Resource Management for Data Intensive Tasks on Grids

by

Imran Ahmad, M.A.Sc. (EE), B.Sc. (EE)

A thesis submitted to the Faculty of Graduate Studies and Research in
partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering (OCIECE)

Department of Systems and Computer Engineering

Carleton University

Ottawa, Ontario, Canada, K1S 5B6

©Copyright 2010, Imran Ahmad

Library and Archives
Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file *Votre référence*
ISBN: 978-0-494-70536-0
Our file *Notre référence*
ISBN: 978-0-494-70536-0

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

Distributed systems such as Grids, aim to enable the sharing, selection, and aggregation of a wide variety of resources that are geographically distributed and often owned by different organizations. These resources collaborate for performing complex tasks. Without efficient resource management, the benefits of a Grid system cannot be realized, especially for large-scale computational and data intensive tasks. The efficient management of distributed resources to perform a complex task is important. In a Grid, a resource management system is responsible for managing the available resources for a given task to be performed. This thesis proposes an effective resource management system called BiLeG, which can be used for performing resource intensive tasks in a Grid computing environment. This thesis focuses on the problem of allocating resources for a group of a particular type of resource intensive tasks termed Processable Bulk Data Transfer (PBDT) tasks. A PBDT task involves the transfer of a very large volume of data that has to be processed in some way before it can be used at a remote set of sink nodes. In BiLeG, the resource management system is bifurcated into separate upper and lower decision making levels and separate responsibilities are assigned to each decision making level. The upper decision making level of BiLeG, called the Task Resource Pool Selector (TRPS), is concerned with selection of a resource-pool for the given task. The lower decision making level, called Resource Allocator (RA) is responsible for allocating resources out of the resource-pool chosen by TRPS. At TRPS, a policy determines the way the resource-pool is chosen for each of the tasks. At RA, an algorithm which determines the allocation of the resources from its resource-pool for the task selected by TRPS is deployed. Note that the resource allocation problem considered in this research focuses on achieving a good balance between multiple

performance metrics. Although existing research has addressed various issues in resource allocation, none of the existing works has dealt with resource allocation for PBDT tasks addressing multiple performance metrics that this research focuses on.

Acknowledgements

Many people have contributed to the completion of my research leading to the doctorate degree. Technical inputs from my advisor Prof. Shikharesh Majumdar and encouragement from family members and friends have had a significant impact on my research work and I am thankful to all for that.

First and foremost, I would like to express my sincere thanks to my advisor Prof. Shikharesh Majumdar whose guidance during each phase of the thesis was invaluable.

I am grateful to all my family members for their continuous support and encouragement. Without their support and encouragement this long journey of research would not have been possible. Especially, I am indebted to my parents and father-in-law for their dedication and many years of support. Although they live thousands of kilometres away in Pakistan, they always remained in touch and shared the frustrations and excitements that I faced on the path to complete this thesis. I am also thankful to my elder brother Irfan Ahmed who continuously reminded me the importance to finish up my research work.

In the end, I would like to acknowledge the efforts made by my wife, Naheed, and my son, Omar that went a long way for me to complete this research. During these years, I never needed to go to the university to find a quiet working environment. Their presence and attitude continuously motivated me and helped me focus on the research work. I do not have words to express my gratitude for them. This thesis is dedicated to Naheed and Omar.

Table of Contents

Chapter 1	Introduction.....	1
1.1	Overview	1
1.2	Motivations for this Research	2
1.3	The Bi-level Grid Resource Management System.....	4
1.3.1	Introduction.....	4
1.3.2	Classification of RA Algorithms	5
1.3.3	TRPS Policies	6
1.4	Goals for an Efficient Resource Management System.....	7
1.5	Contributions of the Thesis	9
1.5.1	BiLeG Grid Management System Architecture.....	9
1.5.2	TRPS Policies	10
1.5.3	RA Algorithms.....	10
1.5.3.1	ATSRA Class of Algorithms for RA.....	11
1.5.3.2	List Scheduler	11
1.5.3.3	Round Robin Job Replicator (RRJR)	12
1.6	Papers Resulting from this Thesis.....	12
1.7	Layout of the Thesis	13
Chapter 2	Background.....	14
2.1	Grid Computing.....	14
2.2	The Grid Architecture	15
2.3	The Globus Toolkit	16
2.4	Grid Concepts.....	17
2.4.1	Types of Resources	17
2.4.2	Tasks and Applications	18
2.4.3	Resource Management.....	18
2.4.4	Monitoring and Discovery Service	19
2.4.5	Data Management	20
2.5	Types of Grids.....	21
2.5.1	Classification of Grids According to Organizational Setup	21
2.5.2	Classification of Grids According to the Type of Resources	22

2.5.3	Cloud Computing and the Grid.....	22
2.5.3.1	Relationship Between Grid and Cloud Computing	27
2.6	Approaches to Resource Allocation for Tasks on Grids	28
2.6.1	Traditional Schedulers and Resource Brokers	29
2.6.2	Policy Based Resource Allocation.....	31
2.6.3	Workflow Based Resource Allocation.....	32
2.6.4	Scheduling Strategies.....	33
2.6.5	Static Scheduling	33
2.6.6	Dynamic Scheduling	34
2.6.7	Hybrid Scheduling	35
2.6.8	Classes of Resource Allocation Algorithms	36
2.6.8.1	Knowledge-based algorithms	36
2.6.8.2	Knowledge-free Algorithms	36
2.6.9	Examples of Resource Allocation Algorithms	36
2.6.9.1	Workqueue Algorithm	37
2.6.9.2	Workqueue with Replication	37
2.6.9.3	Suffrage and XSuffrage	38
2.7	Grid Resource Brokering Systems	38
2.7.1	Condor-G	39
2.7.2	AppLeS Parameter Sweep Template (APST).....	40
Chapter 3	BiLeG Resource Management System	41
3.1	Processable Bulk Data Transfer (PBDT) Tasks	42
3.1.1	Introduction.....	42
3.1.2	Characteristics.....	42
3.1.3	Types.....	44
3.1.4	Practical Applications	44
3.1.4.1	Podcast Rendering	45
3.1.4.2	Multimedia Encoding	47
3.1.4.3	Particle Physics Data Grids	48
3.1.4.4	International Virtual Data Grid Laboratory	49
3.2	Terminology	49
3.3	The BiLeG Architecture.....	52
3.4	Flexibility	54
3.5	Task Resource-pool Selection Policies	55

3.5.1	Static Policies.....	56
3.5.1.1	States of a PBDT Task in the Static TRPS Policies	56
3.5.1.2	Static Resource-Pool with Single Partition (SRP_{sp})	57
3.5.1.3	Static Resource-Pool with Multiple Partitions (SRP_{mp})	61
3.5.2	Dynamic Policies	63
3.5.2.1	States of a PBDT Task in the Dynamic TRPS Policies.....	63
3.5.2.2	Reducing Resource-pool Algorithm.....	64
3.5.2.3	Dynamic Resource-Pool with Single Partition (DRP_{sp})	66
3.5.2.4	Dynamic Resource-pool with Multiple Partitions (DRP_{mp})	68
3.5.2.5	Dynamic Resource-pool with Proportional Partitions (DRP_{mp-pro})	69
3.5.3	Hybrid TRPS Policy	72
3.5.3.1	Static Resource-pool with Single Partition & Backfilling (SRP_{sp+BF}).....	72
3.6	Resource Allocator (RA).....	76
3.6.1	Architectural Templates.....	76
3.6.1.1	2-Tier Architectural Templates	77
3.6.1.2	3-Tier Architectural Templates	79
3.6.1.3	4-Tier Architectural Templates	80
3.6.2	ATSRA Algorithm.....	84
3.6.2.1	$ATSRA_{org}$ Algorithm	85
3.6.2.2	Constraints Relaxation.....	93
3.6.2.3	$ATSRA_{ssr}$ Algorithm	94
3.6.2.4	$ATSRA_{bsr}$ Algorithm	95
3.6.3	List Scheduling (LS).....	96
3.6.4	Round Robin Job Replicator (RRJR).....	97
Chapter 4	Performance Analysis of the RA Algorithms	99
4.1	Experimental Setup	99
4.2	Performance Metrics	100
4.3	The Performance Analysis of the ATSRA Class of Algorithms.....	101
4.3.1	Introduction.....	101
4.3.2	Workload Parameters.....	101
4.4	Performance of the ATSRA Algorithms.....	104
4.4.1	Selection of the TRPS Policy.....	104
4.4.2	Algorithm-running-overhead.....	105
4.4.3	Makespan-without-overhead.....	106

4.4.4	Total Cost.....	106
4.4.5	Makespan-total.....	108
4.4.6	Cost-mstotal	110
4.5	Performance Analysis of the Low Overhead RA Algorithms.....	112
4.5.1	Choice of the TRPS Policies.....	113
4.5.2	Effect of Increasing the Number of Nodes on $t_{ms-total}$	113
4.5.3	Effect of Increasing the Number of Nodes on t_{cost}	114
4.5.4	Effect of Increasing the Variance in Unit Communication Times on $t_{ms-total}$	116
4.5.5	Effect of Increasing Mean Unit Communication Time on $t_{ms-total}$	117
4.6	Summary	118
Chapter 5	Performance of the TRPS Policies.....	122
5.1	Effect of Increasing the Size of Δ on the Performance of the TRPS Policies.....	122
5.1.1	Effect of Increasing the Size of Δ on t_{ms-WOH}	123
5.1.2	Effect of Increasing the Size of Δ on t_{cost}	127
5.2	Algorithm-running-overhead	130
5.2.1	Effect of Algorithm-running-overhead on $t_{ms-total}$	130
5.2.2	Experimental Analysis of Algorithm-running-overhead	132
5.2.3	Effect of Increasing the Size of Δ on Algorithm-running-overhead.....	135
5.2.4	Impact of Overhead of Running the RA Algorithm	138
5.2.4.1	Effect of Increasing CPU Speed on Backfilling.....	139
5.2.4.2	Effect of Increasing CPU Speed on Multiple Partitioning	143
5.3	Achieving a Balance between Makespan and Total cost	143
5.4	Performance of Various TRPS Policies under Different Workload Conditions.....	147
5.4.1	Effect of Increasing the Variance of the Lengths of Raw Data Files of the Constituent Tasks On Performance	148
5.4.1.1	The Makespan-without-overhead.....	148
5.4.1.2	Total Cost	151
5.4.2	Effect of Increasing the Mean Length of the Raw data Files of the Constituent Tasks On the Performance	152
5.4.2.1	Total Cost	153
5.4.2.2	Makespan-without-overhead	154
5.5	Discussion	156
Chapter 6	Performance Analysis of the Allocation-plans	159
6.1	Introduction	159

6.2	Computation of the Lower Bound on t_{cost}	160
6.3	Experimental Setup	166
6.4	Experimental Results.....	166
6.4.1	Minimizing t_{cost}	167
6.4.2	Techniques to Reduce $t_{\text{ms-total}}$	169
6.4.2.1	Objective A: To Control Algorithm-running-overhead.....	170
6.4.2.2	Objective B: To Control Resource-contention	179
6.5	Summary	184
Chapter 7	Conclusions.....	187
7.1	Summary	187
7.2	Conclusions	189
7.3	Future Works.....	191
References	194	
Appendix A	204	
Appendix B	205	

List of Tables

Table 2.1:	Comparison of Grid and Cloud Computing	30
Table 3.1:	Salient Features of the Architectural Templates	79
Table 4.1:	Default Values of the Parameters	104
Table 6.1:	Strategies for the Performance Analysis of the Allocation-plans	172
Table A1:	Probability Distributions Used for Unit Communication & Processing Times	206

List of Figures

Fig. 2.1	The Layered Grid Architecture	16
Fig. 2.2	Attributes of Elasticity	26
Fig. 3.1	Partitioning and Processing of a PBDT Raw Data File	45
Fig. 3.2	Podcast Rendering	48
Fig. 3.3	Multimedia Encoding	50
Fig. 3.4	BiLeG Architecture	55
Fig. 3.5	Different States of a Task for Static Policies	59
Fig. 3.6	Mapping Phase of SRP_{sp}	61
Fig. 3.7	Execution Phase of SRP_{sp}	61
Fig. 3.8	Task Completion Event Handler for SRP_{sp}	61
Fig. 3.9	Mapping phase of SRP_{mp}	64
Fig. 3.10	Different States of a Task for Dynamic Policies	65
Fig. 3.11.	Reducing Resource-pool Algorithm	67
Fig. 3.12	Dynamic Resource-Pool Single Partition Policy	69
Fig. 3.13	Task Completion Event Handler for DRP_{sp}	69
Fig. 3.14	Dynamic Resource-Pool Multiple Partition Policy	71
Fig. 3.15	Task Completion Event Handler of DRP_{mp}	71
Fig. 3.16	Dynamic Resource-Pool Multiple Proportional Partitioning	72
Fig. 3.17	Task Completion Event Handler of DRP_{mp-pro}	72
Fig. 3.18	Execution Phase of $SRP_{sp} + BF$	76
Fig. 3.19	Task Completion Event Handler of $SRP_{sp} + BF$	77
Fig. 3.20	2-Tier Architectures	80
Fig. 3.21	3-Tier Architecture	82
Fig. 3.22	4-Tier Architecture	83
Fig. 3.23	Stage-1 of the $ATSRA_{org}$ Algorithm	88
Fig. 3.24	Stage-2 of the $ATSRA_{org}$ Algorithm	88

Fig. 3.25	RRJR Algorithm	100
Fig. 4.1	Impact of Increasing Number of Nodes on t_{OH}	109
Fig. 4.2	Effect of Increasing Number of Nodes on t_{ms-WOH}	109
Fig. 4.3	Impact of Increasing Number of Nodes on t_{cost}	111
Fig. 4.4	Impact of Increasing Number of Nodes on $t_{ms-total}$	111
Fig. 4.5	Impact of Increasing Number of Nodes on cost-mstotal	114
Fig. 4.6	Impact of Increasing Number of Nodes on $t_{ms-total}$	117
Fig. 4.7	Effect of Increasing Number of Nodes on t_{cost}	117
Fig. 4.8	Impact of Increasing Variance in Unit Transfer Time on $t_{ms-total}$	119
Fig. 4.9	Effect of Increasing Mean Unit Communication Time on $t_{ms-total}$	120
Fig. 5.1	Impact of Increasing No. of Nodes on t_{ms-WOH} for TRPS Policies	125
Fig. 5.2	Impact of Increasing No of Nodes on t_{cost} for TRPS Policies	130
Fig. 5.3	Effect of Increasing No of Nodes on $t_{ms-total}$ for TRPS Policies	135
Fig. 5.4	Impact of Increasing n on t_{OH-on} for Different TRPS Policies	139
Fig. 5.5	Impact of Increasing n on t_{ms-OFF} for Different TRPS Policies	139
Fig. 5.6	Impact of Increasing Number of Nodes on t_{OH} for Different TRPS Policies	142
Fig. 5.7	Impact of Increasing n on $t_{ms-total}$ (with RA Algorithm running at 2X)	143
Fig. 5.8	Impact of Increasing n on $t_{ms-total}$ (with RA Algorithm running at 4X)	143
Fig. 5.9	Impact of Increasing n on $t_{ms-total}$ (with RA Algorithm running at 1X)	144
Fig. 5.10	Effect of Increasing No. of Nodes on cost-ms-WOH for Different TRPS Policies	148
Fig. 5.11	Effect of Increasing No. of Nodes on cost-mstotal for Different TRPS	148
Fig. 5.12	Effect of Increase of Variance of the Lengths of Raw Data Files of the Constituent Tasks on t_{ms-WOH}	152
Fig. 5.13	Effect of Increase of Variance of the Sizes of Raw Data Files of the Constituent Tasks on t_{cost}	155
Fig. 5.14	Effect of Increase of Mean Length of Raw Data Files of the Constituent Tasks on t_{cost}	157

Fig. 5.15	Effect of Increase of Mean Length of Raw Data File of the Constituent Tasks on t_{ms-WOH}	157
Fig. 6.1	Comparison of $\langle SRP_{sp}, ATSRA_{org} \rangle$ with the Lower Bound	170
Fig. 6.2	Effect of Increasing Number of Nodes on t_{OH}	175
Fig. 6.3	Effect of Increasing Number of Nodes on $t_{ms-total}$	175
Fig. 6.4	Effect of Increasing Number of Nodes on t_{cost}	175
Fig. 6.5	Effect of Increasing Number of Nodes on $cost-mstotal$	180
Fig. 6.6	Impact of Increasing Number on Nodes on t_{cost}	183
Fig. 6.7	Impact of Increasing Number on Nodes on t_{ms-WOH}	184
Fig. 6.8	Effect of Increasing Number of Nodes on $cost-mstotal$	185
Fig. B.1	Impact of Increasing Number on Nodes on t_{ms-WOH}	202
Fig. B.2	Impact of Increasing Number on Nodes on $t_{ms-total}$	207
Fig. B.3	Impact of Increasing Number on Nodes on t_{cost}	208

Glossary of Terms

API	Application Programming Interface
ATSRA _{org}	Architectural Template Selection and Resource Allocation, Original
ATSRA _{bsr}	Architectural Template Selection and Resource Allocation, Bi- step Relaxation
ATSRA _{ssr}	Architectural Template Selection and Resource Allocation, Single-step Relaxation
BiLeG	Bi-level Grid Resource Management System
CA	Certificate Authorities
CE	Compute Elements
CPU	Central Processing Unit
DRP _{mp}	Dynamic Resource-pool with Multiple Partitions
DRP _{mp-pro}	Dynamic Resource-pool with Multiple Proportional Partitions
DRP _{sp}	Dynamic Resource-pool with Single Partition
GRAM	Globus Resource Allocation Manager
GRIS	Grid Resource Information Service
LDAP	Lightweight Directory Access
LS	List Scheduling
MDS	Monitoring and Discovery Service
NWS	Network Weather Stations
PBDT	Processable Bulk Data Transfer Tasks
PKI	Public Key Infrastructure
QoS	Quality Of Service
RA	Resource Allocator
RLS	Replica Location Service
RRJR	Round Robin Job Replicator

SE	Storage Elements
SOA	Service Oriented Architectures
SRM	Storage Resource Manager
SRP _{mp}	Static Resource-Pool with Multiple Partitions
SRP _{sp}	Static Resource-pool with Single Partition
SRP _{sp} +BF	Static Resource-pool Single Partition and Backfilling
SSL	Secure Sockets Layer
TRPS	Task & Resource-pool Selector
WMS	Workload Management System
WQ	Workqueue Algorithm
WQR	Workqueue with Replication

List of Symbols

T	Set of tasks in the bag-of-tasks
Δ	Set of available Grid nodes
κ_s	Set of sink nodes
Γ	Set of nodes in a resource-pool
n_{src}	Source node
n	Number of available Grid nodes
w	Number of tasks in the bag-of-tasks
k	Number of sink nodes
L_i	Length of the raw data file in GBs of the task T_i
q	Number of partitions in multiple partitioning policies
ρ_i	Number of partitions in the raw data file of the task T_i
L_q	Length of the raw data file in GB of the q^{th} task in the given bag-of-tasks
ϵ_q	Processing factor associated with the q^{th} task
β	Processing cost in dollars per hour
α_{in}	Transfer-in cost in dollars per hour
α_{out}	Transfer-out cost in dollars per hour
α_{Δ}	Transfer-within cost in dollars per hour
$\alpha_{src-sink}$	Transfer source-sink cost in dollars per hour
$t_{ms-total}$	Makespan total
t_{ms-WOH}	Makespan-without-overhead

t_{OH}	Algorithm-running-overhead
t_{cost}	Total cost
C_{p_i}	Processing time per unit data at n_i
$d(n_1, n_2)$	Transfer time per unit data between n_1 and n_2
$[C_{c_{src-\kappa_s}}]$	Vector representing unit communication times from source to κ_s
$[C_{c_{src-\Delta}}]$	Vector representing unit communication times source to Δ
$[C_{c_{\Delta-\kappa_s}}]$	Vector representing unit communication times from Δ to κ_s
$[C_{c_{\Delta}}]$	Vector representing unit communication times between all the nodes in Δ
$[C_{p_{\kappa_s}}]$	Vector representing unit processing times at κ_s
$[C_{p_{\Delta}}]$	Vector representing unit processing times at Δ
$d^-(n_{src}, \kappa_s)$	$\min [C_{c_{src-\kappa_s}}]$
$d^-(\Delta, \kappa_s)$	$\min [C_{c_{src-\Delta}}]$
$d^-(n_{src}, \Delta)$	$\min [C_{c_{src-\Delta}}]$
$C_{p_{\Delta}}^-$	$\min [C_{p_{\Delta}}]$
$C_{p_{\kappa_s}}^-$	$\min [C_{p_{\kappa_s}}]$

Chapter 1 Introduction

1.1 Overview

A Grid is a collection of homogeneous or heterogeneous resources that may belong to multiple administrative domains and may be spread over multiple sites connected by a wide-area network. These resources are coordinated through secure mechanisms to solve a common problem and are sufficiently integrated to deliver a certain level of quality of service. Grids allow sharing of heterogeneous distributed resources across administrative and geographical boundaries [FOS99a]. By sharing these distributed resources many complex distributed tasks can be performed in a cost effective way, as it provides the possibility to combine the available inexpensive computing resources from multiple domains to form a large parallel processing machine [LEE07]. When Grids are used to process complex tasks, efficient resource allocation holds a pivotal importance for satisfactory results [VIS07]. To perform efficiently, the resource allocation mechanism has to take into account many factors, such as, the system and workload parameters, type of the task to be performed and the requirements of the end user. For improving the efficiency of the resource allocation, it may be useful to classify the given tasks into predefined types based on similarities in their expected resource needs, for example, some tasks can be characterized as compute-intensive tasks; while others can be put in the data-intensive category [ANG06]. This classification of tasks into various types provides the possibility to customize the resource allocation process according to a particular group of similar tasks.

In a Grid, usually a resource management system is responsible for handling the allocation of the available resources to the tasks to be performed [TOY06] [CAS00]. An efficient Grid resource

management system can greatly enhance the performance of a Grid system by intelligently allocating the available resources. This research concerns the devising of an effective resource management system and focuses on a type of resource-intensive tasks called *Processable Bulk Data Transfer* (PBDT) tasks in this research. The common trait among PBDT tasks is the transfer of a very large amount of data which has to be processed in some way before it can be used at a set of designated sink nodes. Typically, these tasks can be broken down into sub-tasks called *jobs*. Various multimedia applications and High Energy Physics (HEP) experiments can give rise to PBDT tasks. The processing operation involved in these tasks may be as simple as applying a compression algorithm to a raw video file in a multimedia application; or, as complex as isolating information about particles pertaining to certain wavelengths in HEP experimentations [BUN03]. Performing PBDT tasks requires extensive processing facilities and involves bulk data transfers. Examples of various PBDT tasks are provided in Chapter 3 (see Section 3.1.4).

1.2 Motivations for this Research

Processing of large geographically distributed datasets, also referred to as distributed data-intensive computing [CHE01], has emerged as an important area in recent years. Scientific discoveries are increasingly being facilitated by analysis of very large datasets in distributed environments. Since careful management of storage, computing, and networking resources is required for efficiently analyzing very large datasets, the task of processing data stored in remote repositories becomes a Grid problem.

Grid computing provides the provision of combining ordinary workstations to form a powerful parallel machine in a cost effective way. For achieving high performance these workstations

must be connected using high-speed connections. Also, an efficient run-time environment must be provided. Even if all data is available at a single repository, it is not possible to perform all analysis at the site hosting such a shared repository. Thus, an application that processes data from a remote repository needs to be broken into several stages, including a data retrieval task at the data repository, a data movement task and a data processing task at a computing site. Because of the high volume of data and the large processing requirements, it is often required to use multiple nodes for data storage and computing. It may be desirable that both the data repository and computing sites are independent nodes. This can further complicate resource management for the data processing applications.

Various initiatives have tried to address the issues discussed in the previous paragraph. The need to integrate services that can handle the processing of large datasets across distributed, heterogeneous and dynamic environments of Grids has led to the definition of a broad service architecture called *Open Grid Service Architecture* (OGSA) [GLO09]. One of the implementation standards that conform to OGSA is *Web Service Resource Framework* (WSRF), which extends Web Service Definition Language (WSDL) and XML Schema to enable the standardized use of Grid services. Globus Toolkit [GLO08] is an open-source implementation of WSRF capable of hosting Grid services and is a widely known Grid application framework. The Globus Toolkit middleware provides the set of services and software libraries that facilitate the development of Grid applications and services. A more detailed discussion of the existing work is presented in Chapter 2.

The focus of this research is on devising an efficient Grid resource management system for a particular class of resource-intensive tasks, called PBDT tasks. As discussed in Section 3.1.2,

for PBDT tasks, data is collected and stored in a repository at one location. This data is processed and delivered to a set of user nodes. The thesis devises efficient and scalable Grid Resource Management techniques that can transparently manage the access and processing of data from nodes processing PBDT tasks. The thesis proposes a bi-level resource management system for achieving its objectives. A discussion of the proposed system is provided in the next section.

1.3 The Bi-level Grid Resource Management System

1.3.1 Introduction

To provide an efficient mechanism for resource allocation in Grids, this research proposes a bi-level Grid Resource Management System abbreviated as *BiLeG*. In *BiLeG*, the decision making module is divided into two separate modules. The upper level decision making module is called the *Task & Resource-pool Selector* (TRPS) and the lower level decision making module is called the *Resource Allocator* (RA). The workload for *BiLeG* is a *bag-of-tasks* which is defined as a set of tasks grouped together, each of which can be run independent of the others. TRPS selects a task from a given bag-of-tasks for which resources are to be assigned and chooses a partition of resources available for this chosen task for assignment (called the *resource-pool* of this task) which is typically a subset of all the resources available. RA uses an assignment algorithm to decide how the resources (from the allocated resource-pool) are allocated to the jobs constituting the chosen task. Various algorithms can be used at RA and various policies can be deployed at TRPS. A particular combination of a TRPS policy and a RA resource allocation algorithm deployed at a time is called an *allocation-plan*. The following notation is used in this thesis to describe an allocation-plan: $\langle \text{TRPS Policy, RA-Algorithm} \rangle$. An appropriate allocation-plan for

a particular Grid depends on many factors such as the type of the task to be performed, user requirements, system and workload parameters. An appropriate allocation-plan can help maximize the overall performance of the underlying Grid system. An effective allocation-plan can be provided by using a suitable algorithm at RA and an appropriate policy at TRPS.

For devising an algorithm for RA, the inherent trade-off between the complexity of the algorithm chosen to allocate resources and the performance benefits obtained by using it, should be carefully considered. The complexity of a resource allocation algorithm is directly related to the overhead of running the algorithm itself. The benefits in terms of performance gains achieved by running a particular algorithm should be considerably higher than the overhead of running the algorithm. Another important factor that may affect the performance of the RA algorithm is the accuracy with which various system parameters can be predicted in a Grid system. Performance of some RA algorithms depends on the existence of an accurate mechanism to predict various system parameters such as the processing and communication times per unit data.

1.3.2 Classification of RA Algorithms

Two types of resource allocation algorithms: knowledge-based and knowledge-free have been considered. The knowledge-based resource allocation algorithms use the predicted value of various system parameters to allocate resources. In certain Grid environments such as cluster Grids, all the resources are owned and controlled by a single administrator. In such environments these system parameters can be accurately predicted [FOS99b]. On other Grid systems, mechanisms such as Network Weather Stations can be used to predict the values of system parameters. If a particular resource allocation algorithm depends on predicted values of the system parameters then inaccurate estimations may lead to an unsatisfactory overall system

performance. In such situations knowledge-free resource allocation strategies, which can allocate the available resources irrespective of the ability to predict the system parameters, are a reasonable choice. Thus, to choose an appropriate resource allocation algorithm at RA, the accuracy with which various system parameters can be predicted is important to consider. Four of the resource allocation algorithms proposed in this research are knowledge-based and depend upon accurate predictions of the system parameters. They are more suitable for dedicated resource environments such as cluster Grids where the value of various system parameters, such as bandwidth, can be accurately predicted. A knowledge-free resource allocation algorithm has also been proposed in this research.

To devise efficient high performance RA algorithms, this research has introduced the concept of an *Architectural Template*. An Architectural Template specifies the way a given task is to be decomposed into its constituent jobs and assigns a role to each node for executing the jobs. A detailed discussion on Architectural Templates can be found in Section 3.6.1.

1.3.3 TRPS Policies

At the upper decision making level of the BiLeG (TRPS), a policy provides the following two functions:

1. Selection of the next task to be run.
2. Selection of the resource-pool for each task to be run.

A particular TRPS policy details these two functions for a given bag-of-tasks. The choice of the suitable policy is dependent on many factors. For example, it may depend upon whether or not an *offline processing facility* is available. An offline processing facility is a set of processing nodes which is not the part of the Grid system that is executing the workload. If the resource

allocation overhead is expected to occur before the first task in the given bag-of-tasks starts executing, the offline processing facility may be used for running the algorithm for resource allocation for the given bag-of-tasks. In this case the resource allocation overhead incurred at the offline resource allocation facility does not affect the total makespan for the bag-of-tasks as explained in Section 4.4.2. The choice of an appropriate TRPS policy is also dependent on user requirements and task characteristics. A comparative study of various TRPS policies is presented in detail in Chapter 5. This thesis proposes six different TRPS policies (discussed in Section 3.5) which can be used in combination with different RA algorithms (presented in Section 3.6). It also discusses which allocation-plan would be better to use in a particular circumstance.

1.4 Goals for an Efficient Resource Management System

In order to design an efficient Resource Management System to remotely process data in Grids, the proposed BiLeG architecture is based on the following goals:

Parallel Processing: An important goal of an efficient remote data processing system is the support for parallel processing of data. An efficient Grid resource management system should be able to support parallel jobs in a task [JAR03]. It should be able to coordinate the execution of these parallel jobs. The support for parallel processing can greatly enhance the performance of the system.

For example, many multimedia processing applications, such as video encoding and movie rendering are resource intensive applications and parallel computing can be effectively used to speed them up, making the system much more efficient [ZAK99].

Remote Data Repository and Grid Computing Standards:

Typically retrieving data from a remote data repository can be difficult, because of the lack of user control over the environment. Data on such repositories is stored using standard data server software, which offers access functionality through a client API. It is important that standardized data retrieval and movement functionality is used.

Performance Enhancement by Balancing Computational Load:

For an efficient resource management system, interesting load balancing and scheduling considerations may arise for applications processing remote data from geographically distributed sources. Typically, data may be partitioned in a number of ways and a set of geographically distributed resources may be available for processing data. Note that the resource utilization and balanced parallel execution becomes critical to application performance.

Research has been conducted to provide a run-time environment that can process remote data efficiently while addressing the issues mentioned earlier. Existing research has led to the creation of various technologies. In order to provide a low cost/performance ratio, these systems optimize the overall execution time (or makespan) of resource-intensive tasks. This requires efficient allocation of the resources for the jobs of the PBDT tasks running on the nodes of the system. The problem of allocating resources for these jobs to optimize the overall execution time of an application is a well-known NP-complete problem [GAR79][WAN06]. To tackle it, heuristic algorithms that can generate effective solutions to problems in polynomial times are required.

This thesis proposes a bi-level architecture and presents six TRPS policies and five resource-allocation algorithms that are based on heuristics and can be used in a Grid for efficient resource management system.

1.5 Contributions of the Thesis

The contributions of this thesis can be summarized as:

- 1- A new bi-level resource management system architecture for Grids named BiLeG is proposed.
- 2- Six different policies for the upper decision making module TRPS have been proposed.
- 3- Five different RA algorithms are proposed. They consist of four knowledge-based algorithms, including Architectural Template Selection and Resource Allocation (ATSRA) class of algorithms which is based on linear-programming. A knowledge-free algorithm called Round Robin Job Replicator (RRJR) is also proposed. The merits and de-merits of these algorithms have been analyzed under various system and workload parameters.
- 4- In order to manage the overhead that is generated when complex RA algorithms are executed, various strategies that result in overall performance improvement have been developed.

These contributions are discussed in more detail in the following sections.

1.5.1 BiLeG Grid Management System Architecture

This thesis introduces a new way of managing Grid resources by introducing a bi-level architecture. The proposed bi-level architecture works by dividing the decision making

architecture into two levels. The upper decision making level is concerned with the resource-selection and the lower decision making level is concerned with resource allocation. This thesis also discusses the mechanisms needed for these two separate decision making modules to interact with each other and with the user of the Grid system.

Note that although existing work on Grids has used a bi-level architecture for resource allocation, none of the existing works has used the bi-level architecture to allocate resources for the PBDT tasks. Existing works on Grids that use bi-level architecture has focused on performing computation or data transfers separately and none of the existing works uses it for the resource management for PBDT tasks that require both processing as well as data movement.

1.5.2 TRPS Policies

This thesis introduces six different policies that can be used at the upper decision making level. The proposed policies can be divided into three classes: static, dynamic and hybrid. Decisions are made by a static policy before task execution commences in the system whereas dynamic policies can make their decisions even after task execution has started. The hybrid policies combine attributes of both the static and the dynamic policies. The TRPS policies are discussed in Chapter 3.

1.5.3 RA Algorithms

Five different resource allocation algorithms are proposed. The proposed RA algorithms can be divided into two categories: knowledge-based and knowledge-free. A brief overview of the RA algorithm is presented next. A detailed discussion is included in Chapter 3.

1.5.3.1 ATSRA Class of Algorithms for RA

A class of knowledge-based algorithms called ATSRA is proposed. These algorithms are based on the linear programming model. ATSRA algorithms perform the following two functions.

1. Selecting the most appropriate Architectural Template for each task in a given bag-of-tasks in such a way that a cost function is minimized.
2. Allocating the resources for the chosen Architectural Template for a given task.

In this class of algorithms, $ATSRA_{org}$ is the basic algorithm. As the number of nodes in the resource-pool allocated to RA by TRPS increases, the time for executing the ATSRA algorithms increases as well. This increases the resource allocation overhead for an ATSRA algorithm. In such situations approximations are introduced to reduce the performance overhead. Achieving the right balance between cost incurred and the benefits obtained is a key here. There are two algorithms proposed, $ATSRA_{ssr}$ and $ATSRA_{bsr}$, which add different levels of approximations to $ATSRA_{org}$.

Note that ATSRA algorithms have been specifically developed during this research for the resource allocation of the PBDT tasks. The ATSRA algorithms are designed to minimize both data transfer and data processing costs using linear programming. To the best of my knowledge the use of linear programming for the resource allocation of the PBDT tasks has never been used in any of the existing research works and is a major contribution of this research.

1.5.3.2 List Scheduler

List Scheduler is a knowledge-based algorithm that sorts the available nodes according to their processing speeds and then assigns the jobs of the PBDT tasks to the nodes, giving priority to the

higher performing nodes.

1.5.3.3 Round Robin Job Replicator (RRJR)

RRJR is a knowledge-free algorithm which is proposed in this research. It uses the task replication to achieve reasonable performance.

Note that RRJR is a new replication algorithm proposed in this research. None of the existing replication based algorithms is designed for a bi-level architecture. Designing a bi-level replication based algorithm allows precise control of replication of jobs as discussed in Chapter 3.

1.6 Papers Resulting from this Thesis

So far, the following refereed publications have resulted from this thesis research:

- 1- I. Ahmad and S. Majumdar, “Efficient Resource Assignment for Processable Bulk Data Transfer Tasks”, in the special issue on Federated Resource Management in Grid and Cloud Computing Systems, Future Generation Computer Systems (FGCS), the International Journal of Grid Computing and eScience (Submitted).
- 2- I. Ahmad and S. Majumdar, “Performance of Resource Management Algorithms for Processable Bulk Data Transfer Tasks in Grid Environments”, in the Proceeding of the 7th ACM international Workshop on Software and Performance, Princeton, USA, 2008 [AHM08a].
- 3- I. Ahmad and S. Majumdar, “A Two Level Approach for Managing Resource and Data Intensive Tasks in Grids”, in the Proceedings of the Conference on Grid computing, high-

performance and Distributed Applications (GADA) Monterrey, Mexico, 2008 [AHM08b].

- 4- I. Ahmad and S. Majumdar, "Efficient Management of Grid Resources Using a Bi-level Decision-Making Architecture for Processable Bulk Data", in the Proceeding of the Conference on Grid computing, High-performance and Distributed Applications (GADA) Vilamoura, Algarve, Portugal, 2007 [AHM07].
- 5- I. Ahmad and S. Majumdar, "An Adaptive High Performance Architecture for Processable Bulk Data Transfers on a Grid," in the Proceedings of the 2nd International Conference on Broadband Networks (Broadnets), Boston, USA, 2005 [AHM05].

1.7 Layout of the Thesis

This thesis consists of six additional chapters. Chapter 2 provides an overview of Grids and the current state-of-the-art in resource management on Grids. Chapter 3 gives details of the BiLeG architecture proposed in this research. Chapter 4 presents the experimental setup and discusses the performance results for the RA algorithms. Chapter 5 analyses the performance of the TRPS policies. In Chapter 6, the performance of various allocation-plans is investigated and the choice of an appropriate allocation-plan, under a specific set of system and workload parameters, is discussed. Chapter 7 concludes this thesis and discusses direction for future research.

Chapter 2 Background

2.1 Grid Computing

The presence of Internet as well as the availability of low-cost powerful computers and high-speed network technologies are changing the way computing is carried out. Increasingly, it is becoming more feasible to use widely distributed computers for solving large-scale problems. In terms of computations and data requirements, these problems are often resource intensive due to their size and complexity. They may also involve the use of a variety of heterogeneous resources that are not usually available in a single location. These factors have led to the emergence of what is known as Grid computing which can be viewed as a seamless and collaborative environment [TOY06][BEY02]. Usually a Grid uses a Grid resource management system to manage the submitted tasks which performs resource discovery, scheduling, and assignment of the available Grid resources. Grid allows users to solve larger-scale problems by pooling together resources that were not been able to be used collectively [PEN04][NUD05][KIC04]. Thus, Grid is a technology that can bond and unify remote and diverse distributed resources.

