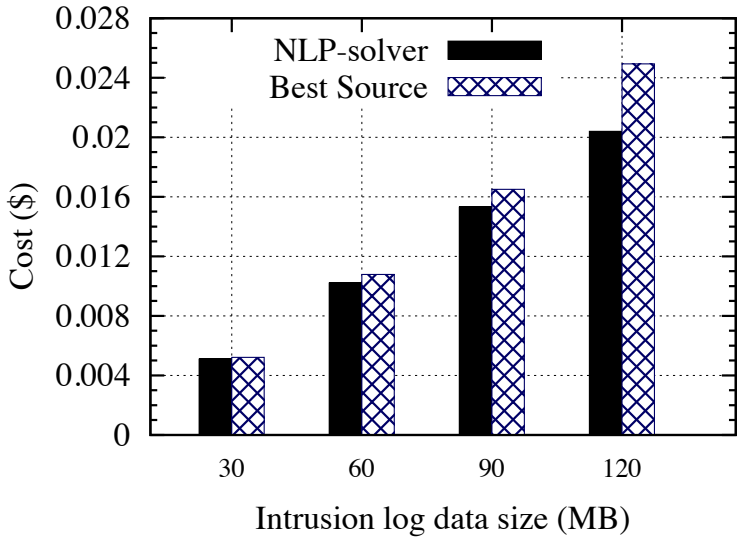



**Figure 4.6:** *Comparison of transfer cost with no execution cost.*

**Scenario 1: Data in Cloud and execution on public compute resources**

Figure 4.6 compares the cost of transferring data to compute resources between NLP-solver based source selection and single source selection given by CloudFront. We set the execution cost to zero for comparing only the transfer cost. The results show that the total data transfer cost is minimized when using NLP-solver based storage host selection for all

size of data. As the size of data increases from 30MB to 120MB, the benefit of transferring data using NLP compared with CloudFront increases. For the total size of 120MB data, using the CloudFront would cost $0.025, whereas using NLP the cost decreases to $0.020. The difference in cost is huge for large experiments, as analyzed later in subsection 4.6.2.

The reason for the decrease in cost is NLP-solver transfers partial data in proportion to the cost of communication, as the data transfer cost is divided among all the cheapest links. CloudFront selects the a single best source for data transfer. Transferring data using CloudFront becomes more expensive as the size of these data increases.

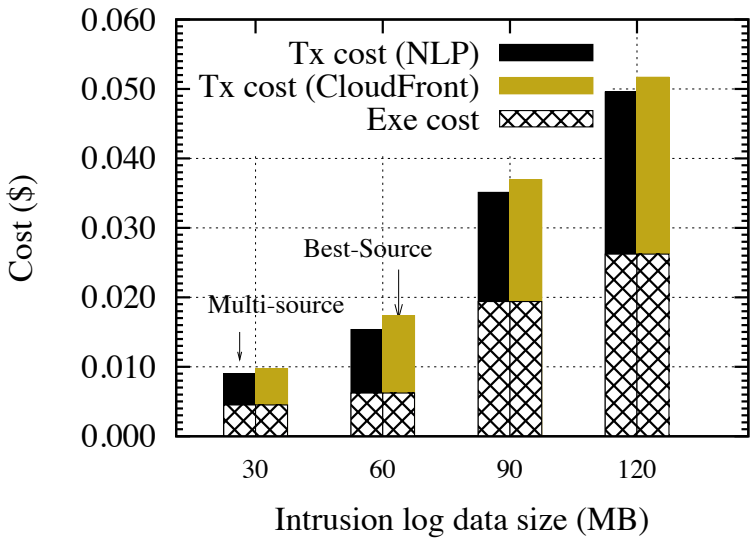



**Figure 4.7:** *Comparison of total cost when both computation and transfer costs are non-zero.*

**Scenario 2: Data and execution on Cloud resources**

Figure 4.7 depicts the total cost of executing the intrusion detection workflow on Cloud resources when using NLP-solver based task-resource mapping and (a) NLP-solver based data source selection (labelled as NLP in the figure), (b) CloudFront based data source selection (labeled as CloudFront in the figure). The NLP based task-resource mapping was used to make a fair comparison on data transfer cost between our approach and CloudFront. In this case, the NLP-model embeds the minimization of both the execution costs and data transfer costs into one objective function to be minimized, as listed in Figure 4.4. As two costs were involved, the total cost increased when compared to only the data transfer cost depicted in Figure 4.6. Nevertheless, partial data transfers based on NLP-based data source selection incurred the minimal cost for all range of data sizes.

Even when the task-resource mapping was based on NLP, the total cost savings for 120MB of data processed was $0.02 on average for 1 execution. If both task-resource mapping and data retrievals were based on existing heuristics (earliest finish time for

**Table 4.3:** *Distributed compute resources used for evaluating NLP model based scheduling heuristic*

| Physical Compute Nodes | cores | Ex cost $/hr | DTx cost $/GB | Nearest Region |
|---|---|---|---|---|
| rotegg.dps.uibk.ac.at | 1 | $0.10 | $0.170 | Europe |
| aquila.dacya.ucm.es | 1 | $0.10 | $0.170 | Europe |
| tsukuba000.intrigger.omni.hpcc.jp | 8 | $0.80 | $0.221 | Japan |
| omii2.crown-grid.org | 4 | $0.40 | $0.210 | China |
| snowball.cs.gsu.edu | 8 | $0.80 | $0.170 | US |
| node00.cs.binghamton.edu | 4 | $0.40 | $0.170 | US |
| belle.csse.unimelb.edu.au | 4 | $0.40 | $0.221 | Japan |
| manjra.csse.unimelb.edu.au | 4 | $0.40 | $0.221 | Japan |

compute and best resource for data), our approach would have had more savings.

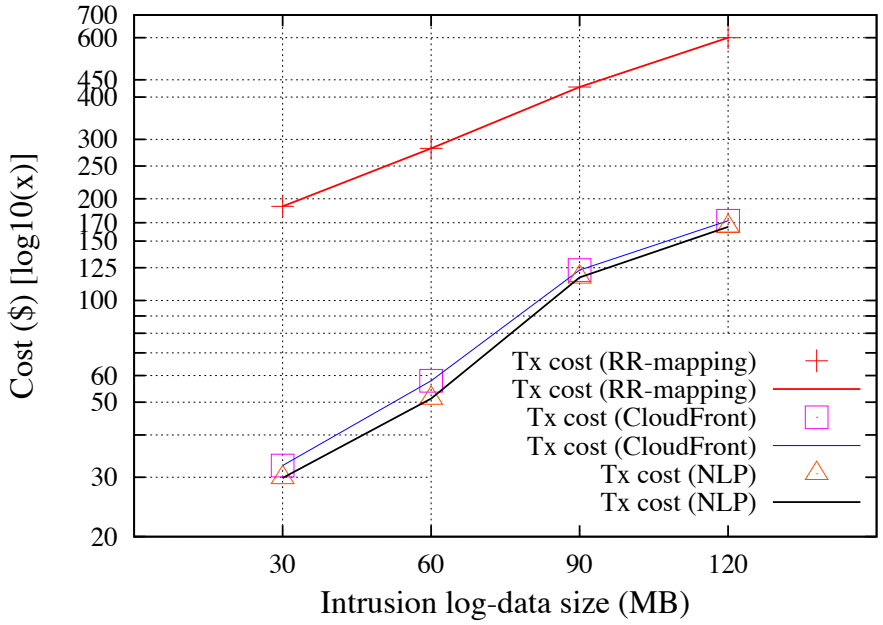



**Figure 4.8:** *Comparison of total execution cost between NLP based mappings, with and without CloudFront (round-robin based mapping is used for comparison of the upper-bound only).*

### Total cost savings

Figure 4.8 depicts the cost of executing the real-time analysis section, depicted as Block C in Figure 4.3), 10,000 times (Blocks A and B are usually computed only once for each set of data). The cost values for each data group were derived from the cost of 10 executions multiplied by 1000. The most costly approach was when using round-robin based task-resource mapping algorithm and nearest source data retrievals. This value should be interpreted as an upper bound for comparison purposes only. This cost was reduced by 77.8% ($466.8) when we used the NLP-solver based mapping and multi-source partial

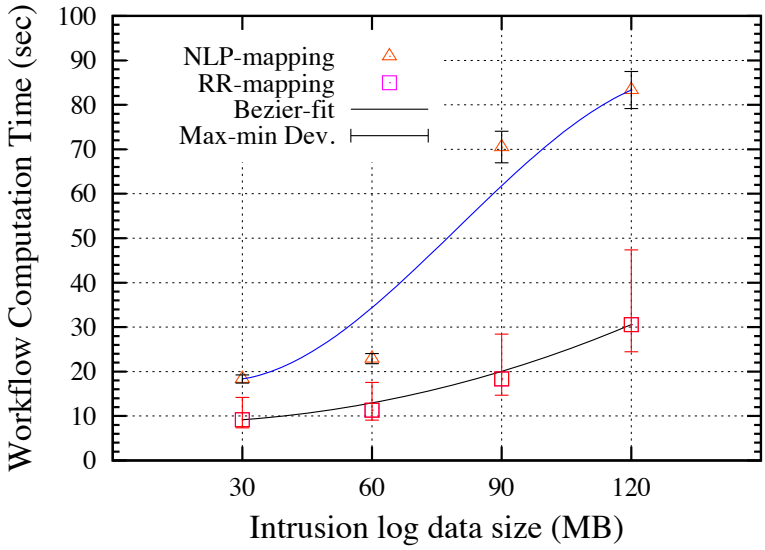



**Figure 4.9:** *Comparison of execution time between NLP based mapping and Round-Robin based mapping.*

data retrieval; and by 76.2% ($457.2) when we used NLP-solver based mapping and data retrieval from CloudFront's best source. This would amount to savings of three-quarters of the total expenditure if intrusion detection systems were to be executed on Clouds using our model. When the costs obtained by using NLP based approach was compared to CloudFront's, NLP was able to reduce the cost by $7.1 on average. This cost savings would cumulate to be higher for larger data and repeated experiments. Thus, for all scenarios, the total cost incurred when using NLP-solver is lower than the cost incurred when using Amazon's CloudFront based data retrieval.

**Workflow computation time:** We measured the time taken for computing (excluding data transfer time) the workflow under two scenarios:(a) when cost is minimized using our NLP-model, and (b) when time is minimized by a simple round-robin based selection of resources, depicted in Figure 4.9. When compared against a simple task mapping algorithm such as round-robin, NLP-model based heuristic takes additional time, which increases as the size of the data-set increases as evident from Figure 4.9. Figure 4.9 also depicts the maximum and minimum values of execution time for various data sizes. When the compute resource list was randomized, the deviation for the NLP-mapping (mainly due to CPU load) was lower than the RR-mapping (change in type of CPU). The computation time can be reduced by increasing the number of compute resources used, while still using Clouds for data storage and distribution.

We tabulated the cost of computation and data transfer according to Amazon's current pricing policy in Table 4.3. The highest computation cost of Amazon Cloud resources

is more than the highest data transfer cost[6]. Armbrust et al. [7] have compared the normalized cost of computing resources and WAN bandwidth between 2008 and 2003. Their data clearly shows that the cost/performance improvement is 2.7 times and 16 times for WAN bandwidth and CPU hours, respectively. This trend hints to the fact that data transfer costs are not decreasing as much as computation cost. Hence, for data intensive applications, total cost savings on communication is a necessity as compared to computation cost.

## 4.7 Related Work

Armbrust et al. [7] described the benefits of moving to Cloud computing. These benefits include lower operating costs, physical space savings, energy savings and increased availability.

Deelman et al. [39] presented a case study for examining the cost-performance trade-offs of different workflow execution modes and provisioning plans for Cloud resources. They concluded that data intensive applications can be executed cost-effectively using Cloud computing infrastructure. In contrast to their work, we focus on the minimization of communication cost using globally distributed Cloud edge-servers and compute nodes.

Amazon CloudFront uses edge locations in United States, Europe, and Asia to cache copies of the content for faster delivery to end users. It provides users address in the form of a HTTP/HTTPS uniform resource locator (URL) . When a user requests one of these data from any site, Amazon CloudFront decides which edge location is 'best' able to serve the request to that user's location. However, users do not have control over the amount of data to get from each edge servers, to minimize cost, unless they access the URL from a different geographic location. We compare our approach with Amazon's 'best' location approach.

Wu et al. [158] presented the design and implementation of Collaborative Intrusion Detection System (CIDS) for efficient intrusion detection in a distributed system. They used a manager framework for aggregating alarms from different detectors to provide a combined alarm for an intrusion. They claim that aggregate information is more accurate than elementary data for intrusion detection. Zeller et al. [174] presented the advantages of using Cloud computing for data mining applications, especially when the size of data is huge and globally distributed. These application scenarios add to the necessity for executing intrusion detection algorithms at multiple locations, which in turn produces data at multiple computing sites facilitating data retrieval and scheduling algorithms such as the one presented in this chapter.

Broberg et al. [20] introduced MetaCDN, which uses 'Storage Cloud' resources to deliver content to content creators at low cost but with high performance (in terms of

[6]assuming transferring 1GB from Amazon takes 1 hour of CPU time

throughput and response time). This work also adds credibility to our assumption that data can be indeed replicated and hence downloaded from multiple locations, including from storage clouds. MetaCDN presented the delivery mechanism but did not address the cost optimization problem for application or the client side. This chapter presented one algorithm to minimize the cost of distributed data retrieval.

Microsoft has a Windows Workflow Foundation for defining, managing and executing workflow as part of its .NET services. With the .NET services, workflows can be hosted on Clouds for users to access it from anywhere [95]. The service facilitates transparent scalability for persistence stores and distribution of load between hosts. At the time of writing this thesis, their services did not address cost optimization. Hence, this chapter addresses the cost optimization problem by modeling the problem and solving it.

A number of work in Grid computing, especially those related to Data Grids, have focused on optimal selection of data sources while scheduling applications [151; 93]. Also, some existing workflow systems [46; 56; 91] use a variety of optimization metrics such as the execution time, efficiency, economical cost, or any user-defined QoS parameter for scheduling workflow applications. In Grids, users were not able to provision required type of resources at specified locations as demanded by applications. In Clouds, however, users can first choose the set of compute and storage resources they want for their application and then use our model for minimizing the total cost. The initial selection may be based on user's budget allocated for executing the application in Clouds.

## 4.8 Conclusions

In this chapter, we presented the execution of an intrusion detection application workflow using Cloud resources, with an objective of minimizing the total execution cost. We modeled the cost minimization problem and solved it using a non-linear program solver. Based on the solution, we retrieved data from multiple data sources to the compute resource where a task was mapped, unlike previous approaches, where data was retrieved from the 'best' data source. Using our NLP-model we achieved savings of three-quarters of the total cost as compared to using CloudFront's 'best' data source selection, when retrieving data.

We conclude that by dividing data retrievals to distributed data centers or storage Clouds in proportion to their access cost (as laid out in our model), users can achieve significant cost savings than when using existing techniques.

The NLP produced near optimal results in terms of task to resource mapping and multi source data retrievals. However, the computation time for obtaining the near optimal result increased as we increased the size of input data. NLP model would be highly suitable for moderate size mapping problems, where the comparisons as a result of fewer resources and data files. It would not be practically feasible for a large

problem, where number and size of data files are significantly larger. In the latter case, the computation time of NLP would significantly outweigh the benefits of obtaining near optimal schedules. In order to avoid such problems, we propose heuristics based algorithms in the following chapters where performance and optimality is balanced for both small and large sized problems.

# 5

# Static and Dynamic Heuristics-Based Scheduling Algorithms

Many large-scale scientific experiments collaborate with researchers and laboratories located around the world so that they can leverage high-tech infrastructures present at those locations and collectively perform experiments quicker. Data produced by these experiments are thus replicated and cached inadvertently 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 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 and incorporate this in our scheduling heuristic. In this chapter, we propose 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. Hence, with a combination of data retrieval and task-resource mapping technique, we show that our heuristic can achieve time-efficient schedules that are better than existing heuristic based techniques, for scheduling application workflows.

## 5.1 Introduction

Selecting compute and storage nodes based on their location in the network is a basic building block for many distributed systems [156]. Applications that require only a handful of resources and storage space, are not significantly affected in terms of performance by the locality of nodes. However, for large scale scientific experiments such as the Compact Muon Solenoid (CMS) experiment for the Large Hadron Collider (LHC) at CERN,

the Laser Interferometer Gravitational-Wave Observatory's (LIGO) science runs, projects at Grid Physics Network etc., selecting compute and data sources based on locality is critical to their performance.

Multiple data sources are created at various locations around the world as a result of scientists carrying out repeated experiments. 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 [146], Giggle framework [32] 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 [48; 178].

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 chapter. 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 chapter, we present two scheduling heuristic that leverage multi-source parallel data-retrieval techniques. We experiment with existing (a) probe-based[178], (b) greedy [178], 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.

## 5.2 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.

### 5.2.1 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. [19].

**Algorithm 3** Enhanced Static Mapping Heuristics (ESMH)

1: **for** each task starting from the root **do**
2: Form resource set $\{R\}$ that has $min(tr(\{f_i\}, r_i \in R))$ 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: **end if**
10: **end for**
11: **if** $t_i$ not assigned **then**
12: Mark $R_m \notin \{R\}$ that has minimum $tr$ time for the largest file required by task $t_i$
13: $EFT_{file} = min\_avg\_comp(t_i, R_m) + tr(\{f\}_{ti}, R_m)$
14: **if** $[EFT_{file}] \leq [EFT_{min\_avg}]$ **then**
15: Map $t_i$ to $R_m$
16: **else**Map $t_i$ to $R_k$ /* last option */
17: **end if**
18: **end if**
19: Update resource availability information based on the mapping of tasks
20: **end for**

**Heuristics:** Algorithm 3 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) $min(tr(\{f_i\}, r_i \in \{R\}))$ 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

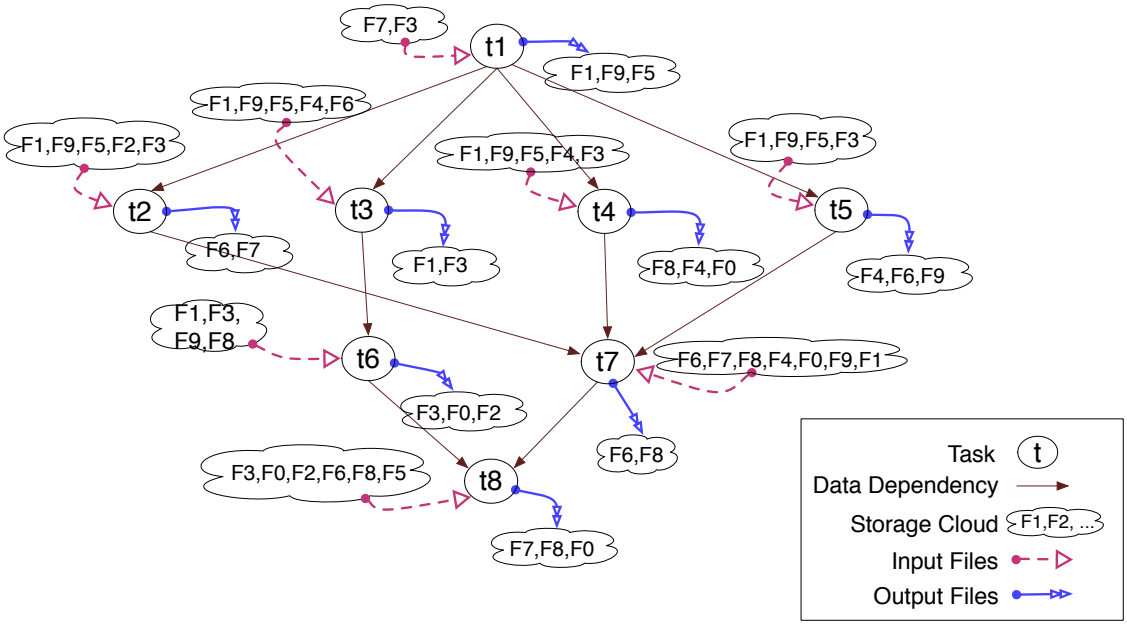
($EFT_{min\_avg}$) by adding the average of the minimum Execution Time (ET) given a resource for the task and the transfer time of all input files required by the task to that resource. This EFT value is an average value since we take the average of all the minimum computation time (*min_avg_comp*) of tasks on that resource. We assume the minimum computation time of every task varies on a resource as the size of input files vary.

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.




a) Task execution time on resources

| Tasks | R1 | R2 | R3 |
|---|---|---|---|
| 1 | 8 | 23 | 40 |
| 2 | 8 | 41 | 43 |
| 3 | 12 | 4 | 21 |
| 4 | 29 | 39 | 49 |
| 5 | 35 | 8 | 22 |
| 6 | 2 | 27 | 16 |
| 7 | 26 | 43 | 29 |
| 8 | 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 |

| Tasks | R1 | R2 | R3 | start | finish |
|---|---|---|---|---|---|
| 1 | - | 680 | - | 0 | 680 |
| 2 | 384 | - | - | 680 | 1064 |
| 3 | - | - | 4224 | 680 | 4904 |
| 4 | 4266 | - | - | 1064 | 5330 |
| 5 | 510 | - | - | 5330 | 5840 |
| 6 | 8698 | - | - | 5850 | 14538 |
| 7 | - | 12145 | - | 5840 | 17985 |
| 8 | - | 20833 | - | 17985 | 38818 |

**Makespan = 38818**

c) A schedule generated by using ESMH

**FIGURE 5.1:** *An example workflow, matrices with estimated values, and a schedule generated by ESMH.*

**Example Workflow**

Figure 5.1 shows an example of a workflow with input and output files and data dependencies between tasks. This workflow resembles a search for periodic Gravitational Waves (GW) from Sco X-1, led by Dr. Melatos at The University of Melbourne. Sco X-1 is the low-mass X-ray binary with the strongest X-ray emission. In Sco X-1, strong X-ray bursts occur frequently, and GWs can be emitted along with the X-ray bursts. Gravitational waves are ripples thought to occur in the fabric of space-time that result from interstellar collisions, explosions, or movement of large and extremely dense objects such as neutron stars. Those ripples can then pass through the space-time that Earth occupies, causing a distortion which Advanced LIGO is meant to pick up. Currently, several interferometric Gravitational Wave detectors around the world such as LIGO, VIRGO, GEO600, TAMA300 have been collecting data that could then be used by scientists for searching GWs.

