Managing Opportunistic and Dedicated Resources in a Bi-modal Service Deployment Architecture

by

Shah Asaduzzaman

School of Computer Science
McGill University, Montréal, Canada

October 2007

A Thesis Submitted to McGill University
in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

This document is dedicated to my father
who inspired the never-ending quest for knowledge in my life.

Acknowledgement

Since I joined the highly dynamic research group in Advanced Networking Research Lab (ANRL) lead by Prof. Muthucumaru Maheswaran, I have worked with a few excellent people whose valuable contributions in various ways has made this thesis a reality. It is a great pleasure to convey my gratitude to them.

First, I would like to express my heartiest thanks to my supervisor Prof. Maheswaran whose extra-ordinary guidance, leadership and motivation have helped me unravel my own capabilities, which I was not aware of. Through continuous discussion and arguments he provided a really agile and creative research environment, which inspired and enriched my growth as a student and a researcher.

I am grateful to the members of my PhD committee, Prof. Carl Tropper and Prof. Hans Vangheluwe, who have contributed with their valuable comments and ideas from the very beginning of my PhD research and took the patience to carefully review all of my works.

Heartiest thanks to all my colleagues in ANRL, especially to Maniy and Arindam, for constant support and feedback in all of my research projects. Also, many thanks to Julien for helping me in French translation.

My sincerest regards to my mother (Mrs. Halima Khatun) whose absolute support and unconditional love throughout my life has brought me in this position, and my father (late Mr. Mortuza Ali), who enlightened my heart and opened my eyes to explore the universe.

Finally, my heartiest love for my wife Rumana, whose tremendous patience and constant support made me progress through my endeavors. And, all my affections to our daughter Areeba, the most precious gift we have ever had in our life, whose ever-smiling face is the biggest inspiration for everything I do.

Abstract

With the emergence of service oriented computing, hosting platforms are becoming key elements of the Internet. One popular example of service hosting platform is Internet Data Center that relies on statically dimensioned centralized pools of server resources. Although direct control over the resources is a key advantage here, such platforms are often unable to handle highly varying workloads of many applications. On the other hand, popular peer-to-peer systems that opportunistically aggregate idle resources from widely distributed end-user computers demonstrate the huge potential of these public resources to build highly scalable, low cost and easily deployable platforms. However, due to the uncertain availability of these resources, it is hard to guarantee any performance for the deployed services on such peer-to-peer platforms.

In this thesis, we introduced a new bi-modal architecture for a geographically distributed and cost-effective service hosting platform. The proposed architecture utilizes a combination of statically provisioned dedicated resource pools and widely available opportunistic public resources to provide quality assured services. The core idea is that through dynamic management of a combination of these two classes of resources, one can gain from the scalability of the public resources and achieve assured quality services by masking their unreliable behavior with the controlled performance of the dedicated resources.

We have explored the combination in two different dimensions – bi-modal computing resources and bi-modal communication links. In the first case, a combination of a dedicated cluster of computers and idle capacities of user computers have been exploited to build a platform to serve compute intensive applications with response time guarantees. In the second case, a collection of geographically distributed dedicated servers inter-connected with a combination of dedicated links and variable capacity public network paths have been

utilized to serve distributed stream processing applications that requires simultaneous allocation of computing and communication resources. In both cases, we have observed that by designing appropriate resource management policies, the combined sets of resources can be utilized to increase the overall resource utilization and throughput of the system as well as to increase the client satisfaction in terms of fulfillment of the service agreements.

Résumé

Le gain de popularité de l'informatique orienté service fait des plateformes hôtes pour ces services un élément essentiel de l'Internet. Un exemple de plateforme est le centre de traitement de l'information qui se fie à des réserves de serveur-ressource statiques. Le contrôle direct des ressources permis par ce genre de système est un avantage, mais ces systèmes peuvent difficilement gérer des charges variables. Les systèmes "peer-to-peer" permettent de redistribuer les ressources libres parmi les ordinateurs du réseau de façon opportuniste. Ce genre de système a le potentiel de créer une plateforme extensible, facile á déployer, et abordable. Par contre, la disponibilité incertaine des ressources rend très difficile de garantir la performance pour les applications informatiques distribuées sur ce genre de plateforme.

Cette thèse présente une architecture bi-mode pour une plateforme hôte géographiquement réparti et rentable. L'architecture utilise une combinaison de ressources attribuées fixes et de ressources publiques attribuées de façon opportuniste afin d'assurer la performance du système. À la base, ce système cherche à gérer de façon dynamique ces deux types de ressources afin d'obtenir l'extensibilité des ressources publiques et la performance assuré du système fixe tout en masquant comportement incertain.

La combinaison a été exploré de deux manières différentes: les ressources informatiques bi-mode et les liens de communication bi-mode. Dans le premier cas, une combinaison d'ordinateurs dédiés et de ressources publiques provenant des ordinateurs des usagers a été exploité pour créer une plateforme visant les applications ayant de lourdes demandes de ressources tout en gardant une garantie sur le temps de réponse. Dans le second cas, une collection de serveurs dédiés réparties géographiquement connectés à une combinaison de liens dédiés et de liens publiques variables a été utilisé afin de servir des applications

continues nécessitant des ressources de computation et de communication. Dans les deux cas, nous avons observé qu'avec de bonnes règles de gestion des ressources, les ressources combinées peuvent augmenter l'utilisation et le débit du système et aussi d'augmenter la satisfaction des clients.

Contents

Acknowledgement **iii**

Abstract **iv**

Résumé **vi**

1 Introduction **1**

1.1 Overview 1

1.2 Problem Definition 3

1.3 Thesis Contributions 5

1.4 Thesis Roadmap 7

1.5 Published Articles 8

2 Background on Services and Hosting Platforms **10**

2.1 Overview 10

2.2 Service Oriented Computing and Hosting Platforms 10

2.3 Application Taxonomy 11

2.4 Hosting Platform Architectures 12

3 Proposed System Architecture **15**

3.1 Overview 15

3.2 Layered Architecture 16

3.3 Bi-modal Organization of Computing Nodes 18

3.3.1 Organization and Characteristics of the Computing Resources . . . 19

3.3.2 Organization of the Resource Manager . . . 23
3.3.3 Users and SLA . . . 24
3.3.4 Job Characteristics . . . 25
3.3.5 Migration and Virtual Machines . . . 26
3.4 Bi-modal Organization of Communication Links . . . 27
3.4.1 Application model . . . 27
3.4.2 Organization and Characteristics of the Link Resources . . . 28
3.4.3 Organization of the Resource Manager . . . 31
3.4.4 Users and SLA . . . 33

4 The Resource Management Problem in Bi-modal Architecture 35
4.1 Overview . . . 35
4.2 Managing Bi-modal Organization of Processing Nodes . . . 36
4.2.1 State-oblivious Scheduling . . . 38
4.2.2 State-aware Scheduling . . . 39
4.3 Managing Bi-modal Organization of Network Resources . . . 39
4.3.1 Mapping of the Composition . . . 40
4.3.2 Dynamic Scheduling . . . 42

5 Related work 44
5.1 Service Hosting Platforms . . . 44
5.1.1 Cluster Based Platforms . . . 45
5.1.2 Distributed Platforms with Controlled Resources . . . 45
5.1.3 Peer-to-Peer platforms . . . 46
5.1.4 Research on Scheduling Problem . . . 47
5.1.5 Research on QoS Assurance . . . 48
5.1.6 Virtual Machines and Service Migrations . . . 49
5.2 Bi-modal Architectures . . . 50
5.3 Stream Processing Platforms . . . 51

6 State Oblivious Management of Public Computing Resources 53
6.1 Overview . . . 53

6.2 The Resource Management Problem 55
6.3 Heuristic Solutions for Resource Management 57
6.3.1 PCU Heuristic: An Online Resource Allocator 57
6.3.2 Least Laxity First and Greedy Heuristics 59
6.4 Simulation Results 60
6.4.1 Comparative Study 61
6.4.2 Response to Parameter Changes 65
6.5 Summary 71

7 State Aware Management of Public Computing Resources 73
7.1 Overview 73
7.2 Design of the Scheduling Policies 75
7.2.1 Defining the Problem 75
7.2.2 Preemptive Migrations 76
7.2.3 Priority Functions 77
7.2.4 Updating Remote States and Failure Detection 79
7.2.5 Replication 79
7.2.6 Analyzing the Design Parameters 80
7.3 Simulation Study 84
7.3.1 Simulation Model 85
7.3.2 Workload Data Source 86
7.3.3 Choosing Between Scheduling Schemes 87
7.3.4 Setting the Design Parameters 89
7.3.5 Design Guidelines 93
7.3.6 Performance Comparison with Grid Systems 94
7.4 Summary 95

8 Allocation of Network and Node Resources for Stream Processing 98
8.1 Overview 98
8.2 The Resource Allocation Problem 101
8.2.1 Capacity Constrained Graph Mapping Problem 102
8.2.2 Constrained Path Mapping Problem 104

8.2.3 Computational Complexity of the Problem . . . 106
8.3 Algorithm for Path Mapping Problem . . . 107
8.3.1 Correctness of the CCPM Algorithm . . . 109
8.3.2 Complexity of the Algorithm . . . 112
8.3.3 Distributed Version of the Algorithm . . . 113
8.3.4 Heuristic Approaches to Reduce Complexity . . . 113
8.4 Performance of the Heuristics . . . 115
8.5 Summary . . . 121

9 Dynamic Management of Bi-modal Network for Stream Processing 122
9.1 Overview . . . 122
9.2 Mapping and Reservation Protocols . . . 123
9.3 Dynamic Scheduling of Links . . . 125
9.4 Simulation Study . . . 128
9.4.1 Simulation Model . . . 129
9.4.2 Results . . . 132
9.5 Summary . . . 139

10 Conclusion 140
10.1 Summary of Contributions . . . 140
10.2 Future Extensions . . . 144
10.2.1 Computing Platform . . . 144
10.2.2 Stream Processing Platform . . . 146

Bibliography 147

List of Figures

2.1 Application taxonomy 11

3.1 The Galaxy Architecture. 17
3.2 An illustration on formation of PCU, by augmenting one or more dedicated cluster of compute servers with public resources and serving its subscribers with quality services 20
3.3 Histogram of frequencies of hosts of different floating point capacities (FLOPS) that are participating in SETI@Home project across the Internet [5] 22
3.4 Markov model for host availability characteristics [73]. AV=Available, NA=Not Available. Numbers inside the states are state duration and steady state probability respectively 22
3.5 Illustration of a distributed stream processing platform containing five application/data servers interconnected with public network as well as dedicated links 29

6.1 Variation of mean throughput with offered load values for mean public resource throughput $\mu = 0.80$, mean number of parallel components $P = 25$, total number of dedicated resources $M = 100$, and total SLA booking, $\sum \rho V = 100$. 62
6.2 Variation of penalty per unit revenue with offered load for $\mu = 0.80$, $P = 25$, $M = 100$, and $\sum \rho V = 100$. 63
6.3 Variation of net profit earned by the PCU provider with offered load for $\mu = 0.80$, $P = 25$, $M = 100$, and $\sum \rho V = 100$. 63

6.4 Upper bound on PCU throughput assuming future behavior of public resources is known for $\mu = 0.80$, $P = 25$, $M = 100$, and $\sum \rho V = 100$. . . . 64
6.5 Utilization of dedicated resources versus offered load for $\mu = 0.80$, $P = 25$, $M = 100$, and $\sum \rho V = 100$. 64
6.6 Penalty per unit revenue earned at different levels of SLA booking for $\mu = 0.80$, $P = 25$, and $M = 100$. 66
6.7 Penalty per unit revenue earned at different levels of SLA booking for $\mu = 0.80$, $P = 25$, and $M = 100$. 66
6.8 Throughput gain at different public resource characteristics, with respect to a dedicated pool only system for $P = 25$, $M = 100$, and $\sum \rho V = 100$. . . . 67
6.9 Throughput gain at different public resource characteristics, with respect to the greedy resource allocation policy on combined pools for $P = 25$, $M = 100$, and $\sum \rho V = 100$. 67
6.10 Mean throughput at varying degree of parallelism for $\mu = 0.80$, $M = 100$, and $\sum \rho V = 100$. 68
6.11 Throughput gain at different degrees of parallelism, with respect to a dedicated pool only system for $\mu = 0.80$, $M = 100$, and $\sum \rho V = 100$. 68
6.12 Throughput gain at different degrees of parallelism, with respect to the greedy resource allocation policy on combined pools for $\mu = 0.80$, $M = 100$, and $\sum \rho V = 100$. 69
6.13 Throughput gain at different amount of laxity in deadline, with respect to a dedicated pool only system for $\mu = 0.80$, $P = 25$, $M = 100$, and $\sum \rho V = 100$. 70
6.14 Comparing delivered throughput to 2 clients having different max-load defined in SLA for $\mu = 0.80$, $P = 25$, and $M = 100$. 70

7.1 Correlation of Goodput with SLA MaxLoad parameter: Static Allocation. Pearson's correlation coefficient, $r = 0.1655306734$ 88
7.2 Correlation of Goodput with SLA MaxLoad parameter: Dynamic Priority Preemption. Pearson's correlation coefficient, $r = 0.3835632358$ 88

7.3 Comparing PCU throughput at different loads with Erlang Loss systems with equivalent number of resources. $N_d = 100$, $N_p = 10000$, $r = 2$, $P_{av} = 0.7$, $\frac{T_p}{T_d} = 0.34$ 89
7.4 How Goodput is affected by number of dedicated resources at different loads 89
7.5 Load on the bottleneck network link due to preemptive migrations 90
7.6 Time required for a migration through the bottleneck link 90
7.7 Downtime due to failure and preemptive migration, for increasing degree of replication 91
7.8 Higher degree of replication reduces elongation, because it increases the chance of getting higher capacity public resource 91
7.9 Downtime due to failure and preemptive migration at different lengths of scheduling epoch 93
7.10 How elongation is affected by the length of the scheduling epoch 93
7.11 Comparing PCU with DAS2 Grid: gain in resource utilization and increase in running time 95
7.12 Amount of work done on public resources 95

8.1 An example resource network 103
8.2 An example data-flow computation with a DAG topology 104
8.3 An example data-flow computation with a path topology 105
8.4 The ratio of the cost of heuristic derived solutions to the lower-bound cost of the optimal solution, across different sizes of networks. 118
8.5 The ratio of the cost of heuristic derived solutions to the lower-bound cost of the optimal solution, across different number of components in the stream processing tasks. 118
8.6 The ratio of the cost of heuristic derived solutions to the lower-bound cost of the optimal solution, across different load to capacity ratios. 119
8.7 Percentage of false negative results, across different load to capacity ratios. 119
8.8 Percentage of false negative results, across different sizes of the server network. 119

8.9 Percentage of false negative results, across different number of components in the stream processing tasks. 119
8.10 Total number of map-extension messages exchanged, across different sizes of the server network. 120
8.11 Total number of map-extension message exchanged, across different number of components in the stream processing tasks. 120

9.1 Proportion of offered jobs accepted (arrival rate = 60 tasks/hr) 133
9.2 Overall system throughput in bi-modal and uni-modal systems (arrival rate = 60 tasks/hr) 133
9.3 Server utilization in three different cases of resource combinations (with 99 dedicated links) 134
9.4 Server utilization in bi-modal and uni-modal systems with increasing installation of dedicated links (arrival rate = 60 tasks/hr) 134
9.5 Utilization of dedicated links at three different cases of resource combinations (with 99 dedicated links) 134
9.6 Utilization of the dedicated links in bi-modal and uni-modal systems, with increasing installation of dedicated links (arrival rate = 60 tasks/hr) 134
9.7 Deviation from the contracts of the accepted jobs for three different cases of resource combinations (with 99 dedicated links) 135
9.8 Deviation from the contracts of the accepted jobs with increasing installation of dedicated links (arrival rate = 60 tasks/hr) 135
9.9 Mean task execution time in three different cases of resource combinations, showing the elongation of execution time when using public links (with 99 dedicated links) 136
9.10 Overall throughput of the bi-modal system, with or without dynamic scheduling (with 99 dedicated links) 137
9.11 Dynamic scheduling increases the capacity of the system and hence the task acceptance ratio (with 99 dedicated links) 137
9.12 Dynamic scheduling increases the utilization of dedicated links (number of dedicated links = 99) 138

9.13 Dynamic scheduling reduces the deviation from the target delivery rate (number of dedicated links = 99) 138

List of Tables

2.1 Characterization of different service hosting platform architectures 13

List of Algorithms

1 Skeleton scheduler 78
2 *Pathmap(P_J, G_R)* 109
3 **subroutine** *Relax(u,v)* 110
4 **subroutine** *Extend(m, j, x, v)* 111
5 *ProcessMap(u, m)* 114
6 Link re-allocation algorithm 127

1

Introduction

1.1 Overview

Constant improvements in computer communications and microprocessor technologies are driving the development of new classes of distributed computing systems. *Service Oriented Computing* is becoming a dominant paradigm in distributed application development to support the growing trend in outsourcing and to benefit from the economies of scale in service hosting platforms. The hosting platforms bring a large number of resources and application services together in a virtual system to serve its clients. Typically, platforms are built by connecting the resources and services to a *resource management system* (RMS) that itself is implemented either centrally or federally. The RMS allocates resources to the client requests such that some measure of delivered performance is maximized, subject to fairness constraints. The hosting platforms have diverse designs based on various parameters including target applications, organization of the RMS, classes of resources managed by the platform, and levels of services offered to the clients.