Ian Foster, one of the pioneers of the Grid, provided a three point check-list that must be satisfied by a system for it to qualify as a Grid [FOS99a]. According to this check-list a Grid is a system that:

- 1) Coordinates resources that are not subject to a centralized control.
- 2) Uses standard, open, general-purpose protocols and interfaces.
- 3) Delivers non-trivial qualities of service.

2.2 The Grid Architecture

A standard high-level layered Grid architecture, shown in Fig. 2.1, was specified in [FOS99a] to identify the requirement for general classes of components. The components within each layer in Fig. 2.1 share common characteristics and build upon capabilities provided by lower layers. The fabric layer in the Grid layered architecture consists of Grid resources to which shared access is mediated by Grid protocols. A Grid resource may be a physical entity such as a CPU, a sensor or a storage component or it may be a logical entity such as a distributed file system. The connectivity layer defines communication and authentication protocols for Grid network transactions. The resource layer uses the functionality provided by the connectivity layer for the secure negotiation, initiation, monitoring and control of operations on individual fabric layer resources. While the resource layer is focused on interactions with a single resource, the collective layer coordinates multiple resources and provides application specific distributed services. The application layer consists of Grid applications that use the services of lower layers to achieve a particular goal. Each layer has well-defined protocols and provides useful services to the application such as resource discovery, resource management and data access.

Fig. 2.1: The Layered Grid Architecture [FOS99a]

2.3 The Globus Toolkit

The Globus project provides an open-source software toolkit [FOS97] [GLO09] that can be used to build Grid systems. The Globus architecture has three main groups of services accessible through a security layer. These groups are resource management, data management, and information services [CZA98] [CZA99]. The local-services layer contains the operating-system services and network services [KES99] [OLE04].

The Grid security infrastructure (GSI) [GLO09] provides methods for authentication of Grid users and secure communication. It is based on Secure Sockets Layer (SSL), Public Key Infrastructure (PKI), and X.509 certificate architecture. GSI provides services, protocols, and libraries to achieve the following aims for Grid security:

- Single sign-on for using Grid services through user certificates
- Resource authentication through host certificates
- Data encryption
- Authorization
- Delegation of authority

Users gain access to resources by having their Grid certificate subjects mapped to an account on the remote machine by its system administrators. This also requires that the Certification Authority that signed the user certificate be trusted by the remote system. Access permissions have to be enforced through restrictions on the remote user account.

2.4 Grid Concepts

In this section, various Grid concepts, components, and terms are introduced in more detail.

2.4.1 Types of Resources

Grid resources include both computing resources and communication resources. The computing resources are provided by the processors of the machines on the Grid. The processors can vary in speed, architecture, software platform, and other associated factors, such as memory, storage, and connectivity. There are two primary ways to exploit the computation resources of a Grid. The first and simplest one is to use it to run an existing application on an available machine on the Grid rather than locally. The second is to use an application designed to split its work in such a way that the separate parts can execute in parallel on different processors. A Grid provides the mechanisms needed to utilize the distributed computing resources to perform a given task.

Another important resource of a Grid concerns data communication. This includes communications within the Grid and external to the Grid. Communications within the Grid are important for sending jobs and their required data to nodes within the Grid [OLE04]. Some jobs require a large amount of data to be processed and it may not always reside on the machine running the job. The bandwidth available for such communications can often be a critical resource that can limit utilization of the Grid. Redundant communication paths are sometimes needed to better handle potential network failures and excessive data traffic [SAN05]. In some cases, higher speed networks must be provided to meet the demands of jobs transferring larger amounts of data [CHU05].

Another important resource of a Grid concerns data storage devices. This includes persistent storage devices such as hard-disks that are present in the individual computers that comprise of a Grid system.

2.4.2 Tasks and Applications

Although various kinds of resources on the Grid may be shared and used, they are usually accessed via an executing "application" or "tasks". In this research, the term task has been used to represent this highest level of a piece of work. Typically, tasks may be broken down into a number of individual jobs. They may perform computation, execute one or more system commands or move and collect data. A task may consist of jobs which may have specific dependencies that may prevent them from executing in parallel in some cases. For example, a job may require some specific input data that must be copied to a machine before the job is run on that machine. Some jobs may require the output produced by certain other jobs and cannot be executed until those predecessor jobs have completed executing. This workflow can create a hierarchy of tasks and jobs. Finally, the results of all of the jobs must be collected and appropriately assembled to produce the ultimate answer for the task.

2.4.3 Resource Management

The Grid system is responsible for sending a task to a given machine to be executed [JEO04]. In the simplest of Grid systems, the user may select a machine suitable for running his task and then execute a Grid command that sends the task to the selected machine. More advanced Grid systems would include a task "scheduler" that automatically finds the most appropriate machine

on which to run any given task that is waiting to be executed as described in [PEN04]. Schedulers react to current availability of resources on the Grid. To create more predictable behavior, Grid machines are often "dedicated" to the Grid and are not pre-empted by outside systems. Having a dedicated environment allows schedulers to compute the approximate completion time for a set of tasks, if the system and workload parameters are known. In a Grid system optimal scheduling, is a difficult problem. Therefore, such schedulers may use heuristics to optimize the scheduling metric.

The resource management subsystem enables resource allocation through task submission, staging of executable files, task monitoring, and result gathering. Globus Resource Allocation Manager (GRAM) provides remote execution capability of tasks and reports status for the course of the execution. A client requests a task submission from the gatekeeper daemon on the remote host. The gatekeeper component is a software module that intercepts all incoming requests at a particular domain node and performs authentication and authorization. The gatekeeper daemon checks to see if the client is authorized (i.e., the client certificate is in order and there is a mapping of the certificate subject to an account on the system). Once authentication is performed, the gatekeeper starts a job manager that initiates and monitors job execution.

2.4.4 Monitoring and Discovery Service

Monitoring and Discovery Service (MDS) provides support for publishing and querying of resource information [GLO09]. Within MDS, a schema defines classes that represent various properties of the system. MDS has a 3-Tier structure at the bottom of which is the information providers that gather data about resource properties and status and translate them into the format

defined by the object classes. The Grid Resource Information Service (GRIS) forms the second tier and is a daemon that runs on a single resource. GRIS responds to queries about the resource properties and updates its cache at intervals defined by a “time to live”, by querying the relevant information providers. At the topmost level, GIIS (Grid Information Index Service) indexes the resource information provided by other GRISs and GIISs that are registered with it.

The GRIS and GIIS run on the Lightweight Directory Access Protocol (LDAP) back end in which the information is represented as a hierarchy of entries, each entry consisting of 0 or more attribute–value pairs. The standard set of information providers provides data on CPU type, system architecture, number of processors, and memory available, among others.

2.4.5 Data Management

The data management package provides utilities and libraries for transmitting, storing, and managing massive datasets that are part of many scientific computing applications [ALL05]. The components of this package include:

GridFTP Module: It uses an extension of the standard FTP protocol that provides secure, efficient, and reliable data movements in Grid environments. In addition to standard FTP functions, GridFTP provides support for authenticated data transfer, third-party transfer invocation, and partial data transfer support.

Replica Location and Management Module: This component supports multiple locations for the same file throughout the Grid. Using the replica management functions, a file can be registered with the replica location service (RLS) and its replicas can be created and deleted.

2.5 Types of Grids

There are two basic ways to classify Grids; based on the organizational setup and based on the nature or type of resources they use. This section briefly discusses both types of Grids.

2.5.1 Classification of Grids According to Organizational Setup

Grids can be classified into the following types based on the organizational setup:

Cluster Grids: Cluster Grids are formed by a group of computers clustered together in a network to perform a particular project. A Cluster Grid is usually designed for a specific project keeping in mind the resource requirements of that particular project [CHE05]. A characterizing feature of a Cluster Grid is that the resources used by a project on a Cluster Grid are exclusively dedicated to that project.

Enterprise Grids: Enterprise Grids consist of resources spread across an enterprise and provide service to all users within that enterprise. An enterprise Grid is deployed within a large corporation that has a global presence or a need to access resources outside a single corporate location. Enterprise Grids can enable multiple projects or departments within an enterprise to share computing resources in a cooperative way [SPO03].

Extraprise Grids: Extraprise Grids are established between companies, their partners, and their customers. The Grid resources are generally made available through a virtual private network. Usually these Grids are established between organizations within similar industries, which have a need to collaborate on projects and use each other's resources as a means to reach a common goal [CAN03a].

Global Grids: Global Grids are the general purpose Grids established over the public

Internet. Global Grids provide the power of distributed resources to users anywhere in the world for computing and collaboration. They provide the ability to share compute and data storage resources across the public Web [CHE05].

2.5.2 Classification of Grids According to the Type of Resources

Grids can be classified into the following types according to the type of resources they use.

Compute Grids: Compute Grids are designed for compute-intensive tasks. Compute Grids usually consists of high speed computers. High speed communication links are not a requirement for a compute Grid as the amount of data that is transferred among constituent nodes in a Compute Grid is typically very small [CHE05].

Data Grids: The architecture of the Data Grids is suitable for data-oriented operations. Data Grids are characterized by the transfer of large volumes of data. Some sort of processing of data by the Grid may be required before it can be used [CIR01].

Utility Grids: Utility Grids are the commercial compute resources that are maintained and managed by a service provider. Customers typically purchase “cycles” from a utility Grid to complete the tasks they need to execute [BAS02].

2.5.3 Cloud Computing and the Grid

One major paradigm that has its roots in the advancement of Grid technologies is Cloud Computing. Cloud Computing is a computing paradigm in which tasks are assigned to a combination of connections, software and services accessed over a network [RAJ08][DOA10]. This network of servers and connections is collectively known as “Cloud.”

Ian Foster is considered one of the pioneers of Grid computing. He has defined Cloud Computing as [FOS08]:

“A large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet.”

This definition highlights the two important characteristics of Cloud Computing; virtualization and scalability. Cloud Computing abstracts from the underlying hardware and system software through virtualization. The virtualized resources are provided through a well-defined abstracting interface which may be an Application Programming Interface (API) or a service. Thus, at the raw hardware level, resources can be added or withdrawn according to demand posted through the interface, while the interface to the user remains the same. This architecture enables scalability and flexibility on the physical layer of a Cloud without having an impact on the interface to the end user. Clouds become a large pool of easily usable and accessible virtualized resources, such as hardware, development platforms and/or services. These resources can be dynamically reconfigured to adjust to a variable load, allowing also for an efficient resource utilization. This pool of resources is typically offered by a pay-per-use model.

Cloud Computing is based on four attributes: elasticity, pay as you go, self-provisioning of resources and scalability [JOH10]. Each of these is briefly described:

- 1) **Elasticity:** Users can rapidly increase and decrease their computing resources as needed, as well as release resources for other uses when they are no longer required. This capability allows users to increase and decrease their computing resources as needed, as Fig. 2.2 illustrates. There is always an awareness of the baseline for computing resources, but predicting future needs is difficult, especially when demands are constantly changing. Cloud Computing can offer a means to provide IT resources on demand and address spikes in usage.
- 2) **Pay as you go:** Users pay for only the resources they actually use and for only the time they require them.
- 3) **Self-provisioning of Resources:** Cloud Computing incorporates self-provisioning of resources, which is the ability of a system to expand and contract its pool of resources and optimize resources like CPU, memory, disk space and communication links, to provide continuous, quality bound services.
- 4) **Scalability:** Scalability is the ability of a system to continue to perform well as the load on the system is increased. Although organizations might have hundreds or thousands of systems, Cloud Computing provides the ability to scale to tens of thousands of systems, as well as the ability to massively scale bandwidth and storage space [JOH10].

Fig. 2.2: Attributes of Elasticity [MAT09]

Interest in the Cloud is growing because Cloud solutions provide users with access to supercomputer-like power at a fraction of the cost of buying such a solution outright. More importantly, these solutions can be acquired on demand; the network becomes the supercomputer in the Cloud where users can buy what they need when they need it. Cloud Computing identifies where scalable IT-enabled capabilities are delivered as a service to customers using Internet technologies.

Cloud Computing is used in two broad categories of applications: batch applications and interactive applications. Batch applications refer to programs that can run to completion without human interaction in a typically long (many minutes to hours or even more) process, and may involve intensive computations and/or handling large datasets. On the other hand, the interactive application needs human interaction and is typically a simple application consisting of

undemanding tasks. Cloud Computing has been proposed as a viable model in which a wide range of finer grained commercial, business, and scientific applications would tap into the Grid resources on an as-needed basis, extending the reach and utility of Grid computing far beyond its current user base. This vision of computing as a utility is expected to change not only the way scientists and businesses work, but also the way they think about computing resources. There are many companies that provide Cloud Computing. EC2 platform by Amazon is one of the most popular facilities available [AMA09]. Sun Grid Compute Utility [SUN09] is a Cloud Computing infrastructure made for batch applications. Google App Engine [LLO08] is a Cloud Computing platform suitable for interactive web applications. Windows Azure [AZU09] is a general Cloud Computing environment platform recently introduced by Microsoft.

The recognized working group with the mandate to standardize the Cloud technology is the Open Cloud Consortium [RIT10]. Its task is to develop a framework for interoperability among various Clouds. The OCC supports the development of benchmarks for Cloud Computing and is a strong proponent of open source software to be used for Cloud Computing. OCC manages a testing platform and a test-bed for Cloud Computing called the Open Cloud Test-bed. The group also sponsors workshops and other events related to Cloud Computing [SKO10]. An architecture for the Clouds that was popularized by a series of Google technical reports describes a storage Cloud providing a distributed file system, a compute Cloud supporting the MapReduce application, and a data Cloud supporting table services [WIS10] [DEA04]. The open source Hadoop system follows this architecture. These types of Cloud architectures support the concept of on-demand computing capacity [HAD10].

2.5.3.1 Relationship Between Grid and Cloud Computing

It can be observed that there seems to be many similarities in the description of Grid and Cloud Computing. There are many discussions around the question whether Grids and Clouds are the same or if there are substantial differences between Grid and Cloud Computing. Differences among Grid and Cloud Computing in various aspects are identified in [FOS08]. According to [JHA09], what makes Cloud Computing new and differentiates it from Grid Computing is virtualization, because Cloud Computing, unlike Grid, leverages virtualization to maximize computing power. While Grid Computing achieves high utilization through the allocation of multiple servers onto a single task or job, the virtualization of servers in Cloud Computing achieves high utilization by allowing one server to compute several tasks concurrently [HAR08]. An important point to note is that beside these technological differences between Grid and Cloud, there are differences in the typical usage pattern. Grid is typically used for job execution, e.g. the execution of a particular High Energy Physics application for a limited time. Cloud Computing supports a job usage pattern but is more frequently used to support long-running services [EGE10]. Some researchers seem to think that Cloud Computing has evolved from Grid Computing and that Grid Computing is the foundation for Cloud Computing [FOS08]. The relationship between Grid and Cloud Computing is described in [FOS08] as follows:

“We argue that Cloud Computing not only overlaps with Grid Computing, it is indeed evolved out of Grid Computing and relies on Grid Computing as its backbone and infrastructure support. The evolution has been a result of a shift in focus from an infrastructure that delivers storage and compute resources (such is

the case in Grids) to one that is economy based aiming to deliver more abstract resources and services (such is the case in Clouds).”

Thus, Cloud and Grid computing can be considered as complementary. Grid interfaces and protocols can enable the interoperability between resources of Cloud infrastructure providers. Grid solutions for job computing can run as a service on top of a Cloud having a distributed virtualized infrastructure [LLO08]. In addition, the potential benefits of simplicity offered by Cloud technologies, such as higher-level of abstractions [JHA09] may help to better serve current Grid users, attract new user communities, accelerate Grid adoption and importantly reduce operations costs [EGE10]. A summary of the comparison between Grid and Cloud Computing is shown in Table 2.1.

2.6 Approaches to Resource Allocation for Tasks on Grids

Different researchers have taken various approaches to resource allocation of Tasks on Grids. The approaches to allocate resources in Grids can be divided into three broad categories.

- 1) Traditional Schedulers and Resource Brokers
- 2) Policy based Resource Allocation
- 3) Workflow based Resource Allocation

Each of these approaches is discussed in a following subsection.

Table 2.1: Comparison of Grid and Cloud Computing

	Grid Computing	Cloud Computing
Means of utilization [HAR08]	Allocation of multiple servers onto a single task or job.	By virtualization of servers; one server may compute several tasks concurrently.
Typical usage pattern [EGE10]	Typically used for job execution, i.e. the execution of a program for a limited time.	More frequently used to support long-running services.
Level of abstraction [JHA09]	Expose high level of detail.	Provide higher level abstractions.

2.6.1 Traditional Schedulers and Resource Brokers

One of the traditional approaches is to use a Grid resource broker which selects suitable resources by interacting with various middleware services. Venugopal describes such a Grid resource broker that discovers computational and data resources running diverse middleware through distributed discovery services [VEN07]. However, any mechanism for breaking given task into parallel jobs for processing, is not present.

There are some schedulers that include the mechanisms to estimate the value of system parameters and the scheduling decisions depends on the estimation. Note that the performance of these scheduler is dependent the accuracy of the predictions of the system parameters. One of such scheduler called is called Prophet [BAS02]. The predictions of the system parameters is supplied by an auxiliary system of Prophet called the Network Resource Monitoring System. Prophet uses the information provided by the Network Resource Monitoring System to select the best number of computing nodes for a particular task. Prophet uses a callback mechanism to obtain application-specific information, and utilizes dynamic system information about CPU and

network usage at runtime obtained from the Network Resource Monitoring System to schedule the given task.

Decker and Diekmann have described CoPA in [DEC07], which is an environment that uses a special library to trace the execution of a group of tasks on Grids and analyze their behavior. Using heuristics CoPA searches for effective task mappings. However, CoPA is designed for compute intensive tasks only and ignores communication costs of data transfer.

Another scheduler is proposed by Subhlok et al. in [DEW07]. This scheduler uses an application specification interface, through which applications can specify their computation and communication requirements. The scheduler can invoke node selection algorithms that maximize computation and communication capacity.

YarKhan and Dongarra [YAR02] have also performed scheduling experiments in a Grid environment using simulated annealing. To evaluate the schedules generated by the simulated annealing algorithm they use a Performance Model, a function specifically created to predict the execution time of the program. Generating such a Performance Model requires detailed analysis of the program to be scheduled.

TITAN [CAO99] is a multi-tiered scheduling architecture that employs the PACE performance prediction system to improve resource usage efficiency. PACE uses a Performance Specification Language (PSL) to describe workloads for both sequential and parallel parts of an application. Also, PACE uses a Hardware Model Configuration Language (HMCL) to describe hardware characteristics. Furthermore, PACE uses a parametric evaluation engine and workload and hardware descriptions to provide execution time estimates. TITAN uses these estimates for scheduling purposes, e.g. as fitness function values for a genetic algorithm-based scheduler.

Another effort worth mentioning is Grid Application Development Software (GrADS) Project [BER05]. At the heart of the GrADS architecture is an enhanced execution environment which continually adapts the application to changes in the Grid resources, with the goal of maintaining overall performance at the highest possible level. A number of resource allocation algorithms can be used at GrADS to schedule a given bag-of-tasks in Grid environments. Due to the NP-complete nature of the task resource allocation problem the majority of proposed solutions are heuristic algorithms [CAO01] [BUR03] [GAR79].

2.6.2 Policy Based Resource Allocation

For resource allocation in Grids, some researchers have also proposed policy based resource allocation mechanisms. Sander et al. [SAN01] propose a policy based architecture for QoS configuration for systems that comprise different administrative domains in a Grid. They focus on making decisions when users attempt to make reservations for network bandwidth across several administrative network domains that are controlled by a bandwidth broker. The bandwidth broker acts as an allocator and establishes an end-to-end signalling process that chooses the most efficient path based on the available bandwidth. The work presented in [SAN01] is concerned with data transmission costs only; whereas the research presented in this research needs to consider both computation and communication costs associated with the PBDT tasks. Verma. et al. [VER02] has also proposed a technique in which resource allocation is performed based on a predefined policy. But in this research, the resource allocation is not based on any performance measure. Another policy based resource management system has been presented in [SUN02]. This work focuses on access control and takes care of user authentication and authorization. It does not concern resource allocation and it is the responsibility of the user to allocate resources in an efficient manner.

2.6.3 Workflow Based Resource Allocation

Many recent efforts have focused on scheduling of workflows in Grids. [GHA02] presents a QoS-based workflow management system and a scheduling algorithm that match workflow applications with resources using event condition action rules.

Pandey and Buyya have worked on scheduling scientific workflows using various approaches in the context of their GridBus workflow management effort [PAN08] [THA05]. [SEN05] has developed an architecture to specify and to schedule workflows under resource allocation constraints. Also, many of the data Grid projects that support distributed processing of remote data have proposed workflow scheduling [JAN03] [BRE03].

The topic of scheduling data-intensive workflows has also received attention recently. Shankar and DeWitt [SHA07] have augmented the popular Condor distributed computing system with support for a workflow planning framework that is cognizant of data requirements. Other data-centric workflow management systems were ZOO [IOA97], which concentrated on supporting data querying and visualization in scientific workflows, and GridDB [LIU04], which uses a functional data language to model workflows and relational tables to model program I/O in the context of the Condor system. GriPhyN [DEE02] was another project aiming at simplifying Grid computing through providing work and dataflow management functionality; when the problem size increases.

There are a number of reasons why scheduling programs or the tasks that comprise the programs is important. For users it is important that the programs they wish to run are executed as quickly as possible (lower makespan). On the other hand the owners of computing resources would wish to optimize their machine utilization. These two objectives, lower makespan and lower machine

utilization, are not always complementary. Owners are not usually willing to let a single user utilize all their resources, and users are not usually willing to wait an arbitrarily long time before they are allowed access to particular resources. Effective scheduling tries to satisfy the users' and the owners' requirements.

Note that this research is different from all the efforts mentioned above. In addition of allocation resources based on computational constraints, resource-allocation in this research is also based on data retrieval constraints. Furthermore, the data processing jobs of PBDT tasks considered in this research run in parallel configurations.

2.6.4 Scheduling Strategies

There are different approaches to the selection of resources for a given task to be performed. In the static approach, the resources are selected for the tasks before the execution of the first task. With the dynamic scheduling approach the resources are assigned at run-time depending on the availability. With the hybrid scheduling model, a combination of both static and dynamic scheduling strategies is used [LIN05].

Each of these approaches is described in the following sub-sections.

2.6.5 Static Scheduling

In the static approach, the resources in the Grid system are assigned to the tasks before the set of given tasks starts executing [JEO04]. An estimate of the time taken for computation and communication of the constituent jobs can be estimated *a priori*. One of the main benefits of the static model is that it is easier to implement from a scheduling and mapping point of view. Since the mapping of tasks is fixed *a priori*, it is easy to monitor the progress of computation.

Likewise, estimating the time that will be spent processing the jobs on a specific processor can be calculated. On completion of the processing of a group of tasks, the precise time that was actually spent in processing is used in making more accurate estimates of computing and communication costs for the new tasks [JEO04].

The static scheduling approach has a few drawbacks. It is based on an approximate estimation of processor execution times and inter-processor communication times. The actual execution time of a program may often vary from the estimated execution time and sometimes may result in a poorly generated schedule [JEO04]. This model also does not consider node and network failures.

2.6.6 Dynamic Scheduling

Dynamic scheduling operates on two levels: the local scheduling level and a load distribution level [TIE00]. The local scheduling level is responsible for managing the resource within a node only. For example, an individual node may consist of multiple processing units and local scheduling policy would determine the timing and order of the jobs assigned to it. The load distribution level determines how tasks would be placed on remote machines. It uses an information policy to determine the kind of information that needs to be collected from each machine, the frequency at which it needs to be collected and also the frequency at which it needs to be exchanged among different machines. In a traditional dynamic scheduling model, the jobs of a task are assigned to processors based on whether they can provide an adequate quality of service. The meaning of quality of service is dependent on the application. If a processor is assigned too many tasks, it may invoke a transfer policy to check to see if it needs to transfer tasks to other nodes and if so, to which ones [THA05]. The transfer of tasks can be sender

initiated or receiver initiated. If it is receiver initiated, a processor that is lightly loaded will voluntarily advertise to offer its services to heavily loaded nodes [SPO03]. Dynamic scheduling is more efficient and fault tolerant as compared to static scheduling because it can quickly react to the unexpected changes in the system parameters at runtime.

2.6.7 Hybrid Scheduling

Static scheduling algorithms are easy to implement and usually have a low schedule generating cost. However, since static scheduling is based on estimated execution times, it may not always produce the best schedules. On the other hand, dynamic scheduling uses run-time information in the scheduling process and generates potentially better schedules. But dynamic scheduling suffers from very high running times and may take prohibitively long time while trying to schedule very large applications with tens and thousands of jobs. Since both the scheduling techniques have their own advantages, researchers have tried to combine them to create a hybrid scheduling technique. In hybrid scheduling, the initial schedule is typically obtained using static scheduling and the jobs are mapped onto the respective processors. However, after execution commences, the processors use run-time information to check and see if the tasks can be mapped to better processors to yield a better system performance.

All scheduling strategies (static, dynamic and hybrid) use one of the Grid resource allocation algorithms to allocate resources. The difference among these scheduling strategies is the time when the resource allocation algorithm is run. The performance of a scheduling strategy is dependent on the resource allocation algorithm it is using and the selection of a suitable resource

allocation algorithm according to the system and workload parameters is pivotal for the satisfactory performance of a particular strategy.

2.6.8 Classes of Resource Allocation Algorithms

The Grid resource allocation algorithms found in literature can be divided into two classes.

2.6.8.1 Knowledge-based algorithms

This is the class of resource allocation algorithms that needs accurate knowledge of the system state (such as processing and communication times per unit data) during the time-slot in which the tasks are expected to be executing. The accurate knowledge of the future system conditions can be estimated by using a prediction mechanism; which predicts the future system parameters based on the past behaviour. Network Weather Stations (NWS) is one such mechanism [ANG06] that is widely used in general-purpose Grid systems. The accuracy of the predicted system conditions is a key factor in the efficient performance of the knowledge-based algorithms.

2.6.8.2 Knowledge-free Algorithms

This is the class of the resource allocation algorithms that can function without any need for predictions of the system state.

2.6.9 Examples of Resource Allocation Algorithms

Some algorithms (both knowledge-free and knowledge-based) that can be used to allocate resources in Grids are presented.

2.6.9.1 Workqueue Algorithm

Workqueue Algorithm (WQ) is one of the simplest algorithms used for resource allocation in a Grid computing environment. In WQ, the tasks in the given bag-of-tasks are chosen in an arbitrary order for processing [VEN07]. The chosen task is allocated randomly to a resource from the available pool of resources. If no resource is available, the chosen task waits in a queue until a resource becomes available. This process is repeated until all tasks in the given bag-of-tasks are processed. The problem with WQ algorithm arises when a large job is allocated to a slow node towards the end of the schedule. When this occurs, the completion of the task will be delayed until the complete execution of this job. Note that the WQ algorithm is a knowledge-free algorithm which does not need the knowledge about system parameters to function.

2.6.9.2 Workqueue with Replication

The Workqueue with Replication (WQR) is an extension of the WQ algorithm [VEN07]. The main disadvantage of the resource allocation using WQ is that if a large job gets allocated to a slow node towards the end of the schedule, this could increase the overall makespan of the application. In WQR, each of the tasks in the given bag-of-tasks is arbitrarily assigned one of the available nodes. If the number of nodes is less than the number of tasks, the remaining tasks wait in a queue. In this case, WQR behaves similar to WQ. But if there are nodes left after assigning nodes for all the tasks, then WQR algorithm starts to replicate the currently running tasks until all the available nodes are used. As soon as the data from a particular task reaches the sink nodes, all other replicas of that task are killed and the freed nodes are used to create replica of the tasks currently being processed. Note that WQR is also a knowledge-free algorithm which does not need the knowledge about the system parameters to function.

2.6.9.3 Suffrage and XSuffrage

Suffrage and XSuffrage [VEN07] are knowledge-based resource allocation algorithms that are used to schedule data-intensive tasks. The Suffrage algorithm assigns a task to a machine that would "suffer" the most in terms of earliest completion time, if a particular machine were not assigned to it. The suffrage value is defined as the difference between the best and the second best completion times for the task [TAK03]. The task that has the maximum suffrage value is given priority. An enhancement of Suffrage algorithm is the XSuffrage algorithm which considers data location in allocating resources for the task. The XSuffrage algorithm tries to minimize unnecessary data movement by grouping together the resources having high bandwidth. These groups of resources are called sites. While calculating the suffrage value, XSuffrage algorithm considers the resources within a site only.

2.7 Grid Resource Brokering Systems

Based on the resource allocation approaches and algorithms presented in Sections 2.6 many Grid resource broking systems have been developed. These systems provide uniform resource management and job submission, large-scale data management and transfer, and resource allocation and scheduling. Grid resource brokering systems are part of the Grid middleware and mediate between users and the underlying Grid fabric consisting of heterogeneous computing and storage resources connected by networks of varying capabilities.

This section presents examples of existing Grid middleware.

2.7.1 Condor-G

Condor-G is a resource management system that allows users to manage multi-domain, heterogeneous resources running Globus Toolkit [GLO09] and Condor middleware, as if the resources belong to a single domain. It combines the harnessing of resources in a single administrative domain provided by Condor with the resource discovery, resource access and security protocols provided by the Globus Toolkit. At the user side, Condor-G provides Application Program Interface (API) and command line tools to submit jobs, cancel them, query their status, and to access log files. A new Grid Manager daemon is created for each job request for submitting the job to the remote Globus gatekeeper that starts a new JobManager process. Condor-G provides “execute once” semantics by using a two phase commit protocol for job submission and completion. Fault tolerance is provided on the submission side by a persistent job queue and on the remote side by keeping persistent state of the active job within the JobManager. Jobs are executed on the remote resource within a mobile sandbox that traps system calls issued by the task back to the originating system and are checkpointed periodically using Condor mechanisms. This technology called “Condor GlideIn” effectively treats a collection of Grid resources as a Condor pool. Resource brokering is provided by matching user requirements with information available from services such as GRIS and GIIS through the ClassAds mechanism [VEN07]. Condor-G is a part of many projects such as EGEE, VDT and UK e-Science among others, and is used by workflow management systems such as Pegasus [ABB06]. Condor can utilize batch queuing systems such as LSF, PBS and NQE but only through Globus GRAM protocols. Condor-G provides strong fault tolerance mechanisms as a result of its close integration with the low-level Grid middleware.

2.7.2 AppLeS Parameter Sweep Template (APST)

APST is an environment for scheduling and deploying large-scale parameter sweep applications on Grid platforms. APST provides mechanisms for deploying applications on different Grid middleware and schedulers that take into account the workload characteristics. APST consists of two processes: the daemon, which deploys and manages applications and the client, which is a console for the users to enter their input. The input is XML-based and no modification of the application is required for it to be deployed on Grid resources. The APST scheduler allocates resources based on several parameters including predictions of resource performance, expected network bandwidths and historical data. The scheduler uses a Data Manager and a Compute Manager to deploy and monitor data transfers and computations respectively. These, in turn, use Actuators to talk to the various Grid middleware. A Metadata Manager talks to different information sources such as Network Weather Service (NWS) [ZHA06] and the Globus Monitoring and Discovery Service (MDS) [GLO09] and supplies the gathered data to the scheduler. APST supports different low-level Grid middleware through the use of Actuators and also allows for different scheduling algorithms to be implemented. However, it is focused towards parameter sweep applications. APST provides the ability to specify data repositories of different types in the input file and has a separate data manager to manage data transfers. However, it does not consider the possibility of multiple sources for any data file other than those created by replication of the data files during the execution of an application.

A number of different approaches for resource management on Grids were presented in this chapter. Existing research has addressed various issues in Grid resource management. To the best of my knowledge none of the existing works has dealt with resource management for PBDT tasks that this thesis focuses on.

Chapter 3 BiLeG Resource Management System

Grid computing focuses on large-scale, multi-institutional, resource sharing to deliver high performance. This sharing of resources can be used to facilitate virtual organizations, for allowing multi-disciplinary, multi-institutional collaborations. Within a Grid, a Grid resource management system plays the role of finding and allocating resources, that includes computing resources, storage and communication resources, in order to satisfy the Grid user requests [WOL03]. The performance of a particular Grid system depends on a well-designed resource management system that can provide the following four functions.

- 1) Discovery of the resources.
- 2) Management of the resources.
- 3) When required, division of each individual task into different jobs and management of the execution of the jobs.
- 4) Allocation of the resources for each constituent job of the tasks.

A number of techniques exists in the literature for providing functions described in Item 1 and Item 2. However there is no existing work that addresses the functions described in Item 3 and Item 4 for PBDT tasks. Thus, this research focuses on Item 3 and Item 4 and proposes an

effective bi-level Grid resource management system, BiLeG. This chapter discusses BiLeG, its various components and the related concepts in detail.

3.1 Processable Bulk Data Transfer (PBDT) Tasks

3.1.1 Introduction

An increasingly important class of Grids involves transfer of large volumes of data which has to be processed in some way before it can be used at the destination. Such type of Grid tasks are both compute-intensive and data-intensive. In this research, these tasks have been classified as Processable Bulk Data Transfer (PBDT) Tasks. PBDT tasks have many important applications (see Section 3.1.4). Due to their heavy resource-usage, efficient resource allocation for the PBDT tasks is pivotal to obtain satisfactory performance. Investigating efficient resource management solutions for the PBDT tasks is the focus of this research.

3.1.2 Characteristics

A task, T_i , is classified as a PBDT task if it has the following three characteristics.

- 1) The task involves large data transfer and the data has to be processed before it can be used at the set of sink nodes.
- 2) Cost of data processing is proportional to the length of the raw data file.
- 3) The unprocessed raw data file is such that it can be either processed as a whole or can be divided into multiple partitions. If divided into partitions, each partition can be processed independently. The resultant processed partitions can later be combined to generate the required processed file. Consider a source file F_i , of length L_i (see Fig. 3.1). F can be divided into p disjoint partitions, with lengths of $\{L_{i1}, L_{i2}, \dots, L_{ip}\}$, such that

$$L_i = \sum_{j=1}^{\rho} L_{ij} \quad (3.1)$$

Then, for a PBDT task, the length of the required processed file is given by:

$$L_i^{\sim} = \sum_{j=1}^{\rho} \varepsilon_i L_{ij} \quad (3.2)$$

where ε_i is a processing factor which is the ratio of the length of the processed partition and that of the original partition.

Fig. 3.1: Partitioning and Processing of a PBDT Raw Data File

For example, a typical multimedia project with an objective of applying a particular encoder to an uncompressed video can be classified as a PBDT task, as it possesses all the three characteristics required for a PBDT task. Uncompressed video is typically a large file, which is required to be encoded. The time taken to encode the file is directly proportional to the length of the file [ABB06]. This multimedia project also meets the third characteristic, because such files can be partitioned and each partition can be independently encoded [SKO10].

3.1.3 Types

PBDT tasks can be further classified based on the type of restrictions that the structure of their raw data files imposes on how they can be partitioned [AHM05] [AHM07] [AHM08a].

Flexibly Partitionable PBDT Tasks ($PBDT_{flexi}$): If the structure of the raw data file of a PBDT task imposes no restriction on how the raw data file is to be partitioned, then the task is called $PBDT_{flexi}$. For such tasks, it is up to the resource management algorithm to decide on the number and length of the partitions of a particular raw data file and allocate the partitions to various available nodes for processing.

Fixed Partitionable PBDT Tasks ($PBDT_{fixed}$): The second type of PBDT tasks have raw data files that can be partitioned within the constraints written in the metadata of the raw data file. Such tasks are classified as $PBDT_{fixed}$ tasks. For such tasks, the algorithm no longer has the freedom to partition the file but it can assign a particular partition to a certain node for processing.

Non-partitionable PBDT Tasks ($PBDT_{no-par}$): The third type of PBDT tasks is $PBDT_{no-par}$ that cannot be partitioned at all.

While deciding to divide a given PBDT task into jobs, the algorithm has to consider the type of the PBDT task to make appropriate partitioning and allocation decisions. The research presented in this thesis focuses on $PBDT_{fixed}$.

3.1.4 Practical Applications

The resource-intensive distributed tasks are becoming increasingly important [ABB06]. They are used in various multimedia, high-energy physics and medical applications. Many of these meet

the criteria that have been laid down for the PBDT tasks (see Section 3.1.2). The following section discusses some of the application examples for the distributed resource-intensive PBDT tasks.

3.1.4.1 Podcast Rendering

Media distribution via podcasts is a relatively new phenomenon which follows a different paradigm compared to traditional modes of content delivery. Podcasts are a push-based mechanism for distributing multimedia files such as videos over the Internet. Podcasting is an important Internet application with roughly six million subscribers [ABB06] and the subscriber population is growing rapidly towards a projected audience of 56 million by the year 2010 [TAN06][HSU05]. Currently big media companies, such as, ABC News, NBC News, ESPN, Disney, MTV, FOX, BBC, Apple, CNN and National Public Radio have introduced podcast programming [ENT08]. One of the requirements of creating a podcast is to encode the raw data files with “podcast-compliant encoders” [ABB06] [AHM05]. Specifically, in the case of video podcasts the requirement is to encode the file with MP4 encoders. Encoding the raw data file is a time-consuming resource-intensive process. One of the efficient ways to expedite the encoding process is to divide the raw data file into multiple partitions and to encode each of the partitions in parallel [ENT08]. The processed partitions are combined to form the required podcast. Fig 3.2 shows this process in detail.