The size and quantity of data produced by the workflow depicted in Figure 5.1 are substantive. For e.g. 10 days of fake data created with *Makefakedata* (using LAL software suite), produces files of 142 KB (each 1800 second Short-time Fourier Transforms in time) and there are 480 of such in a continuous 10 day stretch of data for a single source. This amounts to over 66MB. The *ComputeFStatistic* and *CombSearch*, which are processes under the LIGO search, each produces over 77MB data after processing the data created from *Makefakedata*. Depending on the input parameters, the plot resulting after plotting the points (the task *t8*) also needs further processing to produce an image file (e.g. png, eps, etc.) for visualization. The processing time taken by *ComputeFStatistic* and *CombSearch* depends on the stretch of data, starting frequency, and band of search. Based on this scenario, the matrices in Figure 5.1 list the execution and data retrieval times for the example application.

The table in Figure 5.1c shows the schedule length produced by using ESMH listed in Algorithm 3. 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 5.1a) and b) one that stores values of average transfer time of each data-file from distributed data-centers to each resource (Figure 5.1c) 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.

**Limitations of Static Heuristic**

The static heuristic we have proposed would be most appropriate if all of our assumptions listed in sub-section 5.2.1 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 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 3. To circumvent this limitation, we propose a Steiner Tree based resource selection and apply it for online scheduling, as described in the following section.

### 5.2.2 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 $\rho$.

Let $(X, \rho)$ 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 $(X, \rho)$ 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 $(X, \rho)$ as the total length of $G$:

$$L(G) = L(X, \rho)(G) = \sum_{vv' \in E} \rho(v, v'). \tag{5.1}$$

Thus, the SMT is a tree that connects vertices in $V$ with additional vertices $V_A$ to lower the path length $\rho$. 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 $\rho$ is in metric space highly simplifies the problem of constructing the tree and hence selecting vertices as compared to when using $\rho$ in a non-metric space.

**Forming a Steiner Tree**

The SMT problem is NP-hard, so polynomial-time heuristics are desired [41]. 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 [126]. In this chapter, we use the *incremental optimization heuristic* developed by Dreyer et al. [41]. Even though it does not give an optimal solution, we get a feasible solution at any point in time [1].

The time complexity of constructing SMT with minimal length for a finite set of points $N$ in the metric space $(X, \rho)$ 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:

$$\rho(v, v') = w_1 d(p) + w_2 c(p) \tag{5.2}$$

The weights $w_1, w_2$ are considered as a measure of the significance of each objective in the distance function of Equation 5.2. 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.

The primary objective of constructing a Steiner Tree is to identify neighbors connected to a node and not to find shortest paths between nodes. Finding shortest path in-terms of network distance and latency in a distributed setting is a difficult and time consuming task, the results of which may not be valid for long running applications. However, for short intervals, *traceroute*, which is a network tool for measuring the route path and transit times of packets across an Internet Protocol network, has been used. In the next section,

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

we show that there is a high degree of similarity between nodes' geographic locations and their connectivity with the practical deployment of network clusters in the real world.

### 5.2.3 Steiner Tree Based Resource Selection and Multi-source Data retrieval

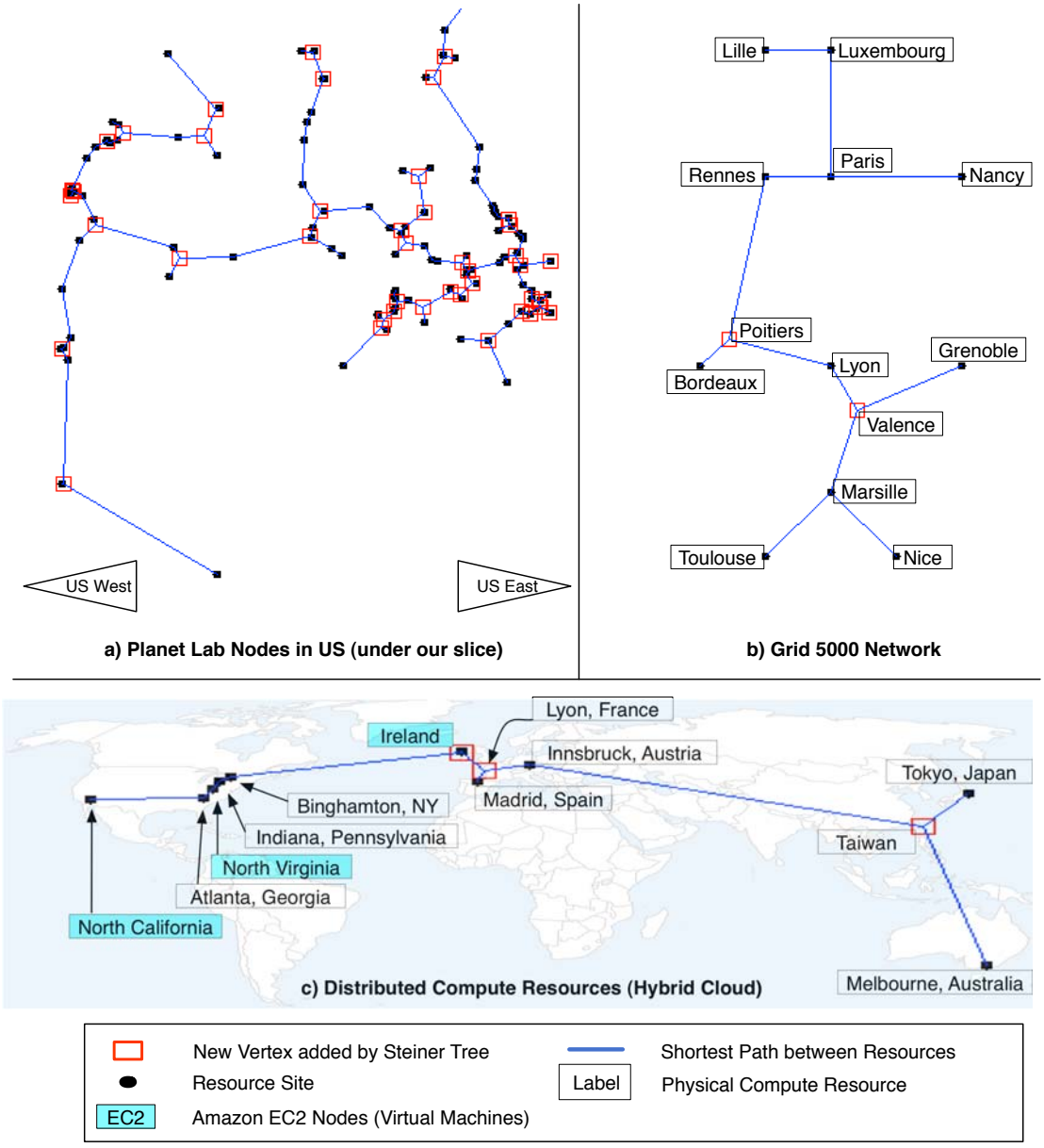



**FIGURE 5.2:** *A Steiner Tree constructed for PlanetLab nodes, Grid'5000 network and distributed resources available for our experiment*

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 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: PlanetLab nodes in US under our slice, the Grid'5000 network, and a distributed compute resources similar to CMS resources in Figure 1.2. These trees are shown in Figure 5.2. 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 5.2(c) when conducting real experiments, as described further in Section 5.3.

In Figure 5.2, 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:

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

In the PlanetLab 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.

### 5.2.4 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 5.2.2. The EDMH is listed in Algorithm 4.

**Algorithm 4** Enhanced Dynamic Mapping Heuristic (EDMH)

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 neighbor $r_i^{neighbor}$ of $r_i$ for calculating instantaneous bandwidth (max of $in_d$ probes)
7: Split input files based on probe values: $\{f^{split-1}, \cdots, f^{split-in_d}\}_{ti} \in \{f\}_{ti}$
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: **end if**
13: **end for**
14: Assign $t_i$ to the $minCR \Rightarrow$ the $r_i$ that gives minimum $ST$
15: Wait for *polling_time*
16: Update the ready task list
17: Distribute output data of task $t_i$ to resources that host the files required by successors of $t_i$
18: **end for**

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 $(X, \rho)$[2].

**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

[2]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 $L(G)$

task, as tasks are scheduled and get completed.

Initially, every ready task's Start Time (ST) is set to a high value (e.g. $\infty$). 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.

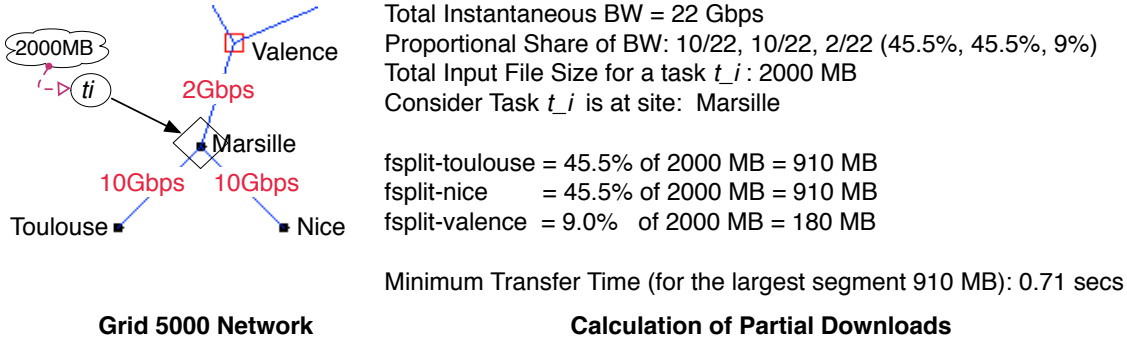



**Figure 5.3:** *An example showing the partitioning of input files in proportion to the available instantaneous bandwidth.*

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 5.3, 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 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 5.2.2).

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 [113].

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.

## 5.3 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.

### 5.3.1 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.

### 5.3.2 Application Workflows

We used three types of workflows: pipelined, complex and hybrid, as depicted in Figure 5.4. Figure 5.4-Montage depicts a workflow similar to the Montage Workflow [66]. 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 5.4-LIGO

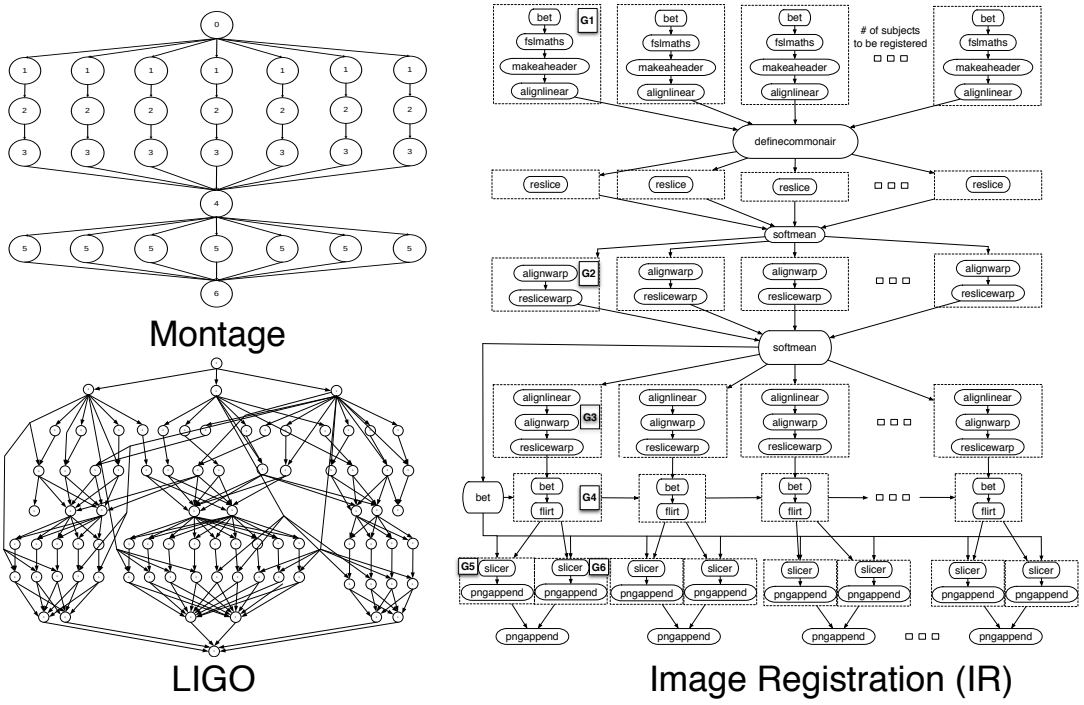



**FIGURE 5.4:** *Application Workflow types: Balanced (Montage), complex (LIGO) and hybrid (IR) workflows*

represents a complex workflow similar to a subset of the workflow used to analyze data from the second LIGO science run for binary inspirals [141]. In this type of workflows, tasks cannot be partitioned into levels easily as they have linked (data dependencies) to/from tasks across levels. Figure 5.4-IR represents a real application workflow used for Image Registration (IR) for fMRI applications [102]. 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 5.1). In this chapter, we label the Montage, LIGO and IR workflows as *WF 1*, *WF 2*, and *WF 3*, respectively.

For the IR workflow depicted in Figure 5.4, we recorded estimates of execution time and data transfer time of each task on compute resources provided by Grid'5000. This work is also explained in Chapter 3 and Appendix A, where we focus on IR experiment [102] 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. However, using random execution times and data sizes may break the symmetry of execution times of the Montage workflow, still keeping the dependency hierarchy intact.

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. Thus, by choosing three different types of workflows, we are interested in measuring the multi-source data retrieval times and their relationship to task execution times at compute resources.

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.

### 5.3.3 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 5.2.4).

### 5.3.4 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.

#### Emulation Setup

We used NS-2[3] as the emulation tool. For simulating the network connecting resources, we constructed a dense network topology by interconnecting *ns* nodes. As an emulator,

[3]http://www.isi.edu/nsnam/ns

we injected workflow execution traffic into the simulator and emitted packets on to live network using NS-2's real-time scheduler. Figure 5.5 depicts the architecture of our emulation platform. The virtual network is connecting a set of User-mode Linux (UML [63]) 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 [90].

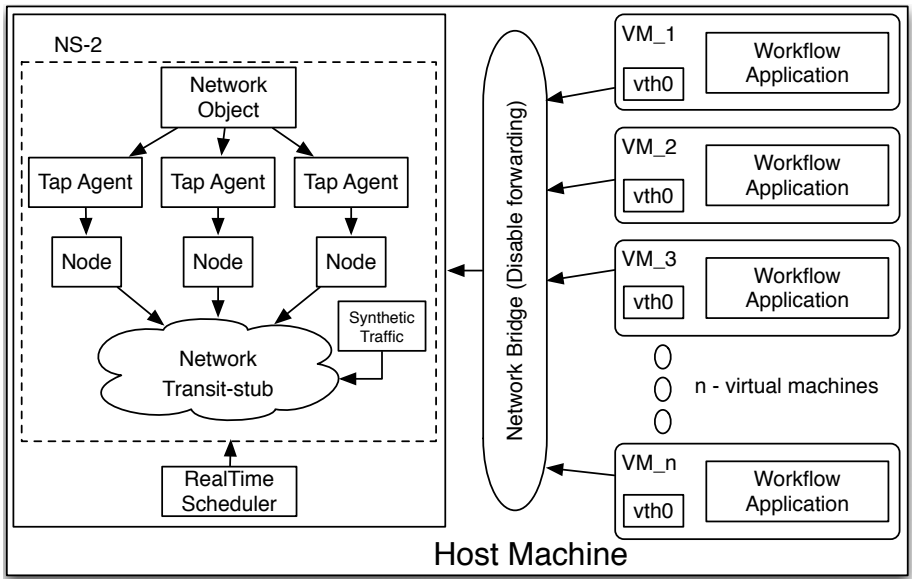



**Figure 5.5:** *Experiment design using NS-2 based emulation*

Cappello et al. [114] 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) [102; 166] as a workflow execution and scheduling engine for executing the workflows depicted in Figure 5.4. 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.

By using virtual machines alone, we would not be able to control the traffic flow between the VMs. In order to maintain traffic and noise levels according to the traffic and noise model of our choice, we needed a platform that would connect these VMs and

enable us to emulate the network. NS-2 provided us the right kind of environment for this purpose.

### Network Topology

We used the GT-ITM[4] internetwork topology generator to generate random graphs. 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 [173] 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.

| | 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 5.6:** *A table summarizing parameters used in the NS-2 based emulation*

### Storage and Compute Resources

Modeling of storage resources is a challenge. However, emerging technologies like Cloud storage systems, as mentioned in Section 5.4, are addressing storage services from a higher level. Since we are not concerned about sub-millisecond 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.

[4]http://www.cc.gatech.edu/projects/gtitm

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 [114]. 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 5.6 summarizes all the parameters used for our emulation.

**Experimental Results**

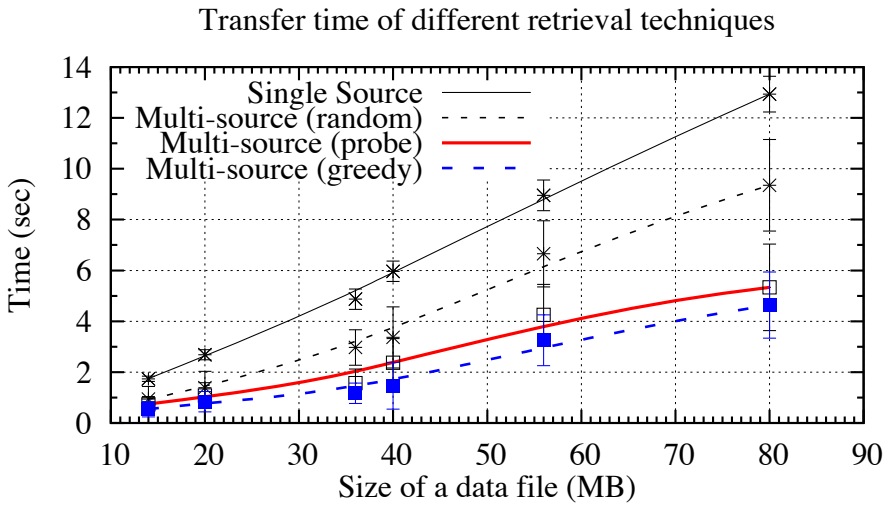



**Figure 5.7:** *Comparison of transfer time and error margins between single-source and multi-source data retrieval techniques using data files of different sizes (files are from IR Workflow experiment only) (Bézier curve fitting used)*

**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 depicted in Figure 5.7. The average data transfer times obtained using these techniques were close to the results listed by Zhou et al.[178]. 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.

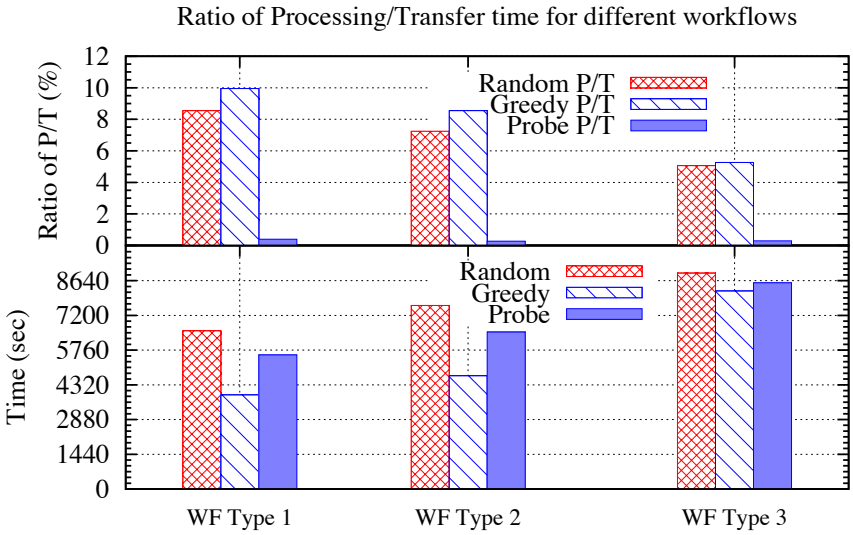



**Figure 5.8:** *Average data transfer and segment-processing times of all files using random, greedy and probe-based retrieval techniques for three workflow (WF 1, WF 2 and WF 3 corresponding to Montage, LIGO and IR workflows in Figure 5.4)*

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 5.8 (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 5.8.

As the workflow structure becomes complex (WF 1 $\rightarrow$ WF 3), both the transfer time and the processing time increased for greedy and random based retrievals, as depicted in Figure 5.8 (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) 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.