Hosting platforms have been successfully deployed to serve a wide range of applications, such as web hosting [43], high-performance computing [40], online gaming [87], content distribution [84], and video-on-demand [96]. While research into hosting platforms is striving to achieve application independent designs [27], target applications continue to cast a strong influence on the design of the platforms. The organization of the RMS is another key design consideration. The organization of the RMS can impact the scalability, extensibility, and fault tolerance of the platform. The platform can manage different classes of resources. For instance, a platform can include dedicated resources that are

owned and exclusively managed by the platform, volunteer resources that are not bound by any contracts, and partially committed resources that are managed through incentives schemes administered by the platform. The levels of services offered to the clients by the platform is another important consideration. In the simplest case, the platform offers *best effort* services to the clients. However, to attract clients with business critical applications the platforms should offer services with *quality of service* (QoS) assurances.

Obviously, implementing services very efficiently on hosting platforms is one of the key requirements for the success of service oriented architectures. Centralized, statically dimensioned dedicated resource based architectures like Internet Data Centers are commonly used as hosting platforms. The key advantage of such dedicated resource based architectures is that all of its resource can be directly monitored and controlled from a centralized location. However, these systems are statically dimensioned, which means accurate resource requirement analysis should be done on the services before provisioning them. Accurate requirement estimation is an especially hard problem for services with geographically dispersed users. As a result, such platforms are often unable to handle highly varying workloads of many applications.

To overcome the above problem, static allocation with over-provisioning or dynamic allocation can be used. For services that have variable resource demands, however, dynamic allocation is desirable. With the presence of planetary scale communications enabled by the Internet, different types of resources can now be shared among a widespread community for various types of applications. This allows new models for service hosting that bring resources together in hitherto unused ways. One such approach is to consider a large collection of *public resources* that are opportunistically available without any service contract, in conjunction with a small set of *dedicated resources* that are contracted or privately installed.

In this research, we propose a new architecture for hosting platforms built from a combination of public and dedicated resources and we refer to it as a *bi-modal architecture*. Because the dedicated resources are contracted, they are expected to have reliable and predictable behavior. The public resources are used *opportunistically* without any contracts and are not statically dimensioned. A huge collection of such public resources results from unused capacity of computers, networks and storages across the Internet. Volume of

such public resources grows with the growth of number of users or the size and spread of the network. This allows true on-demand provisioning because the resource collection is self-scaling with the demand. With its hybrid organization, the bi-modal hosting platform provides several benefits not available with clusters or IDCs. For instance, they provide self-scaling, geographically distributed points-of-presence, increased utilization for dedicated resources, and high compliance levels for service SLAs.

The bi-modal combination of resources can be attained for several types of resources to support various applications. For example dedicated computing servers may be used in conjunction with idle capacities of user computers to support high-throughput computing applications. Dedicated file servers may be combined with shared storage spaces in user computers to create a scalable and highly available storage platform. The best-effort connectivity of the public IP network may be combined with leased or privately installed dedicated links to serve high quality stream processing. Unused uplink bandwidth of end-user nodes may be combined with large bandwidth of dedicated streaming servers, to serve real time TV streaming on Internet. Among several possibilities, we have explored the bi-modal combination of computing resources for high-throughput computing and communication links for distributed stream processing.

The main objective of this research is to investigate the ways to create quality assured services in such bi-modal service hosting platforms built from combination of dedicated and public resources. Devising strategies and algorithms for the resource management system to schedule client requests to appropriate resources is the key to attain quality assured services. Our investigation will cover scheduling, fault-tolerance and resource management strategies for both centralized and federated architectures of the resource management system. Results from our simulation based studies reinforce the benefits of the idea of augmenting dedicated resources with opportunistically available uncontracted public resources, in several dimensions.

1.2 Problem Definition

The basic question we attempted to answer in this thesis research is that whether it is beneficial to use combination of dedicated and public resources in service hosting platforms.

Once the benefit is established, the next issue is how to organize these two types of resources and a resource management system to manage them such that the combination can be exploited in a meaningful way. Given a suitable architecture, the next problem is to devise resource management policies and algorithms to allocate resources to serve the demands.

Although the idea of bi-modal architecture is generic and independent of resource types and applications, the management of resources need to fit with the specific characteristics of different types of resources as well as application demands. In this thesis, we have explored two different types of resource combinations.

In the first case, dedicated compute servers were combined with opportunistically available public computers to serve high throughput computing applications. We have tried to answer several key questions: a) whether a centralized or distributed resource management system would be suitable for such platform, b) whether and how to aggregate the status information of the remote resources, and c) how to utilize the status information for scheduling decisions.We examined the preemptive migration based job scheduling is a feasible strategy for such platforms and what overheads are caused by such a strategy.

In the second case, we explored a combination of public network links with privately installed or leased dedicated links to support distributed stream processing applications. A stream processing task delivers one or more streams of data to one or more destinations after several steps of processing on the data streams. We restricted our study to tasks carrying out the processing on a single stream in a linear array of steps. One major issue in serving such streaming tasks is to map the processing components on different stream processing servers, fulfilling both computational and data transmission capacity constraints. Even after having an efficient algorithm to solve the mapping problem at hand, concurrent arrival of streaming task requests and the variability of the public network bandwidth, demands for adaptive re-allocation of the resources by continuously monitoring the progress of the streaming tasks. The main goal of the adaptive resource management system for the platform that combines the two classes of communication resources is to keep the utilization of the expensive dedicated resources high and to keep the adherence to the target delivery rate of the streaming tasks high.

1.3 Thesis Contributions

In this thesis, we introduce a novel bi-modal architecture for distributed service hosting platforms and investigate the resource management problems in such a platform architecture. The key aspect of the architecture is that the most critical resources used by a certain type of application served by the platform are organized in two different classes – *dedicated* resources that are owned and fully controlled by the platform provider and *public* resources that are not under control of the platform and have their characteristic erratic behavior.

Such dedicated versus public resource combination can be attained for several different types of resources to create platforms that support different types of applications. The resource management policies depend on the characteristics of the resources as well as the desirable performance objectives of the applications. We have methodically explored the bi-modal combination of two different types of resources, supporting two different application types. In the first case, we examined how to manage public computing resources in conjunction with a dedicated privately installed compute-server cluster to serve high-throughput computing applications. In the second case, a distributed set of dedicated servers were connected using a combination of overlay links on public Internet and a small number of privately installed dedicated links, to support distributed stream processing applications.

For the bi-modal high-throughput computing platform, we explored different alternative organizations of the resource management system and developed scheduling algorithms to manage compute-intensive jobs in each organization. The public processing units in these platforms are geographically distributed and assumed to be network accessible. We devised a centralized resource management system that schedules the job requests on the two classes of processing units – the dedicated computers installed in a centralized cluster and the public computers in distributed locations. Aggregation of state information is a hard problem in such a geographically distributed deployment of resources. So, we explored two different scenarios, in one of which the resource manager performs the scheduling activities in a state oblivious manner and in the other, periodic status update is enabled. The trade-off between informed decision and cost of information aggregation is thoroughly investigated. Preemptive migration of jobs from one processing unit to another, although found to be a

very desirable feature, potentially costs a huge amount of communication resources. So, presence and absence of migration is also evaluated in those two studies.

In the distributed stream processing platform, the communication links are organized bi-modally, because they are the most critical resources for the data-intensive stream processing applications. Distributed stream processing requires simultaneous management of computational and communication resources. In the proposed platform architecture, the server nodes that serve different stream processing services, are privately installed dedicated computers. The communication network that interconnects them consists of two different types of links – some privately installed or leased dedicated links and some overlay paths through the public Internet. Resource management in this platform is divided into two phases. In the first phase, a requirement specification of a composite stream processing task is mapped on the server nodes and communication links, fulfilling the computing and transmission capacity requirements. Finding an optimal mapping of the requirement, subject to the given capacity constraints, is a computationally expensive problem. We analyzed the problem in details, outlined centralized and distributed algorithms to solve the problem and developed some heuristics to find out workable solutions in a cost-effective way. In the second phase, the allocation of communication links to the tasks are dynamically altered, due to the inherent variability of the public network links. We performed detailed simulation based evaluation of the scheduling schemes. The results show that considerable synergy can be extracted from the combination of opportunistic and dedicated communication links, and higher utilization of the expensive resources and higher service quality for the distributed stream processing tasks can be attained.

In summary, we have made the following key contributions in this thesis –

- Introduced a bi-modal architecture for service hosting platforms.
- Explored combination of public and dedicated resources for two different types of resources and applications – computing resources for high-throughput computing applications and communication resources for data intensive stream processing applications.
- Explored how the usage of public computing resources can be leveraged to develop a commercially viable hosting platform for compute-intensive applications.

- Examined alternative resource management policies and evaluated the benefits of job-migration and status aggregation in the scheduling process.
- Developed scheduling heuristics and evaluated them through detailed simulation models of the platforms.
- Analyzed the problem of mapping resource requirements of a distributed stream processing task on a network of servers, subject to node and link capacity constraints
- Developed centralized and distributed algorithms to find optimal solution to the mapping problem and proposed several heuristics to minimize the run-time cost of these algorithms.
- Demonstrated the benefits of using a combination of public and dedicated networks links for higher utilization of server and link resources and higher quality assurance for stream processing tasks.
- Developed algorithm for dynamic re-allocation of network links to achieve the benefits of bi-modal network organization.

1.4 Thesis Roadmap

The rest of this thesis is organized as follows. In Chapter 2 we provide some background discussion on service oriented architectures, hosting platforms and a classification of applications that can be served by different hosting platforms. The proposed bi-modal system architecture is described with all its components in Chapter 3. In Chapter 4, we define the resource management problem space pertinent to the proposed architecture that we explore in this thesis. Chapter 5 gives an overview of the existing literature relevant to the problems we tried to solve.

Chapter 6 and Chapter 7 describes the resource management policies and algorithms we developed to manage the processing resources from dedicated and public pools. Chapter 6 deals with the simpler system that lacks features like state-aggregation and capability of migration. The solutions developed in Chapter 7 assume presence of these features and

tries to exploit them. In both cases, the efficiency of the algorithms and the cost-benefit trade-off for different features are explored thoroughly through detailed simulation studies.

When data-streams are processed through multiple processing components, mapping of this components on appropriate processing unit satisfying the CPU-capacity and communication bandwidth constraint becomes an important problem to solve. Analysis of this mapping problem and algorithms to solve the problem, both optimally and approximately, are presented in Chapter 8, along with simulation based performance evaluation of the algorithms.

In Chapter 9, we present the study of different scheduling policies for communication resources of two classes – the public overlay links and the dedicated lease lines. Detailed simulation results are presented in support of performance of the scheduling policies and the benefits of combining two types of resources.

A summary of the thesis with highlights of the contributions is presented in Chapter 10. Possible future extensions of this research is also briefly indicated in the same chapter.

1.5 Published Articles

Various parts of this thesis have been previously published in the following journal and conference articles –

1. S. Asaduzzaman and M. Maheswaran, "Distributed Stream Processing on Network Computing Platforms with Dedicated and Opportunistic Resources" , *22nd IEEE International Parallel and Distributed Processing Symposium*, April 2008, submitted for review. (Chapter 9)

2. S. Asaduzzaman and M. Maheswaran, "Strategies to Create Platforms for Differentiated Services from Dedicated and Opportunistic Resources" , *Journal of Parallel and Distributed Computing*, 67(10), pp. 1119-1134, 2007 (Chapter 7)

3. S. Asaduzzaman and M. Maheswaran, "Utilizing Unreliable Public Resources for Higher Profit and Better SLA Compliance in Computing Utilities" , *Journal of Parallel and Distributed Computing*, 66(6), pp. 796-806, 2006 (Chapter 6)

4. S. Asaduzzaman and M. Maheswaran, "Towards a Decentralized Algorithm for Mapping Network and Computational Resources for Distributed Data-Flow Computations" , *21st IEEE International Symposium on High Performance Computing Systems and Applications: HPCS-2007*, May 2007, Saskatoon, SK, Canada (Chapter 8)

5. S. Asaduzzaman and M. Maheswaran, "Heuristics for Scheduling Virtual Machines for Improving QoS in Public Computing Utilities" , *9th International Conference on Computer and Information Technology :ICCIT-2006*, December 2006, Dhaka, Bangladesh (Chapter 7)

6. S. Asaduzzaman and M. Maheswaran, "Leveraging Public Resource Pools to Improve the Service Compliances of Computing Utilities" , *Springer LNCS(3296): Proceedings of IEEE/ACM International conference on high-performance computing (HiPC)*, pp. 242-251, December 2004 (Chapter 6)

7. M. Maheswaran, B. Maniymaran, S. Asaduzzaman, and A. Mitra, "Towards a Quality of Service Aware Public Computing Utility", *Adaptive Grid Computing Workshop, IEEE NCA* , Cambridge, MA, August 2004 (Chapter 3)

2

Background on Services and Hosting Platforms

2.1 Overview

In this chapter we provide some background knowledge on service oriented computing and hosting platforms, that would be useful for understanding of the problems we have explored in this thesis. We also present a taxonomy of applications that may be deployed on different service hosting platforms and identify the classes of applications that may take advantage of the proposed combination of dedicated and public resources. To set the context of our research achievements, a brief taxonomy of the existing architectures of service hosting platforms is also presented.

2.2 Service Oriented Computing and Hosting Platforms

With the growth of popularity of the World Wide Web along with growing number of active web pages performing some intelligent data processing tasks, the idea of interoperable software components, that can interact among each other and compose into larger distributed application, emerged. Such software components are often termed as *services*, because they provide well defined functionalities through well defined interfaces. Because interoperability is a key issue for such services, much research have been dedicated to standardizing service interfaces and service description languages to facilitate discovery of services in distributed heterogeneous environments.

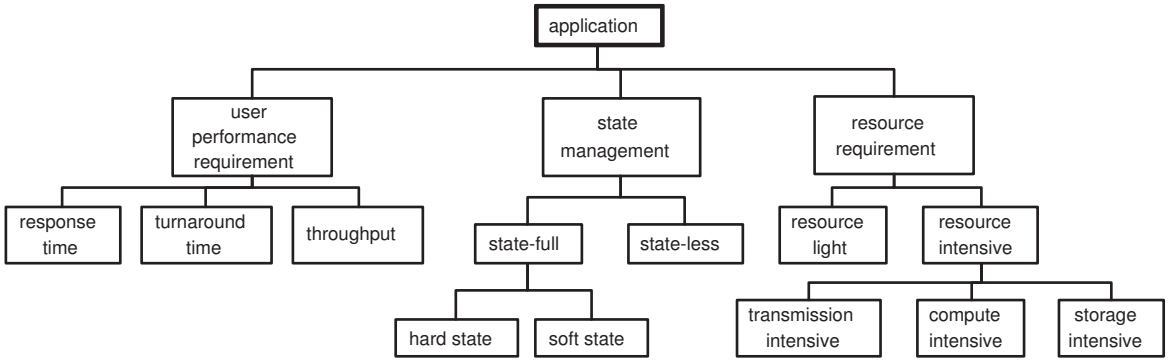

Figure 2.1: Application taxonomy

To achieve the best out of these services, they need to be hosted in widely accessible platforms that is able to manage necessary resources for execution of the services. Such deployment platforms aggregate and manages different types of physical resources and provides some form of performance guarantee for the services. Architecture of the hosting platform can be centralized or distributed. Some of the core functionalities of such hosting platforms include resource discovery, resource allocation, access control and trust management.

2.3 Application Taxonomy

Several design decisions for a distributed and shared resource based hosting platforms are governed by the characteristics of the applications that actually use the system. For hosting purposes, we propose that and application can be characterized along three different axes – *user performance requirement*, *state management*, and *resource requirement* Figure 2.1 gives a brief taxonomy of distributed applications based on this proposal.

Some of the common user performance requirements include *response time*, *throughput*, and *turnaround times*. Response times are essential in interactive applications such as online games. User requirements for some applications such as media streaming are throughput bounded. In case of real-time applications, the total turnaround time might be a critical issue. The user performance requirements impact the choice of resource scheduling algorithms.

Applications can be either *stateless* such as web servers for static documents, or *statefull* such as a network game server. For stateful applications, the volume of state information is an important factor because this affects the amount of time needed for preemptive reallocation of resources. For example, a large weather simulator may initialize a large number of variables in memory and open a number of data files is harder to relocate than a simulator performing a long series of computations on small set of data. The deployment protocol of the stateful application may be based on *soft state* or *hard state*. In case of hard state based systems failure of any node may create an inconsistent system state, whereas soft state deployments can quickly adapt to the failures because they can reconstruct a new consistent state based on available information.

In terms of resource requirements, we classify applications as *resource light* (very low resource requirements) and *resource heavy* (very high resource requirements). Resource heavy applications can be further divided into *compute-intensive*, *data-intensive*, and *transmission-intensive* based on the type of resources requested by the application. The service model is generally considered appropriate for resource heavy applications. For an efficient implementation, the hosting platform should carefully manage the resources heavily used by the application. For transmission intensive applications such as media streaming, communication bandwidth is the most critical resource, the resource management strategies of service platforms for such applications need to be implemented by the routers or the nodes that play a role in allocating bandwidth on an end-to-end path. For compute intensive applications such as protein folding or molecular model simulators, bandwidth is abundant, but the platform need to carefully allocate the available processing capacity to the applications. Data intensive applications require bulk data storage with a desired degree of reliability. The application needs reliable method of performing transactions on databases and maintaining the consistency of large volume of data.