In Step 1, Fig. 3.2, a video file is placed at a source node. Initially, a video is in an uncompressed form with default encoding. This form has been referred to as raw form in this thesis. Different types of encoding may be required to use this raw video on various types of devices.

For example, a mobile device may require a different type of encoding and processing than a desktop computer. In Fig. 3.2, the video has been shown to be processed and encoded in two formats, a Flash format and a MP4 format, which is a typical requirement in various multimedia delivery architectures, as discussed in [ENT08]. The raw data file is divided into various partitions where each of the partitions is encoded with both the MP4 encoder and the Flash encoder in parallel (Step 2). The Flash format data files are delivered to ubiquitous Flash players at various PCs (Step 3). MP4 formats are sent to various data farms (Step 4). Data Farms increase scalability as well as the storage capabilities of the system as discussed in Section 3.6. These multimedia files are transferred to a set of sink nodes where they are used by the end users (Step 5).

Fig. 3.2: Podcast Rendering

Note that this is one of the architectures used for the Podcast rendering. Depending on the requirements, this architecture can be modified. For example, if needed, a separate Data Farm can be added for Desktop Flash Format users as well.

3.1.4.2 Multimedia Encoding

Multimedia encoding is the process of applying a specific codec to an audio/video file through a computing system [SCH07]. Conventional methods use a single system for the conversion. If conventional methods are used, the encoding of the raw video data into MPEG-1, MPEG-2 or MPEG-4 format can take a great deal of time. Using a Grid system is an effective way to speed-up the encoding process. This method is used to encode multimedia in many popular online video related sites, such as Youtube and Facebook, to convert the videos uploaded by the users to the standard flv format [DOA10].

Fig. 3.3 compares the conventional encoding with the encoding process using a Grid. In conventional encoding (“Single Stream” in Fig. 3.3), the source file is in an analog form stored in a video tape which is quantized to produce an uncompressed digital raw data file. Depending on the quality level of the video capture, the data required for a typical one hour tape can create over 10 GB of video data. This raw data file is encoded using a computer system and needs to be compressed to approximately 650 MB to fit on a VideoCD. The compression stage is CPU intensive. If the conventional encoding process is used, the compression process can take a day or more depending on the quality level and the speed of the system being used [ENT08].

When a Grid is used to encode the video (“Using a Grid” in Fig 3.3) the raw video is broken down into multiple partitions and each of the partitions is sent to the Grid. Each of the partitions is then encoded and compressed in parallel (see Fig 3.3). The encoded partitions are transferred

to the destination node where they are combined. Note that as data transfer and data compression is performed in parallel, the whole process takes significantly lower amount time.

Fig. 3.3: Multimedia Encoding

3.1.4.3 Particle Physics Data Grids

Particle Physics Data Grids (PPDG) is a collaborative project concerned with providing next-generation infrastructure for high-energy and nuclear physics experiments [YAN02]. One of the important requirements of PPDG is to deal with the high volume of data that is created during high-energy physics experiments that must be analyzed by a large group of specialists. Data storage, replication, job scheduling, resource management and security components of the Grid must be integrated for use by the different collaborators. Processing of data used in high energy physics experiments require large computing facilities and fast communication capabilities. Grid computing is used for processing tasks related to high energy physics experimentation. Since

typical high-energy physics data impose limitations in the way it can be partitioned and processed [YAN02], PPDG processing tasks can be classified as $PBDT_{fixed}$ tasks.

3.1.4.4 International Virtual Data Grid Laboratory

International Virtual Data Grid Laboratory (iVDGL) has been funded to provide a global computer resource for several leading international experiments in physics and astronomy [ABB06]. The goal is to establish and utilize a virtual laboratory comprising heterogeneous computing and storage resources distributed throughout the world and linked by high-speed networks [CAN03b]. The aim is to provide a laboratory with distributed resources available to scientists located all over the globe. Grid computing is used to realize iVDGL [CAN03b]. Typically, iVDGL is used to process High Energy Physics related applications the tasks in which can be classified as $PBDT_{flexi}$ or $PBDT_{fixed}$. These tasks match the three PBDT characteristics described in Section 3.1.2.

3.2 Terminology

This research focuses on the problem of allocating resources for a given set of related PBDT tasks. Note that many multimedia encoding and High Energy Physics applications comprise a set of related tasks, each of which can be run independent of the other [BUN03][CRO04]. To synthesize a representative PBDT workload in a simulated environment, characteristics of such applications, that can be classified as the PBDT tasks, were carefully studied and it was concluded a single bag-of-tasks is the best way to model a PBDT application. The bag-of-task represents the constituent tasks of a particular PBDT application and has been used in this research to model the constituent tasks of a PBDT application. The bag-of-tasks consists of a set of independent PBDT tasks each of which must be executed successfully. In the BiLeG

architecture, various Grid nodes are used for different specialized functions by a PBDT task. The following terminology has been used in this research to identify various nodes participating in the processing of a PBDT task depending upon their particular function.

Grid node: A computing resource within a Grid system.

Source: The node where the unprocessed raw data file is located.

Sink: A set of nodes where the processed file is to be delivered.

Compute-Farm: A set of Grid nodes dedicated to process the raw data file in parallel.

Data-Farm: A set of Grid nodes dedicated for replicating the data.

The Grid system consists of n nodes. Collectively, these n nodes are represented by a set Δ . Each individual PBDT task in the given bag-of-tasks may be divided into a number of jobs which can be executed in parallel, independent of the other. Each job is used to process a given data partition. As discussed, PBDT tasks are resource-intensive tasks that use a great deal of computing resources and communication bandwidth. Usually, if a node starts processing a PBDT task, pre-emption of this task is counter-productive, as it wastes the effort of transferring the raw data file to the concerned node. Also, due to a high demand of computing power, a node can handle the processing of only one PBDT task at a time. Thus, it is assumed that the constituent jobs of a task are not pre-emptable. Note that this assumption is common in different data-intensive distributed computing environments as it can be costly to perform context switching in such environments (see [RAY04], for example). We have made the following two assumptions regarding the running of constituent jobs of a task on Grid nodes.

- 1- Once a job starts executing on a Grid node, it cannot be pre-empted.

2- Only one job can be executed on a Grid node at a time.

One Giga-Byte (GB) is chosen as a unit of data. The time it takes to process one unit of data by each of the nodes in the system is represented by a vector of length $|\Delta|$ denoted by $[C_{p_\Delta}]$. A matrix $[C_{c_\Delta}]$ of dimensions $|\Delta| \times |\Delta|$ denotes the time to transfer one unit of data between all the nodes in Δ . The emerging on-demand Grid environment model is used to calculate the costs associated with the processing of the given bag of PBDT tasks. In the on-demand environments, the computing and communication facilities are charged based on the total usage of the each individual facility and is currently offered by several utility Grid and Cloud Computing environments [AZU09][AMA09][GOO09]. In this research, $d(i,j)$ denotes the time in hours required to transfer one unit of data from node i to node j . Similarly, the time in hours taken by a particular computing node to process one unit of data is represented by C_{p_m} . Each computing facility or communication link is charged based on the total time for which it is used. The cost of using a computing facility is measured in dollars per hour and is represented by β . Note that in this research dollar is chosen as the currency for cost calculations. But it can be replaced by any other currency based on the user requirements. The cost of using a communication link is also measured in dollars per hour. The cost to transfer one unit of data from the source to a Grid-node is called the transfer-in cost and is represented by α_{in} . The cost to transfer one unit of data from a Grid-node to the sink node is called the transfer-out cost and is represented by α_{out} . The cost to transfer one unit of data between two Grid nodes is called transfer-within cost and is represented by α_Δ . The performance objective of this research is to assign resources in such a manner that the total cost in performing the given bag-of-tasks is minimized. The total cost is defined as the total amount of dollars spent in executing all the tasks in the given bag-of-tasks. A secondary objective is to achieve low completion times for a given bag-of-tasks while minimizing the cost.

3.3 The BiLeG Architecture

The BiLeG resource management system consists of two decision making modules; a lower level decision making module called Resource Allocator (RA) and a higher level decision making module called Task Resource-pool Selector (TRPS). TRPS selects a task T_i from the given bag of PBDT tasks and allocates it a resource-pool which is a subset of all the available resources. A resource-pool of a particular task T_i is represented by Γ_i , where $\Gamma_i \subseteq \Delta$. RA allocates resources for a particular task T_i chosen from its associated resource-pool Γ_i .

First, PBDT tasks are submitted to the system. Each PBDT task consists of an unprocessed raw data file, information about the processing operation that is required to be performed on the raw data file and a set of sink nodes where the processed file is to be delivered. The code for processing each task is stored in each node of the Grid. Typically all the tasks in a given bag-of-tasks are the associated with the same processing operation. The unprocessed raw data file is stored at the source node. The process is started by grouping the constituent tasks of a PBDT application into a bag-of-tasks (Fig. 3.4, Step-1) and by sending an “initiate” signal to the TRPS (Fig. 3.4, Step-2). TRPS determines how many Grid nodes are present in the system (Fig. 3.4, Step-3). The set of nodes in the system is represented by Δ . TRPS determines a resource-pool Γ_i for each of the tasks T_i . Note that not all the Grid nodes available in the system are visible to an individual task T_i in the bag-of-tasks, T . Typically, each individual task has a different resource-pool selected by TRPS according to the policy used (see Section 3.5 for a discussion of TRPS policies). For an individual task, using all the resources of the resource-pool may not be the best option for the most efficient execution. A policy is deployed at TRPS to determine the way in which TRPS chooses a resource-pool for each individual task. This choice depends on the

existing system parameters and resource availability.

Fig. 3.4: BiLeG Architecture

From the resource-pool I_i , allocated by TRPS to T_i , the lower level decision making module (RA) chooses a set of resources that are used to perform T_i . This set of resources is denoted by ω_i . For different systems, different resource allocation algorithms may be best suited at RA. The remaining resources ($I_i - \omega_i$) are returned to TRPS. Based on the resources chosen by the algorithm, RA divides a particular task into different jobs. RA specifies the details of the jobs of a particular task T_i in a file called the *workflow* of T_i . A workflow describes the order in each of the constituent job of a task must be executed. It also specifies the node responsible for executing a particular job. A *workflow engine* provides the run-time environment for activating, managing and executing a workflow generated by RA. A workflow engine interprets events and acts on them. The workflow engine provides two functions. First, it enforces the workflow of a particular task T_i generated by RA by executing each constituent job as specified in the

workflow. Second, it monitors the execution of the jobs in the workflow. Note that the workflow engine is needed to be run on a computer which has a direct access to all the nodes in the system. Source node may be a good choice to be used for running the workflow engine.

A combination of a TRPS policy and an RA algorithm is called an *Allocation-plan* and is represented by $\{<\text{Policy}, \text{Algorithm}>\}$. For a particular task, one of the policies and one of the RA algorithms makes a valid allocation-plan. This thesis explores the factors that determine the choice of an efficient allocation-plan for a given bag-of-tasks.

Note that the visibility of RA for a particular task is limited to the task's resource-pool. RA is myopic in nature and is not concerned with the overall system performance. The objective of RA is to optimize the performance for a particular task only. TRPS is concerned with the global system performance and has the responsibility to choose an appropriate resource-pool for each of the tasks and pass it on to RA. RA assigns a set of resources from the set of resource passed to it.

Note that in the BiLeG architecture, mechanisms to provide security related services are not considered. In an environment in which security is important, a security layer will be needed to provide security related services, such as, single sign-on for using Grid services, authentication, data encryption, authorization and the delegation of authority. The addition of a security layer will have an impact on the overall performance of the system which is not analyzed in this research and may form a direction for future research.

3.4 Flexibility

By bifurcating the overall system into two independent decision making modules and by assigning both decision making modules separate responsibilities, the BiLeG architecture divides

the problem of resource allocation for the given bag-of-tasks into the following two sub-problems.

- 1) Selection of resource-pool I_i for each task T_i in T (assigned to the upper decision making module, TRPS)
- 2) Resource allocation for each constituent job in the given bag-of-tasks from I_i (assigned to the lower decision making module, RA).

The division into two independent sub-problems makes the architecture customizable as these two sub-problems are be solved by two independent algorithms. The provision to change one of these two algorithms independent of the other makes the system modular and increases its flexibility and adaptability.

3.5 Task Resource-pool Selection Policies

A policy defines the way in which the TRPS decision making module chooses the resource-pool for a task. The ability to define a policy at the TRPS decision making module provides the flexibility in the system to handle different types of PBDT tasks differently based on their workload characteristics, system parameters and user requirements.

Based on when this mapping occurs, the TRPS policies can be divided into three different types; static, dynamic and hybrid.

3.5.1 Static Policies

A TRPS policy is said to be static if the resource-pools of all the tasks in the given bag-of-tasks are determined before the system starts executing the tasks. A static policy gives rise to two phases; a *mapping phase* and an *execution phase*. The mapping phase is completed before the execution of the tasks starts. In the mapping phase, first, the resource-pool for a particular task is chosen. Then, the chosen resource-pool and the corresponding task are passed to RA, which decides which resources from its resource-pool will be used to process this task. In the execution phase, the task is executed using the resources allocated to it by RA in the mapping phase.

3.5.1.1 States of a PBDT Task in the Static TRPS Policies

This section explains the lifecycle of a PBDT task in a BiLeG architecture when a static TRPS policy is used. Fig. 3.5 shows the different states that a PBDT task goes through when a static policy is used at TRPS.

Initially, when a task is added to the bag-of-task, it is in “STATE 0: Idle/Unmapped” state. TRPS assigns a resource-pool to the task and uses the RA algorithm to allocate the resources to it. Once resources have been allocated to it by RA, the state of the task state is changed to “STATE 1:Idle/Mapped” (see Fig. 3.5). If the set of resources it has been mapped to are available, its state is changed to “STATE 2: Enabled for execution”. If the complete set of resources allocated to a task is not available, the task has to wait until the set of resources it needs becomes available and its state is changed to “STATE 3: Waiting”. When the set of

Fig. 3.5 Different States of a Task for Static Policies

resources it needs becomes available its state changes to “STATE 2: Enabled for execution”. Once a task is enabled for execution, its constituent jobs start running and its state changes to “STATE 4: Executing”. When all the constituent jobs of a task are complete, the state of the task changes to “STATE 5: Complete”.

The following sub-sections present two static policies investigated in this thesis.

3.5.1.2 Static Resource-Pool with Single Partition (SRP_{sp})

One of the simplest TRPS policies is SRP_{sp}. Being a static TRPS policy, it is divided into a mapping phase and an execution phase. Pseudo code of SRP_{sp} is shown in Fig. 3.6, Fig. 3.7 and Fig. 3.8.

Fig. 3.6 shows the mapping phase for SRP_{sp} . In the mapping phase, a mapping between all tasks and the set of resources they will use, is determined and stored in a mapping set called Ω . Initially the mapping set Ω is empty (Fig. 3.6, Line 2). In SRP_{sp} , each task in the given bag-of-tasks is assigned a resource-pool that includes all the Grid nodes. Specifically, if I_i is the resource-pool associated with a particular task T_i then for each task $T_i \in T$, $I_i = \Delta$ (Fig. 3.6, Line 4). To create the mapping, TRPS iteratively calls the algorithm at RA for each task in T (Fig. 3.6, Line 5). For a particular task T_i , the RA algorithm returns the set of the corresponding resources chosen by RA which is represented by ω_i . T_i and ω_i , are added to the mapping set (Fig. 3.6, Line 6). The state of T_i is changed from “STATE 0:Idle/Mapping” to “STATE 1:Idle/Mapped” (Fig. 3.6, Line 7). Once the mapping for all tasks in T is complete, Ω contains the mapping between all the tasks in T and the set of resources allocated to them.

Fig. 3.7 shows the execution phase for SRP_{sp} . Note that Δ_{free} represents all resources that are available at a particular time. Initially, all resources are available and thus $\Delta_{free} = \Delta$ (Fig. 3.7, Line 2). TRPS iterates through all the tasks in the given bag-of-tasks (Fig. 3.7, Line 3). For each task T_i it checks whether the set of resource it needs to run, ω_i , is available (Fig. 3.7, Line 4). For the first task, T_1 , the set of resource are always be available, i.e. $\omega_i \subseteq \Delta_{free}$ is always true because

```

1: BEGIN Mapping Phase
2:  $\Omega = \{\}$ 
3: FOREACH  $T_i \in T$ 
4:    $\Gamma_i = \Delta$ 
5:    $\omega_i = \text{RA}(\Gamma_i, T_i)$ 
6:   add  $\{\omega_i, T_i\}$  to  $\Omega$ 
7:   state_change $_{T_i}$  [STATE 0:Idle/Unmapped] → [STATE 1:Idle/Mapped]
8: END FOREACH
9: END Mapping Phase

```

Fig. 3.6: Mapping Phase of SRP_{sp}

```

1: BEGIN Execution Phase
2:  $\Delta_{\text{free}} = \Delta$ 
3: FOREACH  $T_i \in T$ 
4:   IF  $(\omega_i \subseteq \Delta_{\text{free}})$ 
5:     state_change $_{T_i}$  [STATE 1:Idle/Mapped] → [STATE 2:Enabled for execution]
6:      $\Delta_{\text{free}} = \Delta_{\text{free}} - \omega_i$ 
7:     state_change $_{T_i}$  [STATE 2:Enabled for execution] → [STATE 4:Executing]
8:     remove  $T_i$  from  $T$ 
9:     IF  $T == \{\}$ 
10:      ALLOCATION_DONE_FLAG = true
11:      BREAK
11:    ENDIF
12:   ELSE
13:     state_change $_{T_i}$  [STATE 1:Idle/Mapped] → [STATE 3:Waiting/Mapped]
14:   ENDIF
15: END FOREACH
16: END Execution Phase

```

Fig. 3.7: Execution Phase of SRP_{sp}

```

1: BEGIN EVENT HANDLER for  $T_i$  COMPLETION
2: state_change $_{T_i}$  [STATE 4:Executing] → [STATE 5:Completed]
3:  $\Delta_{\text{free}} = \Delta_{\text{free}} + \omega_i$ 
4: IF  $T == \{\}$ 
5:   ALLOCATION_DONE_FLAG = true
6:   BREAK
7: ELSE
10:   FOREACH  $T_j \in T$ 
11:     IF  $(\omega_j \subseteq \Delta_{\text{free}})$ 
12:       state_change $_{T_j}$  [STATE 3:Waiting/Mapped] → [STATE 2:Enabled for execution]
13:        $\Delta_{\text{free}} = \Delta_{\text{free}} - \omega_j$ 
14:       state_change $_{T_j}$  [STATE 2:Enabled for execution] → [STATE 4:Executing]
15:       remove  $T_j$  from  $T$ 
16:     ENDIF
17:   END FOREACH
18: ENDIF
19: END EVENT HANDLER for  $T_i$  COMPLETION

```

Fig. 3.8: Task Completion Event Handler for SRP_{sp}

in the first iteration, $\Delta_{\text{free}} = \Delta$ and hence the complete set of the resources is available for T_1 . In the first iteration, TRPS enables T_1 for execution and changes its state to “STATE 2: Enabled for execution” (Fig. 3.7, Line 5). The resources to be used by T_1 are subtracted from Δ_{free} (Fig. 3.7, Line 6). The state of T_1 is changed to “STATE 4: Executing” (Fig. 3.7, Line 7) and T_1 is removed from T to indicate that it has been allocated resources (Fig. 3.7, Line 8). After removing T_1 from the bag-of-tasks T , it is checked that whether T is empty or not (Fig. 3.7, Line 9). If T is empty, a flag named `ALLOCATION_DONE_FLAG` is set to indicate that the resource allocation has been completed for the given bag-of-tasks T (Fig. 3.7, Line 10).

For each subsequent task T_i , TRPS checks whether the set of resource it needs to run, ω_i , is available or not (Fig. 3.7, Line 4). If ω_i is available, the state of task T_i is changed to “STATE 2: Enabled for execution” (Fig. 3.7, Line 5). TRPS removes ω_i from Δ_{free} to reflect that the resources in ω_i are no more available (Fig. 3.7, Line 6). The state of T_i is changed to “STATE 4: Executing” (Fig. 3.7, Line 7) and T_i is removed from T to indicate that it has been allocated resources (Fig. 3.7, Line 8). After removing T_i from the bag-of-tasks T , it is checked that whether T is empty or not (Fig. 3.7, Line 9). If T is empty, `ALLOCATION_DONE_FLAG` is set to indicate that the resource allocation has been completed for the given bag-of-tasks T (Fig. 3.7,

Line 10). If the complete set of resources ω_i is not available, the state of T_i is changed to “STATE 3, Waiting/Mapping” (Fig. 3.7, Line 13).

Fig. 3.8 shows “Completion Event Handler” for one of the tasks T_i . “Completion Event Handler” executes when a particular task T_i is completed. First, the state of the completed T_i is changed from “STATE 4:Executing” to “STATE 5: Completed” to indicate the completion of the task (Fig. 3.8, Line 2). Then, the resources that were used by T_i are returned to Δ_{free} (Fig. 3.8, Line 3). After returning the resources, TRPS checks that whether the given bag-of-tasks, T , is empty or not (Fig. 3.8, Line 4). If T is empty, it means that there is no task waiting for the resources and all tasks are either in “STATE 4:Executing” or in “STATE 5:Complete”. TRPS changes the `ALLOCATION_DONE_FLAG` to true to reflect that each task has been allocated the resources. If T is not empty, TRPS iterates through all the tasks in T and checks for each of the tasks whether the set of resource it needs, ω_i , is available or not. If for a task T_i , ω_i is available, TRPS changes the state of T_i to “STATE 2: Enabled for execution” (Fig. 3.8, Line 12). Then, the set of resources, ω_i is assigned to T_i and is taken off from Δ_{free} (Fig. 3.8, Line 13). The state of T_i is changed to “STATE 4: Executing” (Fig. 3.8, Line 14). T_i is removed from T to indicate that resources needed by T_i have been allocated.

Note that `ALLOCATION_DONE_FLAG` indicates that the resource allocation of the given bag-of-tasks is complete.

3.5.1.3 Static Resource-Pool with Multiple Partitions (SRP_{mp})

In SRP_{mp} , the set of all available resources, Δ , is divided into q equal-sized partitions. These partitions are represented by $\Delta_1, \dots, \Delta_q$. Fig. 3.9 shows the mapping phase for SRP_{mp} . k represents the index of a partition of Δ currently used as the resource-pool by TRPS in a particular iteration.

Initially, the mapping set Ω is empty (Fig. 3.9, Line 2) and the first partition is selected as the resource-pool for task T_1 (Fig. 3.9, Line 3). TRPS iteratively calls the RA algorithm for each task T_i with one of the partitions chosen as the resource-pool. The RA algorithm returns the set of resources chosen for the task T_i represented by ω_i (Fig. 3.9, Line 5). T_i and ω_i , are added to the mapping set (Fig. 3.9, Line 6). The state of T_i is changed from “STATE 0:Idle/Mapping” to “STATE 1:Idle/Mapped” (Fig. 3.9, Line 7).

Note that for first iteration, $k=1$ and the first partition is selected. For each subsequent iteration the value of k is increased until $k>q$. If k becomes greater than q , it is reset to 1 again. (Fig. 3.9, Line 9-10). This process is repeated until a mapping between all tasks and the set of resources they will use, is determined and stored in a mapping set called Ω . Once each of the tasks is mapped to a set of the resources in the mapping phase, it is executed.

```

1: BEGIN Mapping_Phase
2:  $\Omega = \{ \}$ 
3:  $k=1$ 
4: FOREACH  $T_i \in T$ 
5:    $\omega_i = \text{RA}(\Delta_k, T_i)$ 
6:   add  $\{\omega_i, T_i\}$  to  $\Omega$ 
7:   state_change $_{T_i}$  [STATE 0:Idle/Unmapped]  $\rightarrow$  [STATE 1:Idle/Mapped]
8:    $k=k+1$ 
9:   IF ( $k>q$ )
10:     $k=1$ 
11:  ENDIF
12: END FOREACH
13: END Mapping_Phase

```

Fig. 3.9: Mapping phase of SRP_{mp}

The algorithm for the execution phase and the Task Completion Event Handler are exactly the same as the algorithm for the execution phase and the Task Completion Event Handler of SRP_{sp} .

3.5.2 Dynamic Policies

With a dynamic TRPS policy only the resources for the first task in the given bag-of-tasks are determined before the system starts executing. The resources for all other tasks are determined dynamically according to their availability. The resource-pool for the first task is determined and passed on to RA, which chooses a set of resources from the allocated resource-pool and assigns this set to the task.

3.5.2.1 States of a PBDT Task in the Dynamic TRPS Policies

Fig. 3.10 shows the different states of a PBDT task in dynamic TRPS policies. Initially, when a task is added to the bag-of-task, it is in “STATE 0: Idle/Unmapped” state. TRPS checks whether any resources are free in the given set of resources, Δ . If resources are free, TRPS combines these resources to form a resource-pool for the task. TRPS uses the RA algorithm to allocate the

Fig. 3.10 Different States of a Task for Dynamic Policies

resources to the task from the assigned resource-pool. Once the RA algorithm chooses the resource for the task, TRPS changes the state of the task to “STATE 1:Idle/Mapped” (see

Fig. 3.10). If no resources are free in Δ to form the resource-pool, TRPS changes the state of the task to “STATE 3: Waiting/Unmapped” and the task has to wait for some resources to be released. When an executing task completes and some resources are released, the RA algorithm is run, and resources are allocated to the task and TRPS changes the state of the task to “STATE 1:Idle/Mapped”.

Unlike the static TRPS policies, in dynamic TRPS policies, once a task is in “STATE 1:Idle/Mapped” state, its state is always immediately changed to “STATE 2: Enabled for execution”. When a task is enabled for execution, its constituent jobs start running and TRPS changes its state to “STATE 4: Executing”. When all the constituent jobs of a task are complete, TRPS changes the state of the task to “STATE 5: Completed”.

3.5.2.2 Reducing Resource-pool Algorithm

All dynamic TRPS policies are based on a Reducing Resource-pool Algorithm which is shown in Fig. 3.11. The input for the Reducing Resource-pool Algorithm is a set of resources, R and a set of tasks, S (Fig. 3.11, Line 1). Note that both R and S are non-empty sets. The algorithm iterates through all the tasks in S (Fig. 3.11, Line 2). For the first task in S , T_1 , there is always a set of resources available to form a resource-pool as R is a non-empty set of resources. TRPS checks that the number of resources is greater than, ρ_i , the number of partitions of raw data file of T_i (Fig. 3.11, Line 3).

```

1: BEGIN REDUCING_POOL_ALGORITHM (set_of_resources R, set_of_tasks S)
2:   FOREACH  $T_i \in S$ 
3:     IF ( $|R| > \rho_i$ )
4:        $\omega_i = RA(R, T_i)$ 
5:        $state\_change_{T_i} [?] \rightarrow [STATE\ 1: Idle/Mapped]$ 
6:        $state\_change_{T_i} [STATE\ 1: Idle/Mapped] \rightarrow [STATE\ 2: Enabled\ for\ execution]$ 
7:        $R = R - \omega_i$ 
8:        $state\_change_{T_i} [STATE\ 2: Enabled\ for\ execution] \rightarrow [STATE\ 4: Executing]$ 
9:       remove  $T_i$  from S
10:    ELSE
11:      IF  $R == \{\}$ 
12:        FOREACH  $T_k \in S$ 
13:          IF ( $state_{T_k} == STATE\ 0: Idle/Unmapped$ )
14:             $state\_change_{T_k} [STATE\ 0: Idle/Unmapped] \rightarrow [STATE\ 3: Waiting/Unmapped]$ 
15:          ENDIF
16:        END FOREACH
17:        RETURN S
18:      END IF
19:    END FOREACH
20:  FOREACH  $T_k \in S$ 
21:    IF ( $state_{T_k} == STATE\ 0: Idle/Unmapped$ )
22:       $state\_change_{T_k} [STATE\ 0: Idle/Unmapped] \rightarrow [STATE\ 3: Waiting/Unmapped]$ 
23:    ENDIF
24:  END FOREACH
25:  RETURN S
26: END REDUCING_POOL_ALGORITHM

```

Fig. 3.11: Reducing Resource-pool Algorithm

If the number of resources is greater than ρ_i , TRPS runs the RA algorithm, with R assigned as the resource-pool, which results in the allocation of resources to T_1 represented by ω_1 (Fig. 3.11, Line, 4). The state of the first task is changed to “STATE 1: Idle/Mapped” to indicate that the task is now mapped to a set of resources (Fig. 3.11, Line, 5). Once a task is mapped to set of resources, its state is changed to “STATE 2: Enabled for execution” (Fig. 3.11, Line, 6). The set of resources, selected by the RA algorithm for T_1 , is taken out from R (Fig. 3.11, Line 7). The state of T_1 is changed to “STATE 4: Executing” to indicate that it is running (Fig. 3.11, Line 8). The task is removed from the set of tasks S to indicate that the resource allocation for T_1 is complete (Fig. 3.11, Line 9). TRPS checks whether any more resources are available or not

(Fig. 3.11, Line 11). If no more resources are available, the state of each task in S is changed to “STATE 3: Waiting/Unmapped” and S is returned (Fig. 3.11, Line 12-17). If resources are available, and there are tasks left in S , this process is repeated for each of the subsequent tasks. Once the algorithm iterates through all the tasks, it changes the state of each remaining tasks (if any) to “STATE 3: Waiting/Unmapped” and S is returned (Fig. 3.11, Line 21-23).

Note that the Reducing Resource-pool Algorithm returns an empty set S to indicate all the tasks have been assigned resources. Otherwise, it returns the set of tasks that are not assigned any resources.

3.5.2.3 Dynamic Resource-Pool with Single Partition (DRP_{SP})

The algorithm for DRP_{SP} is shown in Fig. 3.12. Initially, the available pool of resources is equal to all the resources in the system (Fig. 3.12, Line 2). TRPS invokes the Reducing Resource-pool Algorithm and provides Δ_{free} as the resource-pool and T as the set of tasks (Fig. 3.12, Line 3). If the Reducing Resource-pool Algorithm is successful in allocating resources for all the tasks in T , an empty set is returned and $ALLOCATION_DONE_FLAG$ is set to true. Otherwise, the set of tasks, which are not allocated the resources, is returned by the Reducing Resource-pool algorithm.

```

1: BEGIN  $\text{DRP}_{\text{sp}}$  POLICY
2:  $\Delta_{\text{free}} = \Delta$ 
3:  $T = \text{REDUCING\_RESOURCE\_POOL\_ALGORITHM}(\Delta_{\text{free}}, T)$ 
4: IF  $T == \{\}$ 
5:    $\text{ALLOCATION\_DONE\_FLAG} = \text{true}$ 
6: ENDIF
7: END  $\text{DRP}_{\text{sp}}$  POLICY

```

Fig. 3.12: Dynamic Resource-Pool Single Partition Policy

```

1: BEGIN EVENT HANDLER for  $T_i$  COMPLETION
2:  $\Delta_{\text{free}} = \Delta_{\text{free}} + \omega_i$ 
3:  $\text{state\_change}_{T_i}$  [STATE 4:Executing]  $\rightarrow$  [STATE 5:Completed]
4: IF  $T == \{\}$ 
5:    $\text{ALLOCATION\_DONE\_FLAG} = \text{true}$ 
6:   BREAK
7: ELSE
8:    $T = \text{REDUCING\_RESOURCE\_POOL\_ALGORITHM}(\Delta_{\text{free}}, T)$ 
9:   IF  $T == \{\}$ 
10:     $\text{ALLOCATION\_DONE\_FLAG} = \text{true}$ 
11:   ENDIF
12: ENDIF
13: END EVENT HANDLER for  $T_i$  COMPLETION

```

Fig. 3.13: Task Completion Event Handler for DRP_{sp}

Fig. 3.13 shows “Completion Event Handler” for T_i . “Completion Event Handler” executes when a particular task T_i is completed. First, the resources used by the task are returned to Δ_{free} . (Fig. 3.13, Line 2). Then, the state of the task T_i is changed to “STATE 5: Completed” (Fig. 3.13, Line 3). TRPS checks that whether there are more tasks left in T (Fig. 3.13, Line 4). If there are tasks left in T , TRPS invokes the Reducing Resource-pool Algorithm and provides Δ_{free} as the list of resources and T as the list of tasks (Fig. 3.13, Line 8). If the Reducing Resource-pool algorithm is successful in allocating resources for all the tasks in T , an empty set is returned and $\text{ALLOCATION_DONE_FLAG}$ is set to true (Fig. 3.13, Line 10).

3.5.2.4 Dynamic Resource-pool with Multiple Partitions (DRP_{mp})

DRP_{mp} limits the maximum number of nodes in the resource-pool of a task T_i to a constant, n_Γ , where $0 < n_\Gamma \leq |\Delta|$. Fig. 3.14 and Fig. 3.15 show the pseudo code for DRP_{mp}. At any instance, Δ_{free} represents all the nodes that are available in the system. Initially all nodes are free, so $\Delta_{free} = \Delta$ (Fig. 3.14, Line 2). TRPS checks that whether n_Γ or $|\Delta_{free}|$ has a lower value and allocates it to a variable m (Fig. 3.14, Line 4). TRPS randomly chooses m nodes from Δ_{free} and combines them as a resource-pool represented by Γ (Fig. 3.14, Line 5). TRPS invokes the Reducing Resource-pool Algorithm with Γ as resource-pool and T as the list of tasks (Fig. 3.14, Line 6). If the Reducing Resource-pool Algorithm is successful in allocating resources for all the tasks in T , the given bag-of-tasks T becomes empty and ALLOCATION_DONE_FLAG is set to true. Fig. 3.15 shows “Completion Event Handler” for T_i . “Completion Event Handler” executes when a particular task T_i is completed. First, the resources used by the task are returned to Δ_{free} (Fig. 3.15, Line 2). Then the state of the task T_i is changed to “STATE 5: Completed” (Fig. 3.15, Line 3). TRPS checks whether there are any more tasks left in T (Fig. 3.15, Line 4). If there are no tasks left in T , ALLOCATION_DONE_FLAG is set to true (Fig. 3.15, Line 5). Otherwise, TRPS checks that whether n_Γ or $|\Delta_{free}|$ has a lower value and allocates it to a variable m (Fig. 3.15, Line 8). TRPS randomly chooses m resources from Δ_{free} and assigns them to the resource-pool Γ (Fig. 3.15, Line 9). TRPS invokes the Reducing Resource-pool Algorithm and provides it Γ as resource-pool and T as the list of tasks (Fig. 3.15, Line 10). If the Reducing Resource-pool algorithm is successful in allocating resources for all the tasks in T , an empty set is returned and ALLOCATION_DONE_FLAG is set to true (Fig. 3.15, Line 12).

```

1: BEGIN  $\text{DRP}_{\text{mp}}$  POLICY
2:  $\Delta_{\text{free}} = \Delta$ 
3: FOREACH  $T_j \in T$ 
4:    $m = \min(n_T, |\Delta_{\text{free}}|)$ 
5:   randomly choose  $m$  nodes from  $\Delta_{\text{free}}$  & assign them to  $T$ 
6:    $T = \text{REDUCING\_RESOURCE\_POOL}(T, T)$ 
7:   IF  $T == \{\}$ 
8:      $\text{ALLOCATION\_DONE\_FLAG} = \text{true}$ 
9:     BREAK
10:  ENDIF
11: END FOREACH
12: END  $\text{DRP}_{\text{mp}}$  POLICY

```

Fig. 3.14: Dynamic Resource-Pool Multiple Partition Policy

```

1: BEGIN EVENT HANDLER for  $T_i$  COMPLETION
2:  $\Delta_{\text{free}} = \Delta_{\text{free}} + \omega_i$ 
3:  $\text{state\_change}_{T_i}$  [STATE 4:Executing]  $\rightarrow$  [STATE 5:Completed]
4: IF  $T == \{\}$ 
5:    $\text{ALLOCATION\_DONE\_FLAG} = \text{true}$ 
6:   BREAK
7: ELSE
8:    $m = \min(n_T, |\Delta_{\text{free}}|)$ 
9:   randomly choose  $m$  nodes from  $\Delta_{\text{free}}$  & assign them to  $T$ 
10:   $T = \text{REDUCING\_RESOURCE\_POOL}(T, T)$ 
11:  IF  $T == \{\}$ 
12:     $\text{ALLOCATION\_DONE\_FLAG} = \text{true}$ 
13:    BREAK
14:  ENDIF
15: END EVENT HANDLER for  $T_i$  COMPLETION

```

Fig. 3.15: Task Completion Event Handler of DRP_{mp}

3.5.2.5 Dynamic Resource-pool with Proportional Partitions ($\text{DRP}_{\text{mp-pro}}$)

DRP_{mp} assigns the same number of nodes to the resource-pool of each task, T_i . $\text{DRP}_{\text{mp-pro}}$ tries to improve resource utilization by using a mechanism to allocate a smaller number of nodes to the resource-pool of the tasks having smaller lengths of the raw-data files. Fig. 3.16 and Fig. 3.17 show the pseudo code for $\text{DRP}_{\text{mp-pro}}$. Initially, all nodes in Δ are free, thus $\Delta_{\text{free}} = \Delta$ (Fig. 3.16, Line 2).

```

1: BEGIN  $\text{DRP}_{\text{mp-pro}}$  POLICY
2:  $\Delta_{\text{free}} = \Delta$ 
3: FOREACH  $T_i \in T$ 
4:    $\text{par}_i = \min\{n_r, (\text{int}) \mu(L_i/L_{\text{total}}) \Delta\}$ 
5:   add  $\text{par}_i$  to  $P$ 
6: END FOREACH
7: FOREACH  $T_j \in T$ 
8:    $m = \min(\text{par}_i, |\Delta_{\text{free}}|)$ 
9:   randomly choose  $m$  nodes from  $\Delta_{\text{free}}$  & assign them to  $I'$ 
10:   $T = \text{REDUCING\_RESOURCE\_POOL}(I', T)$ 
11:  IF  $T == \{\}$ 
12:     $\text{ALLOCATION\_DONE\_FLAG} = \text{true}$ 
13:    BREAK
14:  ENDIF
15: END FOREACH
16: END  $\text{DRP}_{\text{mp-pro}}$  POLICY

```

Fig. 3.16: Dynamic Resource-Pool Multiple Proportional Partitioning

```

1: BEGIN EVENT HANDLER for  $T_i$  COMPLETION
2:  $\Delta_{\text{free}} = \Delta_{\text{free}} + \omega_i$ 
3:  $\text{state\_change}_{T_i}$  [STATE 4:Executing]  $\rightarrow$  [STATE 5:Completed]
4: IF  $T == \{\}$ 
5:    $\text{ALLOCATION\_DONE\_FLAG} = \text{true}$ 
6:   BREAK
7: ELSE
8:   FOREACH  $T_j \in T$ 
9:      $m = \min(\text{par}_i, |\Delta_{\text{free}}|)$ 
10:    randomly choose  $m$  nodes from  $\Delta_{\text{free}}$  & assign them to  $I'$ 
11:     $T = \text{REDUCING\_RESOURCE\_POOL}(I', T)$ 
12:    IF  $T == \{\}$ 
13:       $\text{ALLOCATION\_DONE\_FLAG} = \text{true}$ 
14:      BREAK
15:    ENDIF
16:  END FOREACH
17: END EVENT HANDLER for  $T_i$  COMPLETION

```

Fig. 3.17: Task Completion Event Handler of $\text{DRP}_{\text{mp-pro}}$

Instead of allocating an equal number of resources to all resource-pools, in $\text{DRP}_{\text{mp-pro}}$ the size of a resource-pool for a task is proportional to the length of the raw data file for the corresponding task. This gives a lower proportion of the resources to tasks having small raw data files. The number of nodes in the resource-pool for each task is calculated by using the following formula:

$$\text{par}_i = \min\{n_r, (\text{int}) \mu\left(\frac{L_i}{L_{\text{total}}}\right) \Delta\}$$

where, par_i is the number of nodes in the resource-pool to be assigned to T_i

μ is a scaling factor, such that $0 < \mu < 1$

L_i is the length of the raw data file of task T_i .