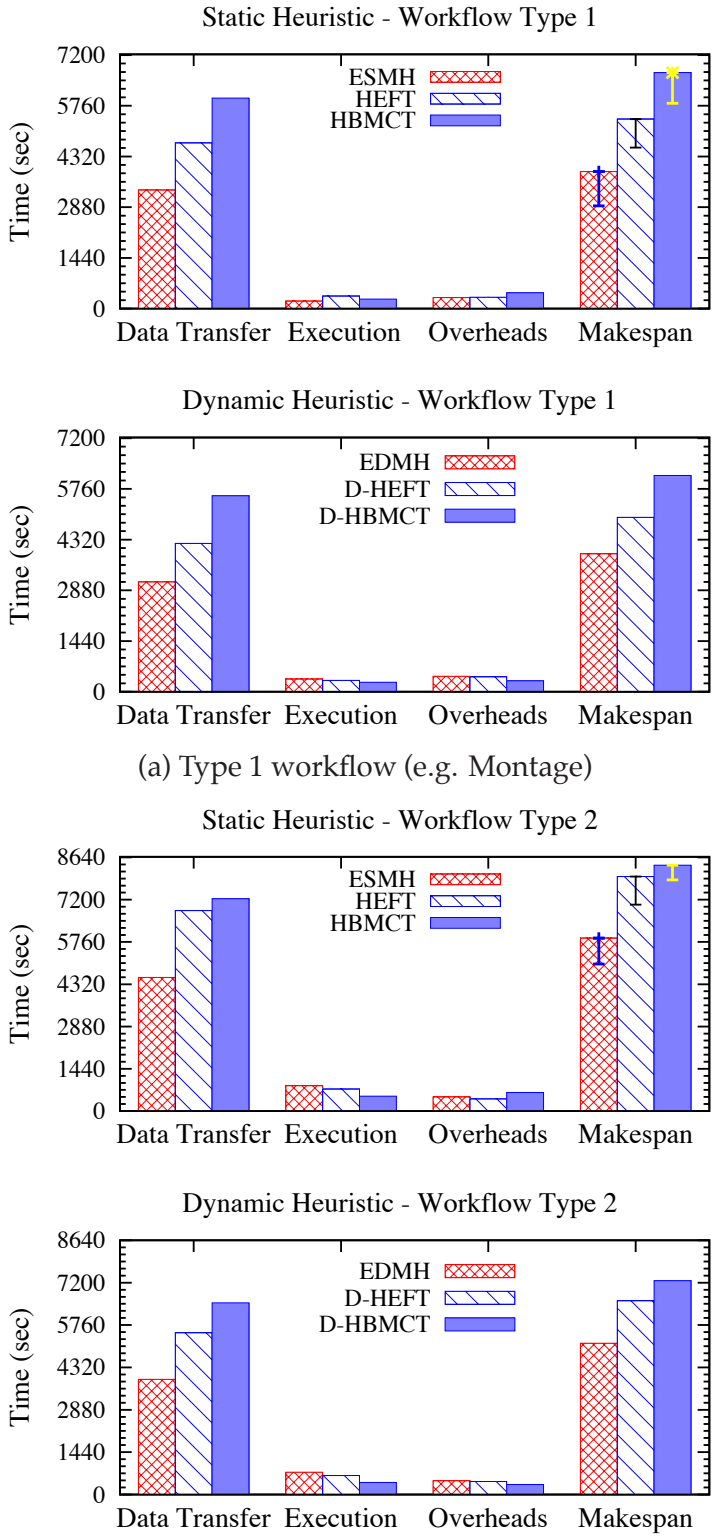
**Comparison of Heuristic Approaches:** We compared the static and dynamic approaches in turn.

*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 5.9 and 5.10. 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 5.9 and 5.10, 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 5.4-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 Figure 5.4-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 5.9 for workflow WF 1 (Montage).

However, for both complex and hybrid workflows (Figure 5.4-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 5.9 and 5.10 for workflow WF




(a) Type 1 workflow (e.g. Montage)

(b) Type 2 workflow (e.g. LIGO)

**Figure 5.9:** *Makespan of Montage and LIGO (WF 1 and WF 2) workflows when using static and dynamic scheduling heuristic.*

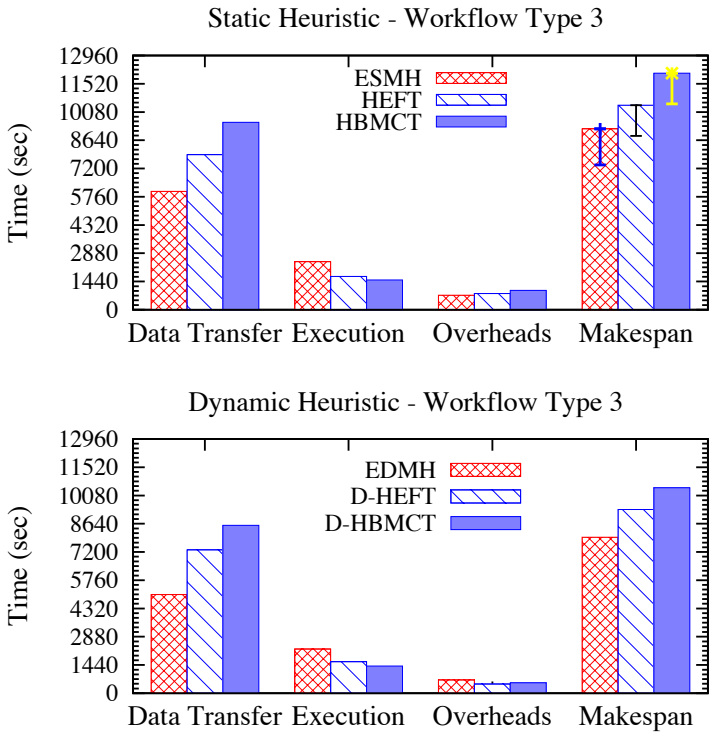



**Figure 5.10:** *Makespan of IR workflow (WF 3) using static and dynamic heuristic.*

2 (LIGO) and WF 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 5.9 and 5.10, 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 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.

### 5.3.5 Real Experiment based Evaluation

In this section, we present the results obtained using a real testbed depicted in Figure 5.2(c). This testbed was selected to match the resource distribution shown in the motivation section in Figure 1.2.

#### Experiment Setup

We formed an experimental testbed consisting of compute resources from worldwide research labs and Amazon EC2, as depicted in Figure 5.2(c). These resources were a combination of real compute nodes and virtual machines (VM) (Amazon EC2 nodes), similar to a hybrid Cloud. The Figure 5.2(b),(c) 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. 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 for our real experiment, with data distribution same as described in sub-subsection 5.3.3.

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 5.2(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.

#### 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 [102]. We varied the input file size for each task from 16KB to 640MB and iterated the experiment for eight times for

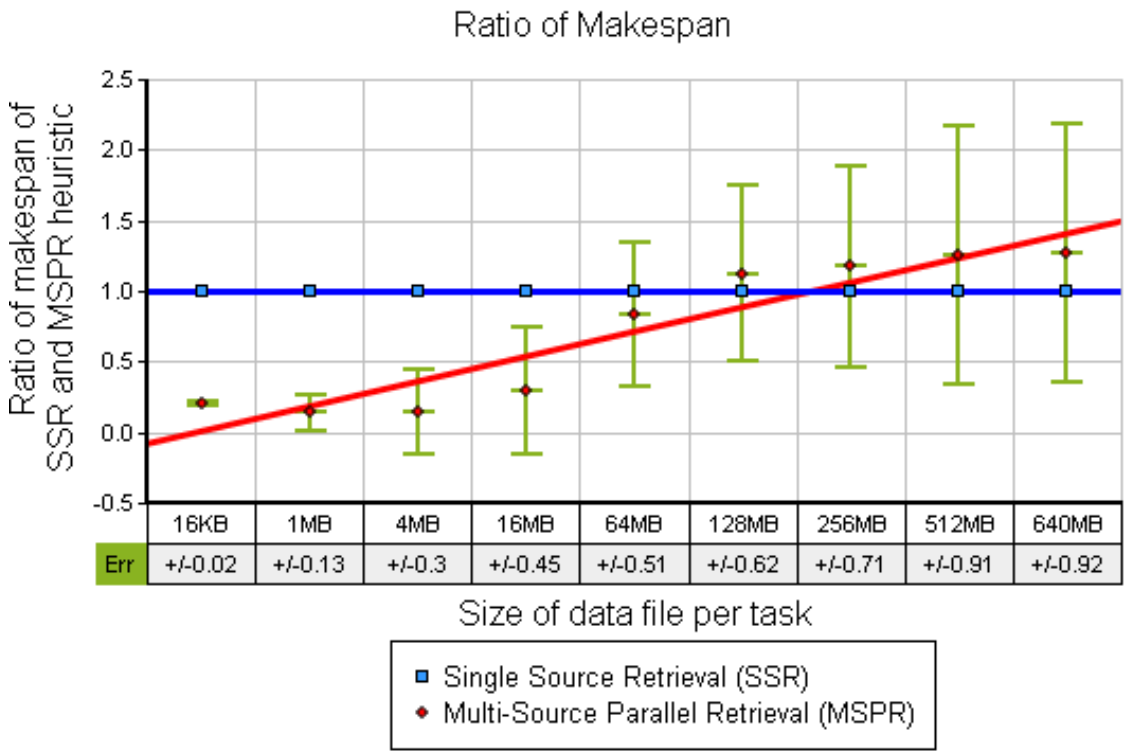



**Figure 5.11:** *Determining the benefit of using multi-source parallel data retrieval by comparing the ratio of makespan for IR workflow (WF type three) in real environment*

each input size (Figure 5.11) 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 5.3:

$$MakespanRatio = \frac{(Makespan_{D-HEFT})}{(Makespan_{EDMH})} \tag{5.3}$$

Figure 5.11 plots the mean values of the ratio from Equation 5.3 for varying file sizes for the IR workflow. It also plots the standard error[5] 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 5.11 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

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

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 5.11. Also, the break-even point (the intersection of the two lines) in Figure 5.11 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.

## 5.4 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. [146] presented design and implementation of high-level replica selection service in the Globus data Grid. Chervenak et al. [32] defined a replica location service (RLS) that maintains and provides access to information about the physical locations of copies. Hu et al. [65] 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. [178] 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. [48] 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) [155] 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. [178] and Feng et al. [48]. Our work leverages these retrieval techniques, namely the greedy and bandwidth proportional partitioned retrieval by Feng et al [48].

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. [150] selected the data-hosts by using one of the solutions to the Set-Coverage problem [9]. In our work, we assume that data are fully replicated and hence set-coverage is guaranteed by every data source. In addition,

we leverage the presence of replicated data for scheduling workflow applications in time efficient manner.

Some work on transport protocols have focused on receiver based flow contention management. Rodriguez et al. [122] 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. [157] 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. [161] and Feng and Humphrey [48]. GridFTP and rFTP [48] 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. [143] 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. In our work, we calculate the instantaneous value for bandwidth and assign the weights to communication channels.

Sakellariou and Zhou et al. [124] 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 algorithms (HBMCT) for comparing with our static scheduling heuristic proposed in this chapter.

Deelman et al. [40] 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. [129] 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. However, this their work data was not partitioned when retrieving. In our work, we partition the data and download it from multiple sources in parallel.

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. [125] 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. This work is orthogonal to our work which also tries to minimize cost of total computation. Even though we do not carry out SR, we do achieve significant savings on total execution time by applying parallel data retrieval technique.

Several work have explored static and dynamic scheduling strategies for workflow execution that focused on: user quality of service and location affinity [16; 13; 10], iterative calls of static algorithms [112; 86; 171], dynamic programming [111] and so forth. Lopez et al. [86] explored several static and dynamic approaches for scheduling workflows and concluded that list-based heuristics significantly outperform non-list based heuristics. Yu et al. [169] 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. Our work addressed this challenge using heuristics based algorithms and showed experimentally that multi-source parallel data retrieval technique can significantly enhance the schedules of data intensive application workflows.

## 5.5 Conclusions

In this chapter, 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 makespan 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.

In this chapter, we focused on minimizing total execution time. In the following chapter, we minimize the total cost of execution using particle swarm optimization based heuristic.

# 6

# Particle Swarm Optimization Based Scheduling Heuristic

Cloud computing environments facilitate applications by providing virtualized resources that can be provisioned dynamically. However, users are charged on a pay-per-use basis. User applications may incur large data retrieval and execution costs when they are scheduled taking into account only the 'execution time'. In addition to optimizing execution time, the cost arising from data transfers between resources as well as execution costs must also be taken into account.

In this chapter, we present a particle swarm optimization (PSO) based heuristic to schedule applications to Cloud resources that takes into account both computation cost and data transmission cost. We experiment with a workflow application by varying its computation and communication costs. We compare the cost savings when using PSO and existing 'Best Resource Selection' (BRS) algorithm. Our results show that PSO can achieve: a) as much as 3 times cost savings as compared to BRS, and b) good distribution of workload onto resources.

## 6.1 Introduction

Modern collaborative scientific experiments in domains such as structural biology, high-energy physics and neuroscience involve the use of distributed data sources. As a result, analysis of their datasets is represented and structured as scientific workflows [58]. These scientific workflows usually need to process huge amount of data and computationally intensive activities, as also described as motivational applications in Chapter 1. A scientific workflow management system [102], such as the one we designed (see Chapter 3), is used for managing these scientific experiments by hiding the orchestration and integration details inherent while executing workflows on distributed resources provided by Cloud service providers.

Cloud computing is a new paradigm for distributed computing that delivers infrastructure, platform, and software (application) as services. These services are made avail-

able as subscription-based services in a pay-as-you-go model to consumers [25; 7]. Cloud computing helps user applications dynamically provision as many compute resources at specified locations (US east1a-d for Amazon[1]) as per their requirements. Also, applications can choose the storage locations to host their data (Amazon S3[2]) at global locations. In order to efficiently and cost effectively schedule the tasks and data of applications onto these Cloud computing environments, application schedulers have different policies that vary according to the objective function: minimize total execution time, minimize total cost to execute, balance the load on resources used while meeting the deadline constraints of the application, and so forth. In this chapter, we focus on minimizing the total execution cost of applications on these resources provided by Cloud service providers, such as Amazon and GoGrid[3]. We achieve this by using a meta-heuristics method called Particle Swarm Optimization (PSO).

Particle Swarm Optimization (PSO) is a self-adaptive global search based optimization technique introduced by Kennedy and Eberhart [72]. The algorithm is similar to other population-based algorithms like Genetic algorithms but, there is no direct re-combination of individuals of the population. Instead, it relies on the social behavior of the particles. In every generation, each particle adjusts its trajectory based on its best position (local best) and the position of the best particle (global best) of the entire population. This concept increases the stochastic nature of the particle and converge quickly to a global minima with a reasonable good solution.

PSO has become popular due to its simplicity and its effectiveness in wide range of application with low computational cost. Some of the applications that have used PSO are: the reactive voltage control problem [163], data mining [136], chemical engineering [100], pattern recognition [87] and environmental engineering [88]. The PSO has also been applied to solve *NP-Hard* problems like Scheduling [171; 83] and task allocation [162; 172].

Our main contributions in this chapter are as follows:

- We formulate a model for task-resource mapping to minimize the overall cost of execution
- We design a heuristic that uses PSO to solve task-resource mappings based on the proposed model

The rest of the chapter is organized as follows: Section 6.5 presents related work. In Section 6.2, we describe the task-resource scheduling problem and its formulation with the help of an example workflow. In Section 6.3, we present our scheduling heuristic that uses PSO and introduce the PSO algorithm. Section 6.4 presents an experimental evaluation of the performance our heuristic. Section 6.6 concludes the chapter.

[1]http://aws.amazon.com
[2]http://aws.amazon.com/s3/
[3]http://www.gogrid.com

## 6.2 Task-Resource Scheduling Problem Formulation

The mapping of tasks of an application workflow to distributed resources can have several objectives. We focus on minimizing the total cost of computation of an application workflow.

We denote an application workflow as a Directed Acyclic Graph (DAG) represented by $G$=($V$, $E$), where $V$=$\{T_1, ..., T_n\}$ is the set of tasks, and $E$ represents the data dependencies between these tasks, that is, $f_{j,k} = (T_j, T_k) \in E$ is the data produced by $T_j$ and consumed by $T_k$. We have a set of storage sites $S = \{1, ..., i\}$, a set of compute sites $PC = \{1, ..., j\}$, and a set of tasks $T = \{1, ..., k\}$. We assume the 'average' computation time of a task $T_k$ on a compute resource $PC_j$ for a certain size of input is known. Then, the cost of computation of a task on a compute host is inversely proportional to the time it takes for computation on that resource. We also assume the cost of unit data access $d_{i,j}$ from a resource $i$ to a resource $j$ is known. The access cost is fixed by the service provider. The transfer cost can be calculated according to the data transferred between the sites, without any time constraints. But, data could then be transferred without any regard to the time taken. Therefore, to value the link between sites and the time taken for transferring data, we have used the cost for transferring unit data between sites, per second. We assume that these costs are non-negative, symmetric, and satisfy the triangle inequality: that is, $d_{i,j} = d_{j,i}$ for all $i, j \in N$, and $d_{i,j} + d_{j,k} \geq d_{i,k}$ for all $i, j, k \in N$.

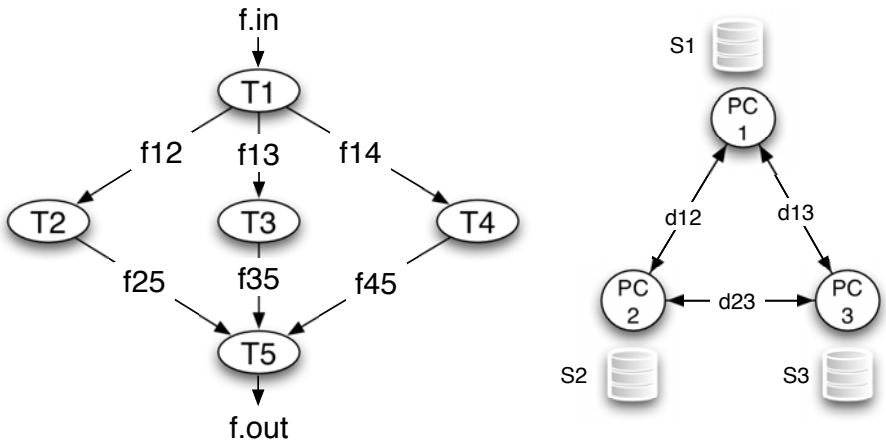



**Figure 6.1:** *An example workflow, compute nodes (PC) & storage (S).*

Figure 6.1 depicts a workflow structure with five tasks, which are represented as nodes. The dependencies between tasks are represented as arrows. This workflow is similar in structure to our version of the Evolutionary Multi-objective Optimization (EMO) application [148]. The root task may have an input file (e.g. *f.in*) and the last task produces the output file (e.g. *f.out*). Each task generates output data after it has completed ($f_{12}$, $f_{13}$, ..., $f_{45}$). These data are used by the task's children, if any. The numeric

values for these data is the edge-weight ($e_{k1,k2}$) between two tasks $k1 \in T$ and $k2 \in T$. The figure also depicts three compute resources ($PC1, PC2, PC3$) interconnected with varying bandwidth and having its own storage unit ($S1, S2, S3$). The goal is to assign the workflow tasks to the compute resources such that the total cost of computation is minimized.

The problem can be stated as: *"Find a task-resource mapping instance M, such that when estimating the total cost incurred using each compute resource $PC_j$, the highest cost among all the compute resources is minimized."*

Let $C_{exe}(M)_j$ be the total cost of all the tasks assigned to a compute resource $PC_j$ (Eq. 6.1). This value is computed by adding all the node weights (the cost of execution of a task $k$ on compute resource $j$) of all tasks assigned to each resource in the mapping $M$. Let $C_{tx}(M)_j$ be the total access cost (including transfer cost) between tasks assigned to a compute resource $PC_j$ and those that are not assigned to that resource in the mapping $M$ (Eq. 6.2). This value is the product of the output file size (given by the edge weight $e_{k1,k2}$) from a task $k1 \in k$ to task $k2 \in k$ and the cost of communication from the resource where $k1$ is mapped ($M(k1)$) to another resource where $k2$ is mapped ($M(k2)$). The average cost of communication of unit data between two resources is given by $d_{M(k1),M(k2)}$. The cost of communication is applicable only when two tasks have file dependency between them, that is when $e_{k1,k2} > 0$. For two or more tasks executing on the same resource, the communication cost is zero.

$$C_{exe}(M)_j = \sum_{k} w_{kj} \qquad \forall M(k) = j \tag{6.1}$$

$$C_{tx}(M)_j = \sum_{k1 \in T} \sum_{k2 \in T} d_{M(k1),M(k2)} e_{k1,k2}$$
$$\forall M(k1) = j \; and \; M(k2) \neq j \tag{6.2}$$

$$C_{total}(M)_j = C_{exe}(M)_j + C_{tx}(M)_j \tag{6.3}$$

$$Cost(M) = max(C_{total}(M)_j) \quad \forall j \in P \tag{6.4}$$

$$Minimize(Cost(M) \quad \forall M) \tag{6.5}$$

Equation 6.4 ensures that all the tasks are **not** mapped to a single compute resource. Initial cost maximization will distribute tasks to all resources. Subsequent minimization of the overall cost (Equation 6.5) ensures that the total cost is minimal even after initial distribution. For a given assignment $M$, the total cost $C_{total}(M)_j$ for a compute resource $PC_j$ is the sum of execution cost and access cost (Eq. 6.3). When estimating the total cost for all the resources, the largest cost for all the resources is minimized (Eq. 6.5). This indirectly ensures that the tasks are not mapped to a single resources and there will be a distribution of cost among the resources.

## 6.3 Scheduling based on Particle Swarm Optimization

In this section, we present a scheduling heuristic for dynamically scheduling workflow applications. The heuristic optimizes the cost of task-resource mapping based on the solution given by particle swarm optimization technique. The optimization process uses two components: a) the scheduling heuristic as listed in Algorithm 5, and b) the PSO steps for task-resource mapping optimization as listed in Algorithm 6. First, we will give a brief description of PSO algorithm.

$$v_i^{k+1} = \omega v_i^k + c_1 rand_1 \times (pbest_i - x_i^k) + c_2 rand_2 \times (gbest - x_i^k), \tag{6.6}$$

$$x_i^{k+1} = x_i^k + v_i^{k+1}, \tag{6.7}$$

where:

| | |
|---|---|
| $v_i^k$ | velocity of particle $i$ at iteration $k$ |
| $v_i^{k+1}$ | velocity of particle $i$ at iteration $k+1$ |
| $\omega$ | inertia weight |
| $c_j$ | acceleration coefficients; $j = 1, 2$ |
| $rand_i$ | random number between 0 and 1; $i = 1, 2$ |
| $x_i^k$ | current position of particle $i$ at iteration $k$ |
| $pbest_i$ | best position of particle $i$ |
| $gbest$ | position of best particle in a population |
| $x_i^{k+1}$ | position of the particle $i$ at iteration $k+1$. |

### 6.3.1 Particle Swarm Optimization

Particle Swarm Optimisation (PSO) is a swarm-based intelligence algorithm [72] influenced by the social behavior of animals such as a flock of birds finding a food source or a school of fish protecting themselves from a predator. A particle in PSO is analogous to a bird or fish flying through a search (problem) space. The movement of each particle is co-ordinated by a velocity which has both magnitude and direction. Each particle position at any instance of time is influenced by its best position and the position of the best particle in a problem space. The performance of a particle is measured by a fitness value, which is problem specific.