2.4 Hosting Platform Architectures

We can classify the service hosting platform architectures based on two aspects of the system: organization and usage. In terms of organization, we can group the systems into centralized and distributed. Similarly we can group the systems into dedicated and shared

Table 2.1: Characterization of different service hosting platform architectures

Issue	Dedicated Resource Pool	Central Shared Resource Pool		Distributed Shared Resource Pool	
		Small	Large	Deterministic Peering	Opportunistic Peering
Installation Cost	increases with size	low	high	low (harder to peer)	low
Running Cost	depends on size	low	low	low	lowest
Resource Exhaustion	depends on size	definitely	no	could be	no
Distributed Point-of-presence	depends on organization	no	no	yes	yes
Scalable	no	no	yes	yes	yes
Performance Isolation	high	low	high	medium	high-medium
Ease of Use	high	medium	medium	low-medium	low-medium

based on usage. In Table 2.1, we have characterized each class of architectures with respect to several design issues. Dedicated resource based architectures are high cost installations of reliable resources and they usually results in either under-utilization or job-overflow, depending on the volume of resource installed. Scheduling applications for such resource pools is comparatively easier task and because the resource behavior is deterministic, it is possible to provide very good service isolation. To overcome the under-utilization of dedicated platforms, one usual approach is to share the same set of resources among different applications, expecting that their peak loads will not overlap and thus yield more utilization.

The bi-modal hosting platform proposed in this thesis falls into the distributed shared model. This model can be divided further into two subclasses: with deterministic peering

and opportunistic peering. When resources installed in different administrative or geographic domains are shared among the service providers based on some off-line contracts, we call it deterministic peering. In such systems the available capacity of the resources to particular provider is deterministic. Globus [42] and other Grid like systems may be classified into this group. But it is often very hard to establish such peering among diverse communities due to the need for offline contracts. An alternative method is to opportunistically share the available resources among the providers without any assurance. Computing platforms such as BOINC [4], Condor [89] may be classified in this group. Although this kind of peering is easy to establish, it is really hard for the platform to provide any quality assured service for its users based solely on such opportunistic resources. The bi-modal architecture proposes for a combination of dedicated resource installations and opportunistic peering among distributed installations. Our results show that such combination can provide better resource utilization and service isolation, compared to any of them individually.

3

Proposed System Architecture

3.1 Overview

In this chapter we outline the architecture of our proposed bi-modal service deployment platform and present the relationship of the *resource management system* (RMS) with other components in the architecture. Then we elaborate on the various components of the RMS that manages the combination of dedicated and public resources.

As introduced earlier, we have explored two different dimensions of combined resource based systems. In one case we have two classes of computing resources and the existence of a well-managed communication network is assumed. The RMS here concentrates on appropriate scheduling of these processing units for compute intensive jobs. In the other case, the RMS is responsible for serving data-intensive applications such as multi-component processing of data streams, by appropriate scheduling of dedicated and public communication links. First, in Section 3.2, we present a generic architecture that can contain the RMS for different classes of resources. The details are then presented for individual cases, the RMS for computing resources in Section 3.3 and the one for communication resources in Section 3.4.

Several issues need to be addressed in developing a full-blown resource manager for

the bi-modal hosting platform. These include (a) co-allocating different resources such as processing bandwidth, storage capacity, and network bandwidth, (b) using network proximities to derive efficient resource allocations that reduce the loading by the platform on the underlying network and at the same time reduce impact of network congestion on the QoS delivered by the platform (c) using trust measures of public resources to derive robust resource allocations, and (d) managing the incentives for the participating volunteer resources so that the performance delivered by such resources can be maximized.

In this thesis, we have explored the resource management issues to manage the bi-modal collection of computing and network resources, with an objective to maximize the quality assurance to the client applications. We have considered different architectures and applications, where either computing or communication resources are organized in bi-modal resource collections. The other problems such as location aware resource discovery, trust and incentive management, and Security and access control have been dealt elsewhere and out of the scope of this thesis. We refer the interested readers to [68, 67] for details on location aware resource discovery, [16] for details on Trust and incentive management, and [72] for security and access control.

3.2 Layered Architecture

In this section we describe a layered architecture of a complete bi-modal service hosting platform, named Galaxy [66].

The proposed architecture for the Galaxy is shown in Figure 3.1. Physical resources including processing units, storage units and communication channels are shown at the bottom of the architecture. The next layer is a P2P overlay network named *resource addressable network* (RAN). All the resources that participate in the platform plug into the

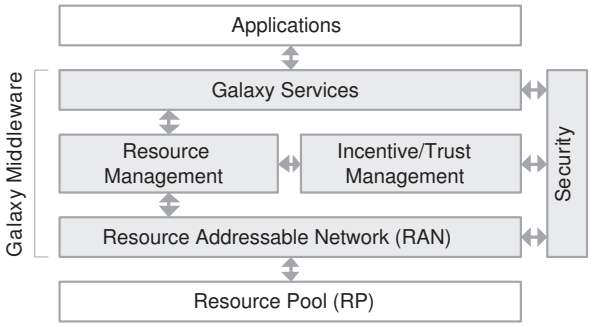

Figure 3.1: The Galaxy Architecture.

RAN. The overlay network is constructed to facilitate location and quality aware discovery of resources. Resource naming, discovery and access are among the services provided by the RAN layer.

In the next upper layer, we have placed the resource management as well as the trust and incentive management modules. The resource management module, named *Galaxy resource management system* (GRMS), is responsible for organization and appropriate allocation of the resources to competing requests. Solving various problems related to the activities of the GRMS module constitutes the core of this thesis. The GRMS may be organized either as a centralized entity or as a collection of distributed entities acting collaboratively for a global objective. In two different cases of resource management we have used in this thesis, the one for managing computing resources are designed as a centralized RMS, whereas the one for managing communication resources are designed as a federated RMS, due to inherent distributed nature of the communication resources. It is also possible to design federated resource management schemes for computing resources, where multiple resource brokers collaborate to meet the requests from users and share resources from different administrative domains.

The module for trust and incentive management is placed at the same layer as GRMS,

because they collaborate with each other. This module computes the trust values for different resources based on their history of accomplishments in performing the assigned jobs and assigns incentive tokens to the resources according to the trust values. The resource manager works hand-in-hand with the trust manager, by prioritizing requesters with higher incentive tokens in presence of multiple competing requests. Also, the trust values of the resources are exploited in the scheduling algorithms to achieve better quality assurances for the services.

The topmost layer in the Galaxy system is composed of services that directly support the user applications. The architecture does not impose any restriction on the organization of this layer. Example Galaxy services include application level QoS managers, shell interfaces, and network file systems. The security layer in the Galaxy architecture spans all the other layers in parallel, to protect the system from malicious activities, both external and internal to the system.

3.3 Bi-modal Organization of Computing Nodes

In this section, we describe the model for the different architectural components of the RMS of a bi-modal service hosting platform that serves compute-intensive applications. We refer this computing resource based platform as *public computing utility* or PCU. The computing resources come from a bi-modal combination of resource pools. Statically dimensioned clusters of reliable dedicated computers are augmented with opportunistically usable public computers that are available in large numbers in distributed locations.

Several large scale network computing applications including *peer-to-peer* (P2P) file sharing systems such as Gnutella [80], voice conferencing system such as Skype [21] and

volunteer computing systems such has SETI@Home [7] have demonstrated the tremendous potential of idle power of desktop computers. The idle capacities of the dedicated computers provisioned by other PCUs may also be used opportunistically during off-peak periods. As in any distributed system, managing this large volume of resources involves three mutually interacting entities - the resources, the resource consumers or the jobs and the resource manager. In the following subsections, we characterize these components and discuss the issues related to them.

3.3.1 Organization and Characteristics of the Computing Resources

The dedicated computers are privately installed and the PCU provider has full control over these resources. On a global scale, of a number of PCU providers may co-exist. Each PCU provider may deploy one or more clusters of dedicated resources, in one or more geographical premises depending on the number of its points of presence. In this thesis, we restrict our studies within the scenario of a single PCU provider deploying a single cluster of dedicated computers. These dedicated computers are highly reliable and their availability extends to almost hundred percent. In addition, we assume that these resources are homogeneous at least within a cluster. The PCU provider has a client base that registers with it in order to get quality assured services. The clients may either purchase the service in exchange of money or redemption of credits earned by donating idle periods of their own resource.

The public resources are opportunistically available idle capacities of computers owned by others. These may be user desktops across the Internet or unused capacity of the clusters installed by other PCU providers. The major difference of these resources from the dedicated ones is that no period of allocation can be reserved on them, because any PCU

job may be preempted if the resource is required by its owner. Although, in theory, any computer from the whole Internet can be used as a public resource for a PCU, we assume only a subset of these resources that reside within a given diameter defined by network delay and bandwidth constraints will be used. It is possible that multiple PCU providers are close to each other and there is contention for public resources. However, resolving these contentions is a separate issue and a topic of further research. In this thesis, we limit our scope to explore the problem of how a single PCU can manage a combination of dedicated and public resources to provide seamless service to its clients.

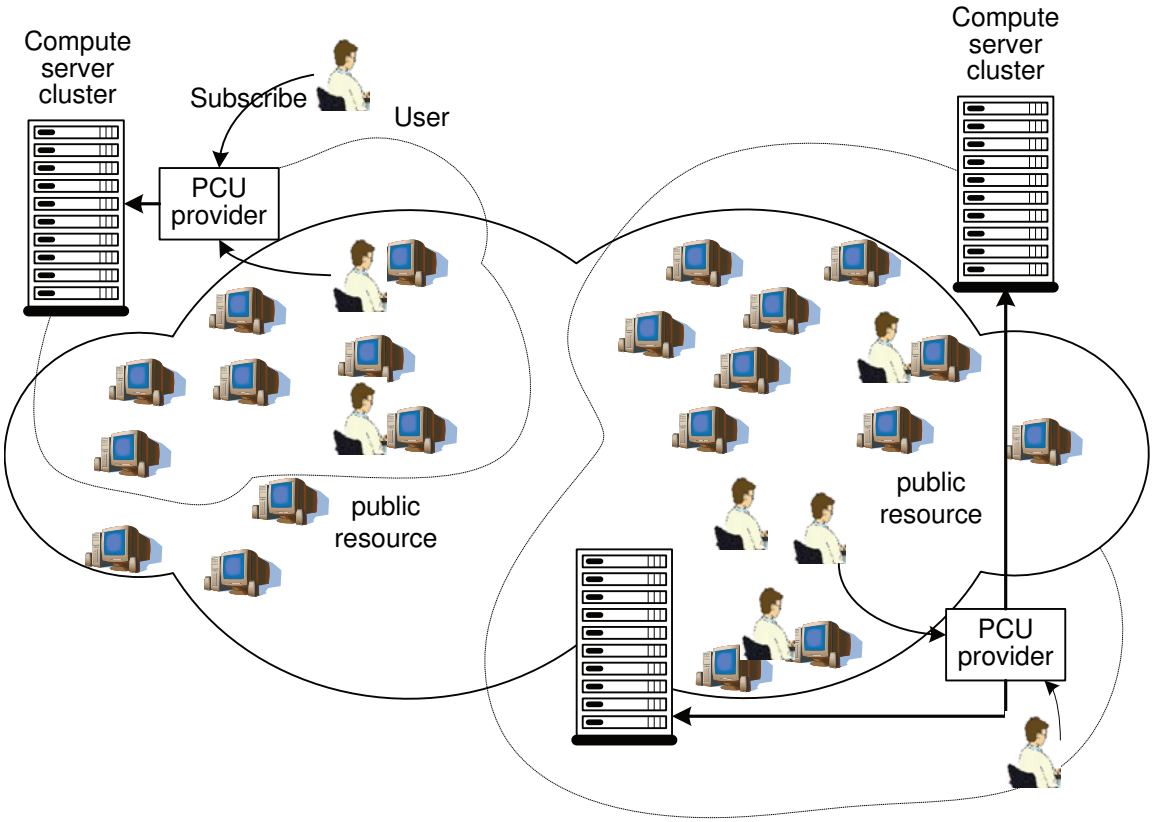

Figure 3.2: An illustration on formation of PCU, by augmenting one or more dedicated cluster of compute servers with public resources and serving its subscribers with quality services

In the current Internet, ISPs are very good candidates to become PCU providers. This is because they already manage the network access for their clients through installations in points of presence and they can easily install dedicated resource clusters in those premises. In addition, they can utilize idle computers from their network clients as public resources, possibly in exchange of some credits. The PCU service can then be sold profitably to subscribers, both from its own network and from outside. Figure 3.2 illustrates the formation of a PCU from public resources available on Internet by augmenting privately installed dedicated cluster of compute servers by a PCU provider.

There can be a wide degree of variability in the public resources, both temporally and spatially. In the spatial dimension, the spectrum of benchmarked capacity of the resources spread from very low capacity desktop machines to high-end multiprocessor machines. Existing data regarding the actual distribution of the resources on the Internet is inadequate. To build a model representative of the actual resources, we used the resource capacity statistics from the projects that use the BOINC software [4]. For the purpose of simulation study we derived an empirical distribution from these statistics shown in Figure 3.3.

In the temporal dimension, the availability of the public resources goes on and off according to the user behavior. Several studies have attempted to record the user idleness and machine availability characteristics in different settings [73, 32] and statistically modeled the distributions [73, 28, 94, 32]. The effect of availability on job execution time distribution have been studied in [53]. In addition to user activity, availability is also affected by system software failure, hardware failure and network disruptions. Although different studies have attempted to model particular aspects of failures, a complete model of public resources is not available yet. For our purpose, we modeled the life of a resource as

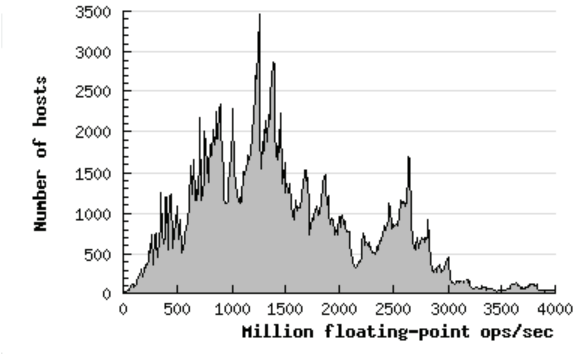

Figure 3.3: Histogram of frequencies of hosts of different floating point capacities (FLOPS) that are participating in SETI@Home project across the Internet [5]

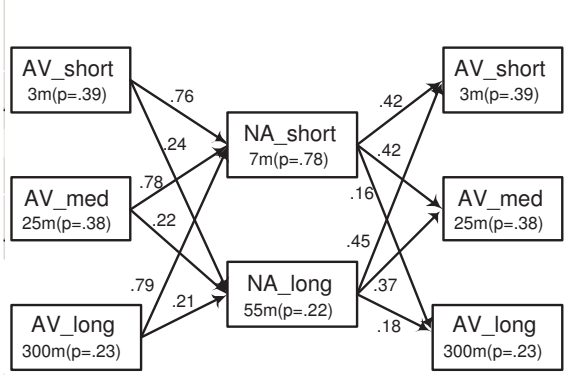

Figure 3.4: Markov model for host availability characteristics [73]. AV=Available, NA=Not Available. Numbers inside the states are state duration and steady state probability respectively

a Markov process that goes through different states of availability according to the transition probabilities and state durations described in [73]. A resource can fail due to various reasons and we assume that whenever a resource recovers after a failure, it is impossible to retrieve the work done by the job on that resource before failure. However, jobs checkpoint their progress periodically, so the job can be re-instantiated from its last recorded checkpoint.

Although a single physical resource can be time-shared by multiple jobs, we assume that each resource, whether dedicated or public, is actually a virtualized unit of resource that executes a single job at a time. It is possible that single physical resource is virtualized into multiple virtual resources, and thus multitasking can occur. The scheduler sees each virtual resource as an independent entity.

3.3.2 Organization of the Resource Manager

As we mentioned earlier, the RMS of the PCU can be organized either in a centralized or a distributed manner. Because we are considering the management of a single cluster augmented with the public resources, a centralized RMS is better suited for controlling the dedicated resources in the cluster. In this centralized organization, the resource manager runs in one of the dedicated computers in the cluster deployed by the PCU provider. PCU *users* (subscribers) submit their jobs to the resource manager for execution and wait for the response. The resource manager is responsible for dispatching all the jobs to appropriate resources, monitoring their progress and, if necessary, rescheduling them on new resources. When a job is spawned on a resource, it is enwrapped in a virtual-machine (or a more lightweight process wrapper) that can measure the resource consumption by the job and communicate the status to the central resource manager.

Scheduling and rescheduling being the main responsibility of the resource manager, it may deploy different queueing and prioritizing policies among jobs in order to optimize its service objectives. One of our goals is to design appropriate policies for scheduling and investigate their effectiveness with respect to the proposed architecture.

Due to very negligible communication delay within the local network that holds the cluster, the resource manager effectively has the current state information of all the dedicated machines. We have explored two different strategies for managing the public resources and the jobs executing on them. Because these public resources are dispersed in geographically diverse region, there is significant communication overhead to aggregate the state information into the central resource manager. Also migration of jobs from one resource to another causes similar overhead. So, in one of the organizations, status aggregation and migration is enabled, and in the other, they are not.

In the case where monitoring and migration is disabled, the RMS gets the notification only when a job assigned to a remote public resource successfully completes. Otherwise, the scheduler either has to restart the job in a dedicated resource or report the failure to the user. In case of monitoring and migration enabled organization, the resource manager relies on periodic status updates from the public resources and include the progress information of the remote jobs in the scheduling decisions.

3.3.3 Users and SLA

PCU can serve a wide range of subscribers or users, coming from both its public resource donors and the outside world, and the users may require the service at different quality levels. The users who require preferential service from the PCU provider, sign SLAs to ensure that service. The SLA for each user specifies four parameters – the maximum workload V_0, service ratio ρ, $0 < \rho \leq 1$, a time window τ to measure the offered load and delivered capacity, and maximum permissible elongation e_{max} for a job.