L_{total} is the sum of the lengths of all the raw data files in the given bag-of-tasks.

Note that the maximum resource-pool size for a task cannot exceed n_r , where $0 < n_r \leq |\Delta|$.

TRPS iteratively calculates par_i for each of the task T_i in T (Fig. 3.16, Line 4) and adds it to a set P (Fig. 3.16, Line 6). TRPS starts iterating for each task T_i in the given the bag-of-tasks (Fig. 3.16, Line 7). In each iteration, TRPS checks that whether par_i or $|\Delta_{free}|$ has a lower value and allocates it to a variable m . (Fig. 3.16, Line 8). TRPS randomly chooses m resources from Δ_{free} and assigns them to resource-pool r_i for T_i (Fig. 3.16, Line 9). TRPS invokes the Reducing Resource-pool Algorithm and provides r_i as the resource-pool and T as the list of tasks (Fig. 3.16, Line 10). If the Reducing Resource-pool algorithm is successful in allocating resources for all the tasks in T , an empty set is returned and `ALLOCATION_DONE_FLAG` is set to true (Fig. 3.16, Line 12).

Fig. 3.17 shows “Completion Event Handler” for one of the tasks T_i . “Completion Event Handler” executes when a particular task T_i is completed. First, the resources used by the task are returned to Δ_{free} (Fig. 3.17, Line 2). Then the state of the task T_i is changed to “STATE 5: Completed” (Fig. 3.17, Line 3). TRPS checks whether there are any more tasks left in T (Fig. 3.15, Line 4). If there are no tasks left in T , `ALLOCATION_DONE_FLAG` is set to true (Fig. 3.15, Line 5). Otherwise, TRPS checks that whether par_i or $|\Delta_{free}|$ has a lower value and allocates it to a variable m (Fig. 3.17, Line 9). TRPS randomly chooses m resources from Δ_{free}

and assigns them to the resource-pool Γ (Fig. 3.17, Line 10). TRPS invokes the Reducing Resource-pool Algorithm and provides it Γ as resource-pool and T as the list of tasks (Fig. 3.17, Line 11). If the Reducing Resource-pool algorithm is successful in allocating resources for all the tasks in T , an empty set is returned and `ALLOCATION_DONE_FLAG` is set to true (Fig. 3.17, Line 13).

3.5.3 Hybrid TRPS Policy

A hybrid TRPS policy is a combination of a static and a dynamic policy as explained in the following sub-section.

3.5.3.1 Static Resource-pool with Single Partition & Backfilling ($\text{SRP}_{\text{SP}}+\text{BF}$)

$\text{SRP}_{\text{SP}}+\text{BF}$ incorporates *backfilling* which is a resource allocation technique that allows a task to begin executing before previously scheduled tasks that are waiting due to insufficient available nodes [KEL00]. Backfilling exploits otherwise idle nodes, thereby increasing system utilization [MUA01]. Many schedulers use backfilling. IBM LoadLeveler [IBM10] and the Maui Scheduler [MAU10] are examples of popular schedulers that incorporate backfilling. $\text{SRP}_{\text{SP}}+\text{BF}$ adds backfilling to SRP_{SP} in order to reduce resource contention, as explained in the following discussion.

$\text{SRP}_{\text{SP}}+\text{BF}$ has a mapping phase and an execution phase as in a static policy. The mapping phase of $\text{SRP}_{\text{SP}}+\text{BF}$ is similar to SRP_{SP} in which a mapping between all tasks and a set of resources is determined. To create this mapping, TRPS iteratively calls the RA for each task in T . In the

execution phase, the first task in the given bag-of-tasks is executed first. TRPS iterates through all the tasks in the given bag-of-tasks and chooses the next task for which the complete set of resources needed is available (and therefore can be executed concurrently).

Fig. 3.18 shows the execution phase for $SRP_{sp}+BF$. Note that Δ_{free} represents all resources that are available at a particular time. Initially, all resources are available and thus $\Delta_{free} = \Delta$ (Fig. 3.18, Line 2). TRPS iterates through all the tasks in the given bag-of-tasks (Fig. 3.18, Line 3). For each task T_i it checks whether the set of resource it needs to run, ω_i , is available or not (Fig. 3.18, Line 4). For the first task, T_1 , the set of resource is always available. In the first iteration, TRPS enables T_1 for execution and changes its state to “STATE 2: Enabled for execution” (Fig. 3.18, Line 5). The resources to be used by T_1 are taken out from Δ_{free} (Fig. 3.18, Line 6). The state of T_1 is changed to “STATE 4: Executing” (Fig. 3.18, Line 7) and T_1 is removed from T to indicate that it has been allocated resources (Fig. 3.18, Line 8). This process is repeated for each task in T . After that TRPS checks whether there are any more tasks left in T (Fig. 3.18, Line 11). If there are no tasks left in T , $ALLOCATION_DONE_FLAG$ is set to true (Fig. 3.18, Line 12) and the algorithm breaks the loop.

At this stage, if any resources are available, $SRP_{sp}+BF$ takes advantage of the unused resource sets by using backfilling. For that first TRPS checks that whether any unused resources are available (Fig. 3.18, Line 15). If resources are available, TRPS initiates backfilling by first ordering the tasks in T in non-decreasing order of the lengths of the raw data (Fig. 3.18, Line 16).

```

1: BEGIN Resource Allocation (Execution Phase)
2:  $\Delta_{free} = \Delta$ 
3: FOREACH  $T_i \in T$ 
4:   IF ( $\omega_i \subseteq \Delta_{free}$ )
5:     state_change $_{T_i}$  [STATE 1:Idle/Mapped]→ [STATE 2:Enabled for execution]
6:      $\Delta_{free} = \Delta_{free} - \omega_i$ 
7:     state_change $_{T_i}$  [STATE 2:Enabled for execution]→ [STATE 4:Executing]
8:     remove  $T_i$  from  $T$ 
9:   ENDIF
10: END FOREACH
11: IF  $T == \{\}$ 
12:   ALLOCATION_DONE_FLAG = true;
13:   BREAK
14: ELSE
15:   IF ( $|\Delta_{free}| > 0$ )
16:     order tasks in  $T$ , in non-decreasing order of lengths of raw-data files
17:      $T = \text{REDUCING\_RESOURCE\_POOL\_ALGORITHM}(\Delta_{free}, T)$ 
18:     IF  $T == \{\}$ 
19:       ALLOCATION_DONE_FLAG = true
20:     ENDIF
21:   ENDIF
22: ENDIF
23: END Resource Allocation (Execution Phase)

```

Fig. 3.18: Execution Phase of $\text{SRP}_{sp} + \text{BF}$

Then, TRPS combines all the resources that are available into a single resource-pool and passes it on to the Reducing Resource Pool algorithm (Fig. 3.18, Line 17). If all tasks have been allocated resource and T becomes empty, the ALLOCATION_DONE_FLAG is set to true (Fig. 3.18, Line 19).

Fig. 3.19 shows the Event Handler code for $\text{SRP}_{sp} + \text{BF}$, which is run when a particular task T_i completes. First, the resources used by T_i are returned (Fig. 3.19, Line 2). Then, the state of the task T_i is changed to “STATE 5: Completed” (Fig. 3.19, Line 3). TRPS checks if there is any task left in T . If T is empty, the ALLOCATION_DONE_FLAG is set to true and the Task

```

1: BEGIN EVENT HANDLER for  $T_i$  COMPLETION
2:  $\Delta_{free} = \Delta_{free} + \omega_i$ 
3: state_change $_{T_i}$  [STATE 4:Executing]→ [STATE 5:Completed]
4: IF  $T == \{\}$ 
5:   ALLOCATION_DONE_FLAG=true
6:   BREAK
7: ELSE
8:   FOREACH  $T_i \in T$ 
9:     IF ( $\omega_i \subseteq \Delta_{free}$ )
10:      state_change $_{T_i}$  [STATE 3:Waiting]→ [STATE 2:Enabled for execution]
11:       $\Delta_{free} = \Delta_{free} - \omega_i$ 
12:      state_change $_{T_i}$  [STATE 2:Enabled for execution]→ [STATE 4:Executing]
13:      remove  $T_i$  from  $T$ 
14:    ENDIF
15:  END FOREACH
16: IF  $T == \{\}$ 
17:   ALLOCATION_DONE_FLAG=true
18:   BREAK
19: ENDIF
20: IF ( $|\Delta_{free}| > 0$ )
21:   order tasks in  $T$  in non-decreasing order of lengths of raw-data files
22:    $T = \text{REDUCING\_RESOURCE\_POOL\_ALGORITHM}(\Delta_{free}, T)$ 
23: ENDIF
24: ENDIF
25: END EVENT HANDLER for  $T_i$  COMPLETION

```

Fig. 3.19: Task Completion Event Handler of $SRP_{sp} + BF$

Completion Event Handler returns (Fig. 3.19, Line 4-6). If T is non-empty, TRPS iterates through all the tasks in T (Fig. 3.19, Line 8). For each task T_i , TRPS checks whether $\omega_i \subseteq \Delta_{free}$ (Fig. 3.19, Line 9). If $\omega_i \subseteq \Delta_{free}$, TRPS enables T_i for execution and changes its state to “STATE 2: Enabled for execution” (Fig. 3.19, Line 10). The resources to be used by T_i are taken out from Δ_{free} (Fig. 3.19, Line 11). The state of T_i is changed to “STATE 4: Executing” (Fig. 3.19, Line 12) and T_i is removed from T to indicate that it has been allocated resources (Fig. 3.19, Line 13). This process is repeated for each task in T (Fig. 3.19, Line 8-15). After that TRPS checks whether there are any more tasks left in T (Fig. 3.18, Line 16). If there are no tasks left in T , ALLOCATION_DONE_FLAG is set to true (Fig. 3.18, Line 17) and the

algorithm breaks the loop (Fig. 3.18, Line 18). If there are tasks left in T while there are some resource available, $SRP_{sp}+BF$ takes advantage of the unused resource sets by using backfilling. In backfilling, first, the tasks in T are ordered in non-decreasing order of the lengths of the raw data files of the tasks (Fig. 3.19, Line 21). Then, TRPS uses backfilling by combining all the resources that are available them into a single resource-pool and pass it on to the Reducing Resource Pool algorithm (Fig. 3.19, Line 22).

3.6 Resource Allocator (RA)

As discussed in Section 3.3, the other decision making component of the BiLeG architecture is RA which works as the lower level decision making module of the BiLeG architecture. Its job is to allocate resources from the resource-pool for a particular task determined by TRPS. A set of algorithms has been proposed for RA. Different algorithms may be suitable for different workload and system parameters, task characteristics and the user requirements. The choice of the most appropriate RA algorithm is the function of these parameters.

The following section presents the concept of Architectural Templates that are used by the resource allocation algorithm deployed at RA.

3.6.1 Architectural Templates

An Architectural Template divides the available resources into different entities and assigns each of them one of the specialized roles mentioned in Section 3.6.1. Note that a particular node may play different roles at different times. For example a resource may be best utilized in a compute-farm for processing a particular job at one time, thus being a part of the compute-farm. But the

same node may be used more effectively in a data-farm for processing a job in another task at another time; thus being a part of a data-farm. The salient features of each of the various architectural templates that are used in this research are enlisted in Table 3.1. The Architectural Templates are explained in the next Section.

Table 3.1: Salient Features of the Architectural Templates

Architectural Template	Salient Features
2-Tier-a	<ul style="list-style-type: none"> - Has a source node and a set of sink nodes without any intermediate nodes. - The raw data file is processed as a whole at the source node.
2-Tier-b	<ul style="list-style-type: none"> - Has a source node and a set of sink nodes without any intermediate nodes. - The raw data file is processed as a whole at each of the sink nodes.
3-Tier	<ul style="list-style-type: none"> - Has a source node, a set of sink nodes and a compute farm. - The raw data file is divided into partitions and each of the partitions is processed at the compute-farm in parallel.
4-Tier	<ul style="list-style-type: none"> - Has a source node, a set of sink nodes, a compute-farm and a data-farm. - The raw data file is divided into partitions and each of the partitions is processed at the compute-farm in parallel. - The processed partitions are combined at the egress node and are stored on the data-farm.

3.6.1.1 2-Tier Architectural Templates

In 2-Tier Architectural Templates, only the source and the set of sink nodes, κ_s , are available for processing and data transfer. There are two different types of 2-Tier Architectural Templates: 2-Tier-a and 2-Tier-b. In 2-Tier-a, the source node is used for data processing. Fig 3.20 (a) shows the process for the 2-Tier-a architecture. The figure shows the upper decision making module, TRPS and the lower decision making module, RA. TRPS co-ordinates with RA (1) and gives it a PBDT task T_i . RA divides T_i into constituent jobs and generates the workflow file (2). The jobs corresponding to a task include both jobs for processing as well as transfer of data. RA transfers the workflow file to the workflow engine (3). The workflow engine initiates the workflow by

Fig. 3.20: 2-Tier Architectures

signalling the source node to start the execution of task T_i (3₁). The source node processes the raw data file (3₂) and delivers the processed file to each of the nodes in κ_s (3₃₁ to 3_{3k}) where $k \in |\kappa_s|$. After the transfer of processed data is completed, each node in κ_s sends an acknowledgment to the workflow engine to indicate that the processed file has reached it (4₁

to 4_k). Once all nodes in κ_s have sent the completion signals to the workflow engine, T_i completes successfully.

A 2-Tier-b Architectural Template is similar to 2-Tier-a (see Fig. 3.20 (b)). The important difference is that instead of at the source node, the data processing is done at each of the sink

nodes (3_{31} to 3_{3k} in Fig. 3.20 (b)). After the processing of data is completed, each node in κ_s sends a signal to the workflow engine (4_1 to 4_k). Once all nodes in κ_s have sent completion signals to the workflow engine, T_i completes successfully.

3.6.1.2 3-Tier Architectural Templates

In a 3-Tier Architectural Template, the resource-pool of the given task selected by TRPS is used as a compute-farm. The role of the compute-farm is to process the data. Fig. 3.21 shows the details. Initially, TRPS starts the process by interacting with RA (1). It assigns T_i to RA with $\bar{I}_i$ as its resource-pool. RA runs the resource allocation algorithm and chooses ω_i resources that are to be used to execute T_i . RA divides the task T_i into jobs and generates the workflow file (2). RA transfers the workflow file to the workflow engine (3). The workflow engine starts executing the workflow by signalling the source node (3_1). The source node divides the raw data file of the given task into p_i partitions (3_2) and transfer each of the partitions to a node in the compute farm, κ_{CF} (3_{31} to 3_{3n}). Each of the nodes in the compute farm processes the partition assigned to it (3_{41} to 3_{4n}). Once processing of the data is completed at the compute-farm nodes, the processed partitions are transferred to each of the nodes in κ_s (3_{511} to 3_{5nk}).

Fig. 3.21: 3-Tier Architecture

After the transfer of processed data is completed, each node in κ_s sends an acknowledgment to the workflow engine to indicate that the task T_i is complete (4_1 to 4_k).

3.6.1.3 4-Tier Architectural Templates

In a 4-Tier Architectural Template, the resource-pool of the given task selected by TRPS is divided into two sets of nodes: a compute-farm κ_{CF} and a data-farm κ_{DF} . Both the compute-farm and the data-farm have a specific role. The role of the compute-farm is to process the data. Once all the data is processed, it is combined at the Egress node. The role of the data-farm is to

replicate this processed data at chosen nodes to efficiently transfer it to the sink nodes.

Fig. 3.22: 4-Tier Architecture

Initially, TRPS gives RA a PBDT task, T_i and its corresponding resource-pool, I_i (1). After running the resource allocation algorithm, RA chooses ω_i resources that are to be used to execute T_i , and generates a workflow file (2). RA transfers the workflow file to the workflow engine (3). The workflow engine initiates the execution of T_i by sending the signal to the source node (3₁). The raw data file of the given task is divided into p_i partitions (3₂) and each of the partitions is transferred to a node in the compute farm, κ_{CF} (3₃₁ to 3_{3n}). Each node processes the partition assigned to it (3₄₁ to 3_{4n}). Once processing of the data is completed at the compute-farm nodes, the processed partitions are transferred to the Egress Node (3_{51e} to 3_{5ne}) where they are combined to produce the required processed file. The Egress Node sends a signal to RA (4) to indicate the completion of this stage.

The responsibility of the Egress node is to make sure that all the partitions of the raw data file associated with T_i have been successfully processed. Even if a small portion of data is not available due to processing or communication error, the resultant processed file formed by the combination of the constituent processed files may become invalid. Catching such an error at an early stage of processing is desirable as the workflow engine can re-initiate the processing for the faulty data partition only.

From the Egress node, the processed data is transferred to the data nodes chosen by the algorithm in RA (4_{1el} to 4_{1em}). From there it is delivered to each of nodes in κ_{DF} (4_{211} to 4_{2mk}). Once the processed data is delivered to all sink nodes, the workflow engine is notified (5_1 to 5_k). Once all sink nodes notify the workflow engine, T_i is marked as complete.

Note that partitions of the raw data file are transferred to nodes in the compute farm. But a complete processed file (not partitions) is transferred to and replicated on each node in the data-farm.

In industry, many distributed processing and delivery systems have a separate data-farm and a compute-farm [ABB06]. The following discussion highlights the appropriateness of using a separate data-farm for the 4-Tier architecture:

- 1) The value of transfer-within cost (α_Δ) is an important factor to determine the feasibility of a separate data-farm. Usually α_Δ is negligible if both the compute-farm

and the data-farm are provided by the same utility-provider [AMA09] [AZU09]. Also, if both are provided by the same utility provider, typically the compute-farm and data-farm are inter-connected by very high-speed data-links making unit communication time very small.

- 2) In certain situations, replicating the processed data in a data-farm may have both cost and performance benefits. The set of sink nodes may consists of one or more subsets of sink nodes, such that the unit communication time between the nodes of the compute-farm and these subsets of nodes is high. It may occur if each subset is located at a physically remote location with respect to the compute-farm and no high-speed connection is available to connect each subset with the compute farm [ANG06]. In such case, a 4-Tier Architectural Template provides the possibility to create at least one local replica for the each subset of sink nodes by replicating data at one of the nodes that is “local” to a particular subset of sink nodes. If d_{s_a} is the node of the data-farm at which the processed data is replicated because it is local to a subset of nodes s_a , the unit communication time from d_{s_a} to s_a should be comparatively low. Note that instead of $|s_a|$ low speed data-paths from compute-farm farm to s_a , the system will have only one low speed data path from egress node to d_{s_a} . Once replicated at d_{s_a} , the processed data will be delivered from d_{s_a} to all constituent nodes of s_a through high-speed links.
- 3) 4-Tier Architectural Templates incorporates a bi-stage workflow approach that uses a separate data and compute farm. Deployment of separate data and compute farms are also described in [SEN05] [SHA07] [PAN08]. In Fig. 3.22, it can be seen that at Egress node all the partitions of the processed file are combined generating the

required processed file. As soon as the resultant file is generated at the Egress node, a signal is sent to RA indicating that the first stage of the processing of the bag-of-tasks has been successfully completed. If the complete set of partitions fails to reach the Egress node, an error is registered by RA. RA can troubleshoot and locate the particular job that has failed. Only the failed job is required to be re-run again to complete stage-1 of the processing of the given bag-of-tasks. Note that when the error is detected, the corrective action can be taken before the data starts leaving the Grid domain and starts getting transmitted to the designated sink nodes. In contrast, the 3-Tier Architecture Template is based on single-stage workflow approach. If processing of one of the partitions fails, RA is notified of the error only after the processed data is transferred to the sink nodes.

- 4) Separate data-farm nodes, as included in 4-Tier Architectural Template, can be used for the caching of data. Data-farms have capabilities to store large amounts of data. Compute-farms are usually high-speed clusters with limited storage capabilities. Separating data processing from data storage allows specialized hardware to be used at the constituent nodes of the data and compute farms, suitable for their particular roles. Also, if some of the sinks nodes are interested in the same set of processed data at different time-slots, the processed data can be cached at the data-farm. The processed data can be deleted from the data-farm, once it is delivered to all the sink nodes.

3.6.2 ATSRA Algorithm

ATSRA algorithms are the knowledge-based algorithms proposed in this thesis. Being knowledge-based algorithms, ATSRA algorithms need accurate knowledge of the system

characteristics (such as processing and communication times per unit data) during the time in which the tasks in the given bag-of-tasks are expected to be executing. ATSRA algorithms are based on Linear Programming (LP) which is a popular technique for solving optimization problems [AZZ06][GZA04]. In LP, an optimization problem is modeled as a set of linear expressions composed of input parameters and output parameters. The LP solver starts by creating a problem instance of the model by assigning values to the input parameters [CHV80] [DIM98]. The problem instance is then subjected to an objective function, which is also required to be a linear expression. The values of the output variables, which collectively represent the optimal solution, are determined for the best value of the objective function. Based on this approach, three algorithms are presented in this section.

3.6.2.1 ATSRA_{org} Algorithm

A summary of the ATSRA_{org} algorithm is presented in Fig. 3.23 and Fig. 3.24. ATSRA_{org} algorithm is divided into two stages.

Stage-1: Selection of the most appropriate Architectural Template, for T:

The purpose of Stage-1 is to identify the appropriate Architectural Template to process the given bag-of-tasks. The result of Stage-1 is the selection of an Architectural Template for a bag-of-tasks. Stage-1 of the algorithm is always executed before the tasks start executing.

Stage-2: Allocation of the resources for the chosen Architectural Template.

In Stage-2, communication and computation resources are chosen for T based on the Architectural Template chosen in Stage-1. The result of Stage-2 is the mapping of a set of

resources to each of the tasks in the given bag-of-tasks. Depending upon the TRPS policy, the Stage-2 of the RA algorithm may be completed before the tasks start executing, or a portion of it may be run after the tasks start executing.

```

1: BEGIN  $\text{ATSRA}_{\text{org}}$  Stage-1
3: calculate  $\text{cost}_{2\text{-Tier-a}}$  and  $\text{cost}_{2\text{-Tier-b}}$ 
4:  $\text{cost}_{\min} = \min(\text{cost}_{2\text{-Tier-a}}, \text{cost}_{2\text{-Tier-b}})$ 
5: create 3-Tier LP model
6: calculate  $\text{cost}_{3\text{-Tier}}$ 
7: IF ( $\text{cost}_{3\text{-Tier}} < \text{cost}_{\min}$ )
8:    $\text{cost}_{\min} = \text{cost}_{3\text{-Tier}}$ 
9: ENDIF
10: create 4-Tier LP model
11: calculate  $\text{cost}_{4\text{-Tier}}$ 
12: IF ( $\text{cost}_{4\text{-Tier}} < \text{cost}_{\min}$ )
13:    $\text{cost}_{\min} = \text{cost}_{4\text{-Tier}}$ 
14: ENDIF
15: choose Architectural Template associated with  $\text{cost}_{\min}$ 
16: END  $\text{ATSRA}_{\text{org}}$  Stage-1

```

Fig. 3.23: Stage-1 of the $\text{ATSRA}_{\text{org}}$ Algorithm

```

1: BEGIN  $\text{ATSRA}_{\text{org}}$  Stage-2 ( $\Gamma_i$  Resource Pool , Task  $T_i$ )
2: choose  $\omega_i$  = set of allocated resources for  $T_i$  based on the chosen
   Architectural Template
3: RETURN  $\omega_i$ 
4: END  $\text{ATSRA}_{\text{org}}$  Stage-2

```

Fig. 3.24: Stage-2 of the $\text{ATSRA}_{\text{org}}$ Algorithm

In Stage-1, the $\text{ATSRA}_{\text{org}}$ algorithm starts by searching for the most appropriate Architectural Template for the system. In this stage, cost associated with each of the Architectural Templates is calculated and the Architectural Template having the minimum cost is chosen. For PBDT tasks described in this thesis, it starts with the 2-Tier architectures (Fig. 3.23, Lines 3-4) and it is followed by the calculation of the cost for the 3-Tier Template and 4-Tier Template respectively (Fig. 3.23, Lines 5-11). The Architectural Template which is associated with the minimum t_{cost} is chosen as the Architectural Template for the given bag-of-tasks (Fig. 3.23, Line 15).

Fig. 3.24 shows the Stage-2 for the $ATSRA_{org}$ algorithm. For a particular task T_i , it accepts a resource-pool I_i allocated to it by TRPS. Based on the Architectural Template chosen in Stage-1, the Stage-2 of $ATSRA_{org}$ chooses ω_i which is the set of allocated resource for the task T_i (Fig. 3.24, Line 2).

The following methodology is used for the cost calculation in the $ATSRA_{org}$ algorithm.

For calculating the total cost associated with the use of 2-tier-a architecture, let L_q be the length of the raw data file in GB for the task T_q , n_{src} be the source node, n_{sink_i} be the i^{th} sink node, ϵ_q be the processing factor associated with the task T_q , Cp_{src} be the CPU processing cost per data unit at the source node, β be the processing cost in dollars per hour, $\alpha_{src-sink}$ be the transfer cost in dollars per hour between the source and sink nodes, κ_s be the set of sink nodes, then the total cost of performing the a particular PBDT task T_q using the 2-Tier-a architecture is given by:

$$t_{cost-2-Tier-a_q} = L_q \{ \sum_{i=1}^{|\kappa_s|} \epsilon_q \alpha_{src-sink} d(n_{src}, n_{sink_i}) + \beta Cp_{src} \} \quad (3.3)$$

Note that $\sum_{i=1}^{|\kappa_s|} \epsilon_q \alpha_{src-sink} d(n_{src}, n_{sink_i})$ is the cost of transferring one unit of processed data from the source node to all sink nodes. βCp_{src} is the unit cost of processing data at the source node in dollars. By multiplying the summation of these two terms with L_q , the total cost for the task T_q is obtained.

The total cost of performing each of the tasks in the given bag of tasks T is given by:

$$t_{cost-2-Tier-a} = \sum_{q=1}^{|T|} t_{cost-2-Tier-a_q}$$

For calculating the total cost associated with the use of 2-tier-b architecture, let Cp_{Sink_i} represent the cost of processing per data unit at the i^{th} sink node then the total cost of performing a PBDT task using a 2-Tier-b architecture is given by:

$$t_{cost-2-Tier-b_q} = L_q [\sum_{i=1}^{|\kappa_s|} \{\alpha_{src-sink} d(n_{src}, sink_i) + \beta Cp_{Sink_i}\}] \quad (3.4)$$

Note that $\alpha_{src-sink} d(n_{src}, sink_i)$ is the cost for transferring one unit of data from the source node to a particular sink node i . Similarly, βCp_{Sink_i} is the cost of processing one unit of data at a particular sink node i . $\sum_{i=1}^{|\kappa_s|} \{\alpha_{src-sink} d(n_{src}, sink_i) + \beta Cp_{Sink_i}\}$ is the unit cost of transferring and processing the data to all sink nodes and by multiplying it with L_q the total cost of transferring and processing the data of the raw-data file of the task T_q is obtained.

The total cost of performing each of the tasks in the given bag of tasks T using 2-Tier-b architecture is given by:

$$t_{cost-2-Tier-b} = \sum_{q=1}^{|T|} t_{cost-2-Tier-b_q}$$

For 3-Tier cost calculations, the cost function is formulated as an Integer Programming problem. Integer Programming problems are generally harder to solve than Linear Programming problems [LAU98]. For calculating the total cost associated with the use of 3-Tier architecture, let α_{in} is the transfer-in cost in dollars per hour, α_{out} is the transfer-out cost in dollars per hour, ρ_q is the number of partitions of the raw data file for the q^{th} task in the given bag-of-tasks, then for a 3-Tier Architectural Template, the total cost can be formulated as:

$$t_{cost-3-Tier-q} =$$

$$\min L_q \left[\sum_{j=1}^n \frac{1}{\rho_q} \{ \beta C p_j + \alpha_{in} d(n_{src}, n_j) \} x_j + \frac{1}{\rho_q} \sum_{i=1}^n \sum_{j=1}^k \varepsilon_q \alpha_{out} d(n_i, n_{sink_j}) w_{ij} \right] \quad (3.5)$$

where x_j is a binary variable which is 1 if a particular node n_i is assigned to compute-farm and is 0 otherwise. Variable w_{ij} is a binary variable which has the value of 1 if data is transferred from node i to node j otherwise it is 0. L_q is the length of the raw data file for the task T_q . As $PBDT_{fixed}$ tasks with ρ_q equal partitions are being considered, each of these partitions has a size of L_q/ρ_q . Note that $\alpha_{in} d(n_{src}, n_j)$ is the cost in dollars of transferring one GB of data from the source node to a particular node j and $\beta C p_j$ is the cost in dollars of processing one GB data at node j . Thus, the first term within the parenthesis in Eq. 3.5 is the cost in dollars of transferring and processing one GB of data at the nodes in the compute farm. The second term within the parenthesis in Eq. 3.5 is the cost in dollars of transferring one GB of the data from compute farm to all the sink nodes. The product of L_q and the sum of the two terms in parenthesis in Eq.3.5 gives the total cost of transferring all the data from the source node to all the sink nodes.

The feasibility of a particular assignment is determined by the following constraints.

$$\sum_{i=1}^n w_{ij} = \rho_q \quad \forall j = 1 \text{ to } k \quad (3.6)$$

$$\sum_{i=1}^n x_i = \rho_q \quad (3.7)$$

$$x_i = w_{ij} \quad i \in \Gamma, j \in S \quad (3.8)$$

$$x_i \in \{0,1\} \quad (3.9)$$

$$w_{ij} \in \{0,1\} \quad (3.10)$$

The first constraint (Eq. 3.6) specifies that for a particular task, T_q , the number of partitions transferred to each of the sink nodes should be equal to ρ_q . The second constraint (Eq. 3.7) specifies that the number of nodes used in the compute-farm should be equal to the number of partitions of the raw data file of the task T_q . The third constraint (Eq. 3.8) ensures that each of the communication paths chosen by the algorithm is originating from one of the nodes used in the compute farm. The forth and the fifth constraints (Eq. 3.9 and Eq. 3.10) ensure that both x_i and w_{ij} are binary variables.

The total cost of performing each of the tasks in the given bag of tasks, T , is given by:

$$t_{\text{cost-3-Tier}} = \sum_{q=1}^{|T|} t_{\text{cost-3-Tier-}q} \quad (3.11)$$

For 4-Tier cost calculations, the cost function is formulated as a mixed integer/linear programming (MILP) problem. MILP problems are generally harder to solve than LP problems [LAU98]. For a particular task T_q in the given bag-of-tasks, let n_{src} be the source node, n_{sink_j} is the j^{th} sink node, n_{egress} is the egress node and ρ_q is the number of partitions of the raw data file, then for a 4-Tier Architectural Template, the cost associated with the bag-of-tasks can be formulated as:

$$t_{\text{cost-4-Tier}} = \sum_{q=1}^{|T|} t_{\text{cost-4-Tier-}q} \quad (3.12)$$

where

$$t_{\text{cost-4-Tier-}q} = \min L_q \left[\begin{array}{l} \sum_{j=1}^n \frac{1}{\rho_q} \{ \beta C p_j + \alpha_{\text{in}} d(n_{\text{src}}, n_j) + \varepsilon_q \alpha_{\Delta} d(n_j, n_{\text{egress}}) \} x_j \\ + \varepsilon_q \sum_{i=1}^n \alpha_{\Delta} d(n_{\text{egress}}, n_i) y_i + \varepsilon_q \sum_{i=1}^n \sum_{j=1}^k \alpha_{\text{out}} d(n_i, n_{\text{sink}_j}) w_{ij} \end{array} \right] \quad (3.13)$$

where x_i is a node assignment binary variable which is 1 if a particular node n_i is assigned to compute-farm and is 0 otherwise. Similarly y_i is a node assignment binary variable for the data-farm. y_i is 1, if a node i is used in the data-farm and is 0 otherwise. Variable w_{ij} is the fraction of the processed file that a sink gets from a particular node in data-farm. As $PBDT_{\text{fixed}}$ tasks with ρ_q equal partitions are being considered, each of these partitions has a size of L_q/ρ_q . Note that $\alpha_{\text{in}}d(n_{\text{src}}, n_j)$ is the cost in dollars of transferring one GB of data from the source node to a particular node j and βCp_j is the cost in dollars of processing one GB data at node j . $\varepsilon_q \alpha_{\Delta}d(n_j, n_{\text{egress}})$ is the cost in dollars of transferring one GB of data from a particular node j to the Egress node, n_{egress} . Thus, the first term within the parenthesis in Eq. 3.13 is the cost in dollars of transferring 1 GB data to n_{egress} . The second term within the parenthesis in Eq. 3.13 is the cost in dollars of transferring one GB of the data from n_{egress} to all the sink nodes. The product of L_q and the sum of the two terms in parenthesis in Eq. 3.13 gives the total cost of transferring all the data from the source node to all the sink nodes.

The feasibility of a particular assignment is determined by the following constraints.

$$\sum_{i=1}^n w_{ij} = 1 \quad \forall j = 1 \text{ to } k \quad (3.14)$$

$$\sum_{i=1}^n x_i = \rho_q \quad (3.15)$$

$$x_i \in \{0,1\} \quad (3.16)$$

$$y_i \in \{0,1\} \quad (3.17)$$

$$w_{ij} \leq y_i \quad i \in \Gamma, j \in S \quad (3.18)$$

$$w_{ij} \geq 0 \quad i \in \Gamma, j \in S \quad (3.19)$$

The first constraint (Eq. 3.14) specifies that the sum of all parts of the files being transferred from the data-farm to a particular sink node should add up to form the full length of the file. The second constraint (Eq. 3.15) specifies that the number of nodes used in the compute-farm should be equal to the number of partitions of the raw data file of task T_q . The third and the fourth constraints (Eq. 3.16 and Eq. 3.17) ensure that both x_i and y_i are binary variables. The fifth constraint (Eq. 3.18) makes sure that the solution proposed by the algorithm has a non-zero value of w_{ij} , if and only if $y_i > 0$. For example, consider node n_3 and a particular sink node s_7 . w_{37} (that represents the portion of the total processed file that s_7 gets from n_3) should only have a non-zero value if $y_3 = 1$ (that is n_3 is being used as a node in the data-farm). The last constraint (Eq. 3.19) prevents negative values for w_{ij} .

$$\text{Let } Cp_j^{\sim} = \beta Cp_j + \alpha_{in} d(n_{src}, n_j) + \varepsilon_q \alpha_{\Delta} d(n_j, n_{egress}) \quad (3.20)$$

$Cp_j^{\sim}$ represents the total cost of sending a unit data from a source node to a node n_j in the compute-farm, processing it and sending it to the egress node. Eq. (3.13) and Eq. (3.20) yield:

$$t_{\text{cost-4-Tier-q}} = \min L_q \left[\sum_{j=1}^n \frac{1}{\rho_q} Cp_j^{\sim} x_j + \varepsilon_q \sum_{i=1}^n \alpha_{\Delta} d(n_{egress}, n_i) y_i \right] + \varepsilon_q \sum_{i=1}^n \sum_{j=1}^k \alpha_{out} d(n_i, n_{sink_j}) w_{ij} \quad (3.21)$$

Note that the input of the $ATSRA_{org}$ algorithm includes the unit processing and communication times and processing and communication costs of all the nodes. The output is the solution matrix which represents the values of solution variables, i.e.

$$x_i, y_i \quad \forall i = 1 \text{ to } n$$

$$w_{ij} \quad \forall i = 1 \text{ to } n, j=1 \text{ to } k$$

It is important to note that in 4-Tier Architecture there is nothing that prevents a node to be part of both a compute-farm and a data-farm. For example, if the solution matrix has $x_3=1$ and $y_3=1$ and then n_3 is used both in the data-farm and the compute-farm.