The PSO algorithm is similar to other evolutionary algorithms. In PSO, the population is the number of particles in a problem space. Particles are initialized randomly. Each particle will have a fitness value, which will be evaluated by a fitness function to be optimized in each generation. Each particle knows its best position *pbest* and the best

position so far among the entire group of particles *gbest*. The *pbest* of a particle is the best result (fitness value) so far reached by the particle, whereas *gbest* is the best particle in terms of fitness in an entire population. In each generation the velocity and the position of particles will be updated as in Eq 6.6 and 6.7, respectively.

PSO algorithm provide a mapping of all the tasks to a set of given resources based on the model described in Section 6.2.

**Algorithm 5** PSO based scheduling heuristic.

1: Calculate average computation cost of all tasks in all compute resources
2: Calculate average cost of (communication/size of data) between resources
3: Set task node weight $w_{kj}$ as average computation cost
4: Set edge weight $e_{k1,k2}$ as size of file transferred between tasks
5: Compute PSO($\{t_i\}$) /* a set of all tasks $i \in k$*/
6: **repeat**
7: **for** all "ready" tasks $\{t_i\} \in T$ **do**
8: Assign tasks $\{t_i\}$ to resources $\{p_j\}$ according to the solution provided by PSO
9: **end for**
10: Dispatch all the mapped tasks
11: Wait for *polling_time*
12: Update the ready task list
13: Update the average cost of communication between resources according to the current network load
14: Compute PSO($\{t_i\}$)
15: **until** there are unscheduled tasks

**Scheduling Heuristic:** We calculate the average computation cost (assigned as node weight in Figure 6.1) of all tasks on all the compute resources. This cost can be calculated for any application by executing each task of an application on a series of known resources. It is represented as TP matrix in Table 6.1. As the computation cost is inversely proportional to the computation time, the cost is higher for those resources that complete the task quicker. Similarly, we store the average value of communication cost between resources per unit data, represented by PP matrix in Table 6.1, described later in the chapter. The cost of communication is inversely proportional to the time taken. We also assume we know the size of input and output data of each task (assigned as edge weight $e_{k1,k2}$ in Figure 6.1). In addition, we consider this cost is for the transfer per second (unlike Amazon CloudFront which does not specify time for transferring).

The initial step is to compute the mapping of all tasks in the workflow, irrespective of their dependencies (Compute PSO($t_i$)). This mapping optimizes the overall cost of computing the workflow application. To validate the dependencies between the tasks, the algorithm assigns the "ready" tasks to resources according to the mapping given by PSO. By "ready" tasks, we mean those tasks whose parents have completed execution and have provided the files necessary for the tasks' execution. After dispatching the tasks

to resources for execution, the scheduler waits for *polling_time*. This time is for acquiring the status of tasks, which is middleware and application dependent. The main drawback of polling is that newly submitted tasks may wait in the queue upto the polling time, before being scheduled. However, the system must rely on the polling time as it cannot accurately predict the completion time of tasks on every compute resource. The polling time is, however, adjustable if the scheduling system is aware of the application execution times, which may be derived from historic executions. Generally, a scheduling system (such as PBS) assumes it may not have this information as users can submit any number and type of tasks to the system for execution.

Depending on the number of tasks completed, the ready list is updated, which will now contain the tasks whose parents have completed execution. We then update the average values for communication between resources according to the current network load. As the communication costs would have changed, we recompute the PSO mappings. Also, when remote resource management systems are not able to assign task to resources according to our mappings due to resource unavailability, the recomputation of PSO makes the heuristic dynamically balances other tasks' mappings (online scheduling). Based on the recomputed PSO mappings, we assign the ready tasks to the compute resources. These steps are repeated until all the tasks in the workflow are scheduled.

**Algorithm 6** PSO algorithm.

1: Set particle dimension as equal to the size of ready tasks in $\{t_i\} \in T$
2: Initialize particles position randomly from $PC = 1, ..., j$ and velocity $v_i$ randomly.
3: For each particle, calculate its fitness value as in Equation 6.4.
4: If the fitness value is better than the previous best *pbest*, set the current fitness value as the new *pbest*.
5: After Steps 3 and 4 for all particles, select the best particle as *gbest*.
6: For all particles, calculate velocity using Equation 6.6 and update their positions using Equation 6.7.
7: If the stopping criteria or maximum iteration is not satisfied, repeat from Step 3.

The algorithm is dynamic (online) as it updates the communication costs (based on average communication time between resources) in every scheduling loop. It also recomputes the task-resource mapping so that it optimizes the cost of computation, based on the current network and resource conditions.

**PSO:** The steps in the PSO algorithm are listed in Algorithm 6. The algorithm starts with random initialization of particle's position and velocity. In this problem, the particles are the task to be assigned and the dimension of the particles are the number of tasks in a workflow.

The value assigned to a each dimension of a particles are the computing resources indices. Thus the particle represent a mapping of resource to a task. In our workflow

(depicted in Figure 6.1) each particle is 5-D because of 5 tasks and the content of each dimension of the particles is the compute resource assigned to that task. For example a sample particle could be represented as depicted in Figure 6.2.

The evaluation of each particle is perform by the fitness function given in Eq. 6.5. The particles calculate their velocity using Eq. 6.6 and update their position according to Eq. 6.7. The evaluation is carried out until the specified number of iterations (user-specified stopping criteria).

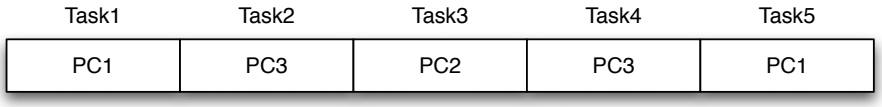



**Figure 6.2:** *A sample particle for the workflow shown in Figure 6.1 .*

## 6.4 Experimental Evaluation

In this section, we present the metric of comparison, the experiment setup and the results.

### 6.4.1 Performance metric

As a measure of performance, we used cost for complete execution of application as a metric. We computed the total cost of execution of a workflow using two heuristics: PSO based cost optimization (Algorithm 5), and best resource selection (based on minimum completion time by selecting a resource with maximum cost).

### 6.4.2 Data and Implementation

We have used three matrices that store the values for: a) average computation cost of each task on each resource (TP-matrix), b) average communication cost per unit data between compute resources (PP-matrix), and c) input/output Data Size of each task (DS-matrix), as depicted in Table 6.1.

The values for PP-matrix resemble the cost of unit data transfer between resources given by Amazon CloudFront[4]. We assume PC1 to be in US, PC2 in Hong Kong (HK) and PC3 in Japan (JP), respectively. We randomly choose the values in the matrix for every repeated experiment, but keep these values constant during the PSO iterations.

The values for TP-matrix varies for two classes of experiments. While varying the size of data, we choose the TP-matrix values to resemble the Evolutionary Multi-objective Optimization (EMO) [148] application. While varying the processing cost, we use the Amazon EC2's[5] pricing policy for different classes of virtual machine instances. E.g.

[4]http://aws.amazon.com/cloudfront/
[5]http://aws.amazon.com/ec2/

**Table 6.1:** *The TP-matrix, PP-matrix and DS-matrix. The values shown are an example of 1 instance of the experiment run.*

$$TP[5 \times 3] = \left[\begin{array}{c|ccc} & PC1 & PC2 & PC3 \\ \hline T_1 & 1.23 & 1.12 & 1.15 \\ T_2 & 1.17 & 1.17 & 1.28 \\ T_3 & 1.13 & 1.11 & 1.11 \\ T_4 & 1.26 & 1.12 & 1.14 \\ T_5 & 1.19 & 1.14 & 1.22 \end{array}\right]$$

$TP[i, j] = Cost\ of\ execution\ of\ T_i\ at\ PC_j$
($EC2\ price\ of\ resources\ for\ High\ CPU\ instance$)
($Example\ matrix\ values\ are\ in\ the\ range$ \$1.1 – \$1.28/$hr$)

$$PP[3 \times 3] = \left[\begin{array}{c|ccc} & PC1 & PC2 & PC3 \\ \hline PC1 & 0 & 0.17 & 0.21 \\ PC2 & 0.17 & 0 & 0.22 \\ PC3 & 0.21 & 0.22 & 0 \end{array}\right]$$

$PP[i, j] = Cost\ of\ communication\ between\ PC_i\ \&PC_j$
($Values\ in$ \$/$MB/second$)

$$DS_{T2,T3,T4}[2 \times 2] = \left[\begin{array}{c|c} & totaldata \\ \hline i/p & 10 \\ o/p & 10 \end{array}\right]$$

$$DS_{T5}[2 \times 2] = \left[\begin{array}{c|c} & totaldata \\ \hline i/p & 30 \\ o/p & 60 \end{array}\right]$$

$row_1 = i/p\ data\ size(MB), \quad row_2 = o/p\ data\ size(MB)$

if we were to use small+medium instances of Linux machines in both US and Europe, the TP-matrix would have values between \$0.1-\$0.3/hr, assuming all the tasks complete within 1 hour.

As each task has its own DS-matrix, the sum of all the values in the matrix varies according to the size of data we experiment (64-1024 MB). The total data is divided among tasks such that if $x$ is the output data size of $T1$, then tasks $T2$, $T3$, &$T4$ each receive $x$ data as input and produce $x$ data as output. Finally, task $T5$ consumes $3x$ data and produces $6x$ data.

We used the JSwarm[6] package to conduct our simulation experiments in PSO. Table 6.2 gives the experimental setup of the PSO algorithm. The number of executions represent the number of independent experiments done in order to calculate the Confidence Interval

[6]http://jswarm-pso.sourceforge.net/

(CI) of the results.

**Table 6.2:** *Statistics of PSO executions.*

| |
|---|
| Number of particles = 25 |
| Number of iterations = 45 |
| Number of executions = 30 |

### 6.4.3 Experiments and Results

We evaluated the scheduling heuristic using the workflow depicted in Figure 6.1. Each task in the workflow has input and output files of varying sizes. Also, the execution cost of each task varies among all the compute resources used (in our case $PC1 - PC3$). We analyze the performance of our heuristic by varying each of these in turn.

We plot the graphs by averaging the results obtained after 30 independent executions. In every execution, the x-axis parameters such as total data size (e.g. 1024MB), range of computation cost (e.g. 1.1-1.3 $/hour) remain unchanged, while the particle's velocity and position change. The graphs also depict the value of the plotted points together with the CI (represented as "+/-" *value*).

#### Variation in Total Data Size of a Workflow

We varied the size of total data processed by the workflow in the range 64-1024 MB. By varying the data size, we compared the variance in total cost of execution and the distribution of workload on resources, for the two algorithms as depicted in Figure 6.3 and Figure 6.4, respectively. We fixed the compute resource cost in the range 1.1 – 1.3$/hr for the experiments in the sub-section 6.4.3 and sub-section 6.4.3.

**Total Cost of Execution:** Figure 6.3 plots the total cost of computation of the workflow (in the **log** scale) with the increase in the total data processed by the workflow. The graph also plots 95% Confidence Interval (CI) for each data point.

The cost obtained by PSO based task-resource mapping increases much slower than the BRS algorithm. PSO achieves at least three times lower cost for 1024MB of total data processed than the BRS algorithm. Also, the value of CI in cost given by PSO algorithm is +/- 8.24, which is much lower as compared to the BRS algorithm (+/- 253.04), for 1024 MB of data processed by the workflow.

The main reason for PSO to perform better than the 'best resource' selection is the way it takes into account communication costs of all the tasks, including dependencies between them. When calculating the cost of execution of a child task on a resource, it adds the data transfer cost for transferring the output from its parent tasks' execution node to that node. This calculation is done for all the tasks in the workflow to find the

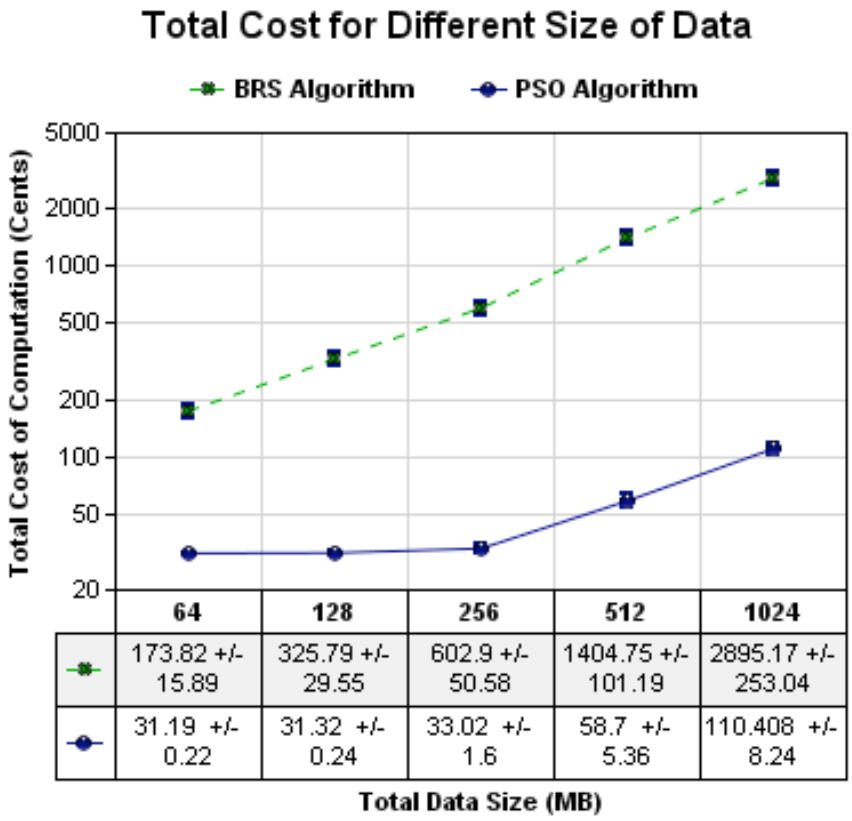



**Figure 6.3:** *Comparison of total cost between PSO based resource selection and best resource selection algorithms when varying total data size of a workflow.*

near optimal scheduling of task to resources. However, the BRS algorithm calculates the cost for a single task at a time, which does not take into account the mapping of other tasks in the workflow. This results in PSO based algorithm giving lower cost of execution as compared to BRS based algorithm.

If this cost of mapping to the current resource is higher than previous best, PSO does not map the task to this resource. Hence, a global minima is found that ensures all tasks' mapping is near optimal. As the mappings are based on global values, they do not change significantly for PSO, which explains the lower deviation values in Figure 6.3. The deviation is quite large in case of 'best selection' as the algorithm just calculates the local minima for each task which changes for every change in values of PP-matrix (communication cost).

**Distribution of Load:** We calculated the distribution of workflow tasks onto available resources for various size of total data processed, depicted in Figure 6.4. This evaluation is necessary as algorithms may choose to submit all the tasks to few resources to avoid communication between resources as the size of data increases, thus minimizing communication cost to zero. In our formulation, equation 6.4 restricts all tasks being mapped

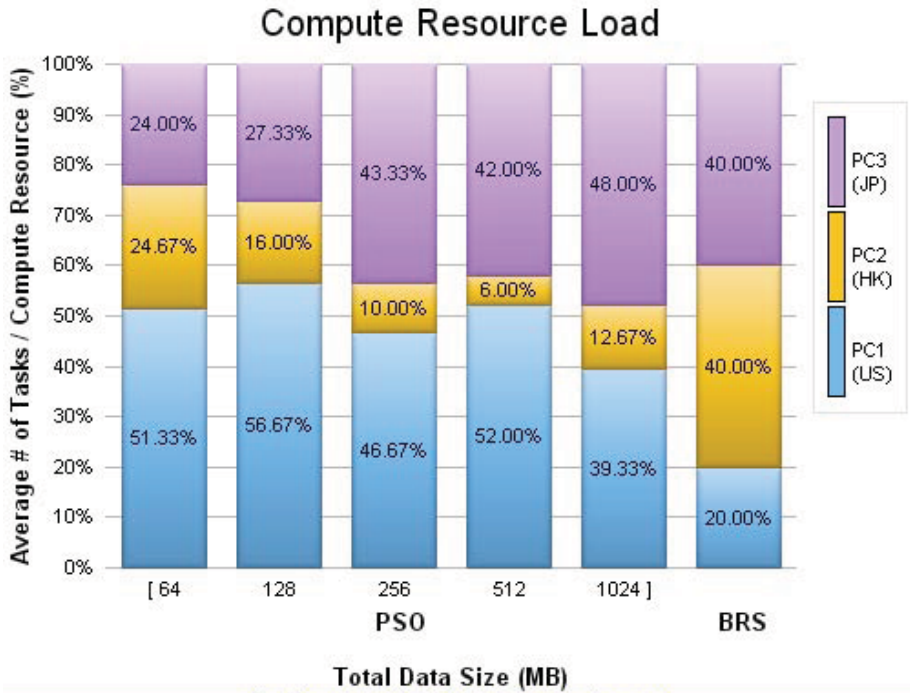



**Figure 6.4:** *Distribution of workflow tasks on available processors.*

to the same resource, so that tasks can execute in parallel for increased time-efficiency. In Figure 6.4, The X-axis represents the total size of data processed by the workflow and the Y-axis the average number of tasks (expressed as percentage) executed by a compute resource for various size of data.

The figure shows that PSO distributes tasks to resources according to the size of data. When the total size of data is small (for 64-126 MB), PSO distributed tasks proportionally to all the resources ($PC1 - PC3$). However, when the size of data increased to (and over) 256MB, more tasks were allocated to $PC1$ and $PC3$.

As the cost of compute resources was fixed for this part of experiment, the BRS algorithm does not vary task-resource mapping. Also, it is indifferent to the size of data. Hence, BRS's load distribution is a straight line as depicted in Figure 6.4, with $PC1$, $PC2$ and $PC3$ receiving 20%, 40% and 40% of the total tasks, respectively.

The distribution of tasks to all the available resources in proportion to their usage costs, ensured that hotspots (resource overloading) were avoided. Our heuristic could minimize the total cost of execution and balance the load on available resources.

### Variation in Compute Resource Cost

We experimented the performance of PSO by varying the cost of computation of all compute resources. This variation is practically justifiable as different Cloud service providers (e.g. Amazon, GOGRID) can have varying pricing policies depending on the

type and capabilities of their resources (virtual machines).

Figure 6.5 depicts the change in total cost of computation of applications for different range of compute resource prices (price range are similar to Amazon EC2 instances in US and Europe combined). The plotted values are an average of 30 executions. We use curve fitting to plot the lines along the points to show the trend: rise in cost in comparison to rise in compute resource cost for the two algorithms. The workflow processed a total of 128MB of data.

Clearly, PSO based mapping has much lower cost (at least 10 times) and CI values (lower than 0.3) as compared to that given BRS based mapping. In addition, the slope of the trend line shows that PSO based mapping increases the cost linearly, whereas BRS increases exponentially.

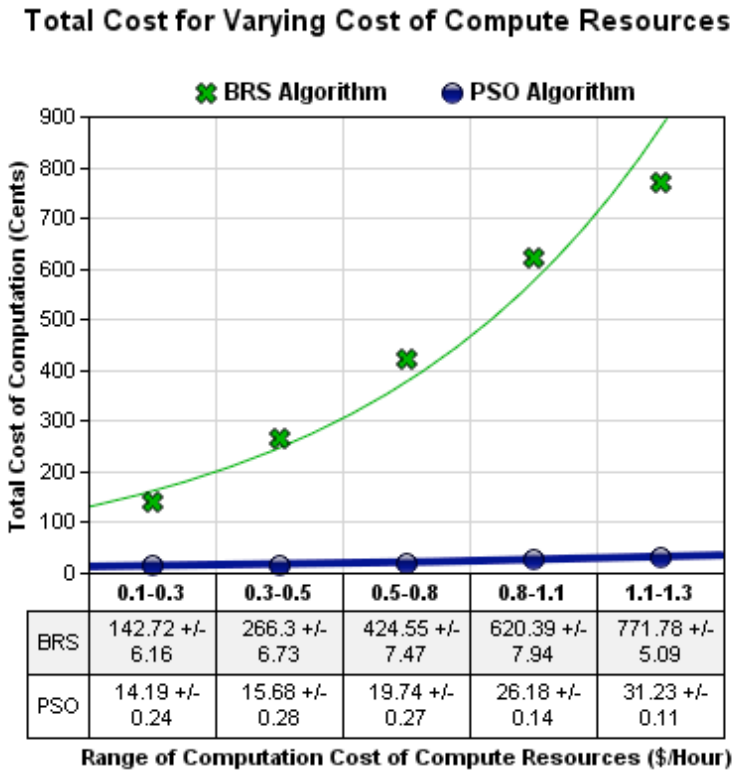



**Figure 6.5:** *Comparison of total cost between PSO based resource selection and best resource selection algorithms when varying computation cost of all the resources ( for 128MB of data).*

The reason for PSO's improvement over BRS is due to PSO's ability to find a near optimal solutions for mapping all tasks in the workflow to the given set of compute resources. The linear increase in PSO's cost also suggest that it takes both computation and communication cost into account. However, BRS simply maps a task to the resource that has minimum completion time (a resource with higher frequency, lower load and thus having higher cost). As the resource costs increase, the use of BRS leads to more costs due to the affinity towards better resource, irrespective to the size of data. Whereas,

PSO minimizes the maximum total cost of assigning all tasks to resources.

### Convergence of PSO

Figure 6.6 plots the convergence of total cost computed by PSO over the number of iterations for different sizes of total data processed by the workflow in Figure 6.1. Initially, the particles are randomly initialized. Therefore, the inital total cost is always high. This initial cost corresponds to the $0^{th}$ iteration. As the algorithm progresses, the convergence is drastic and it finds a global minima very quickly. The number of iterations needed for the convergence is seen to be 20-30, for our application environment.

We use PSO as it has a faster convergence rate than GA. Also, it has fewer primitive mathematical operators than in GA (e.g. reproduction, crossover, mutation), making applications less dependent on parameter fine-tuning. Moreover, using discrete numbers, we can easily correlate particle's position to task-resource mappings. Thus, when the system has a large number of tasks and/or large number of resources, the computing time of PSO algorithm is significantly lower than existing iterative techniques, such as GA and SA (Simulated Annealing) [106; 138].