The parameter V_0 specifies an upper limit on the workload a user can request the PCU to execute, in terms of resource consumption per unit time(e.g., MFLOPS). The PCU is bound to deliver resource capacity at the rate ρ min(B, V_0), where B is the actual workload requested. If the PCU is not compliant with this rule, it needs to pay penalty or rebate to the user in proportion to the degree of deviation. The user must keep its offered workload B within the maximum workload limit V_0 to be eligible for this rebate. The offered load B is measured continuously over the specified history time window τ. The delivered goodput V (defined below) is also measured over the same time window τ.

Each job submitted to the PCU has a nominal resource consumption rate, C (say, in FLOPS), which implies a nominal execution time T_0 if adequate resource is provided

throughout the execution time. However, given that the computational jobs are elastic, the PCU may deviate from delivering this exact capacity and the actual execution time may be T. We define the quantity $e = \frac{T-T_0}{T_0}$ as *elongation* for the job. The SLA parameter e_{max} is used the classify the completed jobs into two categories. A job counts towards revenue generation only if $e \leq e_{max}$. Otherwise, the PCU provider does not earn anything for that job.

We define the quantity *goodput* V, as the resource capacity delivered to a particular user counting the revenue-generating jobs only. The SLA enforces that $V \geq \rho \times \min(B, V_0)$. If the PCU deviates from this, then it should pay the penalty in proportion to $\max\{0, (\rho \min(B, V_0) - V)\}$. A natural objective of the PCU is to minimize this SLA deviation. So, this quantity is used as one of the metrics to prioritize among the jobs competing for high-throughput dedicated resources.

3.3.4 Job Characteristics

For this part of the study we consider compute-intensive jobs only. Each job may have concurrent threads, however, we assume them to be independent and do not consider any inter-thread communication. Many of the bag-of-task scientific applications fit in this criteria [63, 8]. Each job is launched in a process wrapper that is able to take a snapshot of the job's progress periodically and store the checkpoint. Also, the wrapper is able to migrate to a newly allocated resource based on instructions from the scheduler. To minimize the down-time, the migration is done in two phases, first the bulk of the process's memory and filesystem footprint is transmitted while the process is live, and then the process is frozen and residual footprint is migrated. This technique has been successfully tested recently with Xen virtual machines [33].

3.3.5 Migration and Virtual Machines

When we enable monitoring, checkpointing and migration, there are several important concerns about live migration of the jobs being executed in a remote machine. One issue is to encapsulate the job so that the interaction between the remote job and host kernel is done in a controlled manner. Besides other techniques, virtual machine can be used for encapsulation. There have been several research works on efficient implementation of virtual machines and almost local equivalent performance is now achievable.

Although virtual machine is the best option for secure isolation of the guest processes, one problem is that a full-blown virtual machine is too heavyweight for migration. Considering this fact, several research projects have developed efficient process migration techniques that can successfully live-migrate server applications [70]. However, virtual machine based techniques are gaining popularity because of the residual dependency [49] and infrastructure incompatibility [71] problems of process migration. Virtual machines can now be live-migrated that greatly reduces the actual down-time of the applications. Recently, heavily loaded web severs and game severs running on Xen virtual machine has been live migrated with down time in the order of 100ms, although the total migration time spans over $2 - 3$ minutes [33]. Still we believe, to enable deployment of PCU, further research need to be done on virtual machine techniques to achieve lower memory footprint along with guest process isolation. Such lightweight techniques are also important for the widespread availability of virtual machines on end-user desktops. With the introduction of hardware assisted virtualization mechanisms in the mainstream processors such as Intel [59] and incorporation of VM monitors into the common desktop OS kernels [57], widespread availability of VM enabled desktop machines can be expected in near future.

3.4 Bi-modal Organization of Communication Links

The system we describe in this section was designed as a proof of concept for utilizing combination of dedicated and public communication links in data intensive application. Distributed processing of data streams have been found in many applications such as processing continual query on sensor data streams [56] or multi-step processing (encoding, decoding, embedding) of multimedia streams [48, 92]. Appropriate allocation of both computing and communication resources are critical for such applications. Several architectures and allocation algorithms have been presented in the literature [56, 92, 48, 61, 85, 38] for management of these resources for stream processing. However, we are proposing a system that exploits the combination of leased or privately installed dedicated links and opportunistic usage of links on public IP network to provide service quality assurance such as data-rate guarantee to the streaming applications, which has not been studied yet.

In the following subsections, we explain the characteristics and organization of the constituents of this bi-modal stream processing system.

3.4.1 Application model

As mentioned above, the system is built to support multi-hop processing of continuous data streams. The basic criteria of a stream processing task is that one or more data streams originating from one or more data sources are to be delivered to a receiving node after processing and aggregation through multiple stream processing services. A service component may perform any particular type of processing on the data stream. For example, it may serve as a database operator or an embedding or encoding engine for a video stream.

Although it is technically possible to have all the service components installed in a

single server and perform all the processing in a single node, we assume that these components are developed and deployed independently and then advertised in the platform. As a result, different components necessary for processing a single composite task is found in different distributed locations. Also, the data source and the requested delivery destination is not at the same location in most cases. These two facts suggest that different steps of the processing are performed in different server nodes along the path of delivery to the destination.

The components may be arbitrary services on data streams, but we assume that each service consumes server resources, i.e. CPU and memory, proportional to the input rate of data it processes. The proportionality constant is assumed to be characteristic of each type of services. Output data is generated as the processing goes on the input stream. The ratio of output data rate to input data rate is also characteristic to a particular type of service. Thus, processing through a service component may result in either shrinkage or expansion of the data stream based on its characteristics.

The topology of data flow in a stream processing task may be as complex as a directed acyclic graph with multiple sources and multiple destinations. However many interesting processes resemble a simpler topology like a path or tree [61]. Due to the inherent complexity of the problem of mapping a DAG on an arbitrary resource graph, as explained in Chapter 8, we restrict the topology within simple paths and leave the extension of the ideas for more general topologies as future work.

3.4.2 Organization and Characteristics of the Link Resources

Because the service components are developed and deployed independently, we assume the existence of multiple servers spread across diverse geographic locations. Each of these

servers are connected to the Internet, with some specific uplink and downlink capacity. A single server may serve several types of services and any particular type of service may be served by multiple servers.

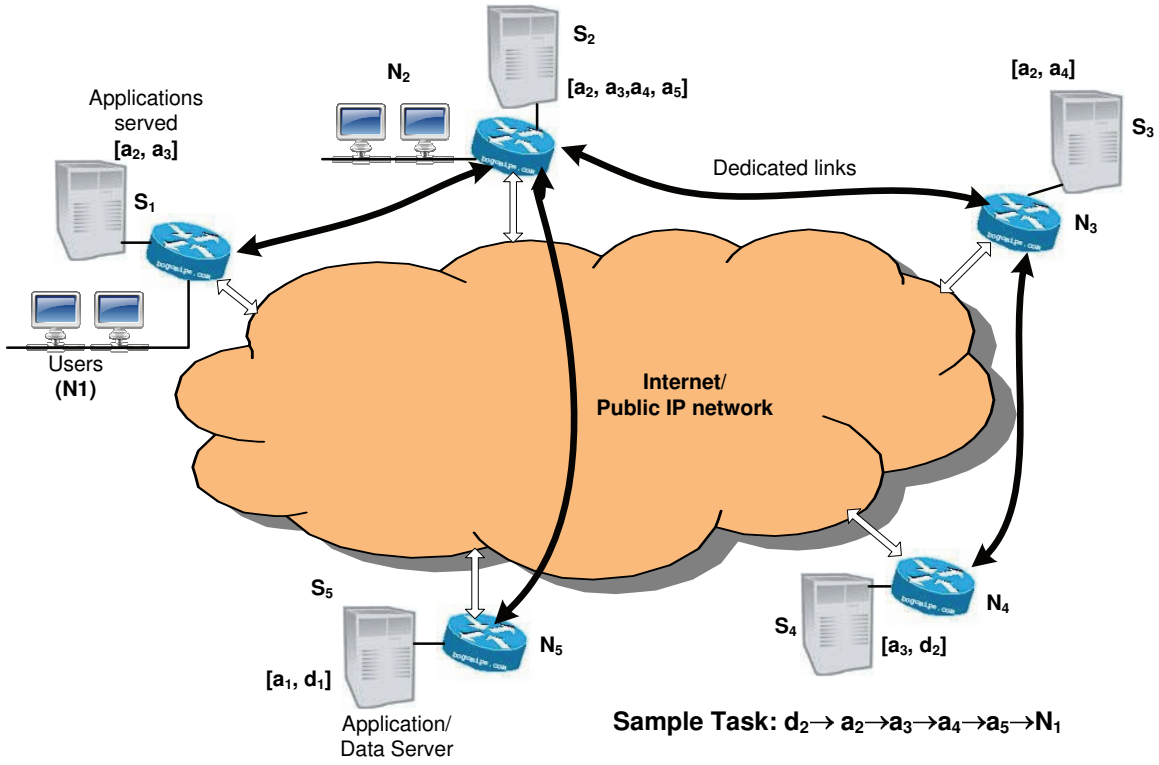

Figure 3.5: Illustration of a distributed stream processing platform containing five application/data servers interconnected with public network as well as dedicated links

Figure 3.5 illustrates a scenario of a stream processing platform containing five servers. All these servers run on dedicated computers. Their total processing and memory capacity is accurately known and their allocation can be controlled solely by the resource manager of the platform. Each service component may need variable amount of processor time and memory. Aggregated chunks of all types of node resources are allocated to a processing component of a particular task, depending on the input flow rate for that task. For this purpose, we assume that a virtualized processing unit, such as a worker thread, is created

that can handle unit rate of input data for a particular service type. The platform agent decides the number of processing threads required for a particular process. The fine grained allocations of different node resources are handled by the local operating system.

Given their connections to the Internet, any server can communicate to any other server as necessary and send data stream to any other server. However, the capacity of the uplink and downlink connections at the two ends limit total rate of transmission. Moreover, since routing of data through such IP-overlay paths are not under control of the stream processing platform, the actual data rates of the flows through such paths vary continuously over time [91]. Thus, a server can only allocate the uplink capacity to competing stream processing tasks, but cannot guarantee the ultimate rate of transmission over a period of time.

To supplement this variability in the IP-overlay flows, some pairs of servers may establish dedicated connections between them. Transmission volume through such links are totally controlled by the sending end of each link. So data can be transmitted at guaranteed rate through these links.

We assume a business model where a server node charges each task proportional to the data volume processed by a service component served by the server. The total cost of executing a task is thus the sum of the amounts payable to all the participating component servers. This cost model suggests a natural growth model for the dedicated link network. The server that hosts many components and have powerful computing resources to serve them, will naturally like to have as many flows as possible to pass through itself, in order to maximize utilization of its computing resources and its revenue generation. This implies that the higher capacity servers will take initiatives in establishing dedicated links with other severs, and eventually they have higher degrees of connectivity than others. This also

implies that the receiving end of the dedicated link is more likely to bear the installation cost, although equal sharing of the cost by two ends is also possible.

To make more utilization of the established dedicated links each server provides data forwarding service besides the processing services. As a result two servers running two consecutive component of the same task may use a data channel that passes through some other servers. Naturally, those other servers will charge for the forwarding service and for the usage of their own dedicated links. Therefore, multi-hop dedicated links are less preferred by the resource manager when alternatives are available.

A sample stream processing task shown in the figure 3.5 requests a data stream from data source d_2 to be processed through services a_2, a_3, a_4 and a_5, and to be delivered to a host in the network N_1. This task may be served by the servers S_4 (serving d_2), S_2 (serving a_2), S_3 (serving a_3 and a_4). Either dedicated link or public network link may be used to transmit the data stream between two consecutive servers.

3.4.3 Organization of the Resource Manager

The resource management system (RMS) of the stream processing platform is completely distributed in organization. The most critical resource the platform attempts to manage is the communication links and the links are inherently distributed in nature. Each of the server nodes participates in the collaborative resource allocation and scheduling algorithms. The RMS agent running in each node is responsible for proper allocation of the CPU capacity of that node and the communication links that originates in that node.

We divided the resource allocation and management for stream processing tasks into two steps. When a task is requested to launch by a user, the RMS agents in all the nodes collaborate to find the feasible mappings of the requested service composition to the available

resources in different nodes and the available communication bandwidth between them. The details of the mapping problem and possible alternative algorithms are described in Chapter 8, where we see that finding the best feasible solution is an NP-complete problem and we need to rely on heuristics that works in most practical cases. According to the algorithms, the discovered feasible mappings are aggregated and the best among them is selected for actual reservation of the resources. The selection criteria include load distribution among the nodes and usage of the dedicated links, among several others. The details of the selection process is described in Chapter 9.

After reservation of the resources along the selected mapping, the execution of the stream processing task begins. Due to variability in the flows through the public network links, the progress of the task deviates from the expectation. To handle this dynamic situation, a second part of the RMS agent in each node implements a dynamic scheduling of the flows passing through them on one of the three possible alternative types of routes. An output stream from a processing component served in the node may be allocated a direct dedicated link if available, may be routed through a multi-hop dedicated link if budget permits, or may be sent out through the public network link. The scheduler is invoked periodically in each server. The invocation is completely asynchronous across different nodes and any node can independently choose the scheduling interval. At each scheduling event, the scheduler decides which of the competing flows to put on which route based on some priority rules. Details of these scheduling policies are explained in Chapter 9.

Among the global objectives of the resource management system are maximizing the overall resource utilization and throughput, and balancing the processing load among servers. The RMS agent in each server node acts on behalf of the server itself and try to maximize

the benefit of the respective server. This is only natural with absence of any centralized authority. Our simulation results demonstrate how far the the global objectives are achieved through the local optimization policies of these decentralized RMS agents.

3.4.4 Users and SLA

In the stream processing platform, users launch requests for composite tasks. The requests are in the form of delivering some data stream originated from some data source node to the user after several processing steps. Naturally, the user resides in the receiving end of the streams. Since there is no central repository of user accounts, we assume that users are registered to any of the servers that can serve as a destination of a flow.

When a request is launched, the user specifies the data source node, the destination node, total volume of data to be extracted from the source, the series of the service types needed for processing of the stream and a required delivery data rate B. All these specifications form the service level agreement (SLA) between the user and the platform. When the platform accepts the task after necessary resource allocation, it is responsible to fulfill the constraints specified in the SLA.

Because each service type has its characteristic resource usage factor and bandwidth shrinkage factor, it is easy to calculate the CPU and link resource requirements for each hop of the composition. Given that many servers may serve any particular type of service component, it is the responsibility of the platform to map the requested components on appropriate nodes subject to fulfillment of the capacity requirements.

Besides defining the requirements, each task request is associated with a total budget per unit of data delivered. This budget factor translates into allocated budget for each component. This budget factor is used, along with deviation from the target rate, to determine

the priority of a task among competing tasks by the RMS agent while scheduling the output flow on different possible outgoing links. The budget factor also restricts the number of hops in a multihop dedicated link if such links are chosen.

To monitor the compliance with the requirement specification, the SLA also includes a time window T across which the delivery rate is measured for conformance. If the actual delivery rate in an measurement interval is $\hat{B}$, the platform is compliant if $\hat{B} >= B$ and the platform is entitled to full price of the service. Otherwise, the platform is penalized at a rate proportional to $\frac{B-\hat{B}}{B}$.

4

The Resource Management Problem in Bi-modal Architecture

4.1 Overview

In the contexts of the architectures of service platforms defined in the previous chapter, we now explore the different research problems in devising effective RMS for the platforms. Our main hypothesis is that the large volume of public resources available in the large scale public networks such as Internet, despite their uncertain behavior, can be opportunistically utilized to create systems that can provide commercially valuable quality assured services, if these public resources are augmented with a set of reliable and fully controllable dedicated resources. Viewed from the other perspective, while creating a large scale service hosting platform with reliable and predictable performance, the cost of deploying a large quantity of expensive dedicated resources, that remains under-utilized except during the peak load times, can be avoided by using the proposed bi-modal architecture. A small set of these controlled and reliable dedicated resources can be augmented by the public resources from the large scale networks and a large scale system with the desirable assurance in performance can be constructed at much lower cost of deployment. This also ensures much higher utilization of the privately deployed expensive resources.

In an architecture where all the resources are dedicated and controllable, managing the resources come down to scheduling the requested jobs to appropriate resources fulfilling the resource capacity constraints. The outcome of the mapping is certain. When the set of jobs arrive online, and the execution time of the jobs are not known a-priori, the lack of knowledge makes the scheduling problem harder. One more dimension is added to the scheduling problem by the uncertainty of the public resources. The scenario is more complicated when the architecture is bi-modal. When a set of dedicated resources is combined with a set of public resources, the resource manager has to decide on how to optimally allocate them to competing jobs such that some constraints mentioned in the SLA are fulfilled as well as utilization of expensive dedicated resources are maximized.

We attacked the resource management problem in bi-modal architecture in two different dimensions. As mentioned in the previous chapter, in one of them we have two classes of computing resources to manage and in the other case we have two classes of communication resources to manage. In the following sections, we discuss the details of the resource management problems particular to each of these two settings.

4.2 Managing Bi-modal Organization of Processing Nodes

As introduced in Chapter 3, we have designed a centralized architecture of the RMS for managing the two types of computing nodes in PCU, because the dedicated pool of computers are installed in a cluster. Users submit their computing jobs to the central resource manager and the main responsibility of the manager is to schedule these jobs on appropriate computing nodes. Globally, the the resource manager receives a series of requests for resources generated by the users across the PCU. In general, we assume that a dedicated resource always has more capacity than any public resource in the PCU. Since they

have predictable performance and they can be exclusively allocated by the scheduler, the dedicated resources are desirable for any job to achieve QoS objectives. However, given the finite number of high capacity dedicated resources, not all jobs may be entitled to a dedicated resource immediately. If there was no public resource involved in the platform, it would be a classical n-server single queue scheduling problem. Any solution to that would apply some priority rule on the queue and schedule the most prior jobs in one of the dedicated resources. The other jobs either remain idle in the queue or are dropped.