3.6.2.2 Constraints Relaxation

For a small number of nodes in the Grid, ATSR_A algorithm is expected to perform well. But as the number of nodes increases the time taken by the algorithm to run also increases considerably. To reduce the time taken by the algorithm to run, the technique of constraints relaxation is used. Constraints relaxation is based on the idea of replacing a “difficult” minimization problem by a “simpler” minimization problem by reducing constraints in a way that its optimal value is at least as small as the original cost. It can give a quick estimate of the optimal solution in the best possible scenario, as explained in the following discussion.

To derive a relaxed or simpler problem from the exact or original LP formulation, two possibilities are considered.

Enlarge the set of feasible solutions. If the set of feasible solutions is represented by P , then we need to find P' such that $P \subseteq P'$.

OR

Replace the minimum objective function by a function that has the same or a smaller value everywhere.

In this research the first approach has been used. A modified optimization problem of the original problem is found, that is a relaxation of the original problem. The new optimization problem uses the same objective function but a larger feasible region P' that includes P as a subset. As the new optimization problem is found by ignoring some of the constraints of the original LP formulation this technique is called *constraints relaxation*. It simplifies the original problem and reduces the algorithm running time at the cost of accuracy of the LP model, because some of the constraints have been ignored. Because P' contains P , any solution which belongs to P , also belongs to P' as well.

3.6.2.3 ATSR_{SSR} Algorithm

In order to reduce the time taken by ATSR_{org} to run, in ATSR_{SSR} the technique of constraints relaxation has been used. The pseudo code of ATSR_{SSR} is similar to the pseudo code of ATSR_{org}, detailed in Fig. 3.23. But, when the 3-Tier and 4-Tier models are created (Fig. 3.23, Line 5 and Line 10) instead of creating the exact model, a simplified model using the technique of constraints relaxation is used. Note that the relaxed model is used in Stage-1 of the algorithm only. Once an Architectural Template has been chosen, the exact model is used for resource allocation (Fig. 3.24, Line 2).

This algorithm is named as ATSR Single Stage Relaxation (SSR) or ATSR_{SSR}, as the constraint-relaxation is applied only to first stage of the algorithm.

The following constraints are relaxed in the Stage-1 of the ATSR_{SSR} algorithm.

- 1- As discussed in Section 3.6.2.1, in ATSR_{org} the optimization problem for the 3-Tier architecture is an IP problem and the optimization problem for the 4-Tier architecture is a

MILP problem. Both IP and MILP problems take a much higher time to run as compared to LP problems [LAU98]. In order to convert both of these problems to a LP problem constraints relaxation is used. For the 3-Tier model, constraints 3.9 and 3.10, which restrict that x_i and w_{ij} should be binary numbers, are dropped. Similarly for 4-Tier model, constraints 3.16 and 3.17 are dropped.

- 2- For the 4-tier architecture, the constraint 3.18 that stipulates that w_{ij} should always be less than y_i , is also dropped.

The Stage-2 of $ATSRA_{ssr}$ is exactly similar to the Stage-2 of $ATSRA_{org}$.

3.6.2.4 $ATSRA_{bsr}$ Algorithm

In the $ATSRA_{bsr}$ algorithm, constraints relaxation is used at both Stage-1 for choosing the Architectural Template Selection and Stage-2 for the resource allocation. In $ATSRA_{bsr}$, for the 3-Tier model, constraints 3.9 and 3.10, which restrict that x_i and w_{ij} should be binary number, are dropped. Similarly for the 4-Tier model, constraints 3.16 and 3.17 are dropped. Note that constraint 3.18 is relaxed in $ATSRA_{ssr}$, but cannot be relaxed in $ATSRA_{bsr}$ as it produces an invalid solution matrix. This is because by dropping constraint 3.18 (i.e. $w_{ij} \leq y_i$), the variable w_{ij} can be assigned a non-zero value even if the corresponding data-farm node y_i is not assigned. Thus, the resultant solution matrix cannot be used in resource allocation. But in $ATSRA_{ssr}$, this relaxation is used only for the selection of Architectural Template and if the 4-Tier Architectural Template is chosen then the exact LP formulation is used for the actual resource allocation. For $ATSRA_{bsr}$, the constraints chosen for relaxation cannot not produce an invalid solution matrix. Thus, the same resultant solution matrix is used for resource allocation as well.

Note that as the constraints are relaxed in the $ATSRA_{org}$ algorithm, the following observations can be made:

- 1) Time complexity of algorithm is reduced.
- 2) Accuracy of resource-allocation algorithm decreases.

The decrease in runtime is the benefit of using the relaxations. A detailed performance analysis based on simulation experiments and a recommendation for when relaxation should be applied to the $ATSRA$ algorithm are presented in Section 4.4.

3.6.3 List Scheduling (LS)

LS is a simple RA algorithm that has low resource allocation overhead. LS is a knowledge-based algorithm and needs the knowledge of the system parameters to choose resources for the tasks [AHM08b]. The LS algorithm lists the computing resources of the given resource-pool I_i in non-descending order of their unit processing times. It, then, selects the first partition of the raw data file of the task T_i and assigns it to the first computing resource in the sorted list (i.e. to the fastest node). It repeats the process until all the partitions are assigned to the nodes. If the number of partitions of the raw data file of task T_i , p_i , is more than the number of computing nodes in the given resource-pool, then the algorithm assigns computing resources to partitions until all the computing resources are allocated. In this case $(p_i - |I_i|)$ jobs wait in a queue until nodes are freed up. Whenever a node becomes available, it is assigned to a job in the Waiting Queue. On the other hand, if $(p_i < |I_i|)$ the LS algorithm chooses p_i nodes for processing and returns the remaining $(|I_i| - p_i)$ nodes back to TRPS. The LS algorithm does not take into account the unit communication times and the resource allocation decision is based on the unit processing times only.

The RA algorithms discussed so far are the knowledge-based algorithms which need accurate knowledge of the system state during the time-slot in which the tasks in the given bag-of-tasks are expected to be executing. In contrast, we are proposing a knowledge-free algorithm in the following section that can be used by RA.

3.6.4 Round Robin Job Replicator (RRJR)

RRJR is a knowledge-free algorithm which does not need the knowledge of the system characteristics to function. The pseudo-code of RRJR algorithm is shown in Fig 3.25. If there are p_i partitions of the raw data file for task T_i , RRJR starts by creating p_i number of jobs, each of which is responsible to process a partition of the raw data file. The node for each job is chosen randomly from the resource-pool, Γ_i allocated by TRPS. If $|\Gamma_i| \geq p_i$, RA starts replicating the jobs created until all the computing nodes in Γ_i are assigned a job to execute. As soon as the data from a particular job reaches the sink nodes, all other replicas of that job are killed and the nodes that were running these jobs, are marked as free and are returned to TRPS which adds them to Δ_{free} . As the resource-pool allocated to RA consists of a fixed number of nodes and RA uses all of the nodes allocated to it for processing the task. An important thing to note is the inherent trade-off between “parallelism” and “replication” in the RRJR algorithm. The number of partitions of the raw data file determines this trade-off. For example, if the raw data file of a particular task contains only one partition, then the RRJR algorithm starts by creating one job to process this partition at one of the randomly selected nodes in Γ_i . All other nodes are used for running replicas for this job. As the number of partitions of the raw data file of the given task is increased, the parallelism increases (as more jobs are running in parallel) but replication decreases (as lower number of nodes are available for replication).

```

1: BEGIN RRJR Algorithm
2:  $j=1$ 
3:  $\Gamma_i$  = resource-pool of Task  $T_i$  allocated by TRPS
4:  $\rho_i$ = number of partitions of raw data file of Task  $T_i$ 
5: REPEAT
6:     randomly choose a node  $k$  where  $k \in \Gamma_i$ 
7:     allocate  $j^{\text{th}}$  partition to  $k$ 
8:      $j = j+1$ 
9:     IF ( $j \geq \rho_i$ )
10:          $j = 1$ 
11:     ENDIF
12:      $\Gamma_i = \Gamma_i - k$ 
13: UNTIL  $\Gamma_i \neq \{\}$ 
14: END RRJR Algorithm

```

Fig. 3.25: RRJR Algorithm

Chapter 4 Performance Analysis of the RA Algorithms

RA algorithms are used in the lower decision making module of the BiLeG architecture to allocate resources for each of the tasks from the resource-pool allocated to the task by the upper decision making module, TRPS. This chapter investigates the performance of various RA algorithms proposed in the Chapter 3. A series of simulation experiments has been conducted for the detailed performance analysis of the RA algorithms and the performance results are presented in this chapter.

4.1 Experimental Setup

To compare the performance of the RA algorithms, prototypes of various algorithms were constructed and the experiments were run using a set of 18 computers. The results obtained from these experiments were analyzed and preliminary conclusions were drawn. An important aspect of the performance analysis of the proposed RA algorithms is to observe the various performance metrics as the total number of the Grid nodes is increased. Our experimentation was limited by the number of computers available in the Epsilon system (which was available in the Real-Time and Distributed Systems lab for performing the experiments). To measure the performance of the system for a large number of nodes, a simulation model of the actual Grid system was developed. The simulator allows us to scale up to a larger number of Grid nodes in comparison to what was available on the Epsilon system.

Note that the 18 computers in the Epsilon system were used only at the initial stages of the research to validate the resource allocation approach for the PBDT tasks. Only a subset of the TRPS policies and RA algorithms were implemented and tested. The performance results obtained from these tests displayed a similar trend as the performance results of the corresponding simulation experiments.

The performance metrics are captured at the end of each simulation experiment. Each experiment is repeated enough number of times to produce a confidence interval of $\pm 5\%$ for each performance metric of interest at a confidence level of 95%.

4.2 Performance Metrics

To analyze the performance of the various RA algorithms, the following performance metrics are used:

Makespan-total ($t_{ms-total}$): The time (in seconds) required for completing all the tasks in the given bag-of-tasks.

Total Cost (t_{cost}): The total cost (in dollars) spent in executing all the tasks in the given bag-of-tasks. Note that in this research dollar is chosen as the currency for cost calculations, which can be replaced by any other currency based on the user requirements.

Algorithm-running-overhead (t_{OH}): Total time (in seconds) taken by the RA algorithm to perform the resource allocation for all the tasks in the given bag-of-tasks. This is used for analyzing the performance of the algorithms.

Makespan-without-overhead (t_{ms-woH}): The time (in seconds) required for completing all the tasks in the given bag-of-tasks, excluding the algorithm-running-overhead, t_{OH} .

4.3 The Performance Analysis of the ATSRA Class of Algorithms

4.3.1 Introduction

The ATSRA class of algorithms is based on Linear Programming. To analyze the performance of the algorithms, a model of the system characterized by various costs associated with processing and communication is used. The ATSRA algorithms give rise to a trade-off between resource assignment and the overhead associated with running the algorithms. To study the performance of the ATSRA algorithms, a simulation-based investigation is performed. The results are analyzed in this section.

4.3.2 Workload Parameters

Many multimedia encoding and High Energy Physics applications have a set of related tasks, each of which can be run independent of the other [BUN03][CRO04]. To synthesize a representative PBDT workload in a simulated environment, characteristics of such applications that can be classified as the PBDT tasks were carefully studied and it was concluded a single bag-of-tasks was the best way to model a PBDT application [AHM05]. The bag-of-task represents the constituent tasks of a particular PBDT application. The overhead associated with running TRPS policies is assumed to be negligibly small.

PBDT application chosen for this research is to model the rendering of a raw multimedia file, which is to be processed and delivered to a set of designated sink nodes. Note that the rendering of an animation movie usually involves extensive processing of the individual scene frames. In this section, the effects of variation of system and workload parameters on the performance have been analyzed by using a-factor-at-a-time approach. Only one of the parameters is varied while holding all others at their default values listed in Table 4.1. The values of the transfer and

Table 4.1: Default Workload and System Parameters

Parameter	Default Value
Number of nodes in the Grid, $ \Delta $	32
Number of sink nodes, $ \kappa_s $	16
Mean size of raw data files of tasks	650 MB
Variance of sizes of raw data files of tasks, v	5333MB ²
Transfer-in cost in dollar per hour, α_{in}	\$0.17/hr
Transfer-within cost in dollar per hour, α_{Δ}	0
Transfer-out cost in dollar per hour, α_{out}	\$0.17 /hr
Transfer source-sink cost in dollar per hour, $\alpha_{src-sink}$	\$0.17/hr
Processing cost in dollar per hour, β	\$0.17/hr
Processing factor, ε	0.75
Number of tasks in the bag-of-tasks, $ T $	12
Number of partitions in multiple partitioning policies, q	4
Number of partitions in the raw data file	Uniform [1,8]

communication costs are chosen based on the cost model of the popular commercial Cloud and

utility Grid computing services [AMA09][AZU09]. The choice of the workload parameters for the simulation is based on data for a typical animation movie described in [ENT08] and [DOU02]. The unprocessed movie is comprised of various components each of which can be independently rendered. When modelling the rendering of a raw multimedia as a PBDT application, each component represents a PBDT task with a separate raw data file. To synthesize a workload representing data of the unprocessed animation movie, the modeling of the length of the raw data files of each of the tasks is an important workload parameter. The true representative probability distribution of the sizes of the raw or unprocessed data files used in similar tasks has been a subject of discussion over the years in the research community. Researchers seem to be split over characterizing it either with Pareto or Log-normal distributions [WUY04]. After a careful analysis, the Bounded Pareto distribution seems to be a better representative of the PBDT multimedia workloads and is thus used in this research. Note that a Log-normal distribution also closely characterizes the probability of the sizes of the raw data files. Investigating the performance of resource allocation algorithms using such a distribution is worthy of further investigation.

The probability density function for the Bounded Pareto, $B(k,b,a)$ is defined as: [DOW05] [WUY04] [ANG06]

$$f_x(x) = (ak^a x^{-a-1}) / \left\{ 1 - \left(\frac{k}{b} \right)^a \right\} \quad k \leq x \leq b \quad (4.1)$$

where a is the shape parameter, which represents the variation in the length of raw data files of the PBDT tasks; k is the smallest possible length of a raw data file and b is the largest possible

length of a raw data file. To study the effect of changing the variance, the mean is fixed at 650MB and the maximum value, b , is fixed at 4000MB which corresponds to the typical values reported in [DOW05]. In order to keep the mean constant, k is adjusted slightly as value of a changes.

4.4 Performance of the ATSRA Algorithms

In this section, the experimental results of the ATSRA class of algorithms are presented. Each set of experiments focuses on a particular performance metric for each of the three ATSRA algorithms. The result of each set of experiments is discussed in a separate sub-section.

4.4.1 Selection of the TRPS Policy

In all these experiments, the same TRPS policy, SRP_{sp} , is used at the upper decision making module. As the ATSRA algorithms are based on LP model, they take considerable time to perform the resource allocation especially for large number of nodes. In order to objectively quantify the benefits of using a particular ATSRA algorithm, it is important to analyze the time taken by the algorithm to run itself and find ways to reduce this time, if possible. The performance metric, t_{OH} is the total time in seconds taken by the ATSRA algorithm to perform the resource allocation and is used to quantify the overhead produced by running the ATSRA algorithm. The complete separation of the mapping phase and the execution phase in SRP_{sp} makes it possible to precisely isolate the algorithm-running-overhead, which is not the case in any dynamic TRPS policy or a static TRPS policy with backfilling.

Also, by using SRP_{sp} , the largest possible resource-pool is allocated to each of the tasks in the

given bag-of-tasks, which means that the ATSRA algorithm is given the largest solution space for the resource allocation problem for each of the tasks. By assigning the most complex solution space to each of the ATSRA algorithms, the strengths and weaknesses of each of the ATSRA algorithm are highlighted that leads to the in-depth performance analysis under different system and workload parameters.

4.4.2 Algorithm-running-overhead

Fig. 4.1 shows the algorithm-running-overhead (t_{OH}) for each of the three ATSRA algorithms. It can be observed that for the small number of nodes in Δ , there is not much difference in the time taken in running these three algorithms. The LP formulation of $ATSRA_{org}$ in Section 3.6.2 reveals that by increasing number of nodes, the number of variables that the algorithms needs to be solved increases sharply. Thus, due to a higher computational complexity associated with $ATSRA_{org}$, the concomitant overhead is observed to increase sharply with the number of nodes in Δ , resulting in high values of t_{OH} . Although the algorithm-running-overhead associated with $ATSRA_{ssr}$ and $ATSRA_{bsr}$ also increases with the number of nodes, the rate of increase is less sharp in comparison. Results from Fig. 4.1 show that for $ATSRA_{org}$ t_{OH} rises sharply as the number of nodes in Δ becomes higher than 32. The large values of t_{OH} observed at large number of nodes provide motivation for this research to investigate means reducing this overhead. As discussed in Chapter 3, constraint-relaxation is used to control t_{OH} at large number of nodes. In an effort to reduce the value of t_{OH} , different levels of relaxation are applied to $ATSRA_{org}$, leading to $ATSRA_{ssr}$ and $ATSRA_{bsr}$ algorithms. For a given number of nodes, $ATSRA_{org}$ incurs the highest overhead followed by $ATSRA_{ssr}$ and $ATSRA_{bsr}$. For $ATSRA_{bsr}$ it can be observed

that t_{OH} does not rise sharply. And for $ATSRA_{ssr}$, t_{OH} lies between $ATSRA_{org}$ and $ATSRA_{bsr}$, as the single level of relaxation is applied in $ATSRA_{ssr}$. It can be observed that constraint-relaxation fulfils its intended purpose and effectively reduces t_{OH} , but the undesirable side-effects of using constraint-relaxation is the simplification of the LP model which reduces the accuracy of resource allocation and needs to be analyzed.

4.4.3 Makespan-without-overhead

The side-effects of relaxing some of the constraints in $ATSRA_{ssr}$ and $ATSRA_{bsr}$ can be observed by analyzing makespan-without-overhead (t_{ms-WOH}), which is the time required for completing all the tasks in the given bag-of-tasks, excluding t_{OH} . Fig. 4.2 shows the values of t_{ms-WOH} for the three $ATSRA$ algorithms as the number of nodes is increased. It is clear that $ATSRA_{org}$ has the lowest value of t_{ms-WOH} for all values of number of nodes. The lowest value of t_{ms-WOH} is expected for $ATSRA_{org}$, as by using all the constraints associated with the LP formulation in $ATSRA_{org}$, resources are allocated with the highest precision. The precision of allocating resources is decreased in $ATSRA_{ssr}$ and $ATSRA_{bsr}$. In $ATSRA_{ssr}$, as constraint-relaxation is applied at Stage-1 of the algorithms, t_{ms-WOH} increases. It increases further for $ATSRA_{bsr}$ in which constraint-relaxation is applied at both stages of the algorithm, further simplifying the LP model and reducing the accuracy in resource allocation.

4.4.4 Total Cost

Fig. 4.3 shows the t_{cost} for each of the three $ATSRA$ algorithms. As the number of nodes, is

Fig. 4.1: Impact of Increasing Number of Nodes on t_{OH}

Fig. 4.2: Effect of Increasing Number of Nodes on t_{ms-WOH}

increased, the ATSRA algorithms have a larger set of resources to choose from. The larger set of resources allows the ATSRA algorithms to choose cheaper resources which, in turn, aids in lowering t_{cost} . $\text{ATSRA}_{\text{org}}$ gives rise to the lowest t_{cost} , as the resources are allocated with highest precision. t_{cost} for $\text{ATSRA}_{\text{ssr}}$ is more than t_{cost} for $\text{ATSRA}_{\text{org}}$ for all number of nodes as constraint-relaxation is introduced. t_{cost} increases further for $\text{ATSRA}_{\text{bsr}}$ as the level of relaxation is increased. This is because by introducing more constraint-relaxation, the accuracy in resource assignment decreases, which further increases the value of t_{cost} .

4.4.5 Makespan-total

The overall makespan, $t_{\text{ms-total}}$ shown in Fig. 4.4 includes both the algorithm-running-overhead and the execution time for the bag-of-tasks. It captures the trade-off between overhead and the makespan. Note that for a small number of nodes, the solution space for $\text{ATSRA}_{\text{org}}$ is small and, thus, the overhead produced by running the algorithm is small. Thus, for a small number of nodes $\text{ATSRA}_{\text{org}}$ is the best option to be used at RA. For a small number of nodes, t_{OH} for $\text{ATSRA}_{\text{ssr}}$ and $\text{ATSRA}_{\text{bsr}}$ are comparable to that of $\text{ATSRA}_{\text{org}}$; whereas the $t_{\text{ms-WOH}}$ achieved by $\text{ATSRA}_{\text{ssr}}$ and $\text{ATSRA}_{\text{bsr}}$ are inferior to that achieved by $\text{ATSRA}_{\text{org}}$. As $t_{\text{ms-total}}$ is the sum of t_{OH} and $t_{\text{ms-WOH}}$, $t_{\text{ms-total}}$ achieved by using $\text{ATSRA}_{\text{org}}$ is lower than $t_{\text{ms-total}}$ achieved with $\text{ATSRA}_{\text{ssr}}$ and $\text{ATSRA}_{\text{bsr}}$ when the number of nodes is small. It can be observed in Fig. 4.4, that when the number of nodes is less than 24, $\text{ATSRA}_{\text{org}}$ is the best algorithm to be used for minimizing $t_{\text{ms-total}}$.

Fig. 4.3: Impact of Increasing Number of Nodes on t_{cost}

Fig. 4.4: Impact of Increasing Number of Nodes on $t_{ms-total}$

It can also be observed from Fig. 4.4 that $\text{ATSRA}_{\text{ssr}}$ produces the lowest value of $t_{\text{ms-total}}$ for a range of intermediate values of the number of Grid nodes. Note that the accuracy of resource allocation for $\text{ATSRA}_{\text{ssr}}$ lies between that achieved with $\text{ATSRA}_{\text{org}}$ and $\text{ATSRA}_{\text{bsr}}$.

In Fig. 4.4, the high value of $t_{\text{ms-total}}$ at larger number of nodes observed for $\text{ATSRA}_{\text{org}}$ demonstrates the necessity of the need to find ways to reduce the algorithm-running-overhead as the problem size increases. For a large number of nodes in Δ , t_{OH} for $\text{ATSRA}_{\text{org}}$ and $\text{ATSRA}_{\text{ssr}}$ are very high and the benefit of using accurate resource allocation is offset by very high value of t_{OH} .

It can be observed in Fig. 4.1 and Fig. 4.2 that for a large number of nodes, although $\text{ATSRA}_{\text{org}}$ and $\text{ATSRA}_{\text{ssr}}$ give rise to lower $t_{\text{ms-WOH}}$ in comparison to $\text{ATSRA}_{\text{bsr}}$, the advantage is offset by the much higher t_{OH} produced by $\text{ATSRA}_{\text{org}}$ and $\text{ATSRA}_{\text{ssr}}$. The net effect is that $t_{\text{ms-total}}$ achieved by $\text{ATSRA}_{\text{org}}$ and $\text{ATSRA}_{\text{ssr}}$ is inferior to that of $\text{ATSRA}_{\text{bsr}}$ for a large number of nodes (see Fig. 4.4). It can be observed from Fig. 4.4, that as number of nodes is increased beyond 48, $\text{ATSRA}_{\text{bsr}}$ produces the lowest $t_{\text{ms-total}}$ among all ATSRA algorithms.

4.4.6 Cost-mstotal

Minimizing $t_{\text{ms-total}}$ and t_{cost} are somewhat contradictory performance objectives. In addition to focusing on each of these objectives individually, it is also important to study the balance achieved between these two performance objectives by the use of a particular ATSRA algorithm.

cost-mstotal is used to quantify this balance. cost-mstotal is the product of the total cost of executing all the tasks in the given bag-of-tasks (t_{cost}) and the time in seconds required for completing all the tasks in the given bag-of-tasks ($t_{\text{ms-total}}$). In Fig. 4.5, the values of cost-mstotal, which captures the trade-off between $t_{\text{ms-total}}$ and t_{cost} , are plotted for the three ATSRA algorithms.

When the number of nodes is below 72, the solution space given to each ATSRA algorithm is small which produces small t_{OH} (see Fig. 4.5). Note that $\text{ATSRA}_{\text{org}}$ always allocates the resources with most precision irrespective of any system or workload parameters. Thus, for a small number of nodes (below 72), $\text{ATSRA}_{\text{org}}$ allocates the resources with most precision and also generates a small algorithm-running-overhead due to a simpler solution space. The combination of these translates into the lowest value of cost-mstotal (see Fig. 4.5). When the number of nodes increases, t_{OH} achieved with $\text{ATSRA}_{\text{org}}$ increases. As $t_{\text{ms-total}}$ is the sum of t_{WOH} and t_{OH} , the increase in t_{OH} causes an increase in $t_{\text{ms-total}}$. Note that the increased value of $t_{\text{ms-total}}$ results in higher value of cost-mstotal for $\text{ATSRA}_{\text{org}}$. As a result $\text{ATSRA}_{\text{org}}$ produces the highest cost-mstotal, when the number of nodes are higher than 86. It can be observed that at higher number of nodes (above 72), $\text{ATSRA}_{\text{bsr}}$ produces the lowest value of cost-mstotal (see Fig. 4.5). Thus, if it is a small Grid (less than 72 nodes) and the objective is to achieve a balance between $t_{\text{ms-total}}$ and t_{cost} , then $\text{ATSRA}_{\text{org}}$ algorithm should be used. For a larger Grid, (greater than 72) nodes, constraint-relaxations used in $\text{ATSRA}_{\text{bsr}}$ is observed to effectively manage t_{OH} associated with a larger solution space and should be the chosen at RA to reduce cost-mstotal. It is

interesting to note that if the objective is to minimize cost-mstotal, $\text{ATSRA}_{\text{ssr}}$ is not the best choice for any number of nodes in the Grid.

Fig. 4.5: Impact of Increasing Number of Nodes on cost-mstotal

4.5 Performance Analysis of the Low Overhead RA Algorithms

The ATSRA algorithms are based on an exact *a priori* knowledge of unit processing and communication times. The performance of two simple algorithms, LS and RRJR, is discussed in this section. Both LS and RRJR use simple algorithms and have negligible algorithm-running-overhead, t_{OH} . RRJR does not use any knowledge of the system parameters. LS uses only the knowledge of computation costs associated with the resources. An analysis of the performance of

these two RA algorithms is performed to understand the importance of using knowledge of system parameters in resource allocation on performance.

4.5.1 Choice of the TRPS Policies

Only dynamic policies are used at the upper decision making module TRPS for this set of experiments. The reason is that when the RRJR algorithm is used at RA, static TRPS policies cannot be used at TRPS. This is because, RRJR dynamically allocates resources and unlike static TRPS policies, the division of the algorithm in a mapping and an execution phase is not possible.

The TRPS policies used in this set of experiments use multiple partitioning of Δ to allocate resource-pools. As discussed in Section 3.6.4, multiple partitioning at TRPS limits the amount of replication. Use of multiple partitioning TRPS policies allows the analysis of the effect of limiting the replication. The two dynamic policies used are DRP_{mp} and DRP_{mp-pro} . DRP_{mp} uses equal sized partitions whereas DRP_{mp-pro} assigns a larger partition to a task having a larger raw data file; thus effectively providing more replication level to the task having larger raw data file. The combination of RRJR and LS with DRP_{mp} and DRP_{mp-pro} gives rise to four different allocation-plans; $\langle DRP_{mp}, RRJR \rangle$, $\langle DRP_{mp-pro}, RRJR \rangle$, $\langle DRP_{mp}, LS \rangle$ and $\langle DRP_{mp-pro}, LS \rangle$.

4.5.2 Effect of Increasing the Number of Nodes on $t_{ms-total}$

Fig. 4.6 shows the effect of increasing the number of nodes in Δ on $t_{ms-total}$. For RRJR based allocation-plans, it can be observed that as the number of nodes is increased, each of the allocation-plans shows an initial sharp drop in $t_{ms-total}$. This is because, initially, the number of

nodes is not enough to run all the tasks in parallel. As the number of nodes in Δ is increased, more and more tasks start to run in parallel decreasing $t_{ms-total}$. Once maximum possible tasks are running in parallel, increasing number of nodes further has less dramatic effect on $t_{ms-total}$. It can be observed that with $\langle DRP_{mp}, LS \rangle$ and $\langle DRP_{mp-pro}, LS \rangle$, $t_{ms-total}$ decreases initially as the number of nodes in Δ is increased. This is because, initially, the number of nodes is not enough to run all the tasks in parallel. As described in Section 3.6.3, there is no replication in LS-based allocation-plans. Further decrease in $t_{ms-total}$ is attributed to the fact that increasing the number of nodes in Δ increases the probability of finding a faster node. For RRJR-based allocation-plans, $t_{ms-total}$ is lowest at higher number of nodes in Δ . This is attributed to higher level of replication as the resource-pool assigned to each of the task is increased. Generally, $\langle DRP_{mp-pro}, RRJR \rangle$ is better than $\langle DRP_{mp}, RRJR \rangle$ due the proportional resource partitioning which results in better resource utilization and improvement in the $t_{ms-total}$, as observed in the Fig. 4.6.

4.5.3 Effect of Increasing the Number of Nodes on t_{cost}

Fig. 4.7 shows the effect on t_{cost} as the number of nodes is increased. Note that the t_{cost} of RRJR based algorithm is quite high and it increases linearly with the increase in number of nodes. This is because in RRJR, all the nodes in the resource-pool are used to process a particular task. Also, the t_{cost} for $\langle DRP_{mp-pro}, RRJR \rangle$ is lower than $\langle DRP_{mp}, RRJR \rangle$, as the size of the resource-pool is proportional to the length of raw data file of each task. This results in better management of replication in $\langle DRP_{mp-pro}, RRJR \rangle$ which is in line with the need of the each individual task.

Fig. 4.6: Impact of Increasing Number of Nodes on $t_{ms-total}$

Fig. 4.7: Effect of Increasing Number of Nodes on t_{cost}

As there is no replication in LS, t_{cost} associated with the LS-based allocation-plans is much smaller than t_{cost} associated with the RRJR-based allocation-plans. Contrary to RRJR based allocation-plans, in LS-based allocation-plans, t_{cost} drop slightly as the number of nodes is further increased; as there is an increased selection of nodes for choosing the nodes with the lowest unit processing times.

4.5.4 Effect of Increasing the Variance in Unit Communication Times on $t_{\text{ms-total}}$

This set of experiments is conducted to analyze the behavior of the two low overhead RA algorithms, as the variance in the unit communication times is increased. Note that the LS algorithm chooses the resources based on the unit processing times only, as explained in the Chapter 3. Thus, it is expected to perform good resource allocation, only if the variance in the unit communication times is low. Fig. 4.8 shows the effect of increasing the variance of the unit communication times on $t_{\text{ms-total}}$. It can be observed that $t_{\text{ms-total}}$ increases sharply for LS-based algorithms as the variance in unit communication times increases. This is because if the variance of the unit communication times is high, time to perform data communication between two nodes can vary across a wide range of values. As LS takes the resource allocation decision based on unit processing times only, it may choose nodes having low unit processing times but very high unit communication times. The choice of inefficient communication links deteriorates the overall system performance when LS is used. It can be observed in Fig. 4.8 that for the RRJR algorithm an increase in $t_{\text{ms-total}}$ is much less steep as compared to LS as the variance in the unit transfer times is increased. When the variance in the unit communication times is high, there are some very fast and some very slow communication links in the system. Unlike LS algorithm, if

Fig. 4.8: Impact of Increasing Variance in Unit Transfer Times on $t_{ms-total}$

the RRJR algorithm chooses a very slow link, due to replication of resources it may have limited effect on the value of $t_{ms-total}$, unlike LS algorithm. The reason is that due to replication the job of transferring the data through the slow link will be replicated on a redundant link that may be fast, thus, reducing the overall impact of increase in variance in the unit communication times on the value of $t_{ms-total}$.

4.5.5 Effect of Increasing Mean Unit Communication Time on $t_{ms-total}$

Fig. 4.9 shows the effect of increasing mean unit communication time on $t_{ms-total}$. It can be observed that both LS-based allocation-plans perform better than all other algorithm when unit communication times are low. This is because in LS-based allocation-plans, the nodes are

selected based on the unit processing times of nodes only. When the unit communication times are negligible, $t_{ms-total}$ for LS-based allocation-plans is low. $t_{ms-total}$ increases sharply as the mean unit communication times is increased. For RRJR based allocation-plans the increase in mean unit communication time has less drastic effect on $t_{ms-total}$ due the process of replication which results in the choice of redundant communication links between different nodes.

Fig. 4.9: Effect of Increasing Mean Unit Communication Time on $t_{ms-total}$

4.6 Summary

In this chapter the performance of five different RA algorithms has been studied. First, the performance of the three ATSRA-based algorithms has been analyzed. Then, the performance of the two simple RA algorithms has been studied.

For small Grids, having less than 32 nodes, $\text{ATSRA}_{\text{org}}$ algorithm always gives the best performance, irrespective of the performance objective. This is because in small Grids, the solution space of the LP algorithm is not large and, thus, it does not generate significant algorithm-running-overhead. As $\text{ATSRA}_{\text{org}}$ allocates resources with the highest precision, it is able to give the best performance for small Grids.

It is also concluded from the experiments described in this chapter that if the objective is to minimize t_{cost} , $\text{ATSRA}_{\text{org}}$ should be used irrespective of the size of the Grid. Note that the ATSRA-based algorithms are designed with the single objective of achieving the lowest t_{cost} . As $\text{ATSRA}_{\text{org}}$ does not ignore any constraints, the resources are allocated with the maximum precision and, thus, the use of $\text{ATSRA}_{\text{org}}$ gives rise to the minimum t_{cost} as compared to the other two ATSRA-based algorithms.

It is also observed from the experimental results that for large Grids, the use of $\text{ATSRA}_{\text{bsr}}$ is effective in effectively reducing the algorithm-running-overhead. $\text{ATSRA}_{\text{bsr}}$ is recommended to be used at RA if the objective is to minimize $t_{\text{ms-total}}$, or if the objective is to achieve a good balance between t_{cost} and $t_{\text{ms-total}}$ by minimizing cost-ms-total . The use of $\text{ATSRA}_{\text{ssr}}$ is only recommended for a medium sized Grid, having between 16 to 32 nodes, if the objective is to reduce $t_{\text{ms-total}}$.

When choosing between the two low overhead algorithms, if the objective is to minimize t_{cost} , the use of RRJR should always be avoided irrespective of the system and workload parameters. On the other hand, if the objective is to minimize $t_{\text{ms-total}}$, replication of jobs used by RRJR is

observed to be quite effective in reducing $t_{ms-total}$ and RRJR performs well irrespective of the number of nodes in the system.

The performance of the ATSRA-based algorithms and the simpler RA algorithms can be compared by using the experimental results presented in this chapter. Note that the three ATSRA algorithms generate considerable algorithm-running-overhead. The ATSRA algorithms also require accurate knowledge of the unit processing and communication times. On the other hand, RRJR and LS are simple RA algorithms having negligible algorithm-running-overhead. RRJR, being a knowledge-free algorithm, does not need any knowledge of the system and workload parameters. The effort put into the development of the ATSRA algorithms can only be justified if the ATSRA algorithms perform better than the two proposed simple algorithms for a wide range of workload and system parameters. Following observations can be made from the experimental results presented in this chapter, which show that ATSRA-based RA algorithms perform better than RRJR and LS algorithms for a wide range of workload and system parameters.

- 1- Irrespective of the size of the Grid, if the performance objective is to reduce t_{cost} , all three ATSRA algorithms perform better than both LS and RRJR (see Fig. 4.3 and Fig. 4.7).
- 2- If the objective is to reduce $t_{ms-total}$, and the number of nodes is less than 48 then the ATSRA-based algorithms perform better than both the LS and RRJR algorithms (see Fig. 4.4 and Fig. 4.6)

- 3- If the objective is to reduce $t_{ms-total}$, and the number of nodes is greater than 48, $ATSRA_{bsr}$ performs better than LS and RRJR. The performance of $ATSRA_{org}$ and $ATSRA_{ssr}$ is worse than LS and RRJR due to large overheads produced.

A detailed comparison of the performance of the knowledge-based allocation-plans (based on $ATSRA$ algorithms) and the knowledge-free allocation-plans (based on RRJR) is performed in Chapter 6.

Chapter 5 Performance of the TRPS Policies

In the proposed BiLeG architecture, the upper level decision making module, TRPS, is responsible for selecting the resource-pool and each TRPS policy proposes a different strategy for it. To compare the performance of the TRPS policies under various system and workload parameters, a set of experiments has been designed. The set of experiments is divided into various groups such that each group is responsible for investigating a particular aspect of the TRPS policies. A detailed discussion follows the description of each group of experiments. The experimental setup and the workload model presented in Section 4.1 is used in this set of experiments.

5.1 Effect of Increasing the Size of Δ on the Performance of the TRPS Policies

One of the primary functions of the TRPS policies is to assign a resource-pool to each of the tasks in the given bag-of-tasks. A resource-pool of a particular task T_i is represented by Γ_i and is a subset of the complete set of all the Grid nodes, Δ . Different TRPS policies employ various strategies to choose a resource-pool for each of the tasks. To study the effectiveness of the resource-pool selection strategies, the analysis of the effect of increasing the size of Δ on the performance of the TRPS policies is performed. In this section various performance metrics are measured as the size of Δ is increased, while the other parameters related to the workload and the system are held at their default values shown in Table 4.1. For all these experiments, the same RA algorithm is used, so that meaningful comparison among TRPS policies can be performed. The RA algorithm chosen is $ATSRA_{org}$, which is described in Chapter 3. The $ATSRA_{org}$

algorithm is chosen at RA because it provides the lowest value of t_{cost} irrespective of any system or workload conditions.