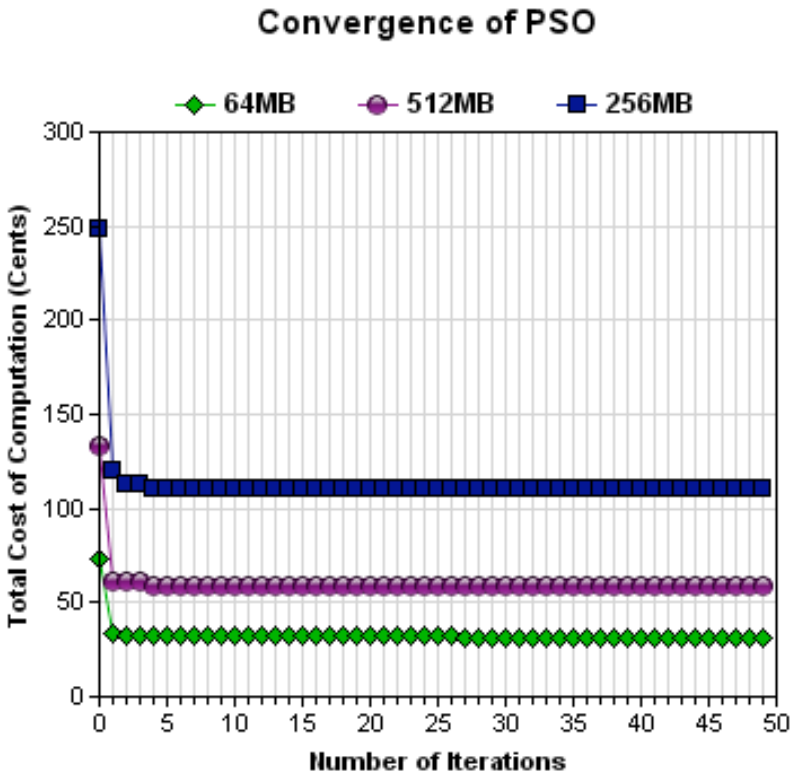



**Figure 6.6:** *The trend of convergence of PSO with the number of iterations for different size of data.*

## 6.5 Related Work

Workflow applications are commonly represented as a directed acyclic graph. The mapping of jobs to the compute-resources is an *NP-complete* problem in the general form [144]. The problem is *NP-complete* even in two simple cases: (1) scheduling jobs with uniform weights to an arbitrary number of processors and (2) scheduling jobs with weights equal to one or two units to two processors [144]. So, past work have proposed many heuristics based approach to scheduling workflow applications. Data intensive workflow applications are a special class of workflows, where the size and/or quantity of data is large. As a result, the transfer of data from one compute node to another takes longer time. This incurs higher transmission and storage cost than computing cost running on these data.

Deelman et al. [40] have done considerable work on planning, mapping and data-reuse in the area of workflow scheduling. They have proposed Pegasus [40], which is a framework that maps complex scientific workflows onto distributed resources such as the Grid. DAGMan, together with Pegasus, schedules tasks to Condor system. Deelman et al. [39] investigated the cost of computing in the Cloud taking Montage workflow as a case study. They experimentally show that the cost of running an application on a cloud depends on the compute, storage and communication resources, which may be different for different execution plans of the same application. By provisioning the right amount of storage and compute resources, their results showed that cost can be significantly reduced with no significant impact on application performance.

Yu et al. [170] also addressed cost-based scheduling of scientific applications. They proposed a cost-based workflow scheduling algorithm that minimized the cost of execution while meeting the deadline. They used Markov Decision Process approach to schedule sequential workflow task execution. That resulted in the algorithm finding the optimal path among services to execute tasks and transfer data. However, they did not take into account the presence of replicated data that can be retrieved from multiple distributed resources.

Other well-known projects on workflow systems include GridFlow [28], ICENI [55], GridAnt [6], Triana [140] and Kepler [89]. Most of these works schedule tasks based on earliest finish time, earliest starting time or the high processing capabilities of computing resources. We term these as "best resource selection" (BRS) approach, where a resource is selected based on its performance and not cost. In our work, we took into account compute resource usage cost and data transfer cost while making scheduling decisions.

Since task scheduling is a *NP-Complete* problem, Genetic Algorithm (GA) has been used for scheduling workflows [169]. However, GA may not be the best approach. Salman et al. [127] have shown that the performance of PSO algorithm is faster than GA in solving static task assignment problem for homogeneous distributed computing systems based on their test cases. Lei et al. [175] have shown that the PSO algorithm

is able to get better schedule than GA based on their simulated experiments for Grid computing. In addition, the results presented by Tasgetiren et al. [138] have provided evidence that PSO algorithm was able to improve 57 out of 90 best known solutions provided by other well known algorithms to solve the sequencing problems.

As PSO is an iterative technique for optimizing multi-objective problems, it can also be used for optimizing other metrics such as: application execution time, storage cost, communication cost, data flow, and so forth [138; 164; 127; 83].

## 6.6 Conclusions

This chapter presented a scheduling heuristic based on Particle Swarm Optimization (PSO). It described a heuristic to minimize the total cost of execution of application workflows on Cloud computing environments. By varying the communication cost between resources and the execution cost of compute resources, the heuristic calculated the average cost of execution. When comparing the results obtained by the PSO heuristic against "Best Resource Selection" (BRS) heuristic, we found that PSO based task-resource mapping could achieve at least three times cost savings as compared to BRS based mapping for our application workflow. In addition, PSO balances the load on compute resources by distributing tasks to available resources. The heuristic we proposed is generic as it can be used for any number of tasks and resources by simply increasing the dimension of the particles and the number of resources, respectively.

# 7

# Conclusions and Future Directions

## 7.1 Summary

In this thesis, we addressed the problem of scheduling data and tasks for data intensive application in Grid and Cloud environments. We proposed several scheduling algorithms that manages workflow applications on distributed resources with an objective to minimize execution time and cost. All the scheduling algorithms were based on multi-source parallel data retrieval technique, which is a novel way of retrieving data and scheduling applications. This method is in contrast to using only one 'best' resource for data retrievals, based on which applications were subsequently scheduled in the past. Through real-experiments, we obtained near-optimal as well as approximate solutions using the proposed algorithms. We also presented several real-world scientific applications that could benefit in terms of minimized execution time and cost when using our proposed algorithms. We demonstrated the execution of the applications on both Grid and Cloud computing environments. Below is the list of major contributions:

- Presented survey of scheduling and management techniques of data intensive workflow applications on distributed platforms
- Designed and implemented a workflow management system for executing scientific applications
- Proposed NLP model based scheduling algorithm for minimizing total time and cost of execution of workflow applications
- Proposed and implemented enhanced static and dynamic heuristics that uses multi-source parallel data retrieval technique to obtain time efficient schedules for workflow applications
- Proposed a fast converging PSO algorithm that also uses multi-source parallel data retrieval technique to schedule workflow applications and minimize total cost of execution on distributed resources

- Executed several real-world scientific workflow applications on Grids and Cloud platforms using the proposed scheduling algorithms to demonstrate the feasibility of both WfMS and algorithms

## 7.2 Conclusions

The work presented in this thesis thus encompasses multi-source parallel data retrieval based scheduling algorithms implemented on a workflow management system and supported by scientific workflow applications. It demonstrated the applicability of well-designed algorithms on real-world scientific applications and made significant contributions towards the advancement of the field.

This thesis described two real-world applications (CMS, LIGO) that are producing, using and sharing data in large quantities across the globe (see Chapter 1). Based on their requirements and execution environment, it listed several challenges that needs to be addressed in order to make the execution of applications on distributed resources time and cost efficient. It then formally defined and described the scheduling problem in the context of data intensive scientific workflows. This study helped identify the problem scenario and clearly defined the path for the thesis.

Following the problem definition based on motivational examples, this thesis presented state of the art work on scheduling and management techniques applied on data intensive workflow applications (see Chapter 2). It classified the techniques into categories according to data locality, data transfer, data footprint, granularity of tasks, execution model, and scheduling platforms; described several works under each category and identified their contributions and shortcomings. This survey provides comprehensive summary of the past work, which can be used by new techniques when conducting comparative performance analysis or making enhancements on algorithms.

In order to address the inadequacy of existing systems to handle data intensive applications, this thesis presented a workflow management system design (see Chapter 3). The WfMS leveraged the existing abstract workflow model [166] and added software components and scheduling algorithms specifically designed for managing data intensive applications. It described the components: workflow editor, the monitoring interface, resource provisioning interface, and interfaces for scientific applications that form the integral part of the workflow portal. It presented the design of workflow engine and its integration within Grid and Cloud computing platforms. The design was supported by two application case studies that used the WfMS when running on global Grids and Cloud resources. The study of WfMS design together with scientific applications proved to be very useful for testing the system performance and demonstrating the feasibility of the design on real platforms. This chapter formed a base where formulation and implementation of core scheduling algorithms could be possible.

This thesis then formulated a non-linear program based model to produce near-optimal schedules for the workflow scheduling problem described in Chapter 1 (see Chapter 4). It used AMPL to model the problem, and used a non-linear program solver to produce near-optimal results. It also demonstrated the need for new techniques for scheduling data intensive workflow applications by showing significant time and cost savings when using optimized scheduling heuristic. Although, the NLP-model based scheduling algorithm provided near-optimal solutions, it was computation intensive to converge to such solutions. Hence, it provided a base line results for comparison purposes. This study emphasizes and also experimentally demonstrates the fact that any WfMS should select multiple data hosts and specific compute resources while scheduling data intensive workflow applications on distributed resources. It also provides a mathematical model of the scheduling problem, which can be used for further enhancements of scheduling algorithms.

To circumvent the problem of algorithm computation time, this thesis presented heuristics based scheduling algorithms (see Chapter 5). In contrast to obtaining computationally expensive solutions (i.e. solution to NLP-model in Chapter 4), this chapter proposed list-based scheduling algorithms that provided approximate solutions to the workflow scheduling problem. It proposed enhanced static and dynamic heuristic that leveraged the multi-source parallel data retrieval technique while scheduling data and tasks of a workflow. The enhanced static mapping heuristic (ESMH) produced time-efficient schedules as compared to existing heuristics as it leveraged multi-source parallel data retrievals. For dynamic scheduling heuristic, it proposed to use Steiner tree based distributed data host selection approach. Using real scientific applications, it presented performance results that was carried out using emulation as well as real experiments on distributed resources. This study experimentally demonstrates the positive impact of using multi-source parallel data retrievals combined with dynamic scheduling heuristic, on data intensive scientific workflow applications. The result is time-efficient schedules as compared to existing algorithms that uses single best source for retrieving data.

Taking further enhancements in computation time and convergence to solutions, this thesis presented a particle swarm optimization based scheduling heuristic (see Chapter 6). The work modeled the scheduling problem using particles as tasks and their positions as task-resource mappings. It also used the multi-source parallel data retrieval technique for transferring data across distributed computing resources. The scheduling heuristic converged to the approximate solutions quicker than other heuristic and produced schedules that minimized total cost of execution when using Clouds. It demonstrated the tolerance and adaptability of the scheduling heuristic in producing cost-efficient schedules under changing execution and communication costs. This study focuses on further enhancing heuristics-based scheduling algorithms in terms of time complexity and accuracy. Building on the WfMS designed in Chapter 3, and the scheduling algorithms

implemented in subsequent chapters, this chapter presents a generic and novel approach for managing data and tasks of data intensive application workflows in Cloud computing environments.

## 7.3 Future Directions

Cloud computing introduces many challenges for system and application developers, engineers, system administrators, and service providers [7; 26; 34]. In this section, we will discuss some of these challenges related to the management and scheduling of data intensive application workflows on Clouds.

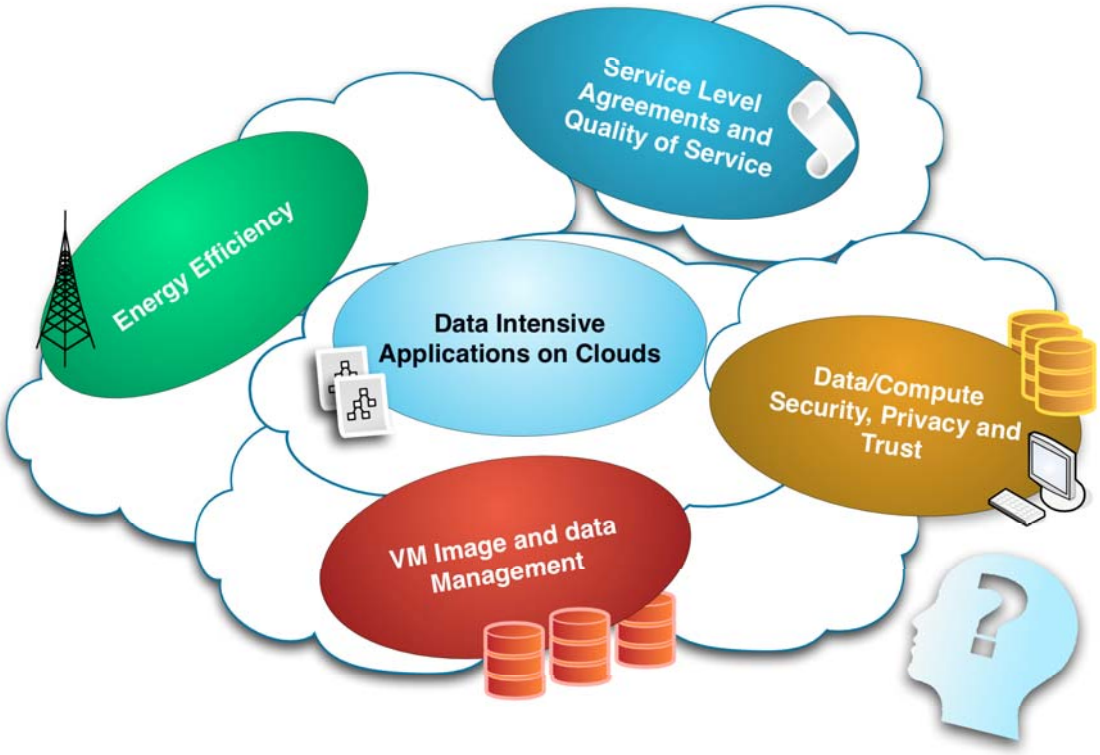



**Figure 7.1:** *Challenges for data intensive applications in Cloud environment.*

### 7.3.1 Scheduling Applications based on Security, Privacy and Trust

One of the major concerns when moving to Clouds is related to security, privacy, and trust. Security in particular, affects the entire Cloud computing stack [109]. The Cloud computing model promotes massive use of third party services and infrastructures to host important data or to perform critical operations. In this scenario, the trust towards providers is fundamental to ensure the desired level of privacy for applications hosted in

the Cloud. The obvious questions are:

- How to schedule tasks and data on the VMs managed by multiple Cloud service providers such that the flow of private data is restricted to only trusted resources through trusted networks?

- How to track the provenance of data and tasks in a virtualized environment? How can scheduling components use this information to apply filters and scanners that restrict security violations? [153]

One of the major concerns for the end users of Cloud computing services is the risk of leakage of data deployed to Cloud computing services. The VM nodes in a data center are managed by a virtual machine manager/resource allocator. As these virtual nodes are deployed on top of physical hardware, there is always a super user (privileged user) from the provider's side who has access to the VM state and the physical node. Any accidental or intentional access/leak of data processed by the VMs cannot be completely ruled out. Both data and computations are susceptible to attacks resulting from any intruder's VM inspection, unauthorized VM migrations to any physical nodes. In addition, many service providers engage in sharing of resources to offload part of its responsibilities (monitoring, identifying and accounting) to third party application vendors. In such cases, a customer's privacy is directly or indirectly affected by the functionality and terms of operation of those tertiary units.Thus the possibility of exposure and sharing of data should be taken into account while scheduling tasks and data on Cloud resources. The challenge is the introduction of these constraints into the scheduling of data and tasks while mapping tasks to resources in the Cloud.

Besides security, there are legal and regulatory issues that need to be taken care of before scheduling sensitive applications in the Cloud. When moving applications and data to the Cloud, the providers may choose to locate them in any of their data centers around the world. The physical location of data centers and clusters determines the set of laws that can be applied to the management of data. For example, specific cryptography techniques could not be used because they are not allowed in some countries. Simply, specific classes of users, such as banks, would not be comfortable to put their sensitive data into the Cloud, in order to protect their customers and their business. At present, a conservative approach is taken for hosting sensitive data. An interesting initiative is the concept of availability zones promoted by Amazon EC2. Availability zones identify a set of resources that have a specific geographic location. Although this initiative is mostly concerned with providing of better services in terms of isolation from failures, network latency, and service down-time, it limits the scheduling system from choosing compute resources and underlying network when making scheduling decisions for large applications which have data distributed at various locations.

### 7.3.2 Service Level Agreements and Quality of Service

Service Level Agreements define the functional and non-functional characteristics of Cloud services that is agreed by both the customer and the provider. The common parameters that define a SLA are: pricing model, usage model, resource metering, billing, and monitoring. In most cases, the desired level of security is also established within a SLA. When a service provider is unable to meet the terms stated in the SLA, a violation occurs. For example, an IaaS Cloud service provider may guarantee a minimum response time from a VM, minimum storage space, reliability of data, etc. However, if a customer does not get the desired response time, runs out of virtual disk space or suffers from frequent service disruptions, the SLA is violated. The SLA also defines a penalty model to compensate the customer in case of violations. At present, the adopted solution for pricing falls into the "pay-as-you-go" model, where users are charged according to the usage of the Cloud services. With constant changing of customer requirements, scheduling algorithms need to address the following challenges:

- How to make scheduling decisions such that Cloud service providers can guarantee QoS satisfactions and prevent SLA violations?

The notion of QoS satisfaction varies across customers as every user has its own requirements. Some general metrics from users' prospective are: amount of aggregate CPU power for the VMs instantiated, minimum bandwidth available, number and size of input/output devices (e.g. storage volumes, virtual hardware, etc.), average response time, etc. Typically, a customer is more inclined to request a statistical bound on most of these parameters than an average [160]. At the moment, no Cloud service providers are guaranteeing the minimum QoS for any of these metrics. From a provider's point of view, it still remains a challenge to provision, manage and predict the use of its Cloud services in the long run. That difficulty obstructs it to state concrete SLA terms (in writing) with its customers. With the increasing number of users, most violations are likely to happen during load fluctuations due to the lack of either sufficient resources or weakness in managing VMs at the providers side. In this direction, Patel et al. [107] have proposed a mechanism for managing SLAs in a Cloud computing environment using the Web Service Level Framework (WSLF) [71]. They propose using dynamic schedulers for measuring parameters, enabling measurements through third parties, and modeling penalties as financial compensations (moderated via a third party), to adapt web SLA to a Cloud environment. The complexity of enabling a SLA is higher in a multi-tenancy environment [159], where many businesses (i.e. tenants) have varying QoS requirements.

On one hand, there are technological challenges; on the other, there are issues with balancing usage cost and services delivered. Cloud service providers are already tussling by advertising attractive pricing policies for luring users of all kinds to use their services (e.g. Amazon, SalesForce, Google, etc.). As the market condition is determined through

tough competitions between vendors, dynamic negotiations and SLA management will play a major role in determining the amount of revenue to be generated for service providers. Similarly, users will be able to choose better services that fit their requirements and budget. They will be evaluating services based on their level of QoS satisfaction, so that they get the right value for the price paid.

### 7.3.3 Scheduling based on Energy Efficiency

Data centers are expensive to operate as they consume huge amount of power [85]. For instance, the combined energy consumption of all data centers worldwide is equivalent to the power consumption of Czech Republic. As a result, their carbon footprint on the environment is rapidly increasing. In order to address these issues, energy efficient resource allocation and algorithms need to be developed. The challenges are as follows:

- How to balance energy consumption and performance of data centers when making scheduling decisions?
- How to choose locality of data centers so that data security, operation cost, and energy consumption meet the terms in the SLA signed with users?

The performance of data centers depends on the provisioning and usage of its hardware devices by the VM management software depending on user needs. As more CPUs are used, the temperature of the hardware increases. This requires cooling of the data center. Hence, performance of the data center and energy consumption are directly related to each other. As the price for commodity hardware and network equipments for a data center is already getting cheaper, significant part of the total cost of operating Cloud services in industrial scale is determined by the amount of energy consumed by the data center. To conserve energy and save cooling costs, data centers could adopt energy efficient scheduling policies. Application schedulers could make use of statistical information obtained from sensors when submitting tasks and data for computation across VMs.

### 7.3.4 Management of VM Images and data as Workflows

Virtualization enables consolidation of servers for hosting one or more services on independent virtual machines in a multi-tenancy manner. When a large number of VMs are created they need to be effectively managed to ensure that services are able to deliver quality expectations of users. That means, VMs need to be migrated to suitable servers to ensure the desired QoS and later get consolidated dynamically to a fewer number of physical servers. Migration thus involves transferring of large amount of data across servers. The migration tasks themselves can overload the system when the migrations

happen or are forced regularly in a large data center. In order to prioritize the migrations, maintain a smooth transition of VMs without disrupting the service and also maintaining the user QoS, migration of tasks and data could be well represented as a workflow. These capabilities draw challenging questions:

- How do service providers process migration of VMs and large amount of data?

It is a customary practice not to disclose the amount of compute/storage resources a service provider has to its customers. On this setting, a customer may choose a particular provider solely based on its reputation and advertised capabilities. When the service provider gets large number of requests, it may have to overload its hardware to fulfill these requests. The challenge here is the capability to manage a sheer number of requests for VMs and the load on the infrastructure [132]. In order to handle these scenarios, providers scale out, replicate VMs [79], migrate VMs to under utilized resources, etc. However, there are software and hardware barriers when trying to instantiate large number of VMs in a data center [68]. Representing this complex process as a workflow and scheduling them is a challenge.