Because we have access to a large number of public resources on the Internet, the less prior jobs can get some progress running on these resources instead of sitting idle in the queue or getting dropped. Given the wide variations in the performance of public resources, this progress is quite unpredictable. To maintain the order of priority, it may be necessary to migrate these jobs back to dedicated cluster as soon as some resource becomes free. Due to the heterogeneous collection of public resources, the scheduling problem becomes harder and analytically intractable. Moreover, in addition to optimizing the per job performance objectives such as response time, the scheduler needs to consider the SLAs contracted with each user and attempt to minimize the penalty paid by the PCU due to deviation. Since status dissemination and migration of jobs create network overhead, the scheduler also needs to minimize the overhead by keeping the number of migrations minimum. When designing a scheduling heuristic for a particular architecture of the resource manager, all these possibly conflicting goals need to be considered.

Different scheduling scenarios may arise depending on the architecture of the resource management system (RMS). In order to investigate a range of architectural options we consider two different models of the RMS, with the status aggregation and job migration features disabled in one case and enabled in another case. In the following two sub-sections

we explain the scheduling problems specific to these two different cases – i) state-oblivious scheduling and ii) state-aware scheduling. It is also possible to have a completely decentralized organization of the RMS where an RMS agent is present in each computing node and act on behalf of the platform provider. However, we have limited our study within the centralized RMS architecture and leave the decentralized case for future research.

4.2.1 State-oblivious Scheduling

In the simplest case there is no status information or progress monitoring when a job is launched on a remote resource. Definitely, in such a case, a scheduler must be a centralized entity if it exists and possibly located inside the dedicated cluster. The only information the scheduler gets from the remotely executing jobs is the notification of termination or completion. In the absence of continuous or periodic status reports from remote public resources, the scheduler needs to statistically model the available capacity of public resources from the history information and sample the current level of availability from the model. However, the scheduler should be able to know the occupancy status of all the dedicated resources. The jobs are also incapable of checkpointing, therefore, whenever a job needs to be rescheduled in a new resource, it must be restarted. Even in this minimally capable model, our simulation studies suggest that it is possible to extract benefit from opportunistically available resources and improve SLA compliance and net revenue of the PCU provider. More detailed analysis of the problem, heuristic solutions and evaluation of performance of the heuristics are presented in Chapter 6.

4.2.2 State-aware Scheduling

The other model we studied includes a smarter wrapper for running jobs capable of progress monitoring, status reporting and checkpointing. The scheduler still works from a central location and a central repository of system state is maintained. Availability of progress status and the capability of migration allows rescheduling the jobs based on current level of progress. This also allows non-clairvoyant scheduling, where no prior estimation of the resource requirement from the user is necessary. An important issue in designing scheduling policies in this scenario is to minimize the overhead on the network due to status dissemination and migrations while maximizing the fulfillment of SLAs for each job and maximizing the overall job throughput. The detailed simulation study based on this model is presented in Chapter 7.

4.3 Managing Bi-modal Organization of Network Resources

The distributed stream processing platform introduced in Chapter 3 is designed to serve data intensive applications such as multi-hop processing of continuous data streams, where both computing nodes and communication channels need to be simultaneously allocated to fulfill the requirements of these composite stream processing tasks. Moreover, the network contains two types of communication links – some private or leased dedicated links and some overlay links over the public network. The public links have variable capacity and have uncertainty in their performance. So, dynamic re-scheduling of the communication channels is required to cope with this variability.

We divided the whole problem of resource management in the distributed stream processing platform into two parts – i) mapping of the requested composite task on the network

of resources and ii) dynamic monitoring and re-scheduling of the communication channels between any two components. Note that the server nodes are privately installed dedicated computers and they do not have variability in their performance. The mapping phase finalizes the assignment of the processing components to the server nodes and these assignments do not change over the course of execution of the stream processing tasks.

4.3.1 Mapping of the Composition

When a request for execution of a composite stream processing task is launched on the platform, the first task is to find out an appropriate mapping of the requested composition on the computing nodes and the communication links connecting them. As we explained earlier in Chapter 3, the data flow topology between the processing components may be an arbitrary DAG in the general case. However, we restrict our discussion within tasks of linear path topology only.

In the mapping problem, we need to find out the feasible mappings that satisfy the bandwidth constraints of the links as well as capacity constraints of the nodes. This problem is different from the problem of establishing a path between a source and a destination node in an arbitrary network, subject to some end-to-end quality constraints. The latter problem has been a topic for active research for a long time. If such path is to be established to satisfy one additive quality requirement such as delay or hop-count, the problem can easily be solved by Dijkstra's shortest path algorithm. Even if some end-to-end min-max constraint such as bandwidth need to be satisfied, still the problem can be solved easily using Wang and Crowcroft's shortest-widest path algorithm [93]. However, it is well known that establishing a path satisfying more than one additive quality constraints is an NP-hard problem [31, 88]. It is important to note that the problem of finding a mapping for

a composite stream processing task requires more than end-to-end constraints, because computational capacity of each of the nodes need to be individually satisfied.

More specifically, the network of servers connected by different communication links can be represented as an arbitrary graph, where each node representing a server has certain capacity limit and each link has certain bandwidth limit. The specification of the composite stream processing task can be represented as a directed path of multiple nodes, where each node specifies certain type of processing service. The source and sink of the path is specified to map on particular node in the resource graph. Based on the data delivery rate requirement of the task and the knowledge of CPU usage factor and bandwidth shrinkage factor of each service type, the node capacity requirement for each processing component and bandwidth requirement for each interconnection can be derived. Based on these information, the specification path need to be mapped on the resource network such that each component node in the path maps on a server node that serves the particular type of service component and has adequate available capacity to serve the component at the desired rate. In addition to the node constraints, the bandwidth requirement between each pair of components must fit in the available bandwidth between the server nodes serving the components. Note that a single server may serve multiple service components, so multiple nodes of the path may be mapped on a single node. When consecutive components are served by a single node, no network communication bandwidth is required between them. Also, two components may be mapped on two servers where there is no direct communication link between them. In that case, the communication channel between the two components may be mapped on a multi-hop path in the resource network.

Given these flexibilities and constraints, the mapping problem is indeed an NP-complete problem, even in the case where the topology and capacities of the whole resource network

is known at a centralized location. The detailed analysis of the problem along with different exact and heuristic solutions is discussed in Chapter 8. There are two additional axes of complexity in the real case. First, the RMS has a decentralized design and the RMS agents in each server node contains the local knowledge only, i.e. its own processing capacity and the services provided, and the bandwidth of the outgoing links to its direct neighbors. So, we need a completely distributed algorithm to find the feasible or optimal mapping. Second, the condition of the network, i.e. the available capacity of the nodes and the links may change while computing the mappings. As a result a feasible mapping may be actually found infeasible while committing the reservation of resources. These complexities are handled in the actual implementation of mapping and reservation protocols. The protocols are described in details in Chapter 9.

4.3.2 Dynamic Scheduling

After initial mapping and reservation, dynamic re-allocation of communication channels is required due to the fact that flows that travel through the public overlay links are highly variable and unpredictable in nature. Since each stream processing task specifies a target delivery rate, each server executing a component service for the task need to process and transmit data to the next component server in the chain in accordance with that target. Because of the variability, some tasks lag more than others in terms of fulfillment of this target. Now, despite being greedy to maximize its own revenue, the RMS agent in each server node will also try to maximize the fulfillment of the committed data rates, because of the penalty measures in the service agreements. In addition, the SLA with each task specifies a total budget or price, which in turn classifies the tasks such that a task paying higher price per byte of data processed will have higher preference by the RMS agent than

the other with a lower price. Given these objectives, we need to design a scheduling policy for the RMS agents such that system-wide performance objectives are maximized.

5

Related work

In this chapter, we discuss the previous works related to the research problems we have addressed in our thesis. First, in Section 5.1, we present an overview of the state of the art service hosting platforms and research works behind them. Several issues related to resource management in hosting platforms is also discussed.

Our primary contribution is the investigation of the usefulness of bi-modal resource management in hosting platforms. There are few other recent works that proposes such bi-modal organization of resources. Section 5.2 provides a brief overview of those works.

We have extended the idea of bi-modal resource management for platforms that serve data intensive applications such as stream processing. There are several works in the current literature that examine various aspects of the resource management problems for stream processing platforms, such as requirements mapping and dynamic re-allocation. These works are discussed in Section 5.3.

5.1 Service Hosting Platforms

There are cluster based centralized hosting platforms, Grid-like distributed platforms where resources are provisioned under strict control, and peer-to-peer platform that works with

public resources with very little central authority or control. We present a brief review of all of these different platform architectures in the following sub-sections. Several resource management issues such as scheduling, QoS assurance, service migration and virtual machines have also been discussed.

5.1.1 Cluster Based Platforms

Networked hosting platforms based on a local area cluster of computers have been widely studied and commercially deployed, especially for web hosting [43]. Several studies have been performed to investigate the resource management problems in such cluster environments to improve load balancing among the computers and fairness among clients. In [10] Aron et al. have introduced the concept of *cluster reserves* as a container of resources, to support service isolation and fairness among competing clients. Provisioning resources in hosting centers based on energy considerations has been studied in [30] . The technique uses an economic approach for sharing resources in such environments and is driven by energy considerations. Resource overbooking in the server clusters have been proposed by [90] to maximize the resource utilization and revenue return. Live migration of servers among the cluster nodes have been proposed by [79] to improve load balancing among the nodes.

5.1.2 Distributed Platforms with Controlled Resources

There have been several studies during the last decade to introduce resources from geographically distributed locations to create a shared service hosting platform. Such resource sharing platforms, commonly termed as Grid computing [42], was primarily motivated by

the massively compute-intensive scientific applications, so that server clusters or supercomputers from different laboratories can be shared among scientists. The roadmap of implementing Grid as a distributed wide area service hosting platform was outlined in [42] and [41]. Several studies have been performed afterwards to investigate the issues including remote resource monitoring, service negotiation, data management and security and trust management. A good survey of grid resource management strategies have been presented in [55]. Azzedin et al. have proposed for integration of trust with the resource management in Grid systems [16]. Economy based models for Grid resource management have been proposed by Buyya et al. in [29]. A good collection of research works on Grid resource management issues is presented in [74].

5.1.3 Peer-to-Peer platforms

Peer-to-peer platforms based on uncontracted public resources, basically by harvesting idle resources from user computers, came into popular use through Internet wide sharing of music and movie files. Although started through file sharing platforms [80], peer-to-peer resource sharing principles have soon been adopted to other types of resources such as network bandwidth for video streaming [77] and CPU cycles for high throughput computing [6].

Several research projects target to harvest idle capacity from public resources such as Condor [89], BOINC [6, 4], OurGrid [8] and Cluster Computing on the Fly [63]. One study in the Condor project investigated the availability pattern of the workstations in a university computing facilities [73]. Later the project developed mechanisms for advertising and discovering matching resources [78] in a Condor like environment. The BOINC project

has developed tools for migrating chunks of parameter datasets for parameter sweep applications to take benefit of idle public resources. OurGrid project has tried to apply Grid computing concepts [42] for constructing Grid systems based of public resources. Cluster computing on the fly project has introduced the notion of wave scheduling [97] in order to take advantage of the variation of idle time based on geographic time zones. A recent project named Global Public Computing [54] have presented the design and implementation of an application independent platform that can transparently aggregate resources from user computers and provides accounting mechanism to trade resources for execution of application codes. Their work demonstrates the scalability of such public resource based architectures, where global scale services can be deployed in less than a minute.

5.1.4 Research on Scheduling Problem

Scheduling a set of jobs on a set of resources is the core problem in any resource management system. Scheduling on a fixed size deterministic set of resources is a well studied problem both in computer science and operations research. Although several optimal algorithms are available for simpler scheduling problems, most of the interesting and practical scheduling problems are computationally intractable [58]. When preemption is possible, there are optimal polynomial time algorithms for scheduling jobs with arrival time and due date constraints on a single processor [20]. Also, for the two processor case, arbitrary jobs with certain precedence constraints can be scheduled in polynomial time [44]. However, scheduling jobs with arrival time and deadline constraints is proven to be a NP-hard problem for more than two processors [45]. In fact [37] proved that optimal scheduling of jobs in multiple processors is impossible if any of the 3 parameters – arrival time, execution time or deadline is unknown. Because in an online scheduling scenario, resource allocations

have to be carried out with incomplete information regarding jobs, heuristic solutions are appropriate for this situation. A good survey of online scheduling heuristics can be found in [86]. Scheduling jobs on heterogeneous collection of resources poses a problem of new dimension due to the variability in the capacity of the resources. A detailed performance analysis of several different heuristics for dynamic scheduling of tasks on heterogeneous computing environment is presented in [65].

5.1.5 Research on QoS Assurance

Assurance of Quality of Service (QoS) is an essential aspect of any successful service hosting platform. The service quality is specified by service level agreements (SLA) between the service provider and the clients. As opposed to best-effort services, the platforms promises adherence to specified values of the quality metrics, and support different quality levels to be specified for different clients.

The QoS issue has been widely studied in the realm of data transport service provided by Internet and several approaches have been proposed [95]. These attempts were guided by the quality requirements for transporting real-time multimedia data over Internet [75, 93]. Different approaches and service models proposed by Internet Engineering Task Force (IETF) for QoS guarantees in data transport include Integrated Services/RSVP model [25, 26], the Differentiated Services model [23], MPLS [81] and Constraint Based Routing [34]. The structure of the SLA for packet delivery services as well as techniques for real-time management of the SLA has been discussed in [24].

There have been several studies on providing response time guarantee for web page access in centralized web hosting platforms [2]. Performance evaluation of scheduling

heuristics for cluster based web hosting platforms are presented in [30, 79, 10] with different optimization goals in different cases. For distributed computing platforms, the issues of SLA negotiation and QoS management have been discussed in [35, 9, 51, 3].

5.1.6 Virtual Machines and Service Migrations

There are several important concerns about executing jobs on remote machines and migrating them live. One issue is to encapsulate the process so that the interaction between the remote job and host kernel is done in a controlled manner. Besides other techniques, virtual machine can be used for encapsulation. There have been several research works on efficient implementation of virtual machines and almost local equivalent performance is now achievable.

Although virtual machine is the best option for secure isolation of the guest processes, one problem is that a full-blown virtual machine is too heavyweight for migration. Considering this fact, several research projects have developed efficient process migration techniques that can successfully live-migrate server applications [70]. However, virtual machine based techniques are gaining popularity because of the residual dependency [49] and infrastructure incompatibility [71] problems of process migration. Virtual machines can now be live-migrated that greatly reduces the actual down-time of the applications. Recently, heavily loaded web severs and game severs running on Xen has been live migrated with down time in the order of 100ms, although the total migration time spans over $2 - 3$ minutes [33]. With the introduction of hardware assisted virtualization mechanisms in the mainstream processors such as Intel [59] and incorporation of VM monitors into the common desktop OS kernels [57], widespread availability of VM enabled desktop machines can be expected in near future.

5.2 Bi-modal Architectures

Although there is a vast body of literature on resource management in cluster, Grid or peer-to-peer hosting platforms, there have been relatively few works that propose combined use of dedicated and public resources. In [52], Kenyon et al. provided some arguments based on mathematical analysis, that commercially valuable quality assured services can be generated from harvested public computing resources, if some small amount of dedicated computers can be augmented with them. With some simple models of available periods of harvested cycles, their work have measured the amount of dedicated resources necessary to achieve some stochastic quality assurance from the platform. However, they did not studied how a bi-modal platform would perform in presence of service level agreements with different clients and how to engineer the scheduling policies to maximize adherence to these agreements.

BitTorrent [77] is a peer-to-peer video streaming platform that capitalizes on unused uplink bandwidth of end-user computers. Recently, in [36], Das et al. have proposed the use of dedicated streaming servers along with BitTorrent, to provide streaming services with commercially valuable quality assurances while maintaining the self scaling property of the BitTorrent platform. With analytical models of BitTorrent and dedicated content servers they have demonstrated how guaranteed download time can be achieved through augmentation of these platforms. However, their proposal does not include the protocols that can be used to achieve these performance improvements.

5.3 Stream Processing Platforms

Architectures and resource management schemes for distributed stream processing platforms have been studied by different research communities including distributed databases, sensor networks and multimedia streaming. The database and sensor network community has approached the problem from the perspective of placing the query operators to nodes inside the network that carries the data stream from source to the viewer [76]. From the multimedia streaming perspective, similar problem arises when we need to perform a series of on-line operations such as trans-coding or embedding on one or more multimedia streams and these services are provided by servers in distributed locations. In both cases, the main problem is to allocate the node resources where certain processing need to be performed, along with the network bandwidths that will carry the data stream through these nodes.

Finding the optimal solution to this resource allocation problem is inherently complex. Several heuristics have been proposed in the literature that attempts for near-optimal solutions. Recursive partitioning of the network of computing nodes have been proposed in [56] and [85] to map the stream processing operators on a hierarchy of node-groups. They have demonstrated that such distributed allocation of resources for the query operators provides better response time and better tolerance to network perturbations compared to planning the mapping at a centralized location.