5.1.1 Effect of Increasing the Size of Δ on $t_{\text{ms-WOH}}$

Fig 5.1 presents the effect of increasing the number of nodes on $t_{\text{ms-WOH}}$ for various TRPS policies. As described in Chapter 4, $t_{\text{ms-WOH}}$ is the time required for completing all the tasks in the given bag-of-tasks, excluding the algorithm-running-overhead, t_{OH} . When the number of nodes in Δ is increased, the $\text{ATSRA}_{\text{org}}$ algorithm used at RA has a larger set of resources to choose from. The larger set of resources helps the $\text{ATSRA}_{\text{org}}$ algorithm to choose cheaper resources, which, in turn, helps in lowering $t_{\text{ms-WOH}}$ for all TRPS policies which can be observed in Fig. 5.1.

Fig. 5.1: Impact of Increasing Number of Nodes on $t_{\text{ms-WOH}}$ for Different TRPS Policies

As shown in Fig. 5.1, SRP_{sp} has the highest t_{ms-WOH} for a given number of nodes. The reason for the highest value of t_{ms-WOH} lies in the way resources are allocated in SRP_{sp} , which introduces considerable resource contention. Resource contention occurs when in the execution-phase, more than one tasks want to use the same set of resources resulting in overlapping resource demands. Note that, as indicated in Section 3.2, for the computing environment used in this research nodes are not shared. Only one of the tasks is allowed to use a particular resource at a time and all other tasks, needing same resource, have to wait until that resource becomes free. In SRP_{sp} the allocation of the resources consists of two completely independent phases; a mapping phase and an execution phase. In the mapping phase, resource allocation of each of the tasks is performed completely independent of the resource allocation of other tasks and the resources are chosen from the same resource-pool. While this mapping strategy gives the “cheapest” set of resources to each task, it also generates an overlap between the chosen sets of resources which are selected for different tasks. In the execution phase, when a particular task is being executed, if there is an overlap in the set of resources assigned to different tasks, other tasks may have to wait for the complete set of resources they have been assigned, to become free. The resource contention adds to the total time taken by the system to process the complete bag-of-tasks and thus increases t_{ms-WOH} (see Fig. 5.1).

In order to improve upon t_{ms-WOH} obtained by using SRP_{sp} , all other TRPS policies use various strategies to reduce the resource contention experienced in SRP_{sp} . The following three strategies have been used in various TRPS policies to improve t_{ms-WOH} .

Strategy 1: Enhancement of the mapping phase of the SRP_{sp} policy, by dividing the resources into different partitions in multiple resource partitioning policies.

Strategy 2: Enhancement of the execution phase of the SRP_{sp} policy, by introducing backfilling.

Strategy 3: Use of Dynamic Resource allocation

The enhancement of the mapping phase of the SRP_{sp} policy by dividing the Grid resources into different partitions results in a TRPS policy named as SRP_{mp} . In SRP_{mp} , Δ is divided into q fixed partitions and the resource-pool for a particular task is created from one of the partitions of Δ , instead from the complete set of available nodes, Δ . This strategy reduces the probability of the overlapping of resources assigned to various tasks, as discussed in Chapter 3, and decreases resource-contention. The decreased resource-contention results in lower t_{ms-WOH} for SRP_{mp} (see Fig. 5.1).

The second strategy of reducing resource-contention to improve t_{ms-WOH} by using backfilling. This strategy is used in $SRP_{sp}+BF$. Note that the mapping phase of $SRP_{sp}+BF$ is the same as SRP_{sp} . In the execution phase, backfilling is triggered if a situation arises when some tasks are waiting for the set of resources, they have been assigned, to become free when there are idle resources in the system. This may be because these idle resources were not chosen to be assigned to any task during the mapping phase because of higher cost associated with these resources. As discussed in Section 3.5.3, backfilling leads to the dynamic creation of a new resource-pool for the waiting task having the smallest raw data file, by using the set of idle resources. The amount

of backfilling depends on the number of idle resources available at any time. As the number of nodes in Δ is increased, the nodes that are not assigned to any task increases, thus, increasing the degree of backfilling. Experimental results show that backfilling significantly reduces t_{ms-WOH} . When the number of nodes in Δ becomes greater than 72, $\langle SRP_{sp}+BF \rangle$ exhibits the lowest t_{ms-WOH} among all TRPS policies (see Fig. 5.1).

The third strategy to reduce resource-contention associated with SRP_{sp} is to use dynamic resource allocation. This strategy is used in three TRPS policies, DRP_{sp} , DRP_{mp} and DRP_{mp-pro} . Dynamic TRPS policies use the dynamic resource allocation at TRPS to eliminate the resource-contention introduced in the static TRPS policies. Note that t_{ms-WOH} for DRP_{sp} in Fig. 5.1 is lower than that for SRP_{sp} for all number of nodes. The drawback of DRP_{sp} is that there is an unfair advantage for the tasks whose resource-pools are assigned earlier by TRPS. The resource-pool for each of the subsequent tasks shrinks. To address this, in DRP_{mp} , Δ is divided into multiple partitions. By dividing the available resources into different partitions, the resource-pool of the first task is reduced to a particular partition of Δ only. t_{ms-WOH} is further improved when the size of each partition is chosen proportional to the size of raw data file in DRP_{ms-pro} as shown in Fig. 5.1.

In multiple partitioning TRPS policies, increasing the number of partitions increases t_{cost} and decreases $t_{ms-total}$. Note that the increase in number of partitions decreases the size of resource-pools available for each of the tasks, which limits the choice of resources for the RA algorithm and thus results in an increased t_{cost} . Increasing the number of partitions improves $t_{ms-total}$ in two different ways. First, the increased number of partitions decreases resource contention which

improves $t_{ms-total}$. Second, by reducing the size of resource-pools, it reduces the complexity of the solution space of the ATSRA algorithm used at RA , thus, reducing $t_{ms-total}$. Based on an initial set of experiments, four partitions seem to strike an effective compromise between t_{cost} and $t_{ms-total}$. Thus four partitions are used in the investigation of all multiple partitioning policies. Detailed analysis of the effect of the number of partitions on $t_{ms-total}$ and t_{cost} and finding an effective way of choosing the most suitable number of partitions for a set of particular system and workload parameters form an important direction for future research.

5.1.2 Effect of Increasing the Size of Δ on t_{cost}

Fig 5.2 shows the effect of increasing the number of Grid nodes on t_{cost} for various TRPS policies. As described in Chapter 4, t_{cost} is the total cost spent in executing all the tasks in the given bag-of-tasks and directly depends on the total usage of the resources. The $ATSRA_{org}$ algorithm (used at RA for this set of experiments) is designed to allocate resources with the objective of minimizing t_{cost} . Generally, as the number of nodes is increased, the $ATSRA_{org}$ algorithm has a larger set of resources to choose from, which helps the algorithm to choose cheaper resources and lowers t_{cost} . This can be observed from Fig 5.2, where t_{cost} tends to decrease as the number of Grid nodes is increased for all TRPS policies, with the exceptions of the $SPR_{sp}+BF$ policy. The different behavior of $SPR_{sp}+BF$ is due to the process of backfilling used in $SPR_{sp}+BF$. Backfilling is used to bring into play the resources that are not otherwise being utilized at runtime. In backfilling a resource-pool is created at runtime by using the resources that are not selected in mapping phase by the RA algorithm due to their higher costs. Thus, backfilling uses “costlier” resources originally left out by the $ATSRA_{org}$ algorithm during

Fig. 5.2: Impact of Increasing Number of Nodes on t_{cost} for Different TRPS Policies

the mapping phase and generates higher values of t_{cost} . In Fig. 5.2, it can be observed that as the number of nodes is below 24, and there is negligible backfilling, the behavior of $\text{SRP}_{\text{sp}}+\text{BF}$ is similar to SRP_{sp} and t_{cost} drops. As the number of nodes is increased above 24 nodes, backfilling increases, driving up the value of t_{cost} (see Fig. 5.2)

Another important observation from Fig. 5.2 is that the lowest value of t_{cost} is associated with SRP_{sp} . When SRP_{sp} is used at TRPS, the resource-pool for each task in the given bag-of-tasks consists of Δ which is the complete set of the available nodes. Thus, the $\text{ATSRA}_{\text{org}}$ algorithm

used at RA can choose the best of the resources for each of the tasks which results in generating the lowest value of t_{cost} for all number of nodes as compared to other TRPS policies.

Multiple partitioning is used in SRP_{mp} in which Δ is divided into q partitions. Each set of nodes in a particular partition m is represented by Δ_m . Note that in SRP_{mp} the resource-pool for each of the tasks is selected from one of the partitions. When the size of a partition Δ_m , is approximately equal to the number of partitions of a task T_i , (i.e. $|\Delta_m| \approx \rho_i$) there are not too many resources to choose from for the RA algorithm and nearly all available resources have to be assigned to a particular task regardless of the costs associated with them. As the size of Δ increases, size of each individual partition increases proportionally, providing a larger resource-pool and more choices for the RA algorithm, thus lowering t_{cost} . This can be observed in Fig. 5.2: for SRP_{mp} , t_{cost} drops sharply as the size of Δ is increased.

For the dynamic TRPS policies, the value of t_{cost} is higher as compared to the static policies (see Fig. 5.2). This is because the resource-pools are chosen only from the nodes which are not busy, instead of the complete set of nodes. As the tasks start running, the resource allocation algorithm chooses all the available resources to form the resource-pool of the first task. For the second task the resource-pool “shrinks” and excludes the resources chosen for the first task as explained in detail in Section 3.5.2.

Among three dynamic TRPS policies, t_{cost} for DRP_{sp} is the lowest as there is no partitioning in Δ and hence the resource-pool for the first task is chosen from the complete set of Grid nodes

available and the resources leading to the lowest cost can be chosen. When partitioning is used in DRP_{mp} , t_{cost} increases as the resource-pool of for the first task is chosen from one of the partitions only, which is a subset of Δ . t_{cost} improves slightly when instead of the equal partitioning proportional partitioning is used, resulting in a better utilization of the available resources.

5.2 Algorithm-running-overhead

Algorithm-running-overhead, t_{OH} , is the total time in seconds taken by the RA algorithm to perform the resource allocation of all the tasks in the given bag-of-tasks. For a given RA algorithm, t_{OH} depends directly on the size of the solution space. For a particular task, the size of its resource-pool is an indicator of the solution space for the RA algorithm. As various TRPS policies use different strategies for selecting the size of the resource-pool, various TRPS policies give rise to different t_{OH} and its analysis forms an important component of the performance analysis of the TRPS policies.

5.2.1 Effect of Algorithm-running-overhead on $t_{ms-total}$

In calculating the makespan for the performance analysis of the TRPS policies conducted in Section 5.1.1, t_{ms-WOH} is used which does not take into account algorithm-running-overhead. This section presents the results for $t_{ms-total}$, which is the makespan obtained when algorithm-running-overhead is included. Note that if an off-site computing facility is not used then $t_{ms-total}$ represents the actual end-to-end time experienced by the end user for the processing of the given bag-of-tasks and is important to analyze.

Fig. 5.3 shows the effect of increasing the number of nodes on $t_{ms-total}$ for the different TRPS policies. It is interesting to observe that $t_{ms-total}$ is minimized for various policies at different number of nodes. When the number of nodes are 128, both the SRP_{sp} and $SRP_{sp}+BF$ policies give rise to one of the highest values of $t_{ms-total}$. The highest value of $t_{ms-total}$ for SRP_{sp} is due to the fact that the solution space provided to the RA algorithm for each of the tasks consists of the largest possible set of nodes, Δ . SRP_{sp} also has highest value of $t_{ms-total}$ when the number of nodes are as low as 8. The high value of SRP_{sp} when the number of nodes is between 8 and 16 is due to resource-contention. When backfilling is added to reduce resource-contention in $SRP_{sp}+BF$, it is seen to be quite effective when the number of nodes is less than 24. In fact, introducing backfilling generates one of the lowest t_{ms-WOH} values for all values of Δ (see Fig. 5.1). But, the use of backfilling has another undesirable side-effect. It introduces considerable algorithm-running-overhead, as the resource allocation algorithm has to run twice; once in the mapping phase and then during backfilling process. As shown in Fig. 5.3, this translates into higher values of $t_{ms-total}$, especially for higher number of nodes. When multiple partitioning is used in SRP_{mp} , it considerably reduces $t_{ms-total}$. The reduction in $t_{ms-total}$ is especially pronounced for high number of nodes (for example 128) and is due to the combination of the two following factors.

- 1) Resource contention is reduced, as overlapping among the different resource assignments made by the RA algorithm is reduced. The reason for the reduction in the overlapping is the assigning of the resource-pool from one of the partitions of Δ in a round-robin fashion. If the number of tasks are more than the number of partitions then there may still be some overlapping as some partitions have to be used

for providing the resource-pool of more than one tasks, but the overlapping will always be less than that of SRP_{sp} , in which all resource-pools are assigned from the single partition, Δ .

- 2) The algorithm-running-overhead is reduced as the RA algorithm is given only a partition of the total resources available as the resource-pool, considerably reducing the solution space and hence the complexity of the problem.

In dynamic TRPS policies, resource-contention is reduced. For DRP_{sp} , the resource-pools for all the tasks are allocated from a single partition. When number of nodes are less than 72, $t_{ms-total}$ obtained by using of DRP_{sp} is low compared to other TRPS policies. As the number of nodes is increased above 72, the solution space of the DRP_{sp} , becomes increasingly complex generating considerable overheads which makes $t_{ms-total}$ quite high. For a high number of nodes (above 72) multiple partitioning in dynamic TRPS policies reduces algorithm-running-overhead and in turn, decreasing $t_{ms-total}$. Thus, when multiple partitioning is used in DRP_{mp} and DRP_{mp-pro} , $t_{ms-total}$ is further reduced as compared the value of $t_{ms-total}$ obtained when DRP_{sp} is used.

5.2.2 Experimental Analysis of Algorithm-running-overhead

This set of experiments is designed to analyze the algorithm-running-overhead when different TRPS policies are used. For analysis of system performance, it is important to know when the system ran the resource allocation algorithm. Specifically, it is important to determine the component of the algorithm-running-overhead that is expected to occur before the first task in the given bag-of-tasks starts executing. The portion of the algorithm-running-overhead that

Fig. 5.3: Effect of Increasing Number of Nodes on $t_{ms-total}$ for Different TRPS Policies

occurs before the first task starts executing on the Grid system is termed as the *algorithm-running-overhead-offline* (t_{OH-off}). The remaining part of the algorithm-running-overhead, which occurs after the first task starts executing on the Grid system is referred to as the *algorithm-running-overhead-online* (t_{OH-on}). If a major portion of t_{OH} consists of t_{OH-off} , it may translate into a reduced $t_{ms-total}$. For example, instead of using one of the Grid nodes to run the resource allocation algorithm, a computing facility outside of the actual Grid (called the *off-site computing facility*) can be used to run the resource allocation algorithm.

There are three potential advantages of using an off-site computing facility:

- 1) A fast computer, not connected to the Grid, may be used as an off-site computing facility. This may result in considerable reduction of t_{OH} due to the increased computing power, which may, in turn, reduce the total makespan, $t_{ms-total}$. The detailed analysis of the effects of reducing t_{OH} on $t_{ms-total}$ by increasing computing power is presented in Section 5.2.4.
- 2) The off-site computing facility, not being part of the Grid, may have a lower value of β (cost in dollars per hour) as compared to the Grid nodes. The lower β value for the node used for processing may have a considerable effect on the total cost associated with processing the given bag-of-tasks.
- 3) If the given bag-of-tasks is scheduled to be processed sometime in the future using the Grid, the off-site computing facility can be used beforehand to complete the mapping phase of the TRPS policy used. This will result in creating the mapping between each of the tasks and the set of resources allocated to it. Once this mapping is created, it can be provided to the Grid at runtime to directly start the execution phase associated with the policy. In this case, at runtime, only t_{OH-on} will be incurred hence reducing the algorithm-running-overhead which, in turn, decreases $t_{ms-total}$ and improves the overall performance of the system.

For a given bag-of-tasks, the value of t_{OH-off} depends on whether the TRPS policy being used is a static TRPS policy or a dynamic one. In a static TRPS policy, all the algorithm-running-

overhead is incurred during the mapping phase. At the end of the mapping phase, all the resources have been allocated and no resource allocation takes place during the execution phase.

Hence for a static TRPS policy,

$$t_{OH-off} = t_{OH} \text{ and } t_{OH-on} = 0$$

Because no algorithm-running-overhead is incurred when the tasks are being processed, static TRPS policies are good candidates for use, when an off-site computing facility is available.

In case of a dynamic TRPS policy, there are no separate mapping and execution phases. The resource allocation occurs dynamically as the tasks are being executed and the algorithm-running-overhead is “embedded” in the total makespan for the given bag-of-tasks. In a dynamic TRPS policy, algorithm-running-overhead-offline, t_{OH-off} , is equal to the time the resource allocation algorithm spends in allocating the resources for the first task in the given bag-of-tasks. If t_{OH1} is the algorithm-running-overhead associated with allocating resources for the first task, then for a dynamic TRPS policy:

$$t_{OH-off} = t_{OH1} \text{ and } t_{OH-on} = t_{OH} - t_{OH1}$$

In a dynamic TRPS policy, the use of an off-site computing facility is not very useful as t_{OH-off} is only a small portion of t_{OH} .

5.2.3 Effect of Increasing the Size of Δ on Algorithm-running-overhead

Fig 5.4, Fig. 5.5 and Fig. 5.6 show t_{OH-on} , t_{OH-off} and t_{OH} respectively for the different TRPS policies. It can be observed that for the two static TRPS policies, SRP_{sp} and SRP_{mp} , t_{OH-on} is

zero. The use of backfilling in $\text{SRP}_{\text{sp}}+\text{BF}$ generates some interesting algorithm-running-overhead patterns. Note that $t_{\text{OH-off}}$ for $\text{SRP}_{\text{sp}}+\text{BF}$ is equal to the $t_{\text{OH-off}}$ of the SRP_{sp} . This is because in both SRP_{sp} and $\text{SRP}_{\text{sp}}+\text{BF}$, mapping phases are similar, thus generating exactly identical plots of $t_{\text{OH-off}}$ as shown in Fig. 5.5. For $\text{SRP}_{\text{sp}}+\text{BF}$, resource-allocation also occurs in the execution phase due to backfilling, which is contrary to SRP_{sp} . Thus $t_{\text{OH-on}}$ is non-zero for backfilling based $\text{SRP}_{\text{sp}}+\text{BF}$ TRPS policy.

As expected, it can be observed that for the dynamic TRPS policies, $t_{\text{OH-off}}$ is small as compared to the static policies. On the other hand, for a dynamic TRPS policy $t_{\text{OH-on}}$ is the time the algorithm takes in allocating resources for all the tasks, except the first task. For DRP_{sp} , $t_{\text{OH-on}}$ increases sharply as the number of nodes is increased in the system (see Fig. 5.4). One of the ways to reduce $t_{\text{OH-on}}$ for DRP_{sp} at large number of nodes is to divide Δ into partitions. It reduces $t_{\text{OH-on}}$ because subdividing the solution space for the LP algorithm into smaller subsets decreases the algorithm-running-overhead. This reduction of $t_{\text{OH-on}}$ at higher number of nodes due to multiple partitioning can be observed from Fig. 5.4 where $t_{\text{OH-on}}$ for DRP_{mp} and $\text{DRP}_{\text{mp-pro}}$ is less than that of DRP_{sp} when the number of nodes in Δ is greater than 108.

Fig. 5.6 shows the value of t_{OH} as the number of nodes is increased. Note that t_{OH} is the summation of $t_{\text{OH-on}}$ and $t_{\text{OH-off}}$. For a small number of nodes, the problem size for the two single partition static policies (i.e. SRP_{sp} and $\text{SRP}_{\text{sp}}+\text{BF}$) is small and not too much algorithm-running-

Fig. 5.4: Impact of Increasing n on t_{OH-on} for Different TRPS Policies

Fig. 5.5: Impact of Increasing n on t_{ms-OFF} for Different TRPS Policies

overhead is produced by the execution of the algorithm (see Fig. 5.6). t_{OH} associated with these two static policies can be observed to rise sharply as number of nodes in Δ is increased beyond approximately 56. Among the static policies, SRP_{SP+BF} has the highest value of t_{OH} . One of the reason for the highest t_{OH} for SRP_{SP+BF} lies in the fact that SRP_{SP+BF} is the only policy in which $ATRSRA_{org}$ is executed twice; once in the mapping phase and then at runtime when Reducing Resource-pool Algorithm is used. SRP_{mp} is observed to effectively control t_{OH} at higher number of nodes but when the number of nodes is increased above 96, t_{OH} increases as each partition size in SRP_{mp} increases proportionally, making the solution space of $ATRSRA_{org}$ more complex. The results for dynamic policies for t_{OH} are similar to the results of dynamic policies for t_{OH-on} that have already been discussed earlier in this section.

5.2.4 Impact of Overhead of Running the RA Algorithm

The CPU speed of the node at which the RA algorithm is run has a direct effect on t_{OH} . One way of decreasing the algorithm-running-overhead, t_{OH} , is to run the resource allocation algorithm on a more powerful computer. In this section, the relative performance of the different TRPS policies is analyzed as the CPU speed of the node running the resources allocation algorithm is changed. Note that if the computing power of the node running the resource allocation algorithm is increased, t_{OH} decreases linearly. The default values of the algorithm-running-overhead is computed by running the RA algorithm on a computer with an Intel Dual Core Pentium 2.0 GHz

processor, 2GB of RAM, running Microsoft XP Professional SP2. The configuration associated with this system has been represented by a label 1X. To quantify the effect of increasing the computing power of the node which is running the RA algorithm on the overall system performance, the following convention is used. Computer configurations are labelled by their computing speed relative to the configuration labelled 1X. A computer configuration is labelled AX if it results in a t_{OH} that is $1/A$ of the t_{OH} achieved with the configuration labelled 1X. Thus, configuration labelled 2X results in halving the t_{OH} achieved with the configuration labelled 1X.

Fig. 5.7 and Fig. 5.8 show the impact of increasing the number of nodes in Δ on $t_{ms-total}$ for 2X and 4X configurations. The $t_{ms-total}$ incurred using the default configuration, 1X, is shown in Fig. 5.9.

Note that $t_{ms-total}$ is the sum of t_{ms-WOH} and t_{OH} . By increasing the computer speed of the node responsible for running the RA algorithm, t_{ms-WOH} remains unchanged but t_{OH} is decreased. Intuitively, it can be said that as the RA algorithm is run on a faster computer, the TRPS policies that generate large t_{OH} will perform better.

It can be observed from Fig. 5.7, Fig. 5.8 and Fig. 5.9 that generally, as the resource-allocation-algorithm is run on a faster computer, amount of t_{OH} decreases and the $t_{ms-total}$ improves for all the policies.

5.2.4.1 Effect of Increasing CPU Speed on Backfilling

Fig. 5.7, Fig. 5.8 and Fig. 5.9 show that $SRP_{sp}+BF$ gives the lowest amount of $t_{ms-total}$ among all TRPS policies for small number of nodes. It can be observed that in 1X configuration, $SRP_{sp}+BF$ generates the lowest amount of $t_{ms-total}$ among all TRPS policies if the number of nodes are less than 24. As the configuration is changed to 2X, $SRP_{sp}+BF$ gives the lowest value of $t_{ms-total}$ if the number of nodes are less than 32; whereas in 4X configuration $SRP_{sp}+BF$ gives the lowest value

Fig. 5.6: Impact of Increasing Number of Nodes on t_{OH} for Different TRPS Policies

of $t_{ms-total}$ if the number of nodes are less than 44. The increased number of nodes, for which $SRP_{sp}+BF$ is the best TRPS policy, shows that the increase in CPU speed has more pronounced effect on $SRP_{sp}+BF$ as compared to other TRPS policies. As discussed in Section 5.2.3, $SRP_{sp}+BF$ produces the highest amount of t_{OH} regardless of the system and workload parameters.

Fig. 5.7: Impact of Increasing n on $t_{ms-total}$ (with RA Algorithm running at 2X)

Fig. 5.8: Impact of Increasing n on $t_{ms-total}$ (with RA Algorithm running at 4X)

Fig. 5.9: Impact of Increasing n on $t_{ms-total}$ (with RA Algorithm running at 1X)

Note that $t_{ms-total}$ is the sum of t_{ms-WOH} and t_{OH} . As the RA algorithm is run on a faster computer, amount of t_{OH} decreases. As $SRP_{sp}+BF$ has highest proportion of t_{OH} in $t_{ms-total}$ among all TRPS policies, $t_{ms-total}$ related to $SRP_{sp}+BF$ improves considerably as compared to the other TRPS policies.

Regardless of the computer speed when number of nodes becomes 120, $SRP_{sp}+BF$ always generates highest value of $t_{ms-total}$. This is because when number of nodes are high, the solution space for the RA algorithm used by $SRP_{sp}+BF$ in the mapping phase becomes complex. The combination of having a complex solution space at mapping phase, combined with t_{OH} incurred

when Reducing Resource-pool Algorithm is used, translates into highest value of $t_{ms-total}$ for $SRP_{sp}+BF$ among all TRPS policies. Although, as the computing speed is increased, $t_{ms-total}$ is observed to be reduced for $SRP_{sp}+BF$, but it remains the worst performing policy among all the TRPS policies at large number of nodes.

5.2.4.2 Effect of Increasing CPU Speed on Multiple Partitioning

For SRP_{mp} , in 1X configuration when the number of nodes is 128, SRP_{mp} has the lowest $t_{ms-total}$ among all the TRPS policies (see Fig. 5.9). Note that in SRP_{mp} , multiple partitioning is added to SRP_{sp} , in order to control t_{OH} . With a higher speed computer running the RA algorithm, as t_{OH} becomes less significant in $t_{ms-total}$, the benefit obtained from reducing the t_{OH} by using multiple partitions in SRP_{mp} is also reduced and SRP_{mp} is no more the best performing TRPS policy. The relative performance of SRP_{mp} in comparison to other TRPS policies drops further when the 4X configuration is used.

5.3 Achieving a Balance between Makespan and Total cost

Experimental results presented in Section 5.1 and Section 5.2, indicate that minimizing makespan and total cost are somewhat contradictory performance objectives. The contradictory nature of these two performance objectives is best highlighted by the performance of SRP_{sp} . When SRP_{sp} is used at TRPS, the lowest value of t_{cost} is incurred for all system and workload parameters. At the same time, the use of SRP_{sp} also generates one of the highest values of t_{ms-WOH} . Also, the use of SRP_{sp} at TRPS generates one of the highest algorithm-running-overhead among various TRPS policies, resulting in high values of $t_{ms-total}$. All other TRPS policies,

employ different strategies to reduce resource contention to decrease t_{ms-WOH} and to limit the algorithm-running-overhead. The decrease in t_{ms-WOH} and the algorithm-running-overhead result in the reduction of $t_{ms-total}$, at the expense of increased t_{cost} . This highlights the inherent trade-off among various performance metrics produced by the TRPS policies. In addition to focusing on each performance metric individually, it is also important to study the various trade-offs in the performance metrics achieved by the different TRPS policies. In utility Grid computing or Cloud Computing, t_{cost} is the total dollar cost incurred to process a given bag-of-tasks and may be related to the resource provider's costs. $t_{ms-total}$ is the total end-to-end time taken by the Grid to process the given bag-of-tasks and is related to the end user's experience. For some applications, the objective may be to strike a good balance between these two performance metrics that translates to providing a good end-user's experience at a reasonable cost. To study the degree to which each of the TRPS policy addresses this inherent trade-off, it is important to quantify the combined objective of minimizing the makespan and the total cost by some new performance metric. Two new performance metrics have been defined to quantify the combined objective.

Cost-Makespan-total (cost-mstotal): The product of the total cost spent in executing all the tasks in the given bag-of-tasks (t_{cost}) and the time required for completing all the tasks in the given bag-of-tasks ($t_{ms-total}$).

Cost-Makespan-without-overhead(cost-ms-WOH): The product of the total cost spent in executing all the tasks in the given bag-of-tasks (t_{cost}) and the time required for completing all the tasks in the given bag-of-tasks excluding the algorithm-running-overhead, (t_{ms-WOH}).

As cost-ms-total is the product of t_{cost} and $t_{\text{ms-total}}$, it quantifies the tradeoff between t_{cost} and $t_{\text{ms-total}}$ achieved by a particular allocation-plan. Similarly, cost-ms-WOH is indicative of the balance achieved between t_{cost} and $t_{\text{ms-WOH}}$ achieved by a given allocation-plan. Fig. 5.10 and Fig. 5.11 present cost-ms-WOH and cost-ms-total for each of the TRPS policies as a function of the number of nodes in the Grid.

It can be observed from Fig. 5.10 that cost-ms-WOH decreases as the number of nodes is increased. This is due to the fact that as the number of nodes is increased, the RA algorithm is able to allocate faster resources to each of the tasks in the bag-of-tasks. Fig. 5.11 shows the results of cost-ms-total as the number of nodes in the Grid is increased. It can be observed that for SRP_{sp} , cost-ms-total first decreases as the number of nodes are increased. This decrease is attributed to the fact that increased number of nodes provides more choices for the RA algorithm to choose resources, thus decreasing t_{cost} . It also reduces resource-contention and thus decreases $t_{\text{ms-total}}$. As cost-ms-total is a product of t_{cost} and $t_{\text{ms-total}}$, the reduction of both t_{cost} and $t_{\text{ms-total}}$ decreases cost-ms-total , as the number of nodes is increased. But, as number of nodes is increased above 64, the high t_{OH} generated by the $\text{ATSRA}_{\text{org}}$ running on the single partition consisting of all available nodes, weighs in. This significantly increases $t_{\text{ms-total}}$ which, in turn, produces an increase in cost-ms-total as the number of nodes is increased further above 64. SRP_{mp} adds multiple partitioning to SRP_{sp} and effectively reduces t_{OH} for large number of nodes, thus resulting in the lowest value of cost-ms-total among all the policies at higher number of nodes.

Fig. 5.10 Effect of Increasing No. of Nodes on cost-ms-WOH for Different TRPS Policies

Fig. 5.11: Effect of Increasing No. of Nodes on cost-mstotal for Different TRPS Policies

SRP_{sp}+BF achieves a good balance between t_{cost} and $t_{\text{ms-total}}$ when the number of nodes are below 32, producing the lowest value of cost-mstotal. Note that SRP_{sp}+BF generates the highest t_{OH} regardless of any system and workload parameters. As the number of nodes are increased beyond 32, the high values of t_{OH} generated by SRP_{sp}+BF due to an increasingly complex solution space drive cost-mstotal high. When the number of nodes is above 88, SRP_{sp}+BF gives the worst balance between t_{cost} and $t_{\text{ms-total}}$ producing the highest amount of cost-mstotal.

For DRP_{sp}, cost-mstotal is high for a large number of nodes due to the high t_{OH} generated. High t_{OH} is generated for DRP_{sp} for a large number of nodes due to the complex solution space provided to the Reducing Resource-pool Algorithm used by DRP_{sp}. When multiple partitioning is used in DRP_{mp}, t_{OH} is effectively reduced and the value of cost-mstotal is lowered indicating a better balance between t_{cost} and $t_{\text{ms-total}}$.

5.4 Performance of Various TRPS Policies under Different Workload Conditions

The BiLeG architecture is designed in a way that it can process any bag-of-tasks as long as each constituent task satisfies the three characteristics of a PBDT task (see Section 3.1.2). The flexible design of BiLeG architecture allows it to be used for various different types of applications, which typically generate workloads which are quite different in characteristics from one another. In order to analyze the performance of various TRPS policies as the workload characteristics change, key performance metrics are measured as the workload parameters are varied. This

section analyzes the performance of various TRPS policies for different values of mean and variance of the lengths of the raw data files of the constituent PBDT tasks.

To investigate the performance of the TRPS policies for different type of resource-intensive applications two performance metrics, t_{cost} and $t_{\text{ms-WOH}}$ are selected. Note that $t_{\text{ms-total}}$ is not selected because changing the mean and variance of the lengths of the raw data file of the constituent tasks has no effect on the algorithm-running-overhead, t_{OH} .

5.4.1 Effect of Increasing the Variance of the Lengths of Raw Data Files of the Constituent Tasks On Performance

This section discusses the effect of increasing variance of the lengths of the raw data files of the constituent tasks on the performance of the various TRPS policies. The two performance metrics, $t_{\text{ms-WOH}}$ and t_{cost} are discussed in separate sub-sections.

5.4.1.1 The Makespan-without-overhead

Fig. 5.12 shows the impact of increasing the variance of the lengths of the raw data files of tasks on $t_{\text{ms-WOH}}$ for various TRPS policies. In general, as the variance of the lengths of the raw data files of the constituent tasks is increased, $t_{\text{ms-WOH}}$ increases for all TRPS policies. The reasons for the increase of $t_{\text{ms-WOH}}$ for static and dynamic policies are quite different. For the dynamic TRPS policies the increase in $t_{\text{ms-WOH}}$ tends to be more pronounced (see Fig. 5.12). The reason for the increase in $t_{\text{ms-WOH}}$ for the dynamic TRPS policies is based on fact that they give rise to a reducing resource-pool explained in detail in Chapter 3 and the resource-pool allocation depends on the position of a particular task in the given bag-of-tasks. Generally, the tasks which are

earlier in the queue are allocated a larger resource-pool. If the variance of the lengths of the raw data files of the constituent tasks is zero, all tasks have equal raw data files and the position of a particular task in the bag-of-task does not matter. As the variance of the lengths of the raw data files of the constituent tasks is increased, the difference among the lengths of the raw data files also increases. When dynamic TRPS policies are used when the variance of the lengths of the raw data files of the constituent tasks is high, it frequently leads to a situation where a task having a small raw data file is allocated a larger resource-pool because it is positioned earlier in the bag-of-task as compared to a task having larger raw data file which is located later in the given bag-of-tasks. By allocating a larger resource-pool to a smaller task, more efficient resources are allocated to tasks having a smaller raw data file, as compared to the tasks having larger raw data files. This “unfair” resource allocation deteriorates the overall resource utilization and leads to higher t_{ms-WOH} . Also, it can be observed from Fig. 5.12 that t_{ms-WOH} for DRP_{sp} increases more sharply as compared DRP_{mp} . DRP_{mp} uses multiple partitioning which limits the maximum size of a resource-pool of a task. For example, if a task having a small raw data file (relative to other tasks) is the first in the bag-of-tasks, the maximum size of resource-pool allocated to it is equal to the size of the partition, and not the complete set of available Grid resources, $|\Delta|$ (which was in the case of DRP_{sp}). Thus, multiple partitioning reduces the degree of advantage that the tasks earlier in the bag-of-tasks can have, resulting in a better distribution of resources, which, in turn, reduces, t_{ms-WOH} as observed in Fig. 5.12. When proportional partitioning is used in DRP_{MP-pro} , it further decreases the value of t_{ms-WOH} . It may be observed that the benefits obtained by proportional partitioning becomes more and more pronounced as

Fig. 5.12: Effect of Increase in Variance of the Lengths of Raw Data Files of the Constituent Tasks on t_{ms-WOH}

the variance of the lengths of the raw data files of the constituent tasks is increased (see Fig. 5.12).

SRP_{sp}+BF uses backfilling for dynamic resource allocation. The process of backfilling allocates resources for the task which is waiting in the queue due to resource contention and has the smallest raw data file, as discussed in Section 3.5.3. Backfilling reduces t_{ms-WOH} by decreasing the amount of waiting time each resource has to experience due to resource-contention. With backfilling, by choosing the task with the smallest raw data file waiting in the queue for resource-pool allocation, the tasks having large raw data files still use the original resource-pool allocated to them during the mapping phase. The on-the-fly resource-pool allocation performed

by backfilling is done for tasks having smaller raw data files. This reduces resource-contention but also uses more expensive resources. An increase in the variance of the lengths of the raw data files of the constituent tasks increases the number of tasks with smaller raw data files. Then tasks with the smaller raw data files can be allocated resources using backfilling which effectively reduces resource-contention and decreases t_{ms-WOH} .

5.4.1.2 Total Cost

Fig. 5.13 shows the effect of increasing the variance of the lengths of the raw data files of the constituent tasks on t_{cost} . It can be observed from Fig. 5.13 that SRP_{sp} has the lowest t_{cost} associated with it, regardless of the variance of the lengths of the raw data files of the constituent tasks. As discussed in Section 3.5.1, SRP_{sp} assigns the largest possible resource-pool to each of the task and t_{cost} associated with SRP_{sp} is always lowest regardless of any system or workload parameters. Another observation to note is that $SRP_{sp}+BF$ is the only policy for which t_{cost} tends to drop as the variance of the lengths of the raw data files of the constituent tasks is increased. In backfilling, the process of resource-allocation starts with the task having the smallest raw data file being selected from the set of tasks waiting for the assigned set of resources to become free. An increase in the variance of the lengths of the raw data files of the constituent tasks results in some very large files and some very small files. In $SRP_{sp}+BF$, the increased number of waiting tasks with smaller raw data files results in the choice of the resources for tasks with smaller raw data files through backfilling. At the same time, the tasks with larger raw data files are handled by static resource allocation components of the TRPS policy. As the variance of the lengths of

the raw data files of the constituent tasks is increased, resources are better utilized when $SRP_{sp}+BF$ is used and t_{cost} is observed to have a slight decreasing trend (see Fig. 5.13).