### 7.3.5 Data Intensive Applications on Clouds

At present there are numerous real-world applications that are running on distributed clusters around the world. However, only a few of them would be able to utilize Cloud resources with minor modifications. This is due to the fact that legacy applications were designed to operate on physical hardware with heavy optimizations targeting storage, input/output, communication etc. Cloud computing offers a different paradigm where traditional assumptions on hardware devices and software models may not always work. Input/output throughput, for example, may be different depending upon the location of the VM instance allocated for an application and the storage hardware used. Similarly, other attributes of an application such as: user experience, distribution, maintenance have new issues when applications are moved to Clouds. The questions that are important to ask before moving applications to Clouds are:

- How to map application attributes to Cloud attributes [36]?

Application attributes, such as data storage requirements, platform, network, distribution, security etc., may be related to different layers of the Cloud stack. These requirements could be satisfied by combining services from multiple Cloud vendors. But, combining different service providers brings along higher cost, risks, managerial difficulties and interoperability issues.

Market oriented computing in industry is getting real as evidenced by the plethora of vendors that provide Cloud computing services. For example, Amazon EC2 started with

flat pricing then moved to pricing based on service difference and recently introduced auction based models. In the next two decades, service-oriented distributed computing will emerge as a dominant factor in shaping the industry, changing the way business is conducted and how services are delivered and managed. This paradigm is expected to have a major impact on service economy, which contributes significantly towards GDP of many countries. The service sector includes health services (e-health), financial services and government services. With the increased demand for delivering services to a larger number of users, providers are looking for novel ways of hosting their application services in Clouds at lower cost while meeting the users quality of service expectations. With increased dependencies on ICT technologies in their realization, major advances are required in Cloud computing to support elastic applications offering services to millions of users, simultaneously.

Software licensing will be a major hurdle for vendors of Cloud services when proprietary software technologies have to be made available to millions of users via public virtual appliances (e.g. customized images of OS and applications). Overwhelming use of such customized software would lead to seamless integration of enterprise Clouds with public Clouds for service scalability and greater outreach to customers. More and more enterprises would be interested in moving to Clouds for cooperative sharing. In such scenarios, security and privacy of corporate data could be of paramount concern to these huge conglomerates. One of the solutions would be to establish a globally accredited Cloud service regulatory body that would act under a common statute for certifying Cloud service providers, standardizing data formats, enforcing service level agreements, handling trust certificates and so forth.

# REFERENCES


[1] H. Afsarmanesh, R. G. Belleman, A. S. Z. Belloum, A. Benabdelkader, J. F. J. van den Brand, G. B. Eijkel, A. Frenkel, C. Garita, D. L. Groep, R. M. A. Heeren, Z. W. Hendrikse, L. O. Hertzberger, J. A. Kaandorp, E. C. Kaletas, V. Korkhov, C. T. A. M. de Laat, P. M. A. Sloot, D. Vasunin, A. Visser, and H. H. Yakali, "Vlam-g: A grid-based virtual laboratory," *Scientific Programming*, vol. 10, no. 2, pp. 173–181, 2002.

[2] E. Alba, E.-G. Talbi, G. Luque, and N. Melab, *Parallel Metaheuristics: A New Class of Algorithms*, ser. Wiley Series on Parallel and Distributed Computing. Wiley, 2005, ch. 4. Metaheuristics and Parallelism, pp. 79–104.

[3] E. Alba, A. J. Nebro, and J. M. Troya, "Heterogeneous computing and parallel genetic algorithms," *Journal of Parallel and Distributed Computing*, vol. 62, no. 9, pp. 1362–1385, 2002.

[4] B. Allcock, I. Foster, V. Nefedova, A. Chervenak, E. Deelman, C. Kesselman, J. Lee, A. Sim, A. Shoshani, B. Drach, and D. Williams, "High-performance remote access to climate simulation data: a challenge problem for data grid technologies," in *Supercomputing '01: Proceedings of the 2001 ACM/IEEE conference on Supercomputing*. New York, NY, USA: ACM, 2001, pp. 46–46.

[5] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web Services - Concepts, Architectures and Applications*. Springer, November 2003.

[6] K. Amin, G. von Laszewski, M. Hategan, N. J. Zaluzec, S. Hampton, and A. Rossi, "GridAnt: A Client-Controllable Grid Work.ow System," in *HICSS '04: Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences - Track 7*. Washington, DC, USA: IEEE, 2004, p. 70210.3.

[7] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[8] E. M. Bahsi, E. Ceyhan, and T. Kosar, "Conditional workflow management: A survey and analysis," *Scientific Programming*, vol. 15, no. 4, pp. 283–297, 2007.

[9] E. Balas and M. W. Padberg, "On the set-covering problem," *Operations Research*, vol. 20, no. 6, 1972.

[10] R. S. Barga, D. Fay, D. Guo, S. Newhouse, Y. Simmhan, and 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*. New York, NY, USA: ACM, 2008, pp. 63–68.

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

[12] F. Berman, H. Casanova, A. Chien, K. Cooper, H. Dail, A. Dasgupta, W. Deng, J. Dongarra, L. Johnsson, K. Kennedy, C. Koelbel, B. Liu, X. Liu, A. Mandal, G. Marin,

M. Mazina, J. Mellor-Crummey, C. Mendes, A. Olugbile, J. M. Patel, D. Reed, Z. Shi, O. Sievert, H. Xia, and A. YarKhan, "New grid scheduling and rescheduling methods in the GrADS project," *International Journal of Parallel Programming*, vol. 33, no. 2, pp. 209–229, 2005.

[13] V. Bhat, M. Parashar, and 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*. Washington, DC, USA: IEEE, 2007, pp. 193–200.

[14] V. Bhat, M. Parashar, H. Liu, N. Kandasamy, M. Khandekar, S. Klasky, and S. Abdelwahed, "A self-managing wide-area data streaming service," *Cluster Computing*, vol. 10, no. 4, pp. 365– 383, 2007.

[15] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy, "Task scheduling strategies for workflow-based applications in grids," in *CCGRID '05: Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05) - Volume 2*. Washington, DC, USA: IEEE, 2005, pp. 759–767.

[16] I. Brandic, S. Pllana, and S. Benkner, "An approach for the high-level specification of QoS-aware grid workflows considering location affinity," *Scientific Programming*, vol. 14, no. 3,4, pp. 231–250, 2006.

[17] I. Brandic, S. Pllana, and S. Benkner, "Specification, planning, and execution of QoS-aware Grid workflows within the Amadeus environment," *Concurrency and Computation: Practice & Experience*, vol. 20, pp. 331–345, March 2008.

[18] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel Distributed Computing*, vol. 61, no. 6, pp. 810–837, 2001.

[19] L. D. Briceno, M. Oltikar, H. J. Siegel, and 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*. USA: IEEE, 2007, pp. 1–14.

[20] J. Broberg, R. Buyya, and Z. Tari, "MetaCDN: Harnessing 'storage clouds' for high performance content delivery," *Journal of Network and Computer Applications*, vol. 32, no. 5, pp. 1012–1022, 2009.

[21] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, *Ptolemy: a framework for simulating and prototyping heterogeneous systems*. Norwell, MA, USA: Kluwer Academic Publishers, 2002, pp. 527–543.

[22] R. Buyya, "Economic-based Distributed Resource Management and Scheduling for Grid Computing," Ph.D. dissertation, Monash University, Melbourne, Australia, April 2002.

[23] R. Buyya, D. Abramson, and J. Giddy, "An Economy Driven Resource Management Architecture for Global Computational Power Grids," in *Parallel and Distributed*

*Processing Techniques and Applications(PDPTA)*, H. R. Arabnia, Ed. CSREA Press, 2000.

[24] R. Buyya, S. Date, Y. Mizuno-Matsumoto, S. Venugopal, and D. Abramson, "Neuroscience instrumentation and distributed analysis of brain activity data: a case for escience on global grids," *Concurrency and Computation : Practice & Experience*, vol. 17, pp. 1783 – 1798, 2005.

[25] R. Buyya, S. Pandey, and C. Vecchiola, "Cloudbus Toolkit for Market-Oriented Cloud Computing," in *CloudCom '09: Proceedings of the 1st International Conference on Cloud Computing*, ser. LNCS, vol. 5931. Springer, Germany, December 2009, pp. 24–44.

[26] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[27] S. Callaghan, E. Deelman, D. Gunter, G. Juve, P. Maechling, C. Brooks, K. Vahi, K. Milner, R. Graves, E. Field, D. Okaya, and T. Jordan, "Scaling up workflow-based applications," *Journal of Computer and System Sciences*, vol. 76, no. 6, pp. 428–446, 2010.

[28] J. Cao, S. A. Jarvis, S. Saini, and G. R. Nudd, "GridFlow: Workflow Management for Grid Computing," in *CCGRID '03: Proceedings of the 3st International Symposium on Cluster Computing and the Grid*. Washington, DC, USA: IEEE, 2003, p. 198.

[29] F. Cappello and H. Bal, "Toward an International "Computer Science Grid"," in *CCGRID '07: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*. Washington, DC, USA: IEEE, 2007, pp. 3–12.

[30] C. Catlett, "The Philosophy of TeraGrid: Building an Open, Extensible, Distributed TeraScale Facility," in *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, ser. CCGRID '02. Washington, DC, USA: IEEE, 2002, pp. 8–.

[31] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: a distributed storage system for structured data," in *OSDI '06: Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*. Berkeley, CA, USA: USENIX Association, 2006, pp. 15–15.

[32] A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunszt, M. Ripeanu, B. Schwartzkopf, H. Stockinger, K. Stockinger, and B. Tierney, "Giggle: a framework for constructing scalable replica location services," in *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*. Los Alamitos, CA, USA: IEEE, 2002, pp. 1–17.

[33] A. Chervenak, E. Deelman, M. Livny, M.-H. Su, R. Schuler, S. Bharathi, G. Mehta, and K. Vahi, "Data placement for scientific applications in distributed environments," in *GRID '07: Proceedings of the 8th IEEE/ACM International Conference on Grid Computing*. Washington, DC, USA: IEEE, 2007, pp. 267–274.

[34] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina, "Controlling data in the cloud: outsourcing computation without outsourcing control," in *CCSW '09: Proceedings of the 2009 ACM workshop on Cloud computing security*. New York, NY, USA: ACM, 2009, pp. 85–90.

[35] P. Couvares, T. Kosar, A. Roy, J. Weber, and K. Wenger, "Workflow Management in Condor," in *Workflows for e-Science*, I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields, Eds. Springer London, 2007, pp. 357–375.

[36] M. Darryl Chantry, "Mapping applications to the cloud," Online, January 2009, http://msdn.microsoft.com/en-us/library/dd430340.aspx (Accessed:January 2010).

[37] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda, "Mapping Abstract Complex Workflows onto Grid Environments," *Journal of Grid Computing*, vol. V1, no. 1, pp. 25–39, March 2003.

[38] E. Deelman and A. Chervenak, "Data Management Challenges of Data-Intensive Scientific Workflows," in *CCGRID '08: Proceedings of the Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*. Washington, DC, USA: IEEE, 2008, pp. 687–692.

[39] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the cloud: the Montage example," in *SC '08: Proc. of the 2008 ACM/IEEE conference on Supercomputing*, Piscataway, NJ, USA, 2008, pp. 1–12.

[40] 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, and D. S. Katz, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, no. 3, pp. 219–237, 2005.

[41] D. R. Dreyer and M. L. Overton, "Two Heuristics for the Euclidean Steiner Tree Problem," *Journal of Global Optimization*, vol. 13, no. 1, pp. 95–106, 1998.

[42] R. Duan, T. Fahringer, R. Prodan, J. Qin, A. V. on, and M. Wieczorek, "Real world workflow applications in the askalon grid environment," in *European Grid Conference (EGC 2005)*, ser. Lecture Notes in Computer Science. Springer Verlag, February 2005.

[43] R. Duan, R. Prodan, and T. Fahringer, "Run-time optimisation of grid workflow applications," in *GRID*. IEEE, 2006, pp. 33–40.

[44] R. E. Ellis and T. M. Peters, "Medical image computing and computer-assisted intervention," in *MICCAI (2)*, 2003.

[45] C. B. et al., "Neurobase: Management of distributed and heterogeneous information sources in neuroimaging," in *Didamic Workshop, MICCAI 2004 Conference*, 2004.

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

[47] M. J. Fairman, A. R. Price, G. Xue, M. Molinari, D. A. Nicole, T. M. Lenton, R. Marsh, K. Takeda, and S. J. Cox, "Earth system modelling with Windows Workflow Foundation," *Future Generation Computer Systems*, vol. 25, no. 5, pp. 586–597, 2009.

[48] J. Feng and 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*. Washington, DC, USA: IEEE, 2004, pp. 359–366.

[49] T. A. Feo and M. G. Resende, "Greedy Randomized Adaptive Search Procedures," *Journal of Global Optimization*, vol. 6, no. 2, pp. 109–133, March 1995.

[50] P. C. Fishburn, *Interval Orders and Interval Graphs–A Study of Partially Ordered Sets*. USA: John Wiley, 1985.

[51] S. Fitzgerald, "Grid information services for distributed resource sharing," in *HPDC '01: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing*. USA: IEEE, 2001.

[52] I. Foster and C. Kesselman, "The Globus toolkit," in *The grid*, I. Foster and C. Kesselman, Eds. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 259–278.

[53] I. T. Foster, J.-S. Vöckler, M. Wilde, and Y. Zhao, "Chimera: Avirtual data system for representing, querying, and automating data derivation," in *SSDBM '02: Proceedings of the 14th International Conference on Scientific and Statistical Database Management*, 2002.

[54] R. Fourer, D. M. Gay, and B. W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press, November 2002.

[55] N. Furmento, W. Lee, A. Mayer, S. Newhouse, and J. Darlington, "ICENI: an open grid service architecture implemented with Jini," in *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, ser. Supercomputing '02. Los Alamitos, CA, USA: IEEE, 2002, pp. 1–10.

[56] A. Geppert, M. Kradolfer, and D. Tombros, "Market-based workflow management," *International Journal of Cooperative Information Systems, World Scientific Publishing Co*, vol. 7, 1997.

[57] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 29–43, 2003.

[58] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, and J. Myers, "Examining the Challenges of Scientific Workflows," *Computer*, vol. 40, no. 12, pp. 24–32, 2007.

[59] T. Glatard, J. Montagnat, D. Lingrand, and X. Pennec, "Flexible and Efficient Workflow Deployment of Data-Intensive Applications On Grids With MOTEUR," *International Journal of High Performance Computing Applications*, vol. 22, no. 3, pp. 347–360, 2008.

[60] K. Gupta, B. Nath, and R. Kotagiri, "Layered Approach Using Conditional Random Fields for Intrusion Detection," *Dependable and Secure Computing, IEEE Transactions on*, vol. 7, no. 1, pp. 35 –49, 2010.

[61] I. F. Haddad, "PVFS: A Parallel Virtual File System for Linux Clusters," *Linux Journal*, vol. 2000, November 2000.

[62] D. Hollingsworth, "The Workflow Reference Model," Workflow Management Coalition, Tech. Rep. TCOO- 1003, 1994.

[63] H.-J. Hoxer, K. Buchacker, and 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, September 2002, pp. 72–82.

[64] T. C. Hu, "Parallel sequencing and assembly line problems," *Operations Research*, vol. 9, no. 3, 1961.

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

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

[67] E. Jaeger, I. Altintas, J. Zhang, B. Ludäscher, D. Pennington, and W. Michener, "A scientific workflow approach to distributed geospatial data processing using web services," in *SSDBM'2005: Proceedings of the 17th international conference on Scientific and statistical database management*. Berkeley, CA, US: Lawrence Berkeley Laboratory, 2005, pp. 87–90.

[68] M. H. Jamal, A. Qadeer, W. Mahmood, A. Waheed, and J. J. Ding, "Virtual Machine Scalability on Multi-Core Processors Based Servers for Cloud Computing Workloads," in *NAS '09: Proceedings of the 2009 IEEE International Conference on Networking, Architecture, and Storage*. Washington, DC, USA: IEEE, 2009, pp. 90–97.

[69] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. P. Berman, and P. Maechling, "Data Sharing Options for Scientific Workflows on Amazon EC2," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '10. Washington, DC, USA: IEEE, 2010, pp. 1–9.

[70] R. Kalyanam, L. Zhao, T. Park, and S. Goasguen, "A Web Service-Enabled Distributed Workflow System for Scientific Data Processing," in *FTDCS '07: Proceedings of the 11th IEEE International Workshop on Future Trends of Distributed Computing Systems*. Washington, DC, USA: IEEE, 2007, pp. 7–14.

[71] A. Keller and H. Ludwig, "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services," *Journal of Network and Systems Management*, vol. 11, no. 1, pp. 57–81, 2003.

[72] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," in *IEEE International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948.

[73] C. Kesselman and I. Foster, *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, November 1998.

[74] L. Kleinrock, "An Internet Vision: The Invisible Global Infrastructure," *Ad Hoc Networks*, vol. 1, no. 1, pp. 3–11, July 2003.

[75] S. Y. Ko, R. Morales, and I. Gupta, "New Worker-Centric Scheduling Strategies for Data-Intensive Grid Applications," in *Middleware*, ser. Lecture Notes in Computer Science, R. Cerqueira and R. H. Campbell, Eds., vol. 4834. Springer, 2007, pp. 121–142.

[76] V. Korkhov, D. Vasyunin, A. Wibisono, A. S. Z. Belloum, M. A. Inda, M. Roos, T. M. Breit, and L. O. Hertzberger, "VLAM-G: Interactive data driven workflow engine for Grid-enabled resources," *Scientific Programming*, vol. 15, no. 3, pp. 173–188, 2007.

[77] T. Kosar and M. Livny, "Stork: Making Data Placement a First Class Citizen in the Grid," in *ICDCS '04: Proceedings of the 24th International Conference on Distributed Computing Systems*. Washington, DC, USA: IEEE, 2004, pp. 342–349.

[78] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Computing Surveys*, vol. 31, no. 4, pp. 406–471, 1999.

[79] H. A. Lagar-Cavilla, J. A. Whitney, A. M. Scannell, P. Patchin, S. M. Rumble, E. de Lara, M. Brudno, and M. Satyanarayanan, "SnowFlock: rapid virtual machine cloning for cloud computing," in *EuroSys '09: Proceedings of the 4th ACM European conference on Computer systems*. New York, NY, USA: ACM, 2009, pp. 1–12.

[80] J. Lee, B. Tierney, and W. E. Johnston, "Data Intensive Distributed Computing; A Medical Application Example," in *HPCN Europe '99: Proceedings of the 7th International Conference on High-Performance Computing and Networking*. London, UK: Springer-Verlag, 1999, pp. 150–158.

[81] W. Lee, S. J. Stolfo, and K. W. Mok, "A data mining framework for building intrusion detection models," in *IEEE Symposium on Security and Privacy*, 1999, pp. 120–132.

[82] T. J. Lehman, A. Cozzi, Y. Xiong, J. Gottschalk, V. Vasudevan, S. Landis, P. Davis, B. Khavar, and P. Bowman, "Hitting the distributed computing sweet spot with tspaces," *Computer Networks*, vol. 35, no. 4, pp. 457–472, 2001.

[83] Z. Lian, B. Jiao, and X. Gu, "A similar particle swarm optimization algorithm for job-shop scheduling to minimize makespan," *Applied Mathematics and Computation*, vol. 183, no. 2, pp. 1008 – 1017, 2006.

[84] D. T. Liu and M. J. Franklin, "GridDB: a data-centric overlay for scientific grids," in *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*. VLDB Endowment, 2004, pp. 600–611.

[85] L. Liu, H. Wang, X. Liu, X. Jin, W. B. He, Q. B. Wang, and Y. Chen, "GreenCloud: a new architecture for green data center," in *ICAC-INDST '09: Proceedings of the 6th international conference industry session on Autonomic computing and communications industry session*. New York, NY, USA: ACM, 2009, pp. 29–38.

[86] M. M. Lopez, E. Heymann, and 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*. Washington, DC, USA: IEEE, 2006, pp. 199–207.

[87] J. Louchet, M. Guyon, M. J. Lesot, and A. Boumaza, "Dynamic flies: a new pattern recognition tool applied to stereo sequence processing," *Pattern Recognition Letters*, vol. 23, no. 1-3, pp. 335–345, 2002.

[88] W. Z. Lu, H.-Y. Fan, A. Y. T. Leung, and J. C. K. Wong, "Analysis of pollutant levels in central hong kong applying neural network method with particle swarm optimization," *Environmental Monitoring and Assessment*, vol. 79, no. 3, pp. 217–230, Nov 2002.

[89] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the Kepler system: Research Articles," *Concurrency and Computation: Practice & Experience*, vol. 18, no. 10, pp. 1039–1065, 2006.

[90] D. Mahrenholz and 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*. Washington, DC, USA: IEEE, 2004, pp. 29–36.

[91] A. Mayer, S. McGough, N. Furmento, W. Lee, S. Newhouse, and J. Darlington, "Iceni dataflow and workflow: Composition and scheduling in space and time," in *UK e-Science All Hands Meeting*. IOP Publishing Ltd, 2003, pp. 627–634.

[92] E. Mayr and H. Stadtherr, "Optimal Parallel Algorithms for Two Processor Scheduling with Tree Precedence Constraints," Technische Universitat Munchen, Tech. Rep. Feb1-1, 1996.

[93] R. McClatchey, A. Anjum, H. Stockinger, A. Ali, I. Willers, and M. Thomas, "Scheduling in data intensive and network aware (diana) grid environments," *Journal of Grid Computing*, vol. 5, no. 1, pp. 43–64, 2007.

[94] L. Meyer, J. Annis, M. Wilde, M. Mattoso, and I. Foster, "Planning spatial workflows to optimize grid performance," in *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*. New York, NY, USA: ACM, 2006, pp. 786–790.

[95] Microsoft, "Windows Azure Platform .NET Services. http://www.microsoft.com/azure/netservices.mspx."

[96] R. Moore, T. A. Prince, and M. Ellisman, "Data-intensive computing and digital libraries," *Commun. ACM*, vol. 41, no. 11, pp. 56–62, 1998.

[97] H. Nakada, S. Matsuoka, K. Seymour, J. Dongarra, C. Lee, and Casanova, "A GridRPC Model and API for End-User Applications," GridRPC Working Group of Global Grid Forum, June 2007.

[98] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li, "Taverna: a tool for the composition and enactment of bioinformatics workflows," *Bioinformatics*, vol. 20, no. 17, pp. 3045–3054, November 2004.