In [92] and [48], the service requirements for multi-step processing of multimedia streams, defined in terms of service composition graphs have been mapped to an overlay network of servers after pruning the whole resource network into a subset of compatible resources. The mapping is performed subject to some end-to-end quality constraints, but the CPU requirements for each individual service component is not considered. Liang and

Nahrstedt in [61] have attempted to solve the mapping problem where both node capacity requirement and bandwidth requirements are fulfilled. However, one of the assumptions made by Liang and Nahrstedt was that the optimization algorithm was executed in a single node and complete state of the resource network is available to that node before execution. In a large scale dynamic network this assumption is hard to realize. If we assume that each node in the resource network is aware of the state of its immediate neighborhood only, we need to compute the solution using a distributed algorithm. In Chapter 8 we present a distributed algorithms to solve the problem.

In all of the abovementioned works, the operator nodes are assumed to interconnected through an application dependent overlay network using the Internet as underlay. In [47], Gu and Nahrstedt presented a service overlay network for multimedia stream processing, where they have shown that dynamic re-allocation of the operator nodes provides better compliance with the service contracts in terms of service availability and response time. However, none of the works have proposed the use of dedicated links in conjunction with IP overlay network for improving adherence to the service contracts,

6

State Oblivious Management of Public Computing Resources

6.1 Overview

In the series of two chapters, this (Chapter 6) and the following one (Chapter 7), we explore the problems in managing a combination of dedicated and public computing resources to serve compute-intensive applications in a public computing utility (PCU). The possible architectural alternatives of the resource management system (RMS) for this purpose were presented in Chapter 3. In this chapter, we explain the problem of state-oblivious resource management in the centralized RMS architecture, develop heuristic and evaluate through simulation studies.

As described in Chapter 3, the PCU model assumes a finite-sized privately installed dedicated resource pool and a very large (nearly infinite) sized public resource pool that can be used in opportunistic manner. The dedicated resource pool is assumed to be installed as a single cluster concentrated at a single network location. The public resource pool, on the other hand, is fully distributed and is organized in a P2P network. A P2P discovery service [83] is assumed to be available to locate the most appropriate set of public resources to satisfy a given request. Although public resources are available in plenty, their service

rates are unpredictable. Estimates of expected performances based on observation of prior engagements provide the only basis for choosing the best candidates from the available public resources.

Given the geographical spread of public resources, continuous monitoring and status aggregation for them is a difficult and costly operation. In this chapter we explore the scenario where the central RMS gets only the report of successful completion of the jobs that run on public resources. Failure or transient variability in available capacity is not observable. Only an aggregate measure of the historical performance of a public resource can be stored by the RMS and can be used to create a model for prediction of future performance. Also it is not possible to migrate the jobs from public resources while it is running. The dedicated resources however, are under total control of RMS. Under these assumptions, in this chapter we devise an online scheduling heuristic for the RMS of the PCU. The PCU online heuristic needs to decide what class of resources (public or dedicated) should be used for serving a given request in addition to determining how best to use the selected resource. Moreover, conformance with the SLA signed with the client that submits the job request must be achieved.

We specifically consider here the high-throughput computing applications with response-time constraints, where each job submitted from a particular client has a deadline before which it should be completely serviced if the client is to receive full benefit. It is well known [58] that even if all the information about the jobs (i.e., arrival time, processing time, and deadline) are known a-priori, finding the optimal non-preemptive schedule that maximizes throughput is a NP-hard problem for a multiprocessor system. In a practical PCU setting, the RMS has to take the allocation decision as soon as the jobs arrive, and the arrival times are arbitrary. Moreover the RMS has to deal with the uncertainty and lack of

control of the public resources, while exploiting them to get some processing done.

The rest of this chapter is organized as follows. Section 6.2 elaborates on the resource scheduling problem being dealt by the RMS in the particular state-oblivious settings. The proposed heuristic solution is presented in Section 6.3 along with description of several other alternatives for comparison. Section 6.4 discusses the results from the simulations performed to evaluate the resource allocation alternatives. Most of the findings presented in this chapter appeared as a conference paper [11] in a concise form and as a journal article [12] in elaborate form.

6.2 The Resource Management Problem

Computational jobs arrive from each client of the PCU service provider at arbitrary points in time with each job consisting of arbitrary number of mutually independent, concurrent components. Along with the jobs, clients are assumed to submit an estimation of the workload of each of the components at the submission time. Component threads may possibly have different execution times. An overall deadline is defined for the job before which all the components must finish their execution.

The SLA that is signed off-line between the provider and a client reserves a throughput guarantee for the corresponding client. The SLA defines various parameters including:

- ρ, the ratio of the client-offered workload that is guaranteed to be carried out by the PCU service provider.
- V, the maximum limit on the workload that can be offered by the client.

From these parameters it can be deduced that when the offered load is not more than V, the delivered throughput should be ρv or more in order to be compliant with the SLA. If offered

load v is greater than V, it is sufficient for the PCU to deliver ρV amount of throughput.

Another service objective of the PCU is meeting the deadlines of the individual jobs presented by the clients. The PCU provider earns revenue in proportion to the total delivered computational work for the jobs that finish completely within their deadline (with all of its components). A global throughput versus price for unit work curve defines this revenue. The curve may be concave to emphasize the fact that the price is higher for work in higher throughput, but the rate of increase is gradually slowed down. Further, there is penalty for violation of the SLA terms and the penalty is proportional to amount of deviation of the delivered throughput from guaranteed throughput, measured over a specified time window. The length of the window and a moving averaging factor α_{sla} that is used to smooth out the burstiness in offered and completed workloads across the windows are defined as SLA parameters.

The job scheduling component of the RMS is invoked at periodic intervals termed as scheduling *epoch*. Guided by the above-mentioned service objectives, the scheduler has precisely two distinct responsibilities at the end of each epoch:

- Accept or reject the jobs that arrived during the last epoch, and start the components of the accepted jobs on the private and/or public resources.
- Migrate some components of some jobs that are vulnerable for deadline violation, from public resources to the dedicated resources. Because there is no checkpointing and no progress monitoring of the jobs running on public machines, the RMS has to *restart* the job from the beginning instead of relocating the remaining portions.

The optimization goal of the job scheduler is to maximize the net revenue (i.e., revenue $-$ penalty) of the PCU service provider. To achieve this objective, the job scheduler has to maximize the amount of work done for jobs that do not violate the deadline. Another goal

is to ensure the fairness among the clients, so that all of them have equal treatment from the scheduler in accordance with the subscriptions agreed upon in the SLAs. The next section describes the heuristic solutions we devised to achieve these goals.

6.3 Heuristic Solutions for Resource Management

In this section, we present three heuristic solutions to resource management problem in the PCU environment described before. The first solution, the PCU heuristic, is proposed as part of this work. The next two solutions are adopted for the PCU environment from the scheduling literature for comparison purposes.

6.3.1 PCU Heuristic: An Online Resource Allocator

The scheduler of the RMS uses an online heuristic to take decisions about allocating available resources to incoming jobs. To reduce the scheduling overhead, the RMS executes the scheduling rules at discrete points of time (i.e., at the end of each scheduling epoch δ). Another component of the RMS, the SLA monitor measures the current deviation D_c of delivered throughput from required throughput for each client c, according to the SLA specified time-window τ_c and moving average factor α_{sla}. Say the total arrived workload in a time-window is W_a and total completed and delivered workload is W_d, both W_a and W_d being smoothed by moving average with the past values. Then,

$$D_c = \max(V_c, W_a \rho_c) - W_d$$

In the above equation, V_c and ρ_c are SLA-defined maximum load and acceptance ratio for client c. The current value of D_c is available to the scheduler at the end of every epoch.

There are two parts of the decision taken by the scheduler at the end of every epoch (i) accept newly arrived jobs and start them on public and/or private resources, and (ii) relocate and restart the deadline vulnerable jobs from public resource to the dedicated resource pool (in absence of checkpointing and progress monitoring, it is impossible to migrate without restarting).

Acceptance of jobs

For each client, the scheduler maintains a priority queue for newly arrived jobs, ordered by highest contributing job first. For a job with total workload W and total available time T_a before deadline, the throughput contribution is $\frac{W}{T_a}$. Every time the foremost job from the queue of the client having highest $D_c - W_c$ value is chosen, where W_c is is the amount of workload so far accepted for client c in current SLA window.

All the jobs are ultimately accepted, and each of them is assigned one of the two different levels of *restart-priority*, which is used later for restarting decisions. The jobs are accepted according to the following rules:

1. As long as available dedicated resources allow, schedule jobs with *critical* components on dedicated and the rest on public resources. The components that are expected to violate deadline if scheduled on a public resources according to their currently estimated expected throughput μ, are identified as critical components. Among the M dedicated resources, M_r are reserved for restarting phase (the ratio $\frac{M_r}{M}$ is a design parameter). Let M_o denotes the number of occupied dedicated resources at any given time and m denotes the number of critical components in the new job. Accepting jobs with this rule continues as long as $M_o + m \leq M - M_r$. Otherwise, the scheduler switches to the rule-2. The restart-priority is set to high for all the jobs

scheduled by rule-1.

2. For the rest of the enqueued jobs all components are scheduled on public resources. For any client c, as long as total accepted workload from that client in the current SLA window is below $\rho_c V_c$, the restart-priority of the accepted job is high, otherwise it is low.

Restart jobs

At the end of every epoch, the scheduler restarts some deadline-vulnerable job-components from public resources. At any given time, a job component is defined to be deadline-vulnerable if it cannot be completed before deadline unless it is allocated a dedicated resource right at that time. A priority queue is maintained for all the vulnerable components. The queue is ordered descending primarily by the restart-priority (explained earlier) and secondly by violation probability (p_v). p_v is computed at the job-launch time from the available information (distribution of the public resource throughput, component size and the deadline). From the queue, high restart-priority components are restarted as long as any dedicated resource is available. Low restart-priority components are restarted as long as available dedicated resource is greater than M_r. The rest of components are left on public resource.

6.3.2 Least Laxity First and Greedy Heuristics

For performance evaluation we compare our PCU heuristic with the well known *Least Laxity First* (LLF) [86] heuristic and a Greedy heuristic. We use the LLF heuristic to schedule the jobs only in the dedicated pool of resources. The laxity is the slack between possible execution finish time and deadline. New jobs from each client enter a separate

priority queue where the priority is given to the job with least laxity. At every epoch, jobs are popped from the queues and scheduled in dedicated resources if available. Otherwise the job waits in the queue until the time after which it becomes infeasible to execute within deadline. After that the job is dropped. As a fairness scheme the queue of the client with highest deviation from SLA is favored when choosing every job.

The Greedy heuristic, another one that we used for comparison, works on the same PCU architecture with a combination of dedicated and public resource pools. The greedy scheduling policy chooses jobs from the arrival queues in every scheduling epoch in the order of highest contributing job of the highest deviating client first. It schedules all components of incoming jobs on dedicated resources in the order of longer component first, as long as there is spare capacity in the dedicated resource pool. All the remaining job-components are scheduled on public resources until all the arrival queues are exhausted.

6.4 Simulation Results

We have evaluated the performance of the PCU heuristic through a simulator written in Parsec [17] by changing different parameters and comparing it with the Greedy and LLF heuristics. In our simulation setup, the PCU provider had a pool of 100 dedicated machines and an infinite pool of public machines. There were five independent clients each feeding a series of parallel jobs that need to be completed within the given deadlines. Each client has its own SLA with the PCU provider. Job arrival is assumed to be a Poisson process, with each job having a random number (k) of parallel components (geometrically distributed with a mean 25, unless mentioned otherwise). Each component of a job also has a random workload that is from a geometric distribution. Each job has a feasible deadline, i.e., it can always be completed if all the parallel components run on dedicated machines. Unless

stated otherwise, the deadline was set to have a certain amount of laxity from the end time of the longest running component. The laxity was chosen randomly between 0.5 and 2 times the mean execution time of the job-components, with uniform distribution. This tight deadline allows one trial on the public pool and failing that it should be restarted on a dedicated resource.

All dedicated machines have homogeneous throughput, completing 1 unit of workload of a component per second. The public resource throughput is sampled from Lognormal distribution with standard deviation 1.0 and mean less than 1.0, generally 0.8 unless stated otherwise. Justification behind using lognormal distribution is that being left skewed, it closely resembles the behavior of the resources in a PCU setting, where most of the public resources may have very low or even 0 throughput.

In this section we show the results of 2 sets of simulation experiments. The first 5 graphs (figure 6.1 to figure 6.5 in subsection 6.4.1) shows the comparative study of the performance of our new heuristic with other standard scheduling algorithms. The next 9 graphs (figure 6.6 to figure 6.14 in subsection 6.4.2) shows the study of how performance of the scheduler is affected by the change of different environment parameters.

6.4.1 Comparative Study

In the first set of experiments the PCU heuristic is compared with LLF and Greedy using throughput (Figure 6.1), SLA compliance (measured using penalty per unit revenue in Figure 6.2) and net revenue in Figure 6.3. The PCU heuristic delivers better throughput than LLF, which implies it is useful to augment public resource in a CU. Also the PCU-heuristic is superior in performance compared to the greedy heuristic in similar setting.

The penalty is higher with the LLF algorithm on dedicated pool only system than the

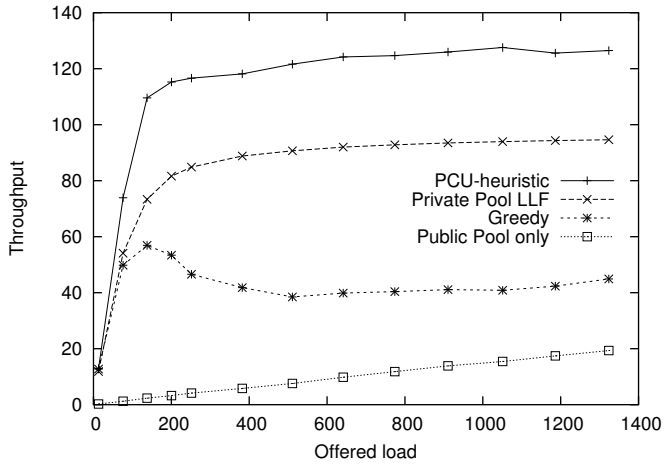

Figure 6.1: Variation of mean throughput with offered load values for mean public resource throughput $\mu = 0.80$, mean number of parallel components $P = 25$, total number of dedicated resources $M = 100$, and total SLA booking, $\sum \rho V = 100$.

PCU heuristic, because jobs are not deprioritized when the client is offering more workload than the SLA upper bound. In case of the greedy algorithm, penalty grows even higher when the client is overloading, because the dedicated pool gets fully occupied and most of the newly arriving jobs are put on public resources. Consequently, only a small portion of the newly arriving jobs can finish before their deadlines.

Figure 6.4 shows that a much higher gain in throughput is achievable, if the exact knowledge of throughput of each public machine is available at schedule time, because then there is no need for restarting jobs. How far of this gain can be achieved without a priori knowledge of public resource characteristics, remains as a problem for future research.

As Figure 6.5 shows, the utilization of dedicated resources is higher for the greedy policy. This is because Greedy uses the dedicated resources exhaustively. The PCU heuristic tries to execute a job-component primarily using public resources unless it becomes vulnerable for deadline violation. Also, in PCU, to allow the restarting of vulnerable components,

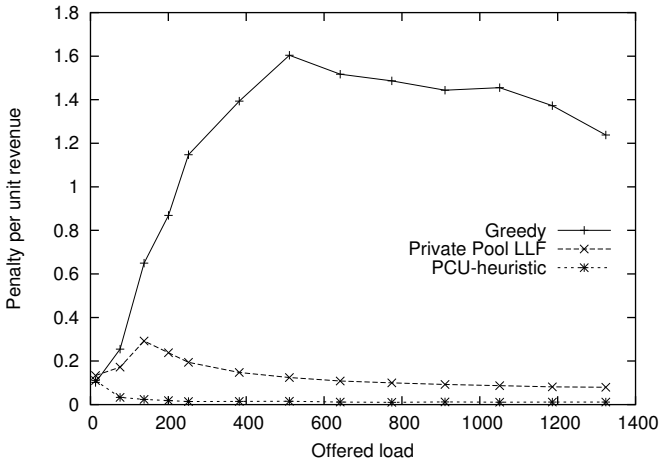

Figure 6.2: Variation of penalty per unit revenue with offered load for $\mu = 0.80$, $P = 25$, $M = 100$, and $\sum \rho V = 100$.

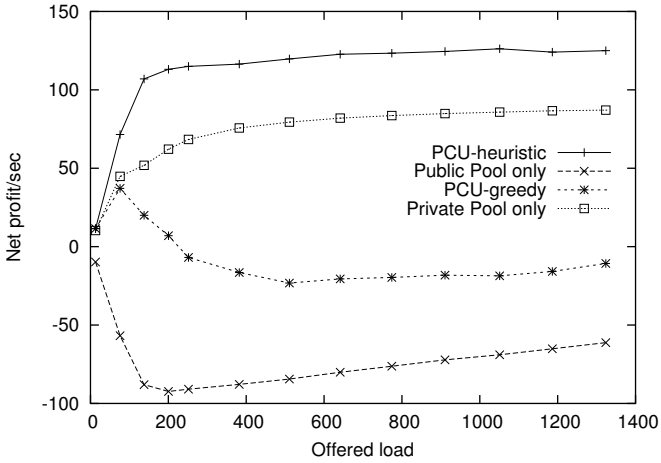

Figure 6.3: Variation of net profit earned by the PCU provider with offered load for $\mu = 0.80$, $P = 25$, $M = 100$, and $\sum \rho V = 100$.