Note that for the three dynamic TRPS policies used, t_{cost} increases as the variance of the lengths of the raw data files of the constituent tasks is increased. Dynamic TRPS policies are based on the reducing resource-pool algorithm in which the size of the resource-pool allocated to a particular task depends on the position of the task in the given bag-of-tasks. When the variance of the lengths of the raw data files of the constituent tasks is high, a particular task having a large raw data file may be allocated a small resource-pool, due to its position in the bag-of-tasks. Due to the limited resources available in a small resource-pool, one of the partitions of the task having a large raw data file may have to be allocated to a computing or communication resource having high t_{cost} . This results in the increase in t_{cost} as the variance of the lengths of the raw data files of the constituent tasks is increased as observed from the plots of DRP_{sp} , DRP_{mp} and DRP_{mp-pro} in Fig. 5.13.

5.4.2 Effect of Increasing the Mean Length of the Raw data Files of the Constituent Tasks On the Performance

As described in Chapter 2, PBDT tasks are responsible for transferring very large amounts of data that need to be processed before it can be used at a set of sink nodes. In this section, the effect of increase in the average length of the raw data files for the PBDT tasks on t_{ms-WOH} and t_{cost} is analyzed for different TRPS policies. Note that the variance of the lengths of raw data files of the tasks in the bag-of-tasks is kept constant while performing this set of experiments.

Fig. 5.13: Effect of Increase in Variance of the Sizes of Raw Data Files of the Constituent Tasks on t_{cost}

5.4.2.1 Total Cost

The effect of increasing the mean length of the raw data files of the constituent tasks on t_{cost} is captured in Fig. 5.14. Intuitively, t_{cost} is expected to increase as the average length of the raw data files of the PBDT tasks increases. This is because this increase leads to an increase in the overall workload for the tasks which directly increases the overall usage of the resources. Recall that t_{cost} is the total cost spent in executing all the tasks in the given bag-of-tasks and directly depends on the total usage of the resources. Thus, as the length of the raw data file of the constituent tasks is increased, t_{cost} increases steadily for all TRPS policies (see Fig. 5.14).

It can be observed from Fig. 5.14 that the lowest value of t_{cost} for any mean length of the raw data files of the constituent tasks is associated with SRP_{sp} . Note that when SRP_{sp} is used at TRPS, the resource-pool for each task in the given bag-of-tasks is the complete set of the available nodes. Thus, the $\text{ATSRA}_{\text{org}}$ algorithm used at RA can choose the best of the resources for each of the tasks which results in generating the lowest value of t_{cost} as compared to other TRPS policies.

Among three dynamic TRPS policies, t_{cost} for DRP_{sp} is the lowest as there is no partitioning in Δ . When partitioning is used in DRP_{mp} , t_{cost} increases as compared to t_{cost} associated with DRP_{sp} as the resource-pool of the first task is chosen from one of the partitions only, which is a subset of Δ . t_{cost} improves when instead of the equal partitioning, proportional partitioning is used, resulting in a better utilization of the available resources.

5.4.2.2 Makespan-without-overhead

The effect as the mean of the length of the raw data files of the tasks on $t_{\text{ms-WOH}}$ is captured in Fig. 5.15. Note that $t_{\text{ms-WOH}}$ is the time required for completing all the tasks in the given bag-of-tasks, excluding the algorithm-running-overhead, t_{OH} . Since larger raw data files take longer to process, $t_{\text{ms-WOH}}$ is expected to increase as the average length of the raw data file of each of the PBDT task increases. As shown in Fig. 5.15, for each TRPS policy, $t_{\text{ms-WOH}}$ increases steadily with an increase in the mean length of the raw data files of the constituent tasks.

Fig. 5.14: Effect of Increase of Mean Length of Raw Data Files of the Constituent Tasks on t_{cost}

Fig. 5.15: Effect of Increase of Mean Length of Raw Data File of the Constituent Tasks on t_{ms-WOH}

It can be observed that SRP_{sp} has the highest t_{ms-WOH} for a given mean length of raw data files of the constituent tasks (see Fig. 5.15). The reason for this highest value of t_{ms-WOH} lies in the way resources are allocated in SRP_{sp} , which introduces considerable resource contention. Resource contention occurs when in the execution phase, multiple tasks want to use the same set of resources resulting in overlapping resource demands. When dynamic allocation of unused resources is added through backfilling in $SRP_{sp} + BF$, performance is improved drastically. For other dynamic policies, DRP_{sp} performs well for smaller lengths of raw data files. But, when the length of the files is increased over 1500MB, the performance is observed to deteriorate sharply. This happens when the unfair advantage that DRP_{sp} gives to the tasks that are handled earlier by the TRPS begins to deteriorate the performance of larger raw data files. The performance improves when this deficiency is minimized through the creation of multiple partitions in DRP_{mp} (see Fig. 5.15). The results are further improved when the size of each of the partitions is chosen proportional to the length of raw data files of the constituent tasks for DRP_{mp-pro} .

5.5 Discussion

This chapter presented a detailed performance analysis of the different TRPS policies. The experimental results presented in this chapter show that the choice of an appropriate TRPS policy depends on three factors that are discussed next.

- 1) **The optimization objective;** i.e. the performance metric to be minimized, $t_{ms-toal}$, t_{ms-WOH} or t_{cost}

2) The workload parameters; i.e. the mean and variance of the lengths of the raw data files of the tasks.

3) The system parameters; i.e. the number of nodes in the Grid

In this research, SRP_{sp} produces the lowest value of t_{cost} . The experimental results confirm that if the objective is to minimize t_{cost} then SRP_{sp} is always the best choice regardless of any system or workload parameters. On the other hand, SRP_{sp} produces one of the highest values of t_{ms-WOH} regardless of the system or workload parameters. SRP_{sp} also generates higher values of $t_{ms-total}$ in comparison to other allocation-plans.

Analysis of the effects of change in various workload parameters on the performance of TRPS policies reveals that variance of the lengths of the raw data files of the constituent tasks is one of the important factors in determining the suitability of a TRPS policy. When the variance of the lengths of the raw data files of the constituent tasks is high, and the objective is to reduce t_{ms-WOH} , using backfilling seems to be an effective strategy. Thus, for high variance workload with an objective of reducing makespan, $SRP_{sp}+BF$ is a good choice.

The process of the choosing an appropriate TRPS policy for various system and workload parameters is be summarized next.

For Large Grids: (Having more than 64 nodes): Multiple partitioning used in static TRPS policies reduces the solution space, giving good results for all performance metrics. Thus, SRP_{mp} is recommended for large Grids.

For Medium Grids: (Having 32 to 64 nodes): DRP_{sp} has the lowest value of $t_{\text{ms-total}}$ and is recommended to be used.

For Small Grids: (Having less than 32 nodes): $\text{DRP}_{\text{mp-pro}}$ is recommended because it is observed to have a good performance for all performance metrics.

Chapter 6 Performance Analysis of the Allocation-plans

6.1 Introduction

Each allocation-plan in the BiLeG architecture represents a particular technique to process a given bag of PBDT task. As observed in Chapter 4 and Chapter 5, different workload and system parameters require different resource allocation techniques to meet a particular performance objective. Before analyzing the performance of the allocation-plans, this thesis presented the performance analysis for each of the two decision making levels of the BiLeG architecture individually (see Chapter 4 and Chapter 5). Analyzing each of the decision making levels in isolation provides an opportunity to study the performance of the algorithms used at a particular decision making level only, before studying the combined effect of different policies and algorithms deployed at both the upper and the lower decision making levels. Following this approach, Chapter 4 has focused on the lower decision making module, RA, and the performance of the RA algorithms was analyzed while a specific TRPS policy was used. In Chapter 5, the focus was changed to the upper decision making module, TRPS, and its performance was investigated by keeping the RA algorithm unchanged for all sets of experiments. The combined effect of using a particular TRPS policy and a specific RA algorithm is presented in this chapter.

First, in this chapter a lower bound for the total cost is calculated. Second, the performance of various allocation-plans is investigated by analyzing system performance under different system and workload parameters. For performance investigation, different allocation-plans have been

grouped according to the resource-allocation strategies they employ. Finally, conclusions drawn from the experimental results are presented.

6.2 Computation of the Lower Bound on t_{cost}

The resource allocation, considered in this research, is a known NP-complete problem. Thus, finding an optimal solution for a given combination of system and workload parameters is very compute-intensive. Instead of comparing the results achieved by using different allocation-plans with the results achieved by an optimal solution, the approach taken in this chapter is to calculate a lower-bound for t_{cost} by using an analytic method and compare t_{cost} achieved by using various allocation-plans with this lower bound, $t_{\text{cost}_{\text{LB}}}$. Note the objective function of the RA algorithm proposed in this research is to minimize t_{cost} incurred while processing the given bag-of-tasks and thus, a lower bound on t_{cost} is calculated.

The approach used in the calculation of the lower bound is to first identify the minimum unit processing and communication times in each of the unit computing and communication time metrics defined in Chapter 3, and assume that all unit processing and communication times are equal to the minimum time. It generates a lower bound, because the lowest possible value of time is used for all computing and communication jobs of each of the tasks.

For calculating the lower bound, let

$$d^-(n_{\text{src}}, \kappa_s) : \min [C_{\text{src}-\kappa_s}]$$

$$d^-(\Delta, \kappa_s) : \min [C_{\Delta-\kappa_s}]$$

$$d^-(n_{src}, \Delta) : \min [C_{c_{src-\Delta}}]$$

$$C_{p\Delta}^- : \min [C_{p\Delta}]$$

$$C_{p\kappa_s}^- : \min [C_{p\kappa_s}]$$

where $[C_{c_{src-\kappa_s}}]$, $[C_{c_{src-\Delta}}]$, $[C_{c_{\Delta-\kappa_s}}]$, $[C_{p\Delta}]$ and $[C_{p\kappa_s}]$ contains unit communication times metrics and correspond to the vector representing unit communication times from source to κ_s , the vector representing unit communication times source to Δ , the vector representing unit communication times from Δ to κ_s , the vector representing unit processing-times at Δ and the vector representing unit processing-times at κ_s respectively.

Recall that for PBDT tasks, the raw data file needs to be processed before it can be used at a set of sink nodes. The data processing facilities are at the following three locations:

- At the source node, n_{src}
- At each of the sink nodes in κ_s
- At each of the nodes in the Grid

Corresponding to each of the choices for processing the data, a particular PBDT task can be processed in one of the following three ways:

- 1) In the first way, the unpartitioned raw data file is processed at n_{src} and sent to the set of sink nodes. All or some of the nodes in κ_s can use this way to process the given PBDT task. If k_1 nodes from the set of sink nodes κ_s use this way for the processing the given PBDT task, the lower bound for t_{cost} for a particular task q , represented by $t_{cost1LBq}$, is computed as:

$$t_{cost1LBq} = L_q \{ k_1 \varepsilon_q \alpha_{src-sink} d^-(n_{src}, \kappa_s) + \beta C_{p_{src}} \} \quad (6.1)$$

Note that the first term within parenthesis on the right hand side is the cost for transferring one unit of processed data from the source node to k_1 nodes. The second term within parenthesis on the right hand side is the unit cost of processing data at the source node. By multiplying the summation of these two terms with L_q , which is the length of raw data file of the task q , the total cost for the task q is obtained.

Reorganizing the terms on the right hand side of Eq. 6.1 yields:

$$t_{cost1LBq} = L_q \beta C_{p_{src}} + [L_q \varepsilon_q \alpha_{src-sink} d^-(n_{src}, \kappa_s)] k_1 \quad (6.2)$$

$$\text{Let } \Phi_1 = L_q \beta C_{p_{src}} \quad (6.3)$$

$$\text{and } \Psi_1 = L_q \varepsilon_q \alpha_{src-sink} d^-(n_{src}, \kappa_s) \quad (6.4)$$

By using Eq. 6.2, Eq. 6.3 and Eq. 6.4, we get:

$$t_{cost1LBq} = \Phi_1 + \Psi_1 k_1 \quad (6.5)$$

- 2) The second way is to send the unprocessed and unpartitioned raw data file to each of the nodes in κ_s where it is processed. If k_2 nodes from the set of sink nodes κ_s use this way for the processing of the given PBDT task, the lower bound for t_{cost} associated this choice for a particular task q , represented by $t_{cost2LBq}$, is calculated as:

$$t_{cost2LBq} = L_q \{ k_2 \alpha_{src-sink} d^-(n_{src}, \kappa_s) + \beta k_2 C_{p_{\kappa_s}}^- \} \quad (6.6)$$

The first term within parenthesis on the right hand side is the cost for transferring one unit of data from the source node to k_2 number of the sink nodes. The second term within parenthesis on the right hand side is total unit cost of processing data at k_2 node.

Reorganizing the terms on the right hand side of the Eq. 6.6 yields:

$$t_{cost2LBq} = [L_q \{ \alpha_{src-sink} d^-(n_{src}, \kappa_s) + \beta C_{p_{\kappa_s}}^- \}] k_2 \quad (6.7)$$

$$\text{Let } \Psi_2 = L_q \alpha_{src-sink} d^-(n_{src}, \kappa_s) + \beta C_{p_{\kappa_s}}^- \quad (6.8)$$

By using Eq. 6.7 and Eq. 6.8, we get:

$$t_{cost2LBq} = \Psi_2 k_2 \quad (6.9)$$

- 3) The third way is to process the data at the Grid nodes. In this case, the raw data file is partitioned and sent to Δ where it is processed. From Δ , the processed partitions are transferred to κ_s . If k_3 nodes from the set of sink nodes κ_s use this way for the processing of the given PBDT task, the lower bound for t_{cost} of a particular task q represented by $t_{cost3LBq}$ is computed as:

$$t_{cost3LBq} = \frac{L_q}{\rho_q} [\rho_q \beta C_{p_{\Delta}}^- + \rho_q \alpha_{in} d^-(n_{src}, \Delta) + (\rho_q k_3) \varepsilon_q \alpha_{out} d^-(\Delta, \kappa_s)] \quad (6.10)$$

The first term within parenthesis on the right hand side is the unit cost of processing ρ_q partitions at the Grid. As discussed in Section 3.1.3, $PBDT_{\text{fixed}}$ tasks are considered in this research. Thus, a task T_q consists of ρ_q number of equal-sized partitions. Recall that, as discussed in Chapter 3, only one of the partitions can be processed by a Grid node at a time. The second term within parenthesis on the right hand side is the unit cost of transferring data from source node to ρ_q nodes in the Grid. The third term within parenthesis on the right hand side is the unit cost of transferring ρ_q processed partitions to k_3 number of sink nodes. Note that the size of each partition is by L_q/ρ_q . By multiplying these three terms with the size of each partition, total cost is obtained.

By reorganizing the terms on the right hand side of Eq. 6.10, we get,

$$t_{\text{cost3LB}_q} = L_q [\{ \beta C_{p_\Delta}^- + \alpha_{in} d^-(n_{src}, \Delta) \} + \varepsilon_q \alpha_{out} k_3 d^-(\Delta, \kappa_s)] \quad (6.11)$$

$$= [L_q \beta C_{p_\Delta}^- + L_q \alpha_{in} d^-(n_{src}, \Delta)] + [L_q \varepsilon_q \alpha_{out} d^-(\Delta, \kappa_s)] k_3 \quad (6.12)$$

$$\text{Let, } \Phi_3 = L_q \beta C_{p_\Delta}^- + L_q \alpha_{in} d^-(n_{src}, \Delta) \quad (6.13)$$

$$\Psi_3 = L_q \varepsilon_q \alpha_{out} d^-(\Delta, \kappa_s) \quad (6.14)$$

By using Eq. 6.12, Eq. 6.13 and Eq. 6.14, we get,

$$t_{\text{cost3LB}_q} = \Phi_3 + \Psi_3 k_3 \quad (6.15)$$

The lower bound for t_{cost} is calculated by finding the values of k_1 , k_2 and k_3 that produce the lowest t_{cost} . The objective of finding the lower bound can be formulated as following:

$$\text{minimize } t_{\text{cost}_{\text{LBq}}} = (j_1\Phi_1 + \Psi_1k_1) + (\Psi_2k_2) + (j_3\Phi_3 + \Psi_3k_3) \quad (6.16)$$

where j_1 and j_3 are binary variable.

Note that this equation assumes that out of the κ_s nodes, k_1 nodes is using the first way of processing the given PBDT task, k_2 nodes are using the second way and k_3 nodes are using the third way. It finds the values of k_1 , k_2 and k_3 that generates the minimum value of $t_{\text{cost}_{\text{LBq}}}$, after observing the following constraints:

Constraints:

$$k_1 + k_2 + k_3 = |\kappa_s| \quad (6.17)$$

$$j_1 = \text{ceiling}(k_1/|\kappa_s|) \quad (6.18)$$

$$j_3 = \text{ceiling}(k_3/|\kappa_s|) \quad (6.19)$$

$$k_1, k_2, k_3 \geq 0 \quad (6.20)$$

$$j_1, j_3 \in \{0, 1\} \quad (6.21)$$

The first constraint (Eq. 6.17) ensures that the sum of sink nodes used in different choices is equal to the total number of sink nodes in the system. The second constraint (Eq. 6.18) specifies that the value of binary variable j_1 should be 1 only if at least one of the sink nodes is using choice 1. Similarly, the third constraint (Eq. 6.19) specifies that the value of the binary variable j_3 should be non-zero only if at least one of the sink nodes is using choice 3. The fourth constraint

(Eq. 6.20) makes sure that the value of k_1, k_2 and k_3 are positive. The fifth constraint (Eq. 6.21) defines j_1 and j_3 as binary variables.

$t_{\text{cost}_{LBq}}$ defines the lower bound on cost for the task q in the given bag-of-tasks. The lower bound on associated with processing all the tasks in the given bag-of-tasks is thus given by:

$$t_{\text{cost}_{LB}} = \sum_{i=1}^{|T|} t_{\text{cost}_{LB_i}} \quad (6.22)$$

6.3 Experimental Setup

The simulation setup for this set of experiments is similar to the one used for investigating the results for the RA algorithms in Chapter 4 and has been explained in Section 4.1 in detail. The effect of each of the parameters on performance is analyzed by using a-factor-at-a-time approach. Only one of the parameters is varied while keeping all others at their default values listed in Table 4.1.

6.4 Experimental Results

The TRPS policies and the RA algorithms both have a significant impact on the performance of the BiLeG architecture. TRPS policies were discussed in Chapter 5, whereas the performance of RA algorithms was discussed in Chapter 4. Various performance improvement strategies have been devised, each corresponding to one or more combinations of TRPS policies and RA algorithms referred to as allocation-plans. Each strategy is devised to minimize a particular performance metric. Unfortunately, when a particular strategy is used to minimize one of the performance metrics, it can negatively affect other performance metrics as explained in Chapter

4 and Chapter 5. In this section, in order to quantify the usefulness of a particular strategy, the allocation-plans sharing the same strategy are grouped together in a set and are analyzed under different workload and system parameters.

In this section, first, the performance of $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$, which always generates the lowest t_{cost} regardless of any system and workload parameters, is analyzed. $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$ can be considered as the base allocation-plan and all other allocation-plans can be considered as the enhancements to $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$ using various strategies in an effort to improve $t_{\text{ms-total}}$ at the cost of increased t_{cost} , as discussed in Section 6.4.2.

The performance of the base-allocation-plan, $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$ is discussed in Section 6.4.1. Different strategies for performance improvement are presented next. Discussion on each strategy and its performance analysis are presented in a separate subsection.

6.4.1 Minimizing t_{cost}

The value of the lower bound on t_{cost} , t_{costLB} , is calculated by using Eq. 6.22 and is plotted in Fig. 6.1. Note that among all allocation-plans, $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$ achieves the lowest value of t_{cost} , as discussed in Section 5.1.2. In $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$, SRP_{sp} is used at TRPS and $\text{ATSRA}_{\text{org}}$ is used at RA. When SRP_{sp} is used at TRPS, for each task the resource-pool consists of the complete set of all the nodes available. The use of the $\text{ATSRA}_{\text{org}}$ algorithm without any constraints relaxation combined with the choice of the largest possible size of the resource-pool for each of the tasks leads to the smallest t_{cost} among all allocation-plans. t_{cost} achieved by using

Fig. 6.1: Comparison of $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$ with the Lower Bound

$\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$ is, therefore, closest to the calculated lower bound and has been chosen to plot against the calculated lower-bound in Fig. 6.1.

It should be noted from Fig. 6.1 that when the number of nodes are small, the difference between the t_{cost} associated with $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$ and the lower bound on t_{cost} is large. The reason is that when the number of nodes is small, the size of the resource-pool allocated to each of the tasks in the given bag-of-tasks is smaller than the total number of partitions. Thus, a smaller number of choices are available for the RA algorithm for resource-selection. As the number of nodes is increased, the size of resource-pool allocated to each of the tasks in the given bag-of-

tasks is also increased, which, in turn, increases the choice of resources available for the RA algorithm. It can be observed that as the number of nodes is increased, there is a significant reduction in the difference between t_{cost} associated with $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$ and the lower-bound on t_{cost} .

Note that the method used for computing the lower bound on t_{cost} leads to a looser bound at lower number of nodes. This is because when the lower bound is calculated it is assumed that all computing and communication costs in the cost matrices are equal to the lowest computing and communication costs respectively. When $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$ is used, the choice of resources is limited for small number of nodes and therefore the RA algorithm is forced to choose resources that have high costs. However, if the number of resources becomes large, the RA algorithm may have a better possibility of finding the resources with costs which are closer to the best costs used in the calculation of the lower bound of t_{cost} .

6.4.2 Techniques to Reduce $t_{\text{ms-total}}$

While $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$ produces the lowest t_{cost} , it also has one of the highest values of $t_{\text{ms-total}}$ (see Fig. 6.3). As discussed in Chapter 3 and Chapter 4, the high value of $t_{\text{ms-total}}$ for $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$ is due to two reasons, resource-contention and the high value of algorithm-running-overhead.

In an effort to improve $t_{\text{ms-total}}$ produced by $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$, different strategies are used at both upper and lower decision making levels, resulting in new allocation-plans. These strategies work at RA and/or TRPS and improve $t_{\text{ms-total}}$ at the cost of increased t_{cost} . For performance

analysis of the allocation-plans, the trade-off between t_{cost} and $t_{\text{ms-total}}$ needs to be studied carefully.

Table 6.1 summarizes the approach used to study the impact of various strategies which are used to reduce $t_{\text{ms-total}}$. Note that in an effort to reduce $t_{\text{ms-total}}$, five strategies have been devised based on two objectives: to control algorithm-running-overhead and to control resource contention. Effectiveness of each of the strategies is analyzed by studying the performance of the particular allocation-plan that best personifies it (see Table 6.1). Note that the performance of a subset of all possible allocation-plans has been plotted and discussed in this chapter to avoid unnecessary clutter in the graphs and is to keep the discussion focussed. For reference, experimental results of all allocation-plans, that are relevant to the experiments described in this chapter, are presented in Appendix B.

Table 6.1: Strategies for the Performance Analysis of the Allocation-plans

	Objective	Strategies Used to Fulfil the Objective	Allocation-plan Used to in Analysis
A	To Control Algorithm-running-overhead	At TRPS: Multiple Partitioning At RA: Constraint-relaxation, Use of Low-complexity RA Algorithm	$\langle \text{SRP}_{\text{mp}}, \text{ATSRA}_{\text{org}} \rangle$, $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{bsr}} \rangle$, $\langle \text{DRP}_{\text{mp-pro}}, \text{RRJR} \rangle$
B	To Control Resource-contention	At TRPS: Backfilling, Use of Reducing Resource-pool Algorithm	$\langle \text{SRP}_{\text{sp}} + \text{BF}, \text{ATSRA}_{\text{org}} \rangle$, $\langle \text{DRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$

6.4.2.1 Objective A: To Control Algorithm-running-overhead

Three different strategies have been developed to decrease $t_{\text{ms-total}}$ by reducing algorithm-running-overhead, t_{OH} , which is the total time in seconds taken by the RA algorithm to perform

the resource allocation of all the tasks in the given bag-of-tasks. Recall that $t_{ms-total}$ is the sum of t_{OH} and t_{ms-WOH} .

Two of the strategies have been devised to be used at RA and the third has been devised to be used at TRPS. These strategies are summarized as:

At TRPS

- 1) **Multiple Partitioning:** Algorithm-running-overhead can be controlled at TRPS by reducing the solution space of the ATSRA algorithm by the division of Δ into partitions. It reduces t_{OH} , as by reducing the solution space for the LP algorithm into smaller subsets, the number of variables that the algorithm has to solve are considerably reduced. $\langle SRP_{mp}, ATSRA_{org} \rangle$ adds partitions to $\langle SRP_{sp}, ATSRA_{org} \rangle$ at TRPS and has been used in this section to analyze the performance of this strategy.

At RA:

- 1) **Constraint-relaxation:** Algorithm-running-overhead can be controlled at RA by relaxing the constraints of the ATSRA algorithm. $\langle SRP_{sp}, ATSRA_{bsr} \rangle$ adds constraints relaxation to $\langle SRP_{sp}, ATSRA_{org} \rangle$ and has been used in this section for the performance analysis of the constraints relaxation strategy.
- 2) **Use of Low Complexity RA Algorithm:** Another strategy used to control the algorithm-running-overhead is the use of a low complexity algorithm at RA which generates negligible algorithm-running-overhead. As discussed in Chapter 3, RRJR is a low-

complexity knowledge-free algorithm. $\langle \text{DRP}_{\text{mp-pro}}, \text{RRJR} \rangle$ has been used in this section for the performance analysis of this strategy. As described in Chapter 3, RRJR can only be used with dynamic TRPS policies, such as $\text{DRP}_{\text{mp-pro}}$ which allocates resources at runtime.

Fig. 6.2 compares the values of t_{OH} produced by the use of these three strategies with t_{OH} produced by $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$. It can be noted that for $\langle \text{DRP}_{\text{mp-pro}}, \text{RRJR} \rangle$, t_{OH} is negligibly small for all number of nodes. This is due to the fact that $\langle \text{DRP}_{\text{mp-pro}}, \text{RRJR} \rangle$ is a low-complexity algorithm and t_{OH} produced by it for all number of nodes can be neglected. It can be observed from Fig. 6.2 that for a small number of nodes, the solution space of $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$ is small. Due to the small solution space, a low overhead is produced by $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$. t_{OH} associated with $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$ algorithm rises sharply as the number of nodes in Δ is increased more than 32. The high value of t_{OH} at large number of nodes also highlights the importance of the effort of finding ways to reduce t_{OH} . Both $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{bst}} \rangle$ and $\langle \text{SRP}_{\text{mp}}, \text{ATSRA}_{\text{org}} \rangle$ are observed to effectively control t_{OH} at higher number of nodes. When the number of nodes is between 32 to 72, t_{OH} generated by $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{bst}} \rangle$ and $\langle \text{SRP}_{\text{mp}}, \text{ATSRA}_{\text{org}} \rangle$ is comparable. When the number of nodes is increased above 72, each partition size in $\langle \text{SRP}_{\text{mp}}, \text{ATSRA}_{\text{org}} \rangle$ increases proportionally, making the solution space of $\text{ATSRA}_{\text{org}}$ considerably more complex. Also, as the $\langle \text{SRP}_{\text{mp}}, \text{ATSRA}_{\text{org}} \rangle$ allocation-plan divides Δ into multiple partitions, the ATSRA algorithm has to run multiple times. Note that ATSRA algorithm runs only once in any of the single

Fig. 6.2: Effect of Increasing Number of Nodes on t_{OH}

Fig. 6.3: Effect of Increasing Number of Nodes on $t_{ms-total}$

partition based allocation-plans, such as $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{bsr}} \rangle$. The combination of these two factors makes $\langle \text{SRP}_{\text{mp}}, \text{ATSRA}_{\text{org}} \rangle$ less effective in reducing t_{OH} at higher number of nodes.

Fig. 6.3 compares the values of $t_{\text{ms-total}}$ produced by the use of these three approaches with the value of $t_{\text{ms-total}}$ produced by $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$. When the number of nodes is small (below 32), both $\text{ATSRA}_{\text{org}}$ based allocation-plans $\langle \text{SRP}_{\text{mp}}, \text{ATSRA}_{\text{org}} \rangle$ and $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$ effectively reduce $t_{\text{ms-total}}$ as compared to other allocation-plans. Note that t_{OH} is one of the components of $t_{\text{ms-total}}$. At smaller number of nodes, both for $\langle \text{SRP}_{\text{mp}}, \text{ATSRA}_{\text{org}} \rangle$ and $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$, the solution space for the $\text{ATSRA}_{\text{org}}$ is small and a small t_{OH} is produced. The benefit obtained by the precision in resource-allocation achieved by use of original ATSRA algorithm outweighs the additional overhead incurred, thus, effectively reducing $t_{\text{ms-total}}$.

It can be observed from Fig. 6.3 that as the number of nodes is increased (greater than 32), $t_{\text{ms-total}}$ associated with $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$ increases sharply. This is because for a large number of nodes, the high value of t_{OH} associated with $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$ offsets the benefit of using accurate resource allocation. It can be observed that although the algorithm-running-overhead associated with $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{bsr}} \rangle$ and $\langle \text{SRP}_{\text{mp}}, \text{ATSRA}_{\text{org}} \rangle$ also increase with an increase in the number of nodes, the rate of increase is less sharp in comparison to $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$. $\langle \text{SRP}_{\text{mp}}, \text{ATSRA}_{\text{org}} \rangle$ reduces t_{OH} by reducing the solution space for the RA algorithm by decreasing the size of the resource-pools and $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{bsr}} \rangle$ reduces t_{OH} at the expense of a decreased accuracy in the LP algorithm. It is interesting to note that although, as far as t_{OH} is concerned, $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{bsr}} \rangle$ outperforms $\langle \text{SRP}_{\text{mp}}, \text{ATSRA}_{\text{org}} \rangle$ for all number of nodes (see Fig. 6.2), at lower number of nodes (below 98) the value of $t_{\text{ms-total}}$ associated with

$\langle \text{SRP}_{\text{mp}}, \text{ATSRA}_{\text{org}} \rangle$ is lower than the value of $t_{\text{ms-total}}$ associated with $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{bsr}} \rangle$ (see Fig. 6.3). When the number of nodes is below 98, $\langle \text{SRP}_{\text{mp}}, \text{ATSRA}_{\text{org}} \rangle$ seems to be a good choice to decrease $t_{\text{ms-total}}$. On the other hand, when number of nodes is above 98, decreasing $t_{\text{ms-total}}$ by introducing constraints relaxation is observed to be more appropriate and $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{bsr}} \rangle$ gives the lowest value of $t_{\text{ms-total}}$.

$t_{\text{ms-total}}$ for the replication-based allocation-plan $\langle \text{DRP}_{\text{mp-pro}}, \text{RRJR} \rangle$ is quite high in comparison to the other allocation-plans. $t_{\text{ms-total}}$ for $\langle \text{DRP}_{\text{mp-pro}}, \text{RRJR} \rangle$ decreases as the number of nodes in Δ is increased, as increasing the number of nodes in Δ increases the level of replication, translating to lower $t_{\text{ms-total}}$. An important observation is that although $\langle \text{DRP}_{\text{mp-pro}}, \text{RRJR} \rangle$ has negligible t_{OH} , by using replication based RRJR algorithm in $\langle \text{DRP}_{\text{mp-pro}}, \text{RRJR} \rangle$, $t_{\text{ms-total}}$ cannot be improved to the degree achieved by using ATSRA-based allocation-plans. It justifies the effort to develop the ATSRA-based algorithms and to devise various strategies targeted at improving specific performance metrics.

Fig. 6.4 shows t_{cost} incurred while using the allocation-plans. Note the use of the performance improvement strategies leads to an increase in t_{cost} from the t_{cost} achieved by the use of $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$. For $\langle \text{SRP}_{\text{mp}}, \text{ATSRA}_{\text{org}} \rangle$ t_{cost} is increased because by subdividing Δ into different partitions, the maximum number of nodes that can be allocated to a particular resource-pool is reduced. Note that whenever the size of the resource-pool associated with a particular task is reduced, t_{cost} associated with that particular task tends to increase. The combined effect of restricting the resource-pools of all individual tasks is the increased value of t_{cost} associated with the processing of the complete bag-of-tasks. In comparison to $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$, t_{cost} also rises

as constraint-relaxation is used in $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{bsr}} \rangle$, decreasing the precision of resource-allocation.

But the increase in t_{cost} associated with constraint-relaxation is observed to be lower than the increase in t_{cost} associated with the use of multiple partitioning at TRPS for all number of nodes (see Fig. 6.4).

Fig. 6.4: Effect of Increasing Number of Nodes on t_{cost}

It can be observed that when the job replication strategy is used in $\langle \text{DRP}_{\text{mp-pro}}, \text{RRJR} \rangle$, increasing the number of nodes in Δ directly increases size of the resource-pools allocated to each of the tasks in the given bag-of-tasks which translates into higher level of replication of the constituent

jobs (as discussed in Section 3.6.4). The higher replication increases the usage of resources for each of the PBDT tasks and it can be observed that t_{cost} associated with $\langle \text{DRP}_{\text{mp-pro}}, \text{RRJR} \rangle$ increases sharply as the number of nodes in Δ is increased (see Fig. 6.4)

Fig. 6.5 shows cost-mstotal for different allocation-plans as the number of nodes is increased. As discussed in Chapter 5, minimizing $t_{\text{ms-total}}$ and t_{cost} are somewhat contradictory performance objectives. In addition to focusing on each of these objectives individually, it is also important to study the balance achieved between these two performance objectives by the use of a particular allocation-plan. cost-makespan-total is used to quantify this balance, which is the product of the total cost of executing all the tasks in the given bag-of-tasks (t_{cost}) and the time in seconds required for completing all the tasks in the given bag-of-tasks ($t_{\text{ms-total}}$).

The first observation is that if the objective is to reduce cost-mstotal, then $\langle \text{DRP}_{\text{mp-pro}}, \text{RRJR} \rangle$ is not a good choice, irrespective of the number of nodes in the system. For $\langle \text{DRP}_{\text{mp-pro}}, \text{RRJR} \rangle$, cost-mstotal initially increases sharply as t_{cost} increases with the increase in nodes, but $t_{\text{ms-total}}$ does not drop proportionally. Once the number of nodes becomes greater than the number of partitions of the raw data files, job replication increases, reducing $t_{\text{ms-total}}$, which, in turn, decreases cost-mstotal.

Among the ATSRA-based algorithm, it can be observed from the plot of $\langle \text{SRP}_{\text{mp}}, \text{ATSRA}_{\text{org}} \rangle$ in Fig. 6.5 that if the objective is to reduce cost-mstotal, then multiple partitioning is not the best choice, irrespective of the number of nodes in the system. If the number of nodes in the system is

below 76 then $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$ provides lowest value of cost-mstotal. This is due to the fact that when number of nodes is small, t_{OH} is small and the benefit obtained by the precise resource-allocation achieved by the use of $\text{ATSRA}_{\text{org}}$ generates low value of $t_{\text{ms-total}}$, which, in turn, translates to low value of cost-mstotal (as t_{cost} is already lowest for $\langle \text{SRP}_{\text{sp}}, \text{ATRA}_{\text{org}} \rangle$ for any system and workload parameters). When the number of nodes is greater than 76, the strategy of relaxing constraints at RA is observed to be the most effective one as in Fig. 6.5 $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{bsr}} \rangle$ has the lowest value of cost-mstotal.

Fig. 6.5: Effect of Increasing Number of Nodes on cost-mstotal

6.4.2.2 Objective B: To Control Resource-contention

Two strategies have been devised to control resource-contention. Resource contention occurs when in the execution-phase, multiple tasks want to use the same set of resources resulting in overlapping resource demands. Note that in this research only one of the tasks is allowed to use a particular resource at a time, so all other tasks, needing the same resource, have to wait until that resource becomes free. Resource contention adds to the total time taken by the system to process the complete bag-of-tasks and thus increases $t_{ms-total}$.

In an effort to reduce $t_{ms-total}$, the following two strategies have been used.

Backfilling: In backfilling a resource-pool is created at runtime by using the resources that are not selected in the mapping phase by the RA algorithm due to their higher costs. In the execution phase, backfilling is triggered if a situation arises when some tasks are waiting for the set of resources, they have been assigned, when there are idle resources in the system. This may be because these idle resources were not chosen to be assigned to any task during the mapping phase. Backfilling leads to the dynamic creation of a new resource-pool for the waiting task having the smallest raw data file, by using the set of idle resources. $\langle SRP_{sp}+BF, ATSRA_{org} \rangle$ adds backfilling to $\langle SRP_{sp}, ATSRA_{org} \rangle$ and is analyzed to study the impact of backfilling on various performance metrics.

Reducing Resource-pool Algorithm: The Reducing Resource-pool Algorithm is used in the dynamic TRPS policies in which the resources for the tasks are decided dynamically according to their availability. The resource-pool for the first task is determined and passed on to RA, which chooses a set of resources from the allocated

resource-pool and assigns this set to the task. The resource-pool of each subsequent task is chosen from the available resources. As the resource-pool of each task is chosen only from the available resources, resource-contention is considerably reduced. $\langle \text{DRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$ replaces the two phase resource allocation in $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$ with the Reducing Resource-pool Algorithm. $\langle \text{DRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$ is analyzed in this section to study the performance impact of the Reducing Resource-pool Algorithm.

Fig 6.6 shows the impact of increasing the total number of nodes on t_{cost} for various allocation-plans. The lower bound on t_{cost} is also included in these figures. It can be observed that irrespective of the system and workload parameters, using these two strategies to improve the performance of $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$ always increases t_{cost} but the amount of increase depends on the system and workload parameters as well as the strategy used.

The use of backfilling is seen to have an interesting effect on t_{cost} . It can be observed from Fig. 6.6 that when backfilling is used in $\langle \text{SRP}_{\text{sp}} + \text{BF}, \text{ATSRA}_{\text{org}} \rangle$, t_{cost} initially decreases and then increases as the number of nodes is increased. Backfilling uses the resources that are not otherwise being utilized at runtime. In backfilling, a resource-pool is created at runtime by using the resources that are not selected in the mapping-phase by the RA algorithm due to the association of higher costs with them. Thus, backfilling uses “costlier” resources originally left by the $\text{ATSRA}_{\text{org}}$ algorithm during the mapping phase and generates a higher value of t_{cost} . In Fig. 6.6, it can be observed that when the number of nodes are below 24, there is negligible backfilling and t_{cost} drops as the number of nodes in Δ is increased. As the number of nodes is increased above 24, backfilling increases, driving up the value of t_{cost} .