[99] S. D. Olabarriaga, P. T. de Boer, K. Maheshwari, A. Belloum, J. G. Snel, A. J. Nederveen, and M. Bouwhuis, "Virtual lab for fmri: Bridging the usability gap," in *E-SCIENCE '06: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, 2006.

[100] C. u. O. Ourique, E. C. J. Biscaia, and J. C. Pinto, "The use of particle swarm optimization for dynamical analysis in chemical processes," *Computers and Chemical Engineering*, vol. 26, no. 12, pp. 1783–1793, 2002.

[101] S. Pandey and 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*. Washington, DC, USA: IEEE, 2008, pp. 548–553.

[102] S. Pandey, W. Voorsluys, M. Rahman, R. Buyya, J. E. Dobson, and K. Chiu, "A grid workflow environment for brain imaging analysis on distributed systems," *Concurrency and Computation: Practice & Experience*, vol. 21, no. 16, pp. 2118–2139, 2009.

[103] S. Pandey, L. Wu, S. M. Guru, and R. Buyya, "A Particle Swarm Optimization-Based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments," in *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications*, ser. AINA '10. Washington, DC, USA: IEEE, 2010, pp. 400–407.

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

[105] S.-M. Park and M. Humphrey, "Data throttling for data-intensive workflows," in *IPDPS '08: Proceedings of the 22nd International Parallel and Distributed Processing Symposium*. IEEE, 2008, pp. 1–11.

[106] K. E. Parsopoulos and M. N. Vrahatis, "Particle swarm optimization method in multiobjective problems," in *Proceedings of the 2002 ACM symposium on Applied computing*, ser. SAC '02. New York, NY, USA: ACM, 2002, pp. 603–607.

[107] P. Patel, A. Ranabahu, and A. Sheth, "Service level agreement in cloud computing," in *OOPSLA '09: Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*. New York, NY, USA: ACM, October 2009.

[108] C. Pautasso, "JOpera: An Agile Environment for Web Service Composition with Visual Unit Testing and Refactoring," in *VLHCC '05: Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing*. Washington, DC, USA: IEEE, 2005, pp. 311–313.

[109] S. Pearson, "Taking account of privacy when designing cloud computing services," in *CLOUD '09: Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*. Washington, DC, USA: IEEE, 2009, pp. 44–52.

[110] R. A. Poldrack and J. T. Devlina, "On the fundamental role of anatomy in functional imaging: Reply to commentaries on in praise of tedious anatomy," *NeuroImage*, vol. 37, pp. 1066–1068, 2007.

[111] R. Prodan and M. Wieczorek, "Bi-Criteria Scheduling of Scientific Grid Workflows," *Automation Science and Engineering, IEEE Transactions on*, vol. 7, no. 2, pp. 364 –376, April 2010.

[112] R. Prodan and 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*. New York, NY, USA: ACM, 2005, pp. 687–694.

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

[114] B. Quetier, V. Neri, and 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, June 2006.

[115] I. Raicu, Y. Zhao, I. T. Foster, and A. Szalay, "Accelerating large-scale data exploration through data diffusion," in *DADC '08: Proceedings of the 2008 international workshop on Data-aware distributed computing*. New York, NY, USA: ACM, 2008, pp. 9–18.

[116] A. Ramakrishnan, G. Singh, H. Zhao, E. Deelman, R. Sakellariou, K. Vahi, K. Blackburn, D. Meyers, and M. Samidi, "Scheduling data-intensiveworkflows onto storage-constrained distributed resources," in *CCGRID '07: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*. Washington, DC, USA: IEEE, 2007, pp. 401–409.

[117] L. Ramakrishnan and D. A. Reed, "Performability modeling for scheduling and fault tolerance strategies for scientific workflows," in *HPDC '08: Proceedings of the 17th international symposium on High performance distributed computing*. New York, NY, USA: ACM, 2008, pp. 23–34.

[118] K. Ranganathan and I. Foster, "Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications," in *HPDC '02: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*. Washington, DC, USA: IEEE, 2002, p. 352.

[119] K. Ranganathan and I. Foster, "Grid resource management," in *Grid resource management*. Norwell, MA, USA: Kluwer Academic Publishers, 2004, ch. Computation scheduling and data replication algorithms for data Grids, pp. 359–373.

[120] K. Ranganathan and I. T. Foster, "Identifying Dynamic Replication Strategies for a High-Performance Data Grid," in *GRID '01: Proceedings of the Second International Workshop on Grid Computing*. London, UK: Springer-Verlag, 2001, pp. 75–86.

[121] D. E. Rex, J. Q. Ma, and A. W. Arthur W. Toga, "The loni pipeline processing environment," *NeuroImage*, vol. 19, pp. 1033–1048, 2003.

[122] P. Rodriguez and E. W. Biersack, "Dynamic parallel access to replicated content in the internet," *IEEE/ACM Transactions on Networking*, vol. 10, no. 4, pp. 455–465, 2002.

[123] M. Romberg, "The UNICORE Architecture: Seamless Access to Distributed Resources," in *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing*, ser. HPDC '99. Washington, DC, USA: IEEE, 1999, pp. 44–.

[124] R. Sakellariou and 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, April 2004.

[125] R. Sakellariou and H. Zhao, "A low-cost rescheduling policy for efficient mapping of workflows on grid systems," *Scientific Programming*, vol. 12, pp. 253–262, December 2004.

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

[127] A. Salman, "Particle swarm optimization for task assignment problem," *Microprocessors and Microsystems*, vol. 26, no. 8, pp. 363–371, November 2002.

[128] R. Sandberg, D. Golgberg, S. Kleiman, D. Walsh, and B. Lyon, "Design and implementation of the Sun network filesystem," in *Innovations in Internetworking*. Norwood, MA, USA: Artech House, Inc., 1988, pp. 379–390.

[129] S. Shankar and 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*. New York, NY, USA: ACM, 2007, pp. 127–136.

[130] S. Shankar, A. Kini, D. J. DeWitt, and J. Naughton, "Integrating databases and workflow systems," *SIGMOD Record*, vol. 34, no. 3, pp. 5–11, 2005.

[131] Y. L. Simmhan, B. Plale, and D. Gannon, "A survey of data provenance in e-science," *SIGMOD Record*, vol. 34, no. 3, pp. 31–36, September 2005.

[132] A. Singh, M. Korupolu, and D. Mohapatra, "Server-storage virtualization: integration and load balancing in data centers," in *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. Piscataway, NJ, USA: IEEE Press, 2008, pp. 1–12.

[133] G. Singh, C. Kesselman, and E. Deelman, "Optimizing Grid-Based Workflow Execution," *Journal of Grid Computing*, vol. 3, no. 3-4, pp. 201–219, September 2005.

[134] G. Singh, K. Vahi, A. Ramakrishnan, G. Mehta, E. Deelman, H. Zhao, R. Sakellariou, K. Blackburn, D. Brown, S. Fairhurst, D. Meyers, G. B. Berriman, J. Good, and D. S. Katz, "Optimizing workflow data footprint," *Scientific Programming*, vol. 15, no. 4, pp. 249–268, 2007.

[135] S. Smith, M. Jenkinson, M. Woolrich, C. Beckmann, T. Behrens, H. Johansen-Berg, P. Bannister, M. D. Luca, I. Drobnjak, D. Flitney, R. Niazy, J. Saunders, J. Vickers, Y. Zhang, N. D. Stefano, J. Brady, and P. Matthews, "Advances in functional and structural MR image analysis and implementation as FSL," *NeuroImage*, vol. 23, pp. S208–S219, 2004.

[136] T. Sousa, A. Silva, and A. Neves, "Particle swarm based data mining algorithms for classification tasks," *Parallel Computing*, vol. 30, no. 5-6, pp. 767–783, 2004.

[137] P. Spellucci, "A sqp method for general nonlinear programs using only equality constrained subproblems," *Mathematical Programming*, vol. 82, pp. 413–448, 1993.

[138] M. F. Tasgetiren, Y.-C. Liang, M. Sevkli, and G. Gencyilmaz, "A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem," *European Journal of Operational Research*, vol. 177, no. 3, pp. 1930–1947, March 2007.

[139] I. Taylor, M. Shields, I. Wang, and R. Philp, "Distributed P2P Computing within Triana: A Galaxy Visualization Test Case," in *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, ser. IPDPS '03. Washington, DC, USA: IEEE, 2003, p. 16a.

[140] I. Taylor, I. Wang, M. Shields, and S. Majithia, "Distributed computing with Triana on the Grid: Research Articles," *Concurrency and Computation: Practice & Experience*, vol. 17, no. 9, pp. 1197–1214, 2005.

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

[142] A. Tiwari and A. K. T. Sekhar, "Workflow based framework for life science informatics," *Computational Biology and Chemistry*, vol. 31, no. 5-6, pp. 305–319, 2007.

[143] H. Topcuouglu, S. Hariri, and M.-y. Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[144] J. D. Ullman, "NP-complete scheduling problems," *Journal of Computer and System Sciences*, vol. 10, pp. 384–393, June 1975.

[145] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *SIGCOMM Computer Communication Review*, vol. 39, pp. 50–55, December 2008.

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

[147] C. Vecchiola, X. Chu, and R. Buyya, *High Speed and Large Scale Scientific Computing*. IOS Press, 2009, ch. Aneka: A Software Platform for .NET-based Cloud Computing, pp. 267–295, iSBN: 978-1-60750-073-5.

[148] C. Vecchiola, M. Kirley, and R. Buyya, "Multi-Objective Problem Solving With Offspring on Enterprise Clouds," in *Proceedings of the 10th International Conference on High-Performance Computing in Asia-Pacific Region (HPC Asia 2009)*. Kaohsiung, Taiwan: IEEE, 2009.

[149] C. Vecchiola, S. Pandey, and R. Buyya, "High-performance cloud computing: A view of scientific applications," in *ISPAN '09: Proceedings of the 2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks*. Washington, DC, USA: IEEE, 2009, pp. 4–16.

[150] S. Venugopal and 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*. Washington, DC, USA: IEEE, 2006, pp. 238–245.

[151] S. Venugopal, R. Buyya, and K. Ramamohanarao, "A taxonomy of Data Grids for distributed data sharing, management, and processing," *ACM Computing Surveys*, vol. 38, no. 1, 2006.

[152] S. Venugopal, R. Buyya, and L. Winton, "A Grid service broker for scheduling e-Science applications on global data Grids," *Concurrency and Computation: Practice & Experience*, vol. 18, no. 6, pp. 685–699, May 2006.

[153] J. Wei, X. Zhang, G. Ammons, V. Bala, and P. Ning, "Managing security of virtual machine images in a cloud environment," in *CCSW '09: Proceedings of the 2009 ACM workshop on Cloud computing security*. New York, NY, USA: ACM, 2009, pp. 91–96.

[154] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. San Francisco: Morgan Kaufmann, 2005.

[155] R. Wolski, N. T. Spring, and J. Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing," *Future Generation Computing Systems*, vol. 15, no. 5-6, pp. 757–768, 1999.

[156] B. Wong, A. Slivkins, and E. G. Sirer, "Meridian: A Lightweight Network Location Service without Virtual Coordinates," in *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2005, pp. 85–96.

[157] R. X. Wu and 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*. Washington, DC, USA: IEEE, 2004, pp. 228–238.

[158] Y.-S. Wu, B. Foo, Y. Mei, and S. Bagchi, "Collaborative intrusion detection system (cids): A framework for accurate and efficient ids," in *Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC 03)*, Purdue University. Applied Computer Security Associates, 2003. [Online]. Available: http://www.ece.purdue.edu/~sbagchi/Research/Papers/cids_acsac03.pdf

[159] E. Wustenhoff, "Service level agreement in the data center," Online, April 2002, http://www.sun.com/blueprints/0402/sla.pdf. [Online]. Available: http://www.sun.com/blueprints/0402/sla.pdf

[160] K. Xiong and H. Perros, "Service Performance and Analysis in Cloud Computing," in *SERVICES '09: Proceedings of the 2009 Congress on Services - I*. Washington, DC, USA: IEEE, 2009, pp. 693–700.

[161] L. Yang, J. M. Schopf, and 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*. Washington, DC, USA: IEEE, 2005, pp. 734–742.

[162] P.-Y. Yin, S.-S. Yu, and Y.-T. Wang, "A hybrid particle swarm optimisation algorithm for optimal task assignment in distributed systems," *Computer Standards and Interfaces*, vol. 28, no. 4, pp. 441–450, 2006.

[163] H. Yoshida, K. Kawata, Y. Fukuyama, and Y. Nakanishi, "A particle swarm optimization for reactive power and voltage control considering voltage stability," in *the International Conference on Intelligent System Application to Power System*, 1999, pp. 117–121.

[164] B. Yu, X. Yuan, and J. Wang, "Short-term hydro-thermal scheduling using particle swarm optimisation method," *Energy Conversion and Management*, vol. 48, no. 7, pp. 1902–1908, 2007.

[165] J. Yu, "QoS-based Scheduling of Workflows on Global Grids," Ph.D. dissertation, University of Melbourne, October 2007. [Online]. Available: http://www.cloudbus.org/students/JiaYuPhDThesis.pdf

[166] J. Yu and 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*. Washington, DC, USA: IEEE, 2004, pp. 119–128.

[167] J. Yu and R. Buyya, "A Taxonomy of Workflow Management Systems for Grid Computing," *Journal of Grid Computing*, vol. 3, no. 3, pp. 171–200, September 2005.

[168] J. Yu and R. Buyya, "Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms," *Scientific Programming*, vol. 14, pp. 217–230, December 2006.

[169] J. Yu, R. Buyya, and K. Ramamohanarao, "Workflow Scheduling Algorithms for Grid Computing," in *Metaheuristics for Scheduling in Distributed Computing Environments*, ser. Studies in Computational Intelligence. Springer Berlin / Heidelberg, 2008, vol. 146, pp. 173–214.

[170] J. Yu, R. Buyya, and C. K. Tham, "Cost-Based Scheduling of Scientific Workflow Application on Utility Grids," in *Proceedings of the First International Conference on e-Science and Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 140–147.

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

[172] A. E. M. Zavala, A. H. Aguirre, E. R. Villa Diharce, and S. B. Rionda, "Constrained optimisation with an improved particle swarm optimisation algorithm," *Intl. Journal of Intelligent Computing and Cyvernetics*, vol. 1, no. 3, pp. 425–453, 2008.

[173] E. W. Zegura, K. L. Calvert, and 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, March 1996, pp. 594–602 vol.2.

[174] M. Zeller, R. Grossman, C. Lingenfelder, M. R. Berthold, E. Marcade, R. Pechter, M. Hoskins, W. Thompson, and R. Holada, "Open standards and cloud computing: KDD-2009 panel report," in *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. NY, USA: ACM, 2009, pp. 11–18.

[175] L. Zhang, Y. Chen, R. Sun, S. Jing, and B. Yang, "A task scheduling algorithm based on pso for grid computing," *International Journal of Computational Intelligence Research*, vol. 4, no. 1, 2008.

[176] Y. Zhao, J. Dobson, I. Foster, L. Moreau, and M. Wilde, "A notation and system for expressing and executing cleanly typed workflows on messy scientific data," *SIGMOD Record*, vol. 34, no. 3, pp. 37–43, 2005.

[177] Y. Zhao, I. Raicu, and I. Foster, "Scientific Workflow Systems for 21st Century, New Bottle or New Wine?" in *SERVICES '08: Proceedings of the 2008 IEEE Congress on Services - Part I*. Washington, DC, USA: IEEE, 2008, pp. 467–471.

[178] X. Zhou, E. Kim, J. W. Kim, and 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*. Washington, DC, USA: IEEE, 2006, pp. 446–453.

# A

# Case Studies of Scientific Applications

Appendix A lists the implementation code of the three scientific applications that this thesis has used in the experiments. The codes are limited to application logic and corresponding workflow representation only.

## A.1 fMRI Image Registration Application

The *bash* code that is used to normalize the fMRI images is given in listing A.1. This code was obtained from the Dartmouth Brain Imaging Center at the Dartmouth College, the University of Chicago.

**Listing A.1:** *Bash code for fMRI Image Registration application*

```
#!/bin/bash -x

TARGET=`ls -1 */ANATOMY/hires.hdr | head -1`
target_dir=`dirname $TARGET`

rm -rf AIR && mkdir AIR
rm -rf RESLICED && mkdir RESLICED
rm -rf ATLAS && mkdir ATLAS
rm */ANATOMY/bhires*
#
echo "BEGIN: `date`"
#
for SOURCE in `ls -1 */ANATOMY/hires.hdr`; do
  subject_dir=`dirname $SOURCE`
  subject=`dirname $subject_dir`
  bet ${SOURCE} ${subject_dir}/bhires -R
  fslmaths ${subject_dir}/bhires ${subject_dir}/bhires -odt short
  makeaheader ${subject_dir}/bhires.hdr 3 256 256 124 .9375 .9375 1.2
  alignlinear ${target_dir}/bhires.hdr ${subject_dir}/bhires.hdr  \
                                        AIR/${subject}.air -m 12
done
#
cd AIR
definecommon_air null null null  .aircommon y *.air
cd ..

for SOURCE in `ls -1 */ANATOMY/bhires.hdr`; do
  subject_dir=`dirname $SOURCE`
  subject=`dirname $subject_dir`
  reslice AIR/${subject}.air.aircommon RESLICED/${subject}-reslice -a ${SOURCE}
done
#
cd RESLICED
softmean atlas-linear y null *reslice.img
#
for SOURCE in `ls *.hdr`; do
```

```
 subject=`basename $SOURCE .hdr`
 align_warp atlas-linear $SOURCE ${subject}.warp -m 6 -q
 reslice_warp ${subject}.warp ${subject}-warp
done

softmean ../ATLAS/atlas y null *reslice-warp.img

cd ..


TARGET=ATLAS/atlas.hdr
for SOURCE in `ls -1 */ANATOMY/bhires.hdr`; do
  subject_dir=`dirname $SOURCE`
  alignlinear ${TARGET}  ${SOURCE} ${subject_dir}/hires.air -m 12 \
              -t1 1000 -t2 1000 -s 81 3 3
  align_warp  ${TARGET} ${SOURCE} ${subject_dir}/hires.warp -m 12 \
              -f ${subject_dir}/hires.air -t1 1000 -t2 1000 -s 81 3 3 -q
  reslice_warp ${subject_dir}/hires.warp ${subject_dir}/nhires
done

#
# determine quality of fit to atlas
#

bet ../../ATLAS/atlas ../../ATLAS/batlas
bet nhires nbhires

flirt -ref ../../ATLAS/batlas.hdr -in nbhires.hdr -out subject_01_fit -applyxfm

slicer  subject_01_fit ../../ATLAS/batlas.hdr -s 2 -x 0.35 sla.png -x 0.45 \
          slb.png -x 0.5 slc.png -x 0.65 sld.png -y 0.35 sle.png -y 0.45 \
          slf.png -y 0.55 slg.png -y 0.65 slh.png -z 0.35 sli.png -z 0.45 \
          slj.png -z 0.55 slk.png -z 0.65 sll.png


pngappend       sla.png + slb.png + slc.png + sld.png + sle.png + slf.png \
                + slg.png + slh.png + sli.png + slj.png + slk.png + sll.png \
                subject_01_fit1.png


slicer ../../ATLAS/batlas.hdr subject_01_fit -s 2 -x 0.35 sla.png -x 0.45 \
        slb.png -x 0.5 slc.png -x 0.65 sld.png -y 0.35 sle.png -y 0.45 slf.png \
       -y 0.55 slg.png -y 0.65 slh.png -z 0.35 sli.png -z 0.45 slj.png -z 0.55 \
       slk.png -z 0.65 sll.png


pngappend sla.png + slb.png + slc.png + sld.png + sle.png + slf.png + \
                slg.png + slh.png + sli.png + slj.png + slk.png + \
                sll.png subject_01_fit2.png

pngappend subject_01_fit1.png - subject_01_fit2.png subject_01_fit.png

echo "DONE: `date`"
```

This thesis designed a workflow using the above application code. This workflow is depicted in Figure A.1 as a series of tasks interlinked according to the data dependencies. An elaborate version of the same workflow is given in Figure 3.13.

The workflow depicted as a DAG in Figure A.1 is formed using XML when submitting to the workflow management system. This XML is formed using the xWfl described in Chapter 3. Listing A.2 presents the XML code for a task as an example. When adding more subjects to the workflow, we can add tasks for each additional subject at each level. For example, "GroupI1" and "GroupI2" are tasks at the first level. An additional task for a new subject would be named "Group13" added alongside these two tasks. However, the synchronizing tasks (a task that accepts the outputs from several tasks) do not get added

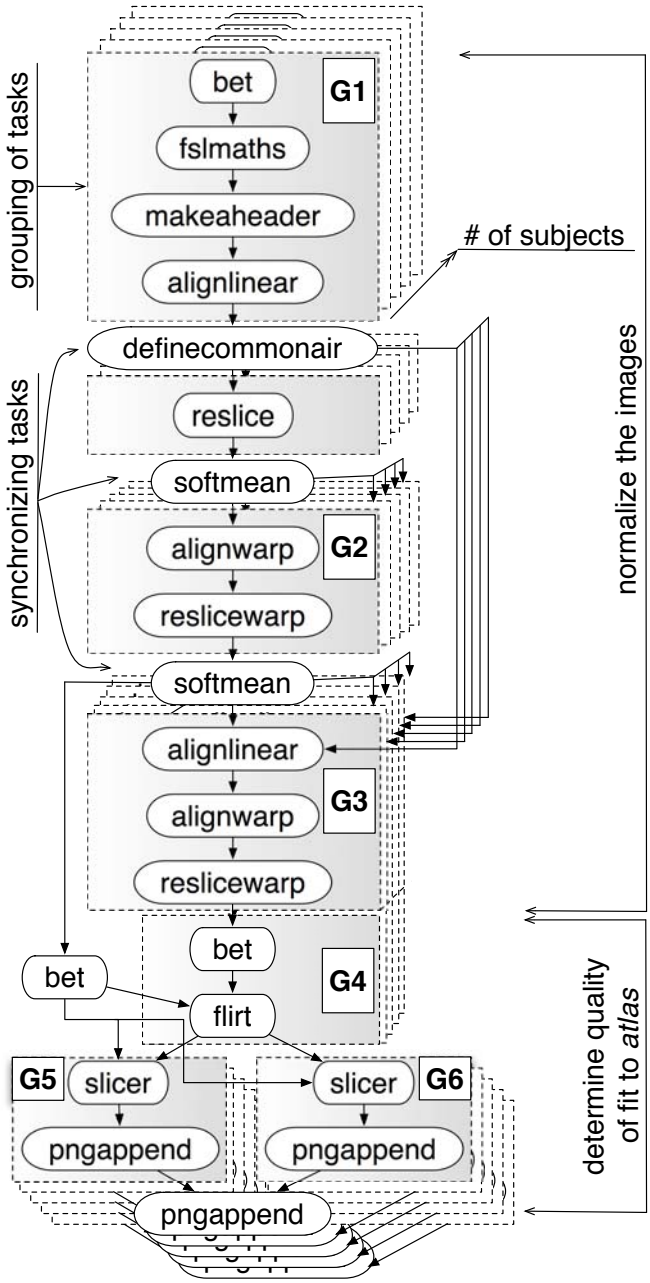



**Figure A.1:** *Image Registration application as a workflow.*

for changing number of subjects (e.g. definecommonair). However, for every addition of subject, these synchronizing tasks accept one additional file as input that was produced by the added task.