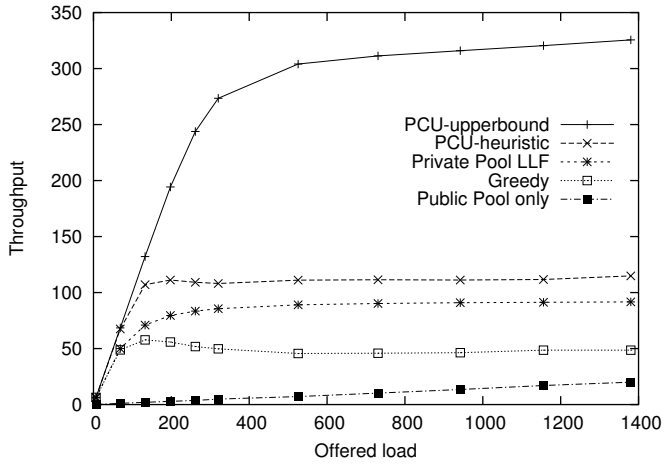

Figure 6.4: Upper bound on PCU throughput assuming future behavior of public resources is known for $\mu = 0.80$, $P = 25$, $M = 100$, and $\sum \rho V = 100$.

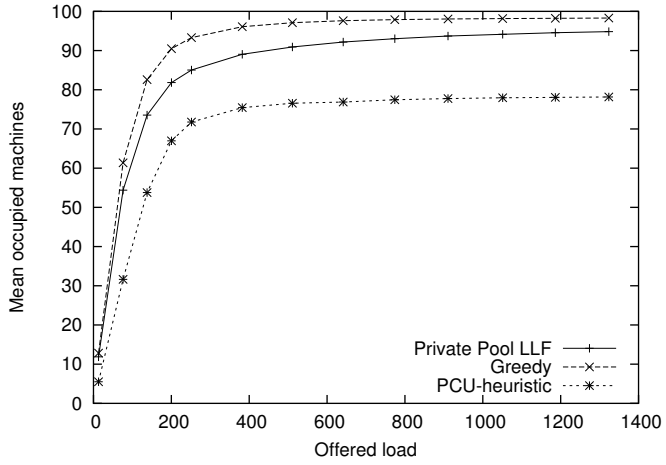

Figure 6.5: Utilization of dedicated resources versus offered load for $\mu = 0.80$, $P = 25$, $M = 100$, and $\sum \rho V = 100$.

it reserves a portion of the dedicated resources (25%) as contingency resources. These factors lower the utilization of dedicated resources in the PCU heuristic. Greedy's utilization is even more than LLF, because, in LLF jobs are not allocated unless the all the components fit in the dedicated resources, whereas, Greedy may put part of a job in dedicated pool and rest in public pool.

6.4.2 Response to Parameter Changes

Here we consider the effects of different environment parameters like public resource capacity, SLA-overbooking, degree of parallelism of the jobs, etc. on the performance of the scheduler. First, to consider the flexibility in SLA overbooking, if the total agreed upon deliverable throughput (ρV) is higher than the maximum system capacity, the SLA deviation goes very high leading to correspondingly high penalties (Figure 6.6). This in turn reduces the net profit earned by the PCU provider. From Figure 6.7 it can be observed that SLA booking should be at 140% of the dedicated pool capacity to maximize the performance for the given PCU configuration.

Figure 6.8 shows that use of PCU-heuristic brings gain in delivered throughput in most region of the spectrum of public resource behavior. It should be noted that with lognormal distribution, even if the mean throughput is equal to that of a dedicated machine, 62% of the public resources have throughput less than that of a dedicated machine. For very low public resource throughput, almost all of the jobs scheduled there need to restart, and since restart is subject to availability in the limited capacity dedicated pool, many jobs get discarded. This explains the less than one throughput-gain with poor quality of public resources. Figure 6.9 shows that PCU-heuristic outperforms the greedy heuristic across the whole spectrum.

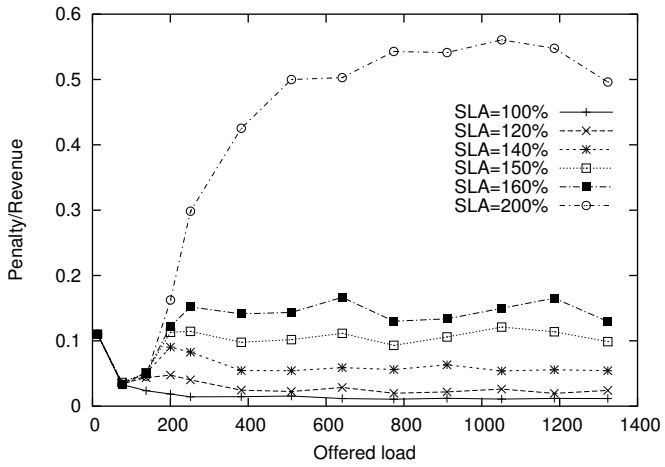

Figure 6.6: Penalty per unit revenue earned at different levels of SLA booking for $\mu = 0.80$, $P = 25$, and $M = 100$.

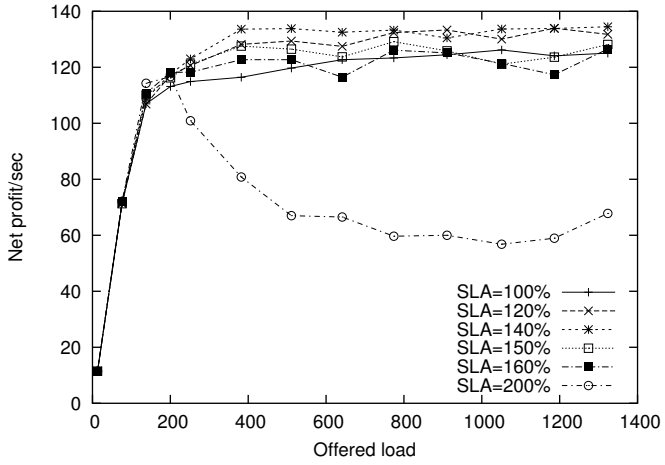

Figure 6.7: Penalty per unit revenue earned at different levels of SLA booking for $\mu = 0.80$, $P = 25$, and $M = 100$.

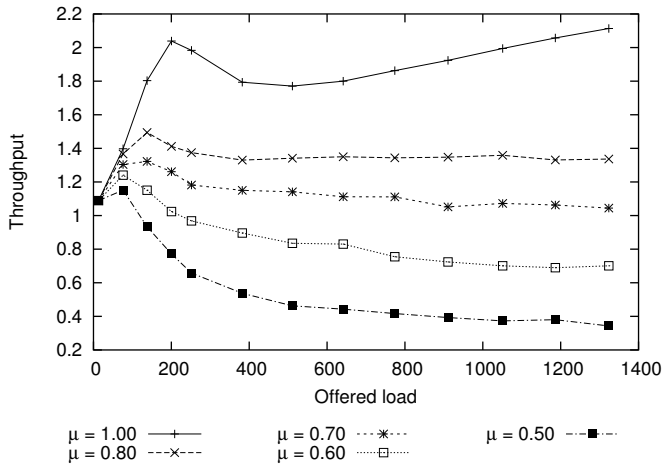

Figure 6.8: Throughput gain at different public resource characteristics, with respect to a dedicated pool only system for $P = 25$, $M = 100$, and $\sum \rho V = 100$.

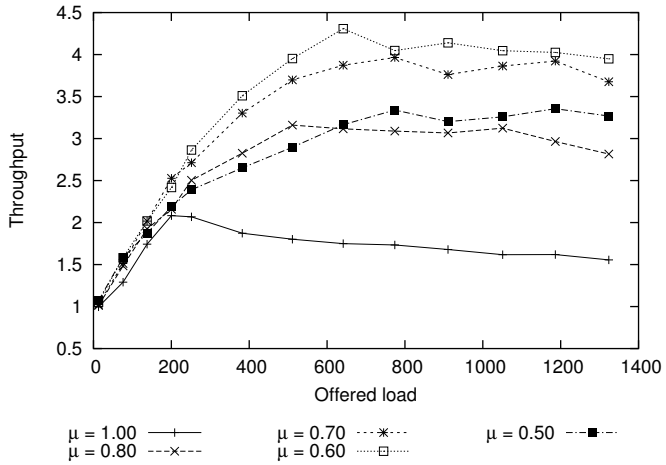

Figure 6.9: Throughput gain at different public resource characteristics, with respect to the greedy resource allocation policy on combined pools for $P = 25$, $M = 100$, and $\sum \rho V = 100$.

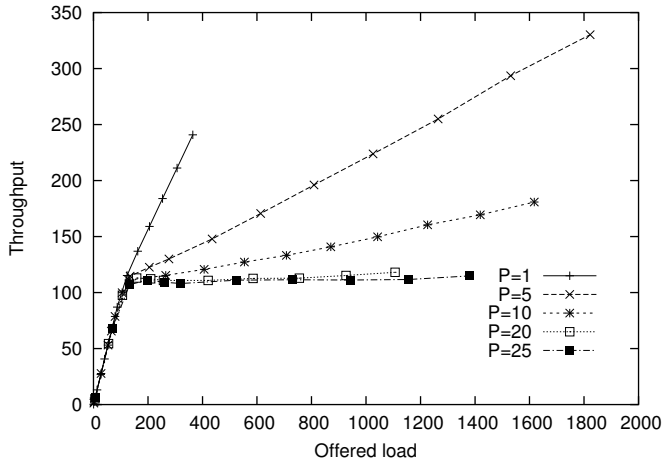

Figure 6.10: Mean throughput at varying degree of parallelism for $\mu = 0.80$, $M = 100$, and $\sum \rho V = 100$.

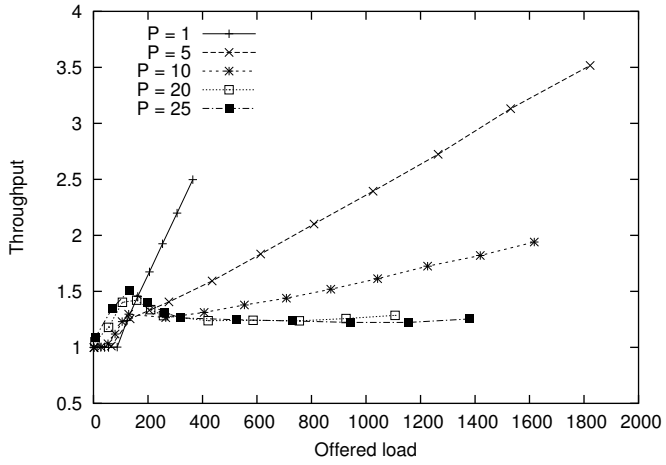

Figure 6.11: Throughput gain at different degrees of parallelism, with respect to a dedicated pool only system for $\mu = 0.80$, $M = 100$, and $\sum \rho V = 100$.

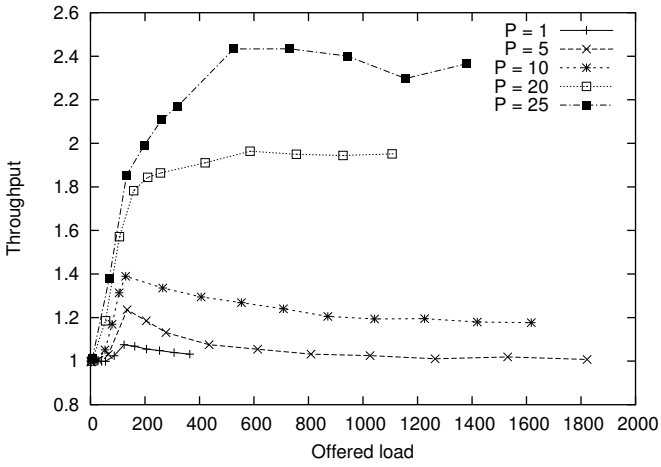

Figure 6.12: Throughput gain at different degrees of parallelism, with respect to the greedy resource allocation policy on combined pools for $\mu = 0.80$, $M = 100$, and $\sum \rho V = 100$.

Studying the effect of parallelism figure 6.10 shows that the effect is insignificant in underloaded situations, but when the system is overloaded, high number of parallel components increase the probability of failure of a whole job due to failure of only one or few components which could not be restarted when necessary. Hence, the total delivered throughput becomes low. Figure 6.11 shows that the throughput with PCU heuristic always outperforms the LLF heuristic at a large degree for jobs with fewer parallel components. As we compare the PCU and the greedy heuristics in figure 6.12, it reveals that greedy heuristic performs much poorer with highly parallel jobs than the PCU heuristic. This is because in greedy, the dedicated pool of resources gets occupied very quickly and a large number of jobs are scheduled on the unreliable public resources.

Study on the effect of laxity before deadline (Figure 6.13)f shows that throughput gain is much higher with relaxed laxity jobs. This is because with relaxed laxity the probability of getting a job component completed before deadline on a public resource increases, which incurs less restarts and better contribution from public resources.

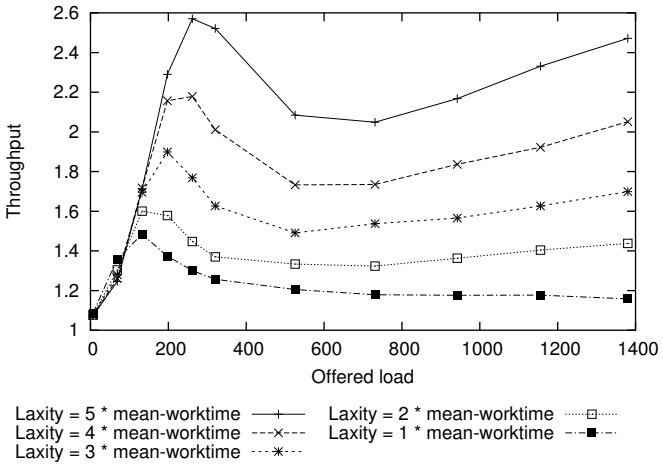

Figure 6.13: Throughput gain at different amount of laxity in deadline, with respect to a dedicated pool only system for $\mu = 0.80$, $P = 25$, $M = 100$, and $\sum \rho V = 100$.

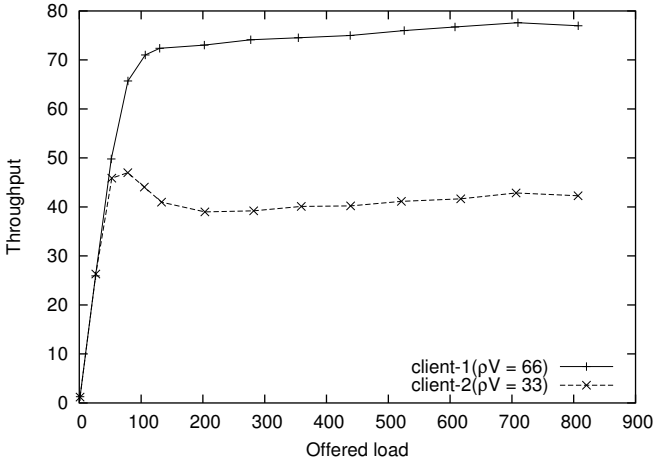

Figure 6.14: Comparing delivered throughput to 2 clients having different max-load defined in SLA for $\mu = 0.80$, $P = 25$, and $M = 100$.

Figure 6.14 demonstrates the fairness of the PCU heuristic. For two different clients having SLA guaranteed max-workload (V) defined at $2:1$ ratio but offering load at similar rate, it shows that the delivered throughput is proportional to the SLA-defined maxload of the clients, in overloaded situations. Thus the algorithm honors the SLA for the clients and distributes the available resources in a fair ratiometric way.

In summary, the results show that the PCU heuristic outperforms the other two standard algorithms in terms of throughput gain, SLA compliance and profit maximization. The current algorithm uses the public resources without any prior knowledge about their performance, which greatly reduces the communication overhead. It is shown that with the accurate a priori knowledge, much higher throughput is achievable. This trade-off between communication overhead and performance gain is not studied in this paper. Resource utilization for the PCU heuristic is lower than other algorithms, because they use dedicated resources more exhaustively. On the other hand this little under-utilization yields higher performance in terms of throughput. It is shown that the performance (throughput and SLA compliance) depends on different factors like the capacity of the public resources, the degree of parallelism of jobs and the amount of laxity allowed between job execution time and deadline. The algorithm also allocate the available resources fairly among competing clients, according to the commitments in SLA.

6.5 Summary

In this chapter, we presented the resource management strategies for a public computing utility that is created by augmenting dedicated clusters with unreliable public resources. One of the significant features of the studied architecture is the minimal monitoring on the public resources. Because public resources are in plenty, this helps to keep the overhead

low. The effect of enabling monitoring of the public resources and job migrations on the delivered performance levels is studied in the following chapter.

We proposed a resource allocation heuristic that uses the public and dedicated pools of resources in an efficient manner. We carried out extensive simulations to evaluate the performance of the proposed heuristic and compare it with two other heuristics.

The results indicate that the PCU concept of using public resources to augment dedicated resources can lead to significant performance improvements both in terms of overall throughput obtainable from the computing utility and the service level compliance of the computing utility. Further, the results indicate that PCU performance depends on two major factors: characteristics of the public resources and characteristics of the workload. For instance, it can be noted that the performance gain from PCU increases if the job has fewer inter-job dependencies or relaxed deadlines. The performance of the proposed heuristic may be further improved by incorporating these parameters in the decision process.

7

State Aware Management of Public Computing Resources

7.1 Overview

This is the second in the series of two chapters that elaborates on the resource management problems in presence of two classes of computational resources in a service hosting platform. Similar to the system organization assumed in the study presented in the previous chapter, a combination of unreliable public computing resources and a privately installed and controllable cluster of dedicated computers forms the resource base for the service platform, name public computing utility (PCU).