Fig. 6.6: Impact of Increasing Number on Nodes on t_{cost}

Fig. 6.6 indicates when Reducing Resource-pool Algorithm is used in $\langle \text{DRP}_{sp}, \text{ATSRA}_{org} \rangle$, t_{cost} increases considerably as compared to t_{cost} incurred by the use of $\langle \text{SRP}_{sp}, \text{ATSRA}_{org} \rangle$. As described in Chapter 3, with $\langle \text{DRP}_{sp}, \text{ATSRA}_{org} \rangle$ the resource-pool allocation for a task depends on the position of the particular task in the given bag-of-tasks. The largest resource-pool is allocated to the first task in the given bag-of-tasks. In comparison, in $\langle \text{SRP}_{sp}, \text{ATSRA}_{org} \rangle$ each task is allocated a resource-pool containing all the nodes in Δ , that are allocated only to the first task in $\langle \text{DRP}_{sp}, \text{ATSRA}_{org} \rangle$. Note that reducing the size of the resource-pool increases t_{cost} as it limits the resource-allocation choices for the RA algorithm.

Fig. 6.7 shows t_{ms-WOH} for different allocation-plans as the number of nodes in Δ is increased. Note that backfilling is used in $\langle SRP_{sp}+BF, ATSRA_{org} \rangle$. Backfilling considerably reduces

Fig. 6.7: Impact of Increasing Number on Nodes on t_{ms-WOH}

resource contention by leading to the dynamic creation of a new resource-pool for the waiting task having the smallest raw data file, by using the set of idle resources. The amount of backfilling depends on the number of idle resources available at any time. As the number of nodes in Δ is increased, the number of nodes that are not assigned to any task increases, thus, increasing the degree of backfilling. Increased backfilling further reduces resource-contention thus decreasing t_{ms-WOH} . Experimental results show that backfilling significantly reduces t_{ms-WOH} as $\langle SRP_{sp}+BF, ATSRA_{org} \rangle$ has the lowest value of t_{ms-WOH} for all number of nodes (see Fig. 6.7).

The use of Reducing Resource-pool Algorithm in $\langle \text{DRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$ also reduces the resource-contention, hence decreasing $t_{\text{ms-WOH}}$, but is less effective then $\langle \text{SRP}_{\text{sp}} + \text{BF}, \text{ATSRA}_{\text{org}} \rangle$ to reduce $t_{\text{ms-WOH}}$.

Finally, to capture the trade-off between $t_{\text{ms-total}}$ and t_{cost} , the values of cost-mstotal are plotted for different allocation-plans in Fig. 6.8. Following observations can be made from Fig. 6.8.

- When the number of nodes are above 32, $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$ yields the lowest value of cost-mstotal. It means that for the workload parameters used in the experiments and a large number of nodes, it is best to use $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$ to achieve a good

Fig. 6.8: Effect of Increasing Number of Nodes on cost-mstotal

balance between t_{cost} and $t_{\text{ms-total}}$.

- When the number of nodes are below 32 $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$ does not generate the lowest cost-mstotal due to the resource-contention. Addition of backfilling at TRPS to $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$ is observed to be the most efficient strategy and gives the lowest value of cost-mstotal (see Fig. 6.8).

6.5 Summary

This chapter presents a technique for the computation of the lower bound on t_{cost} by using an analytical method and then compares the values of t_{cost} associated with the use of various allocation-plans with this lower bound. For a given system, the choice of the most efficient allocation-plan mainly depends on the performance objective. System and workload parameters are also observed to play a key role in choosing the most effective allocation-plan. A particular resource allocation-plan can have one of the following three performance objectives.

- 1- To minimize t_{cost}
- 2- To minimize cost-mstotal
- 3- To minimize $t_{\text{ms-total}}$

The discussion presented in this chapter analyzes the effectiveness of various allocation-plans with respect to these three performance objectives and recommends which allocation-plans should be used for various system and workload parameters. In a possible future extension of this research work, the recommendations for the choice of an allocation-plan can be used to architect a fully automated BiLeG architecture that can automatically choose an allocation-plan based on

the performance objective and existing system and workload parameters. This is further discussed in Section 7.3.

$\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$ is observed to generate a t_{cost} which is closest to the lower bound for all system and workload parameters. Unfortunately, $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$ is also observed to produce one of the highest values of $t_{\text{ms-total}}$ among various allocation-plans which makes it an unreasonable choice for achieving the other two performance objectives. High levels of resource contention and algorithm-running-overhead have been attributed to the high value of $t_{\text{ms-total}}$ associated with $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$ and in an effort to reduce $t_{\text{ms-total}}$, various strategies have been developed. These strategies focus on controlling resource contention and reducing the algorithm-running overhead. Experimental results highlight different trade-offs that are implicit in the use of each of these strategies. The conclusions from the experimental results which are discussed in detail in the previous subsections are summarized:

If the optimization objective is to minimize t_{cost} ,

- irrespective of the system and workload parameters, $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$ should be used (see Fig. 6.4 and Fig. 6.6).

If the optimization objective is to minimize cost-mstotal

- $\langle \text{DRP}_{\text{mp-pro}}, \text{RRJR} \rangle$ should be avoided (see Fig. 6.5).
- If a Grid has a large number of nodes (greater than 76 nodes), the use of $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{bsr}} \rangle$ produces the lowest value of cost-mstotal (see Fig. 6.5).

- If a Grid has a small number of nodes (less than 32 nodes), $\langle \text{SRP}_{\text{sp}} + \text{BF}, \text{ATSRA}_{\text{org}} \rangle$ produces the lowest value of cost-mstotal (see Fig. 6.8).

If the optimization objective is to minimize $t_{\text{ms-total}}$

- If a Grid has a large number of nodes (greater than 92 nodes), $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{bsr}} \rangle$ should be used (see Fig. 6.3).
- If a Grid has a small number of nodes $\langle \text{SRP}_{\text{mp}}, \text{ATSRA}_{\text{org}} \rangle$ gives the lowest value of $t_{\text{ms-total}}$ (see Fig. 6.3).

Chapter 7 Conclusions

7.1 Summary

This research proposes a bi-level Grid Resource management system, called BiLeG, to be used for resource allocation in a Grid computing environment executing a specific type of resource intensive tasks called the PBDT tasks. The contributions of the thesis highlighting difference with existing research have been discussed in Section 1.5. A short summary of the thesis is presented in this section.

BiLeG devises an allocation-plan which reflects the overall resource allocation strategy comprising two parts; a policy used at the higher decision making module, TRPS, which has the responsibility to select a resource-pool for each of the tasks; and a resource allocation algorithm used at the lower decision making module, RA, which actually assigns resources from the resource-pool selected by TRPS for a particular PBDT task. Five RA algorithms and six TRPS policies have been proposed in this research forming different allocation-plans. The suitability of various allocation-plans under different sets of system and workload parameters has been explored.

The proposed RA algorithms include a “family” of three Linear Programming based algorithms called ATSRA. Resource allocation suggested by the ATSRA algorithms is quite accurate and produces effective results. But, as the ATSRA algorithms are complex, they can generate substantial algorithm-running-overhead. Out of the three ATSRA algorithms, $ATSRA_{org}$ is the

most accurate algorithm and gives rise to the minimum value of t_{cost} for a given bag-of-tasks. This is followed by t_{cost} achieved by using $\text{ATSRA}_{\text{ssr}}$ and $\text{ATSRA}_{\text{bsr}}$ respectively. However, due to a higher computational complexity associated with $\text{ATSRA}_{\text{org}}$, the concomitant algorithm-running-overhead is observed to increase sharply with the number of nodes in Δ . For a given number of nodes in Δ , $\text{ATSRA}_{\text{org}}$ incurs the highest algorithm-running-overhead followed by $\text{ATSRA}_{\text{ssr}}$ and $\text{ATSRA}_{\text{bsr}}$.

In addition to the ATSRA class of algorithms, two non-LP based RA algorithms with negligible algorithm-running-overhead, have also been presented. First of them is a replication based algorithm called the Round Robin Job Replicator, RRJR. Note the ATSRA algorithms are based on the assumption that processing and communication times per unit data are known *a priori*. The performance of ATSRA algorithms depends on the accuracy of this information. When, under various circumstances accurate values of unit processing and communication times are not known in advance, RRJR that does not use any *a priori* knowledge of the processing and communication times per unit data, is a good choice. Note that t_{cost} associated with RRJR is quite high as compared to other RA algorithms due to job replication performed by RRJR. The second non-LP based RA algorithm is the List Scheduler, LS. In LS, the selection of appropriate resources does not take into account the unit communication times and, therefore, LS is observed to perform good only if the variance of the unit communication times is low in a system.

For the selection of a resource-pool at the upper TRPS module, six different policies have been proposed. Each policy details a strategy to partition the available nodes into various resource-pools corresponding to each of the tasks. The detailed performance analysis of the TRPS policies is conducted in Chapter 5 that discusses which TRPS policy should be used under a certain set of system and workload parameters.

7.2 Conclusions

Detailed study of the various trade-offs, implicit in the use of different allocation-plans, is one of the focal points of this research. As discussed in Chapter 5, the most suitable allocation-plan not only depends on various workload and system parameters, it also depends on the user requirements and the hardware available. The recommended allocation-plans under a given set of system and workload parameters are presented in Chapter 6. It can be seen that from the performance perspective various trade-offs exist among different allocation-plans and understanding these trade-offs in depth is the focus of the experiments conducted in this research.

For the choice of an appropriate allocation-plan, two of the important considerations that came out of these experimental results are the size of the Grid and the performance metric chosen for optimization. A general recommendation that can be made, irrespective of any system and workload parameters, is that if minimizing t_{cost} is the major objective then replication based allocation-plans such as RRJR should be avoided and ATSRA based allocation-plans should be considered after a careful study of the trade-offs associated with the use of each of them.

Generally, from the results obtained from the experiments conducted in the Chapter 4, Chapter 5 and Chapter 6, it can be concluded that if an allocation-plan tries to minimize one of the performance metrics, it tends to yield higher values of the other performance metrics. For example, $\langle \text{SRP}_{\text{sp}}, \text{ATSRA}_{\text{org}} \rangle$ always gives the lowest value of t_{cost} but it also yields one of the highest values for $t_{\text{ms-WOH}}$, especially for a large number of nodes. Note that at TRPS, three resource-pool allocation techniques, backfilling, dynamic resource allocation and multiple partitioning, have been investigated to enhance the basic TRPS policy, SRP_{sp} . The use of these three techniques gives rise to new TRPS policies. A detailed study of the trade-offs associated with these techniques forms an important direction for the performance investigation conducted in this research. The use of backfilling is observed to reduce $t_{\text{ms-WOH}}$ at the expense of increased t_{cost} . Multiple partitioning is useful typically for large size Grids and addresses two different issues. First, the use of the multiple partitioning technique drastically reduces t_{OH} associated with running the ATSRA algorithms. Second, using multiple partitioning also reduces resource-contention, which reduces $t_{\text{ms-WOH}}$. As discussed in Chapter 5, because $t_{\text{ms-total}}$ is the sum of t_{OH} and $t_{\text{ms-WOH}}$, the reduction of t_{OH} and $t_{\text{ms-WOH}}$ considerably reduces $t_{\text{ms-total}}$, but at the expense of increased t_{cost} .

At RA, the trade-offs associated with reducing the accuracy of the ATSRA algorithm by relaxing some of the constraints in the LP formulation have been studied in detail. The combination of the proposed RA algorithms and TRPS policies gives rise to various allocation-plans. These

allocation-plans can be used under a wide variety of system and workload parameters to maximize the use of available resources according to a pre-determined optimization objective.

Although the research has focused primarily on the Grid systems, the proposed BiLeG architecture can also be used in a Cloud Computing environment. Cloud Computing environments are often classified as public and private Cloud environments [DOA10]. The private Cloud environment is better suited for the BiLeG architecture; as a private Cloud environment uses a dedicated computing infrastructure that provides hosted services to a limited number of users behind a firewall [EGE10] and can, thus, more easily incorporate mechanisms to accurately predict the computing and communication costs. A public cloud is a shared environment; resource management in the context of such shared environments is discussed in the following section.

7.3 Future Works

This research recommends a suitable allocation-plan under a given set of system and workload parameters. Based on these recommendations, future research can not only focus on techniques for automating the choice of the appropriate allocation-plan for a given environment, but it can also devise an effective feedback mechanism to fine-tune certain parameters, such as the size of a partition in multiple partitioning based TRPS policies. Mechanism for making such choices of allocation-plans and for such tuning of parameters at runtime according to the prevailing conditions induces a certain learning behavior in the system that may be perceived as a form of artificial intelligence. Any future research to employ such form of artificial intelligence in the

BiLeG architecture can lead to framework for a commercial product, which can be used for resource-allocation in the utility Grid and Cloud Computing infrastructures.

Moreover, the proposed BiLeG architecture can be extended to other resource-intensive tasks, which cannot be classified as PBDT. The requirements and the needs of the non-PBDT tasks need to be carefully studied and the RA algorithms and the TRPS policies presented in this thesis will be required to be modified accordingly to cater for the needs of the non-PBDT tasks.

Fairness is one of the important properties of a resource allocation algorithm. Fairness is defined in the context of sharing resources by different tasks. Fairness quantifies equability of tasks in allocating resources according to a prearranged agreement [VIS07]. For example, in the context of the BiLeG architecture, the prearranged agreement for fairness may be the allocation of resources proportional to the size of the raw data files of the PBDT tasks. Then, for this particular prearranged agreement a perfectly fair scheduler is the one which allocates the resources to all the tasks proportional to the size of raw data file of each individual task. Intuitively, it can be observed that multiple partitioning based policies should perform well in the context of fairness as they allocate the resource-pool to each task proportional to the size of individual raw-data file. Devising fair allocation strategies is an active area of research. A detailed analysis of fairness associated with the use of various TRPS policies can form a direction for the future research. Future research can also analyze the impact of incorporating fairness on $t_{ms-total}$ and t_{cost} which may lead to further improvements in the design of the proposed TRPS policies.

It may be hard to predict the exact computation and communication times on a real system. Another important direction for future research is to analyze the impact of using approximate values of computation and communication times and study the impact of the error associated with the approximation on the performance of the resource-allocation algorithms. Both mean and distribution of the error are likely to impact system performance. Making measurements on real systems for determining the mean and the distribution is required.

The algorithms presented in this research are based on a dedicated resource environment. To adapt the BiLeG architecture to shared environments, more research is required. For example, in order to use it in a shared resource environment, mechanisms to accurately predict the unit communication and processing times are needed to be incorporated in the BiLeG architecture. Also, in a shared environment, investigating the impact of various types of communication models, such as many-to-one and one-to-many forms, an important direction for the future research.

References

- [ABB06] A. Abbas, "Grid Computing: A Practical Guide to Technology and Applications", Charles River Media, New York, USA, 2006.
- [AHM05] I. Ahmad and S. Majumdar, "An Adaptive High Performance Architecture for "Processable" Bulk Data Transfers on a Grid", in the Proceedings of the 2nd International Conference on Broadband Networks (Broadnets), Boston, USA, 2005, pp. 1482-1491.
- [AHM07] I. Ahmad and S. Majumdar, "Efficient Management of Grid Resources Using a Bi-level Decision-Making Architecture for Processable Bulk Data", in the Proceeding of the Conference on Grid computing, High-performance and Distributed Applications (GADA) Vilamoura, Algarve, Portugal, 2007, pp.1313-1321.
- [AHM08a] I. Ahmad and S. Majumdar, "Performance of Resource Management Algorithms for Processable Bulk Data Transfer Tasks in Grid Environments", in the Proceeding of the 7th ACM international Workshop on Software and Performance, Princeton, USA, 2008, pp. 177-188.
- [AHM08b] I. Ahmad and S. Majumdar, "A Two Level Approach for Managing Resource and Data Intensive Tasks in Grids", in the Proceedings of the Conference on Grid Computing, High-performance and Distributed Applications (GADA) Monterrey, Mexico, 2008, pp. 802-811.
- [ALL05] B. Allcock, A. Chervenak, I. Foster, C. Kesselman and M. Livny, "Data Grid Tools: Enabling Science on Big Distributed Data", Journal of Physics, Conference Series, 16(1), 2005, pp. 571-575.
- [AMA09] Amazon Web Services, <http://aws.amazon.com> (Accessed 2009).
- [ANG06] C. Anglano and M. Canonico, "Performance Analysis of High-Performance File-Transfer Systems for Grid Applications", Journal of Concurrency and Computation Practice & Experience, 18(07), 2006, pp. 807-816.
- [AZU09] Azure Services Platform, <http://www.microsoft.com/azure/default.mspx> (Accessed 2009).

- [AZZ06] A. Aziz and H. El-Rewini, "Grid Resource Allocation and Task Scheduling for Resource Intensive Applications", in the Proceedings of the 2006 International Conference on Parallel Processing Workshops, Columbus, Ohio, 2006, pp. 8-29.
- [BAS02] A. Bassi, M. Beck, G. Fagg, T. Moore, J. Plank, M. Swany and R. Wolski, "The Internet Backplane Protocol: A Study in Resource Sharing", in Proceedings of the Second IEEE / ACM International Symposium on Cluster Computing and the Grid, Piscataway, N.J, 19(4), 2002, pp. 551-561.
- [BEY02] M. D. Beynon, A. L. Sussman, U. Catalyurek, T. Kurc and J. Saltz, "Performance Optimization for Data Intensive Grid Applications", in the Proceedings Third Annual International Workshop on Active Middleware Services, San Francisco, CA 2002, pp. 97-105.
- [BRE03] P. Brezany, J. Hofer, A. Tjoa, and A. Wohrer. Gridminer, "An Infrastructure for Data Mining on Computational Grids", in the Proceedings of Australian Partnership for Advanced Computing Conference (APAC), Gold Coast, Australia, 2003, pp.88-95.
- [BUN03] J. Bunn and H. Newman, "Data-Intensive Grids for High Energy Physics", in Grid Computing: Making the Global Infrastructure a Reality, G. Berman and E. Hey, Eds. John Wiley & Sons, Inc., New York, 2003.
- [BUR03] L. Burchard, H. Heiss and C. A. F. De Rose, "Performance Issues of Bandwidth Reservations for Grid Computing", in the Proceedings of the 15th Symposium on Computer Architecture and High Performance Computing, Raleigh, North Carolina, 2003, pp. 82-90.
- [CAO99] J. Cao, D. J. Kerbyson, E. Papaefstathiou, and G. R. Nudd, "Modelling of High Performance Applications Using PACE", in the Proceedings of 15th Annual UK Performance Engineering Workshop, Bristol, UK, 1999, pp. 322-340.
- [CAO01] J. Cao, D. J. Kerbyson and G. R. Nudd, "Performance Evaluation of An Agent-Based Resource Management Infrastructure for Grid Computing", in the Proceedings of First IEEE/ACM International Symposium on Cluster Computing and the Grid, Brisbane, Australia, 2001, pp. 311-318.
- [CAS00] H. Casanova, A. Legrand, D. Zagorodnov and F. Berman, "Heuristics For Scheduling Parameter Sweep Applications in Grid Environments", in the Proceedings of 9th Heterogeneous Computing Workshop (HCW 2000), Geneva, Switzerland, 2000, pp. 349-363.

- [CAN03a] M. Cannataro and D. Talia, "Knowledge Grid: An Architecture for Distributed Knowledge Discovery", *Communications of the ACM*, 46(1), 2003, pp. 89-93.
- [CAN03b] M. Cannataro and D. Talia, "Towards the Next-Generation Grid: A Pervasive Environment for Knowledge-Based Computing", in the *Proceedings of the Fourth IEEE International Conference on Information Technology: Coding and Computing*, Los Alamitos, Calif., 2003, pp. 347-441.
- [CHE05] S. Chen, C. Yang and K. Li, "Optimizations Of Data Distribution Localities in Cluster Grid Environments", in the *Proceedings of the International Conference on Computational Science and Its Application*, ICCSA 2005, Singapore, 2005, pp. 1017-1027.
- [CHE01] A. Chervenak, I. Foster, C. Kesselman, C. Salisbusy, and S. Tuecke, "The Data Grid: Towards An Architecture for the Distributed Management and Analysis of Large Scientific Datasets", *Journal of Network and Computer Applications*, 23(3), 2001, pp.187-200.
- [CHU05] Chun-Wu Chen, "A Service-oriented Approach for the Parallelization of Data-intensive Algorithms in a Grid-enabled Cluster", in the *Proceedings of the 21st International Conference on Data Engineering Workshop*, Hannover, Germany, 2005, pp. 1154-1154.
- [CHV80] V. Chvatal, "Linear Programming", W.H. Freeman and Company Press , New York, USA, 1980.
- [CRO04] P. Crosby, D. Colling and D. Waters, "Efficiency Of Resource Brokering in Grids for High-Energy Physics Computing", *IEEE Transactions. Nuclear Science*, 51(1), 2004, pp. 884-891.
- [CZA98] K. Czajkowski, I.Foster, N. Karonis, C. Keselman, S. Martin, Warren Smith and Steve Tuecke, "A Resource Management Architecture for Metacomputing Systems", in the *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, Anchorage, Alaska, 1998, pp. 62-82.
- [CZA99] K. Czajkowski, I. Foster, and C. Keselman, "Resource Co-Allocation in Computational Grids", in the *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC)*, Redondo Beach, California, USA, 1999, pp. 222-231.
- [DEC07] T. Decker and R. Diekrmann, "Mapping of Coarse-Grained Applications onto Workstation Clusters", in the *Proceedings of the 5th EUROMICRO Workshop on Parallel and Distributed Processing*, PDP'07, Maastricht, Germany 2007, pp. 121-130.

- [DEA04] J. Dean and S.Ghemawat, "Mapreduce: Simplified Data Processing on Large Clusters", in the Proceedings of Operating Systems Design and Implementation (OSDI), San Francisco, CA, 2004, pp. 137-150.
- [DEE02] E. Deelman, C. Kesselman, G. Mehta, L. Meshkat, L. Pearlman, K. Blackburn, P. Ehrens, A. Lazzarini, R. Williams, S. Koranda, "Grifhyn and LIGO, Building A Virtual Data Grid For Gravitational Wave Scientists", in the Proceedings of 11th IEEE International Symposium on High Performance Distributed Computing, Piscataway, NJ, USA, 2002, pp. 225-234.
- [DEW07] T. Dewitt, T. Gross, B. Lowekamp, N. Miller, P. Steenkiste, J. Subhlok, and D. Sutherland, "Remos: A Resource Monitoring System for Network Aware Applications", in Technical Report CMU-CS-07- 194, School of Computer Science, Carnegie Mellon University, 1997.
- [DIM98] Dimitris Bertsimas, John N. Tsitsiklis, "Introduction to Linear Programming", Athena Scientific, Belmont, Massachusetts, 1998.
- [DOA10] D. Doan, "A Developer's Survey on Different Cloud Platforms", M.S. dissertation, University of California, San Diego, United States, 2010.
- [DOU02] N. Doulamis, A. Panagakis, "Workload Prediction of Rendering Algorithms in Grid Computing", in the Proceedings of European Multigrid Conference, Germany, 2002, pp. 24-30.
- [DOW05] A. B. Downey, "Lognormal and Pareto Distributions in the Internet", Comput. Commun., 28(1) , 2005, pp. 790-801.
- [EGE10] EGEE, An EGEE Comparative Study: Grids and Clouds – Evolution or Revolution, Enabling Grids for E-science report, <https://edms.cern.ch/document/925013>, (Accessed 2010).
- [ENT08] Enterprise Multimedia Systems, <http://enterprise.amd.com/Downloads/Industry/Multimedia>, (Accessed 2008).
- [FOS97] I. Foster, J. Geisler, W. Nickless, W. Smith, and S. Tuecke, "Software Infrastructure for the I-WAY High Performance Distributed Computing Experiment", in the Proceedings of 5th IEEE Symposium on High Performance Distributed Computing, Portland, OR, USA, 1997, pp. 562-571.

- [FOS08] I. Foster , Y. Zhao, I. Raicu, S. Lu, “Cloud Computing and Grid Computing 360-Degree Compared”, in the Proceedings of the Grid Computing Environments Workshop (GCE’08), Austin, Texas, USA2008, pp.10-18.
- [FOS99a] I. Foster and C. Kesselman, “The Grid: Blueprint for a Future Computing Infrastructure”, Morgan Kaufmann Publishers, USA, 1999.
- [FOS99b] I. Foster, C. Kesselman, C. Lee, B. Lindell, K. Nahrstedt and A. Roy, "A Distributed Resource Management Architecture that Supports Advance Reservations And Co-Allocation", in the Proceedings of IWQoS'9, Seventh International Workshop on Quality of Service, New Haven, CT, USA, 1999, pp. 27-36.
- [GAR79] M.R. Garey, D.S. Jonson, “Computer and Intractability, Computers and Intractability: A Guide to Theory of NP-Completeness”, W.H. Freeman Press, New York, 1979.
- [GHA02] M. Ghanem, Y. Guo, A. Rowe, and P. Wendel, “Grid-Based Knowledge Discovery Services for High Throughput Informatics”, in the Proceedings of the Eleventh IEEE International Symposium on High Performance Distributed Computing, Edinburgh, Scotland, 2002, pp. 140-146.
- [GLO08] The Globus Alliance, Research Papers from Globus Alliance Members, <http://www.globus.org/alliance/publications/papers.php> (Accessed 2008).
- [GLO09] Globus, <http://www.globus.org>, (Accessed 2009).
- [GOO09] Google App Engine, <http://code.google.com/appengine> (Accessed 2009).
- [GRI08] GridToday, <http://www.Gridtoday.com/Grid/638845.html>, (Accessed 2008).
- [GZA04] F. Gzara, "Large Scale Integer Programming: A Novel Solution Method and Application", ACM Press, 2004.
- [HAD10] Hadoop, <http://lucene.apache.org/hadoop/www> (Accessed 2010).
- [HAR08] D .Harris, “Why Grid Doesn’t Sell, On-Demand Enterprise”, <http://www.on-demandenterprise.com/blogs/26058979.html> (Accessed 2008).
- [HSU05] C. Hsu, G. Lin, K. Li and C. Yang, "Localization Techniques for Cluster-Based Data Grid," in the Proceedings of the 6th International Conference on Distributed and Parallel Computing, Melbourne, Australia 2005, pp. 83-92.

- [IOA97] Y. E. Ioannidis, M. Livny, A. Ailamaki, A. Narayanan, and A. Therber, "Zoo: A Desktop Experiment Management Environment", 1997, pp 580-583.
- [IBM10] IBM LoadLeveler, <http://www.ibm.com> (Accessed 2010).
- [JAN03] I. Janciak, P. Brezany, and A. Min Tjoa, "Towards theWisdom Grid: Goals and Architecture", in the Proceedings of 4th International Conference on Parallel Processing and Applied Mathematics (PPAM), Czestochowa, Poland, 2003, pp. 796-803.
- [JEO04] S. Jeong, Chan-Hyun Youn and Hyoug-Jun Kim, "Optimal File Replication Scheme (CO-RLS) for Data Grids", in 6th International Conference on Advanced Communication Technology, Austin, Texas, 2004, pp. 1055-1059.
- [JHA09] S.Jha, A. Merzky and G. Fox, "Clouds Provide Grids with Higher-Levels of Abstraction and Explicit Support for Usage Modes. Presentation for OpenGridForum (OFG) 23", http://www.ogf.org/OGF23/materials/1272/grids_hla_cloud.pdf (Accessed 2009).
- [JOH10] W. John and F. Ransome, "Cloud Computing: Implementation, Management, and Security", Auerbach Publications, New York, USA, 2010.
- [KEL00] P. Keleher , D. Zotkin, and D. Perkovic, "Attacking the Bottlenecks in Backfilling Schedulers, Cluster Computing: The Journal of Networks, Software Tools and Applications", 3(4) ,2000, pp. 245–254.
- [LAU98] Laurence Wolsey, "Integer Programming", John Wiley & Sons, Inc., Now York, 1998.
- [LEE07] Y. C. Lee and A. Y. Zomaya, "Practical Scheduling of Bag-of-tasks Applications on Grids With Dynamic Resilience", IEEE Trans. Comput., 56(6), 2007, pp. 815-25.
- [LIN05] C. Lin, Q. Li and Z. Shan, "Performance Modeling and Analysis for Resource Scheduling In Data Grids", in the Proceedings Network and parallel computing. International conference, Beijing, China, 2005, pp. 32-39.
- [LIU04] D. Liu and M. Franklin. Griddb, "A Data-Centric Overlay for Scientific Grids", in the Proceedings of the Thirtieth international conference on Very large data bases, VLDB Endowment, Austin, Texas, USA, 2004, pp. 600-611.

- [MAT09] T. Mather, S. Kumaraswamy and S. Latif, "Cloud Security and Privacy", 1st Edition, O'Reilly Media Inc., New York, 2009.
- [MAU10] Maui Scheduler Open Cluster Software, <http://mauischeduler.sourceforge.net>, (Accessed 2010).
- [MUA01] A. Mualem, and D. Feitelson, "Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBMSP2 with Backfilling", IEEE Transactions on Parallel and Distributed Systems, 12(6), 2001, pp. 529–543.
- [NUD05] G. R. Nudd and S. A. Jarvis, "Performance-Based Middleware for Grid Computing", Concurrency and Computation Practice & Experience, 17(1), 2005, pp. 215-234.
- [PAN08] S. Pandey and R. Buyya, "Scheduling of Scientific Workflows on Data Grids", in the Proceedings of the Eighth IEEE International Symposium on Cluster Computing and the Grid, Boston, USA, 2008, pp. 548-553.
- [PEN04] L. Peng and S. See, "Scheduler Oriented Grid Performance Evaluation", in Parallel and Distributed Computing: Applications and Technologies, Springer Berlin, 2004, pp. 844-847.
- [RAJ08] B. Rajkumar, S. Chee , V. Srikumar, "Market Oriented Cloud Computing: Vision, Hype, And Reality For Delivering IT Services as Computing Utilities", in the Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications, Dalian, China, 2008, pp. 65-73.
- [RAY04] S. Ray and Z. Zhang, "Heuristic-Based Scheduling to Maximize Throughput of Data-Intensive Grid Applications", in Distributed Computing, IWDC, 2004, pp. 63-74.
- [RIT10] J. Rittinghouse, "Cloud Computing: Implementation, Management, and Security", Auerbach Publications, New York, 2009.
- [SAN01] V. Sander, W. A. Adamson, I. Foster and A. Roy, "End-To-End Provision of Policy Information for Network QoS", in the Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing, San Francisco, CA, USA, 2001, pp. 115-126.
- [SAN05] E. Santos-Neto, W. Cirne, F. Brasileiro and A. Lima, "Exploiting Replication and Data Reuse to Efficiently Schedule Data-Intensive Applications on Grids", in the Proceedings of International Conference on the Job Scheduling Strategies for Parallel Processing, Cambridge, MA, USA, 2005, pp. 210-232.

- [SKO10] S.KO, "Efficient On-Demand Operations in Large-Scale Infrastructures", Ph.D. dissertation, Department of Computer Science, University of Illinois at Urbana-Champaign, United States , 2010.
- [SEN05] P. Senkul and I.Toroslu, "An Architecture For Workflow Scheduling Under Resource Allocation Constraints", *Information Systems*, 30(5), 2005, pp. 399-422.
- [SHA07] S.Shankar and David J. DeWitt, "Data Driven Workflow Planning in Cluster Management Systems", in the Proceedings of the 16th international symposium on High performance distributed computing, ACM, New York, USA, 2007, pp. 127-136.
- [SCH07] J. M. Schopf, "Performance Prediction and Scheduling for Parallel Applications on Multi-User Clusters", Ph.D. dissertation, Department of Computer Science, UCSD, 2007.
- [STA04] V.Stankovski, M. May, J. Franke, A. Schuster, D. McCourt, and W. Dubitzky, "A Service-Centric Perspective For Data Mining in Complex Problem Solving Environments", in the Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), Boston, USA, 2004, pp. 780-787.
- [SPO03] D. P. Spooner, S. A. Jarvis, J. Cao, S. Saini and G. R. Nudd, "Local Grid Scheduling Techniques Using Performance Prediction" in *IEE Proc. Comp. Digit. Tech.*, 150(2), 2003, pp. 87-96.
- [SUN02] B. Sundaram and B. M. Chapman, "XML-Based Policy Engine Framework for Usage Policy Management in Grids", in the Proceedings of the 3rd International Workshop on Grid Computing, Baltimore, USA, 2002, pp. 194-198.
- [SUN09] Sun Grid Compute Utility, <http://www.network.com> (Accessed 2009).
- [TAK03] H. Takemiya, K. Shudo, Y. Tanaka and S. Sekiguchi, "Constructing Grid Applications Using Standard Grid Middleware", *Journal of Grid Computing*, 1(2), 2003, pp. 117-131.
- [TAN06] M. Tang, Bu-Sung Lee, X. Tang and Chai-Kiat Yeo, "The Impact of Data Replication on Job Scheduling Performance in the Data Grid", *Future Generation Comput. Syst.*, 7(1) , 2006, pp. 254-268.

- [THA05] C. Tham, J. Yu and R. Buyya, "Cost-Based Scheduling of Scientific Workflow Application on Utility Grids", in the Proceedings of the First International Conference on e-Science and Grid Computing, Washington, DC, USA, 2005, pp. 140-147.
- [TIE00] B. Tierney, W. Johnston, J. Lee and M. Thompson, "A Data Intensive Distributed Computing Architecture for Grid Applications", Special issue on high performance computing and networking Europe, 2000, pp. 473 – 481.
- [TOY06] T. Toyama, Y. Yamada and K. Konishi, "A Resource Management System for Data-Intensive Applications in Desktop Grid Environments", in the Proceedings of the 18th IASTED International Conference on Parallel and Distributed Computing and Systems, Orlando, Florida, 2006, pp. 60-65.
- [VAZ03] S. Vazhkudai, "Enabling the Co-Allocation of Grid Data Transfers", in the Proceedings of the Fourth International Workshop on Grid Computing, Phoenix, Arizona, USA, 2003, pp. 44-51.
- [VEN07] S. Venugopal, "Scheduling Distributed Data-Intensive Applications on Global Grids", Ph.D. dissertation, Department of Computer Science and Software Engineering, The University of Melbourne, 2007.
- [VER02] D. Verma, S. Sahu, S. Calo, M. Beigi and I. Chang, "A Policy Service for Grid Computing", in the Proceedings of the Third International Workshop on Grid Computing, Tampa, FL. 2002, pp. 243-255.
- [VIS07] S. Viswanathan, B. Veeravalli and T. G. Robertazzi, "Resource-Aware Distributed Scheduling Strategies for Large-Scale Computational Cluster/Grid Systems", IEEE Trans. Parallel Distrib. Syst., 18(10), 2007, pp. 1450-1461.
- [WAN06] G. Wang, C. Lin and X. Liu, "Performance Modeling And Analysis for Centralized Resource Scheduling in Metropolitan-Area Grids", in the Proceedings of APWeb 2006 International Workshops: XRA, IWSN, MEGA, and ICSE, Harbin, China, 2006, pp. 583-589.
- [WIS10] G. Wishnie, "A Complex Event Routing Infrastructure for Distributed Systems", M.S. dissertation, Department of Computer Science, University of Kansas, United States, Kansas, 2010.
- [WOL03] R. Wolski, "Experiences with Predicting Resource Performance On-Line in Computational Grid Settings", in Performance Evaluation Review, 30(1), 2003, pp. 41-49.

- [WUY04] Y. Wu, T. Cheng and J. Huang, "Research on the Models of the Distribution of Files on the Networks", in the Proceedings of 2004 International Conference on Communications, Circuits and Systems, Boston, USA, 2004, pp. 108-112.
- [YAR02] A. Yarkhan and J. Dongarra, "Experiments with Scheduling Using Simulated Annealing in a Grid Environment", in Lecture notes in computer science Grid Computing-GRID 2002, Third International Workshop, Springer Verlag, Baltimore, USA, 2002, pp. 232-242.
- [YAN02] K. Yang, A. Galis and C. Todd, "Policy-Based Active Grid Management Architecture", in the Proceedings of 10th IEEE International Conference on Networks (ICON 2002), Austin, Texas, USA, 2002, pp. 243-248.
- [ZHA06] Y. Zhang and Y. Inoguchi, "Influence of Inaccurate Performance Prediction on Task Scheduling in a Grid Environment", IEICE Trans. Inf. Syst., 89(1), 2006, pp. 479-486.
- [ZAK99] M. Zaki, "Parallel and Distributed Association Mining: A Survey", IEEE Concurrency, 7(4), 1999, pp.14-25.

Appendix A

Table A1: Probability Distributions Used for Unit Communication and Processing Times

Parameter	Default Values
$[C_{p-\Delta}]$	Uniform [1,10]
$[C_{p-src}]$	Uniform [1,10]
$[C_{p-sink}]$	Uniform [1,10]
$[C_{c-in}]$	Uniform [10,100]
$[C_{c-\Delta-ks}]$	Uniform [10,100]
$[C_{c-out}]$	Uniform [10,100]
$[C_{c-src-ks}]$	Uniform [40,400]

Appendix B

Fig. B.1: Impact of Increasing Number on Nodes on t_{ms-WOH}

Fig. B.2: Effect of Increasing Number on Nodes on $t_{ms-total}$

Fig. B.3: Impact of Increasing Number on Nodes on t_{cost}