**Listing A.2:** *XML code for fMRI Image Registration using xWfl schema*

```
<task name="Group11" paramsweep="false">
        <paras>
                <para name="X" type="integer" domain="range">
                        <range from="1" to="1" type="step" interval="1" />
                </para>
        </paras>
```

```
                <executable>
                        <name value="GroupI1" I_OModel="many_many" />
                        <service serviceID="a1" />
                        <input>
                                <port number="0" type="file" value="$Xhires.img" arg="false" />
                                <port number="1" type="file" value="$Xhires.hdr" arg="true" />
                                <port number="2" type="msg" value="$Xbhires" arg="true" />
                                <port number="3" type="msg" value="$X.air" arg="true" />
                        </input>
                        <output>
                                <port number="4" type="file" value="$X.air" />
                                <port number="5" type="file" value="$Xbhires.hdr" />
                                <port number="6" type="file" value="$Xbhires.img" />
                        </output>
                </executable>
        </task>
```

## A.2 Intrusion Detection Application

The intrusion detection application performs datamining tasks on three sets of user supplied data: training set, testing set and the real-time data set. These operations are carried out with the help of Weka library for Java. The command line code is given in the listing A.3.

**Listing A.3:** *Command line code for Intrusion Detection application*

```
Step 1:
java    -Xmx1024M -classpath "./weka.jar" \
                weka.attributeSelection.CfsSubsetEval \
                 -S weka.attributeSelection.ExhaustiveSearch -I data/train.arff

Step 2:
java -Xmx1024M -classpath "./weka.jar" \
                weka.filters.unsupervised.attribute.Remove \
                -R 2-3 -I data/train.arff > data/filteredtrain.arff

java -Xmx1024M -classpath "./weka.jar"
                weka.filters.unsupervised.attribute.Remove \
                -R 2-3 -I data/test.arff > data/filteredtest.arff

Step 3a:
java -Xmx1024M -classpath "./weka.jar" \
                weka.classifiers.bayes.NaiveBayes \
                -no-cv -v -o -t data/filteredtrain.arff -d bayes.model

Step 4a:
java -Xmx1024M -classpath "./weka.jar" \
                weka.classifiers.bayes.NaiveBayes \
                -no-cv -v -o -T data/filteredtest.arff -l bayes.model


Step 3b:
java -Xmx1024M -classpath "./weka.jar" \
                weka.classifiers.trees.J48 -no-cv -v \
                -o -t data/filteredtrain.arff -d decisiontrees.model

Step 4b:

java -Xmx1024M -classpath "./weka.jar" \
                weka.classifiers.trees.J48 -no-cv -v \
                -o -T data/filteredtest.arff -l decisiontrees.model

Step 3c:
java -Xmx1024M -classpath "./weka.jar" \
                weka.classifiers.functions.SMO -no-cv -v \
                -o -t data/filteredtrain.arff -d supportvectormachines.model

Step 4c:
```

```
java -Xmx1024M -classpath "./weka.jar" \
               weka.classifiers.functions.SMO -no-cv -v \
               -o -T data/filteredtest.arff -l supportvectormachines.model

Testing Phase:

//sample 5% of the dataset supervised
java -Xmx1024M -cp weka.jar \
               weka.filters.supervised.instance.Resample \
               -i data/train.arff -o train-5%.arff -c last -Z 5

//random selection
java -cp weka.jar -Xmx2048M \
               weka.filters.unsupervised.instance.Resample \
               -i data/train.arff -o train-random-5pc.arff -Z 5
```

In this thesis, we designed a workflow that uses the above code for processing intrusion detection code. This workflow represented as a DAG is depicted in Figure 4.3. Similar to the fMRI workflow described above, the intrusion detection workflow could well be expanded to include multiple training, testing and real-time log files by adding branches of code in the same level. This process is made lot simpler using the workflow editor described in Chapter 3.

## A.3 Distributed Evolutionary Multi-Objective Algorithms

An example execution code for processing the distributed evolutionary multi-objective algorithm is given in listing A.4.

**Listing A.4:** *Command line code for EMO application*

```
emo -t:2 -g:100 -i:100 -oa:1.archive
emo -t:2 -g:100 -i:100 -oa:2.archive
emo -t:2 -g:100 -i:100 -oa:3.archive
emo -t:2 -g:100 -i:100 -oa:4.archive

emomerge -t:2 -i:pops1.in -o:%d.out -n:4

emo -t:2 -g:100 -i:100 -oa:21.archive -ip:0.out
emo -t:2 -g:100 -i:100 -oa:22.archive -ip:1.out
emo -t:2 -g:100 -i:100 -oa:23.archive -ip:2.out
emo -t:2 -g:100 -i:100 -oa:24.archive -ip:3.out

emomerge -t:2 -i:pops2.in -o:final.out -f:false -n:1

emo -t:2 -i:100 -g:100 -oa:final.archive -ip:final.out

# The file pops1.in contains the list of input files
1.archive
2.archive
3.archive
4.archive

# The file pops2.in contains the list of input files
21.archive
22.archive
23.archive
24.archive
```

The above code only depicts a three level iteration. The number of iterations and divisions after each merge operation depends upon the user's application requirements. In this thesis, we constructed a workflow using the above code as depicted in Figure A.2.

Similar to the fMRI and Intrusion Detection applications, this workflow was also designed using the xWfl language described in Chapter 3.

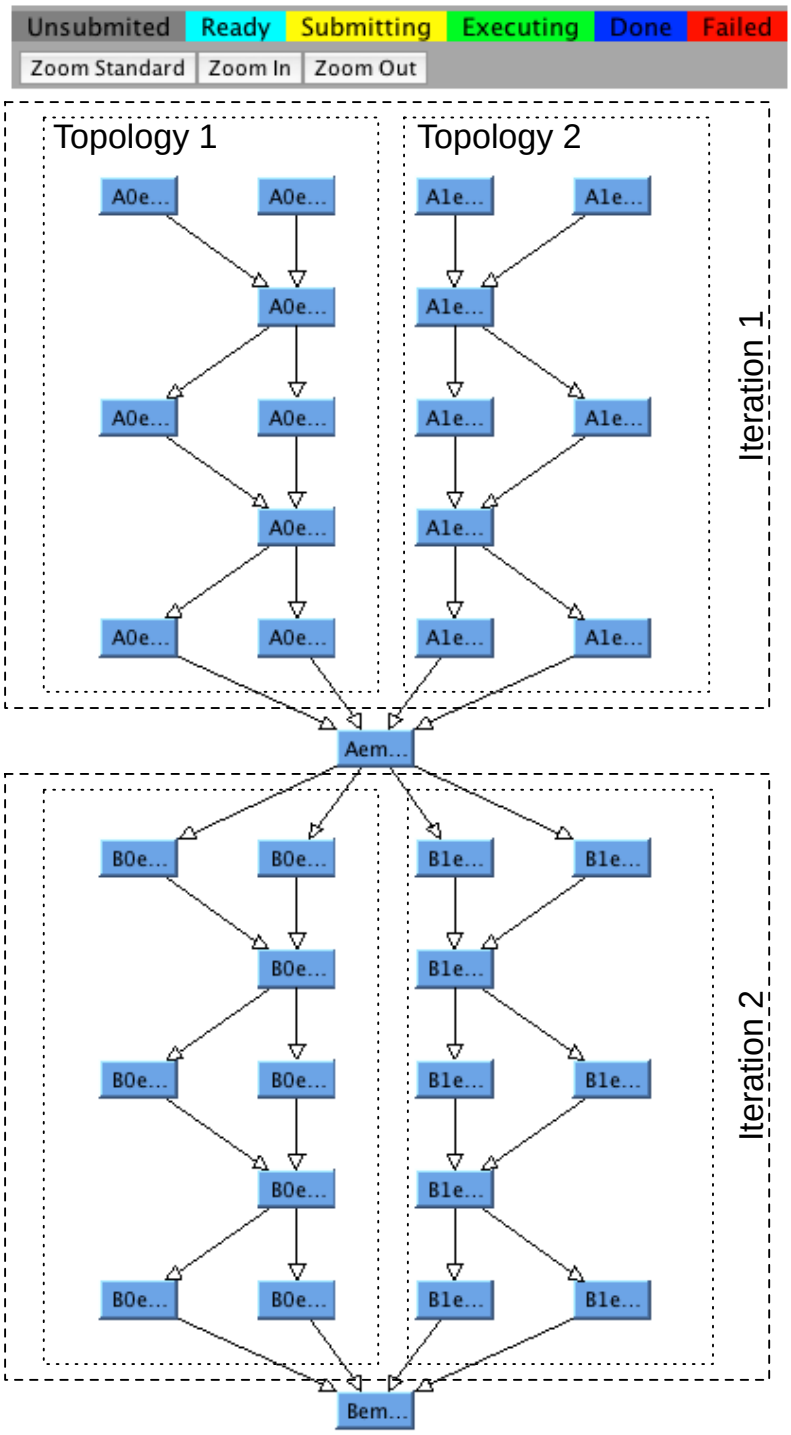



**Figure A.2:** *EMO workflow structure (boxes represent task, arrows represent data-dependencies between tasks)*

In order to parallelize the execution of EMO, we construct a workflow model for

systematically executing the tasks. A typical workflow structure is depicted in Figure A.2.

In our case study, the EMO application consists of 5 different topologies, upon which the iteration is done. These topologies are defined in 5 different binary files. Each file becomes the input files for the top level tasks (A0emo1, A0emo, . . . ). We create a separate branch for each topology file. In the Figure A.2, there are two branches, which get merged on level 6. The tasks at the root level operate on the topologies to create new population, which is then merged by the task named "emomerge". In Figure A.2, we see two "emomerge" tasks in the second level; 1 task in the 6th level that merges two branches and then splits the population to 2 branches again; 2 on 8th and 10th level and the final task on 12th level. In the example figure, each topology is iterated 2 times in a branch before getting merged. The merged population is then split. This split is done two times in the figure. The tasks labeled B0e, B1e is the start of 2nd iteration.

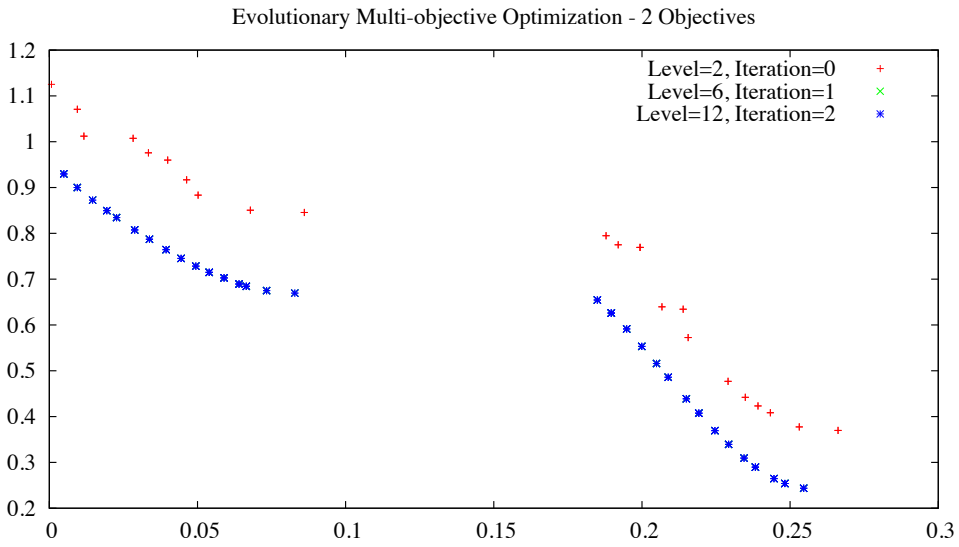



**Figure A.3:** *A graph that plots the pareto-front obtained after executing EMO for ZTD2 test problem*

We use ZDT2 [148] as a test function for the objective function. After executing the EMO workflow, we expect to see optimized values for the two objectives given by the ZDT2 test function. Figure A.3 depicts the graph that plots the front obtained after iterating the EMO workflow depicted in Figure A.2.At the zeroth iteration (level 2) as depicted in Figure A.3, the front does not give the optimal values. After the first iteration is complete, the front approaches the optimized values. Iteration 2 does not significantly change the front, hence the data overlaps for Iteration 1 and 2.

# B
# Non-linear Programming

Appendix B lists the routines in the Modeling Language for Mathematical Programming (AMPL) [54] that was used for solving our NLP-model listed in Chapter 4.

**Listing B.1:** *AMPL Model for solving the NLP problem*

```
set S;
set P;
set T;

var y {k in T, j in P} >=0;
var d {i in S, j in P, k in T} >=0;

param ecost     {k in T, j in P} >=0;
param etime     {k in T, j in P} >=0;

param tdata     { k in T } > 0;

param txcost    {i in S, j in P} >=0;
param txtime    {i in S, j in P} >=0;


minimize Total_cost: sum {i in S, j in P, k in T} ( d[i,j,k] * txcost[i,j]*y[k,j]
                                + ecost[k,j] *etime[k,j]* y[k,j] );

#Objective for consideration
#minimize Total_time: sum {i in S, j in P, k in T} ( d[i,j,k] * txtime[i,j]*y[k,j]
                                  + etime[k,j] * y[k,j] );


#
#       y denotes if a task k is executed in processor j.
#
subject to tasks_execution {k in T}: sum {j in P} y[k,j] = 1 ;


#
#       tdata is the total data each task needs.
#
subject to data_per_task {k in T}: sum {i in S, j in P} y[k,j]*d[i,j,k] = tdata[k];
subject to totaltx : sum {i in S, j in P, k in T} y[k,j]*d[i,j,k] = sum {k in T}tdata[k];

#extensions
#subject to txdata {i in S,k in T, j in P}: y[k,j] * d[i,j,k] <= tdata[k];
#subject to txprocess {k in T,i in S, j in P}: ( d[i,j,k] * y[k,j]  ) >=0;
```

The model represented in listing B.1 uses the data distribution presented in listing B.2.

**Listing B.2:** *Example data distribution used by AMPL model for solving the NLP problem*

```
set S:=         rotegg aquila tsukuba omii snowball node belle manjra;
set P:=         roteggp aquilap tsukubap omiip snowballp nodep bellep manjrap;
```

```
set T:=          as ftrain nbtrain dttrain       ;

param ecost: roteggp aquilap tsukubap omiip snowballp nodep       bellep manjrap :=

as              0.0016   0.0016   0.013    0.006    0.013    0.006    0.006    0.006
ftrain          0.0016   0.0016   0.013    0.006    0.013    0.006    0.006    0.006
nbtrain         0.0016   0.0016   0.013    0.006    0.013    0.006    0.006    0.006
dttrain         0.0016   0.0016   0.013    0.006    0.013    0.006    0.006    0.006    ;



param etime:    roteggp  aquilap  tsukubap omiip   snowballp nodep  bellep   manjrap :=

as              25       30       15       20       14       21       22       18
ftrain          110      120      70       55       30       50       58       33
nbtrain         115      122      75       57       36       56       65       35
dttrain         185      190      150      130      122      125      144      120      ;

param txcost:   roteggp  aquilap  tsukubap omiip  snowballp nodep  bellep  manjrap   :=

rotegg          0        0.1      0.1      0.1      0.1      0.1      0.1      0.1
aquila          0.1      0        0.1      0.1      0.1      0.1      0.1      0.1
tsukuba         0.1      0.1      0        0.1      0.1      0.1      0.1      0.1
omii            0.1      0.1      0.1      0        0.1      0.1      0.1      0.1
snowball        0.1      0.1      0.1      0.1      0        0.1      0.1      0.1
node            0.1      0.1      0.1      0.1      0.1      0        0.1      0.1
belle           0.1      0.1      0.1      0.1      0.1      0.1      0        0.1
manjra          0.1      0.1      0.1      0.1      0.1      0.1      0.1      0        ;


param txtime:   aup      jpp      usp      :=

au              10       500      400
jp              50       10       200
us              250      200      10;



param tdata     :=
as              100
ftrain          100
nbtrain         100
dttrain         100 ;
```

The model and the data can now be executed using the code presented in listing B.3

**Listing B.3:** *Code for executing the AMPL model for solving the NLP problem*

```
model m-t1.txt
data d-id1.txt
option solver donlp2;
option donlp2_options "maxit=4000";
solve;
expand;


display y ;
display d ;

display y > locations.txt;
display d > partialdata.txt;

display Total_cost;
display Total_time;
```

Following the execution of the listing B.3, the mapping of tasks (listing B.4) and mapping of partial data (listing B.5) can be obtained.

**Listing B.4:** *Locations of task given by a solution to the NLP problem*

```
y [*,*] (tr)
:              as          dttrain        ftrain         nbtrain      :=
aquilap     1.74865e-15   1.61296e-15   0             6.63495e-15
bellep      0             0             0             3.27613e-15
manjrap     5.12944e-15   9.52004e-15   0             0
nodep       4.0911e-15    0             7.68236e-15   0
omiip       4.62698e-15   6.18761e-15   0             0
roteggp     1             1             1             1
snowballp   3.82849e-15   0             8.64125e-16   0
tsukubap    0             4.84344e-15   2.1808e-15    0
;
```

**Listing B.5:** *Partial data retrievals given by a solution to the NLP problem*

```
d [*,*,as]
:          aquilap    bellep     manjrap     nodep      omiip     roteggp  snowballp   tsukubap
:=
aquila      12.9344    12.5013    12.4989    12.4993    12.4996       0  12.5011  12.5031
belle       12.4388    12.5076    12.4989    12.4993    12.4996       0  12.5011  12.5031
manjra      12.4388    12.5013    12.5111    12.4993    12.4996       0  12.5011  12.5031
node        12.4388    12.5013    12.4989    12.5064    12.4996       0  12.5011  12.5031
omii        12.4388    12.5013    12.4989    12.4993    12.5079       0  12.5011  12.5031
rotegg      12.4388    12.5013    12.4989    12.4993    12.4996     100  12.5011  12.5031
snowball    12.4388    12.5013    12.4989    12.4993    12.4996       0  12.5037  12.5031
tsukuba     12.4388    12.5013    12.4989    12.4993    12.4996       0  12.5011  12.5056

 [*,*,dttrain]
:          aquilap    bellep     manjrap     nodep      omiip     roteggp  snowballp   tsukubap
:=
aquila      12.947     12.5006    12.4961    12.4977    12.4979       0  12.5  12.4999
belle       12.437     12.5025    12.4961    12.4977    12.4979       0  12.5  12.4999
manjra      12.437     12.5006    12.4991    12.4977    12.4979       0  12.5  12.4999
node        12.437     12.5006    12.4961    12.5004    12.4979       0  12.5  12.4999
omii        12.437     12.5006    12.4961    12.4977    12.5004       0  12.5  12.4999
rotegg      12.437     12.5006    12.4961    12.4977    12.4979     100  12.5  12.4999
snowball    12.437     12.5006    12.4961    12.4977    12.4979       0  12.5003  12.4999
tsukuba     12.437     12.5006    12.4961    12.4977    12.4979       0  12.5  12.5001

 [*,*,ftrain]
:          aquilap    bellep     manjrap     nodep      omiip     roteggp  snowballp   tsukubap
:=
aquila      12.7282    12.5023    12.4891    12.4999    12.5006       0  12.5032  12.5011
belle       12.4666    12.5052    12.4891    12.4999    12.5006       0  12.5032  12.5011
manjra      12.4666    12.5023    12.5772    12.4999    12.5006       0  12.5032  12.5011
node        12.4666    12.5023    12.4891    12.505     12.5006       0  12.5033  12.5011
omii        12.4666    12.5023    12.4891    12.4999    12.5038       0  12.5032    12.5011
rotegg      12.4666    12.5023    12.4891    12.4999    12.5006     100  12.5032    12.5011
snowball    12.4666    12.5023    12.4891    12.4999    12.5006       0  12.5057  12.5011
tsukuba     12.4666    12.5023    12.4891    12.4999    12.5006       0  12.5033  12.5016

 [*,*,nbtrain]
:          aquilap    bellep     manjrap     nodep      omiip     roteggp  snowballp   tsukubap
:=
aquila      12.844     12.504     12.4927    12.4996    12.4994       0  12.4986  12.5003
belle       12.4491    12.5066    12.4927    12.4996    12.4994       0  12.4986  12.5003
manjra      12.4491    12.504     12.5528    12.4996    12.4994       0  12.4986  12.5003
node        12.4491    12.5039    12.4927    12.5031    12.4994       0  12.4986  12.5003
omii        12.4491    12.504     12.4927    12.4996    12.5027       0  12.4986  12.5003
rotegg      12.4491    12.504     12.4927    12.4996    12.4994     100  12.4986  12.5003
snowball    12.4491    12.5039    12.4927    12.4996    12.4994       0  12.5005  12.5003
tsukuba     12.4491    12.504     12.4927    12.4996    12.4994       0  12.4986  12.5009
```