In the previous model, the states of the public resources were considered unobservable due to remoteness and the resource allocation had to be done in a state-oblivious manner. This caused most of the work done on public resource to be wasted. Later, we have observed that periodic monitoring of public resources can be enabled without overwhelming the network. In this chapter, we present the study of resource management policies in a new system organization that enables status monitoring and failure detection of remote public resources to to gainfully engage them within the PCU service platform.

We have studied several approaches to fulfill the application resource requirements

from a PCU and deliver assured services to the clients. Applying our approaches on a compute-intensive application workloads from a research Grid, we have observed that resource scheduling using dynamic priority based preemptive migrations are necessary to achieve differentiated quality of service for the different clients. We demonstrated that these approaches, despite their simplicity, can achieve considerable performance benefits. For example, compared to the heavily provisioned DAS-2 research Grid [1] with a total of 400 CPUs that uses Maui cluster scheduler [50], an model PCU system can be built with only 100 dedicated CPUs, which implies 75% cut in resource provisioning. With opportunistic access to public resources, the same workload as DAS-2 can be carried out in PCU with only 12% increase in job execution time. In addition to these, a detailed study of the communication overhead due to the job migrations and status aggregation is also performed and we have summarized the observations in a set of guidelines for design of a successful PCU.

We have already introduced the organization of the PCU in Chapter 3. In the rest of this chapter we have explained the resource management problems that are unique under the new set of assumptions, develop heuristic solutions and evaluate them through simulation studies. Section 7.2 formally defines the scheduling problems and discuss scheduling strategies to design on-line heuristic solutions. At the end of the section, the relationships among design parameters are analyzed using simplified queueing models. In Section 7.3, we present the results of simulation based experiments that explore several design parameters of a PCU, as well as the conclusions drawn from the experiments. A performance comparison of a model PCU system with the DAS-2 research grid, based on past workload traces is presented at the end of the same section.

Most of the findings presented in this chapter have been published in the journal of

parallel and distributed computing as a regular paper [13].

7.2 Design of the Scheduling Policies

In this section, we explore different design considerations to devise an effective scheduling algorithm for the PCU resource manager, under the assumptions we described in the previous section.

7.2.1 Defining the Problem

Having a series of requests from different users, the prime task of the resource manager is to allocate one resource for each of the jobs. Given the finite number of high capacity dedicated resources, not all the jobs may be entitled to a dedicated resource immediately. If there was no access to any resource other than the dedicated ones, it would be a n-server single queue scheduling problem. Any solution to that would apply some priority rule on the queue and schedule the most prior jobs in one of the dedicated resources. The other jobs either remain idle in the queue or are dropped. There are well studied on-line heuristics for this problem such as EASY Back-Fill [62] or Maui [50]. The optimization goal could vary depending on the type of applications, but most of the schedulers try to minimize the completion time of the jobs.

Because we have access to a large number of public resources on the Internet, low priority jobs can get some progress running on these resources instead of sitting idle in the queue or getting dropped. Given the wide variation in the performance of public resources, this progress is quite unpredictable. To maintain the order of priority, it may be necessary to migrate the jobs running on public resources back to dedicated cluster as soon as some

resource becomes free. However, too many migrations will eventually swamp the network and the down-time for preemptive migrations will grow. The scheduling priorities are to be designed carefully to prevent unnecessary migrations.

Due to heterogeneous public resources, the scheduling problem becomes harder and analytically intractable. In addition to the standard goal of minimizing the response time for each job, the schedule needs to consider the SLAs contracted with each user and attempt to minimize the penalty paid by the PCU due to any deviation. Another goal is to minimize the network overhead to keep the delays in migrations within acceptable limits. Keeping these different goals in mind, we attempt to calibrate the design parameters of the resource manager and study their impact on performance.

7.2.2 Preemptive Migrations

In the absence of any SLA with the users, it would be sufficient to maximize the throughput of jobs in order to bring higher revenue for the PCU. This can be achieved by maximizing the utilization of dedicated resources disregarding priorities of the jobs. A simple algorithm to achieve this is to schedule all the high priority jobs in dedicated resources as long as one is available, and schedule the rest of the jobs on public resources. Since the number of public resources is virtually infinite, there will always be enough resources to schedule the jobs in queue. This scheme would certainly be unfair to long-running jobs that happen to arrive when all the dedicated resources are allocated. Such jobs have their fate stuck with some public resource for life.

In order to consider fairness among jobs in terms of access to the dedicated resources, a straightforward approach would be to allocate the dedicated resources to all the running jobs in round-robin fashion. Although this scheme will ensure uniform progress for all the

jobs, the high number of migrations will cause severe congestion in the link that connects the cluster to the Internet. One better idea is to dynamically and periodically assign priorities to all the running jobs and make sure that prior jobs migrate to the dedicated resources. Then the problem becomes computing the priorities intelligently, such that it does not incur high number of migrations whereas it ensures proportional share of dedicated resources according to the SLA.

7.2.3 Priority Functions

Given that a priority function for controlling preemptive migrations is central to the scheduling algorithm, we try to devise a proper function now. The idea is to assign priorities to each job such that lagging or deprived jobs get higher priority to get scheduled on dedicated resource. Two primary objectives considered in computing the priority are – a) minimize the completion time for each job b) minimize the deviation from SLA for each user. These priorities should be computed from the available information about the jobs. We assume that the resource manager can maintain a job table for all running jobs in the system. Now, based on current level of progress, the requested resource capacity, and the total amount of time elapsed from arrival, the resource manager can compute a relative elongation for all the running jobs. The resource manager also maintains a user table to keep track of current level of SLA deviation for each user. Then a linear combination of the deviation and the elongation values can be used as a priority for each jobs. For our simulations we simply added these two values. There can be other contributors to the priority value. Jobs that are delayed too much and do not have any chance to complete within the SLA defined elongation margin (in order to be counted in the goodput), are de-prioritized by forcing very negative priority values.

Algorithm 1 Skeleton scheduler

```
1:  for Every scheduling epoch do
2:    while jobs in InputQueue do
3:      j = first job in InputQueue
4:      if j is too old then
5:        drop(j)
6:      else if Free dedicated resource found then
7:        Schedule j in dedicated resource
8:      else
9:        Query for k public resource {k is the replication factor}
10:       if 0 < i ≤ k free public resources found then
11:         Spawn i replicas of j on public resources
12:       else
13:         break
14:       end if
15:     end if
16:   end while
17:   Based on life-pulses, detect failure of all remotely running jobs
18:   Compute priority of all the N_r jobs currently running
19:   if Among most prior N_p {number of dedicated resources} jobs m
      are running on public resources then
20:     Start migrating the forerunner replica of these m jobs to dedi-
        cated resource pool
21:     There will be m among the remaining N_r − N_p jobs that are
        running on dedicated resource. Start migrating them to public
        resources
22:   end if
23:   for All jobs j running in public resources and not migrating do
24:     If not all k replicas are up, replenish the replicas
25:   end for
26: end for
```

In the dynamic preemptive migration algorithm, the priorities are recomputed every epoch and a new ordering of the job table is computed. Then the N_p highest priority jobs are mapped on dedicated resources, where N_p is the total number of dedicated resources. Note that a major fraction of these N_p jobs may already be running on dedicated resources, so they need not get rescheduled. Also, the algorithm prevents rescheduling of the jobs that are in migration or recently migrated. Thus, number of migrations every epoch is much smaller than N_p. Algorithm 1 shows a skeleton for the scheduling algorithm. Computing the priorities and reordering the table is not computationally expensive because, we don't need a complete ordering of all the n running jobs in the job table. Selecting top N_p entries will be sufficient and this can be done efficiently in $O(N_p \log(n))$ time (N_p is a constant here) if a heap data structure for jobs entries are maintained.

7.2.4 Updating Remote States and Failure Detection

Once a job is scheduled on a public resource, it is important to get the updated information on resource consumption by that job. We assumed that each job running on a public resource sends a small progress report (not a complete checkpoint) to the resource manager at intervals equal to (but not necessarily synchronized to) scheduling epoch. The resource manager updates its job table using this information. Also, these reports act as life-pulses for the remote jobs, because after a timeout period of no progress reports, the resource manager can assume that the remote job has failed.

7.2.5 Replication

In order to mask the unreliability of the public resources, we have used multiple replicas of a single job running in parallel on multiple public resources. On detection of failure of

any one of the k different replicas, the resource manager re-spawns another replica on a new resource, and thus it always tries to maintain k replicas alive. The idea is to minimize down time due to failure to almost zero, by maintaining at least one replica alive all the time. In case all the k replicas fail at the same time, the resource manager has to re-spawn them from the last recorded checkpoint. Note that jobs running on public resources do not checkpoint their progress periodically, whereas the ones inside the cluster does. This is because the only reliable place to store the checkpoint is inside the cluster, and storing periodic checkpoint from remote jobs would cause huge amount of traffic load on the bottleneck link of the cluster. An estimate of appropriate value for k can be found from the analysis given in Section 7.2.6.

7.2.6 Analyzing the Design Parameters

The parameters of the scheduling scheme discussed above are interrelated along with other system parameters such as size of the dedicated cluster and workload. Understanding of these relationships is important for optimal tuning of these parameters. Due to its complex nature, a realistic model of the system is hard to solve analytically. Nevertheless, it is useful to resolve the dependencies under simplifying assumptions. Here we derive some approximate relationships among the parameters using some simplified queueing models.

Total capacity of the system

If N_d, N_p are number and T_d, T_p are mean throughput of dedicated and public resources respectively, P_{av} is the steady state probability of a public resource being available and r is the number of replica per job, then assuming zero cost of migration, the whole resource collection equates to N dedicated resources, where $N = N_d + \frac{T_p}{rT_d} P_{av} N_p$. Let μ be the

processing rate of one dedicated machine (e.g. 3.5GFLOPS). If we feed an Erlang's loss system of capacity N with unit-workload jobs (1 FLOP) in a Poisson process at mean rate (λ) equal to the workload arrival rate of PCU, we get an estimate of maximum deliverable throughput of the PCU applying Erlang's loss formula [39] on the equivalent system –

$$\text{Throughput} = \lambda \left(1 - \frac{(\lambda/\mu)^N}{N! \sum_{i=1}^{N} \frac{(\lambda/\mu)^i}{i!}}\right)$$

The bottleneck network link

To analyze how the bottleneck link capacity restricts the number of dedicated resources and also the incoming load, let us assume N_d be the total number of dedicated resource, B_{up} (bps) be the uplink bandwidth of the bottleneck link that carries data from the dedicated cluster to the public resources and B_{down} be the downlink bandwidth. Let λ be the arrival rate of jobs and L be the mean execution time of each job, if executed uninterrupted in a dedicated resource. In light load, when $\lambda L < N_d$, all the jobs will be served by dedicated resources, so there will not be any significant amount of transmission through the bottleneck link. At high load when $\lambda L \gg N_d$, most of the incoming jobs will be initially spawned on a public resource, and also, there will be a significant number of migrations every epoch. In worst case, all the incoming jobs go to public resources. So, the transmission load on bottleneck due to new jobs,

$$\lambda_s = km\lambda$$

where, m is the mean size of the memory footprint of a job and k is the replication factor. For the load due to preemptive migrations, there can be N_d inbound and N_d outbound

migrations every scheduling epoch, in the worst case. If the epoch length is δ, total transmission load on the outbound link,

$$\lambda_b = km\lambda + \frac{km}{\delta}N_d$$

For sustained steady state transmission, this load must be less than the capacity, i. e.,

$$\begin{aligned} \lambda_b &< B_{up} \\ \Rightarrow \lambda &< \frac{B_{up}}{km} - \frac{N_d}{\delta} \end{aligned}$$

Here, both N_d and δ are adjustable parameters and the arrival rate λ can be restricted for particular values of N_d and δ. For example, $N_d = 100$, $\delta = 120$sec, $k = 2$, $m = 1$megabytes, $B_{up} = 100$Mbps restricts maximum arrival rate to 5.4jobs/sec. Similarly, the downlink bandwidth B_{down} restricts the transmission load such that,

$$\begin{aligned} \frac{mN_d}{\delta} &< B_{down} \\ \Rightarrow N_d &< m\delta B_{down} \end{aligned}$$

Length of scheduling epochs

Regarding the length of the scheduling epoch, we already mentioned that too long epochs leave failures of public resources unnoticed for a long time. On average, each failure takes $\frac{\delta}{2}$ seconds to be noticed and one migration time $(= T$ seconds$)$ to be respawned. So, down time for each failure is $T + \frac{\delta}{2}$. Say, among all the running jobs on public resources, one jobs faces failure of all the k replicas every $\frac{1}{y}$ seconds. So, in a steady state, number of jobs

down due to failure,

$$n_f = \frac{y}{\frac{1}{T+\delta/2} - y} = \frac{y\delta + 2Ty}{2 - (y\delta + 2Ty)}$$

If, x migrations are initiated every epoch and again each migration takes T seconds to complete, then steady state number of jobs down due to migration,

$$n_m = \frac{x/\delta}{1/T - x/\delta} = \frac{xT}{\delta - xT}$$

If we want to minimize overall downtimes of the jobs due to both failure and migration, we need to find δ that minimizes,

$$f(\delta) = n_f + n_m = \frac{y\delta + 2Ty}{2 - (y\delta + 2Ty)} + \frac{xT}{\delta - xT}$$

For example, for $x = N_d = 100$, $T = 100$ sec, $1/y = 200$ sec, $f(\delta)$ is minimized for $\delta = 180$ seconds.

Number of replicas

It is beneficial to run replicated copies of the jobs on different resources, when they are scheduled on unreliable public machines. From the reliability point of view, the higher the replication factor k is, the better the chance of survival. But higher replication is definitely expensive and wasteful in terms of resources. Also, increasing k would increase the bottleneck network load for checkpointing and migrations. A desired value of k is the one that allows at least 1 replica always up when running on public resources.

Say a job is replicated on k public resources. If we assume that the failure process is memoryless, each resource i has an exponential random runtime before failure with mean time $1/\lambda_i$. Let us assume the mean recovery or migration time is $1/\mu_i$ and the distribution

is exponential. Also failure and recovery of all resources are independent.

For any resource i, its expected uptime $E(up) = 1/\lambda_i$ and expected downtime $E(down) = 1/\mu_i$. According to renewal theory [82], probability that it is available at any given time is:

$$A_i = \frac{E(up)}{E(up) + E(down)} = \frac{1/\lambda_i}{1/\lambda_i + 1/\mu_i} = \frac{\mu_i}{\mu_i + \lambda_i}$$

If all the resources are identical, in that case, say $\forall i A_i = p$. Now, if X is the available number of resources at any given time, then

$$P(X = j) = \begin{pmatrix} k \\ j \end{pmatrix} p^j (1-p)^{k-j}$$

Therefore,

$$P(X \geq 1) = 1 - P(X < 0) = 1 - (1-p)^k$$

$$\Rightarrow k = \frac{log(1 - P(X \geq 1))}{log(1-p)} = \frac{log(1 - P(X \geq 1))}{log(\lambda) - log(\lambda + \mu)}$$

So, if we want, with 95% probability, that 1 resource be available at any time, and we have MTTF $\frac{1}{\lambda} = 5$ minutes and recovery time $\frac{1}{\mu} = 2$ minutes, then,

$$k = \frac{log(1 - 0.95)}{log(1/5) - log(1/5 + 1/2)} = 2.39$$

7.3 Simulation Study

We performed three sets of experiments on simulated PCU systems as described in the following sub-sections. The first set of experiments described in Section 7.3.3 weigh between different scheduling strategies. Having a chosen strategy, the next set of experiments,

described in Section 7.3.4, explore the parameter space of the system to investigate different trade-offs and optimal settings. A set of design principles that is drawn based on the simulation results and the analysis given in Section 7.2.6, is presented in the following subsection 7.3.5. Last, in Section 7.3.6, we evaluate a model PCU system with DAS-2 Grid using an actual computational workload trace. Model parameters for the simulated systems and workload data sources used for different experiments are described in sub-section 7.3.1 and 7.3.2, respectively.

7.3.1 Simulation Model

We developed a discrete event simulation model using Parsec [17] for a PCU system. The dedicated cluster was assumed to have up to 200 Pentium-IV, 3.2GHz machines connected by a Gigabit Ethernet. The benchmark capacity of each Pentium-IV machine was 3.5GFlops. The number of public resources was large compared to the size of the dedicated cluster. We assume that the number of machines reachable from the cluster within acceptable network delay and bandwidth constraints is $10,000$ (for comparison, the total number of participating hosts in a popular BOINC project is in the order of $100,000$ [5]). The network parameters were set to 100ms of latency and 1Mbps bandwidth. This is reasonable if we harvest desktops from home users because they connect to the Internet using 1Mbps or better DSL or cable modem, and the round trip times between machines from North America and Australia is in the range $150 - 250$ms. The public resources are distributed widely across the Internet where the backbone capacity is much higher than the endpoint link. So the background traffic on the backbone will not significantly perturb this minimal bandwidth. The distribution of computational capacity of these public resources are empirically derived (Figure 3.3) from the distribution of capacities of the participant hosts

of the SETI@Home project [5]. The failure model of the public resources are modeled as Markov process schematically described in Figure 3.4 and it corresponds to the analysis given in [73]. The cluster is connected to rest of the Internet through one or more edge routers which creates a bottleneck for traffic in and out of the cluster. We assume the total capacity of the bottleneck link to be 100Mbps.

7.3.2 Workload Data Source

We used three different workloads for different sets of experiments, that is appropriate for the particular experiment objectives. To test the performance of the chosen algorithm on a real workload we chose the workload trace from DAS2 5-cluster research Grid over year 2003 that was presented in [60]. DAS2 is a Netherlands based academic research Grid, made up of 5 clusters of Pentium-III machines spread across 5 universities. Four of the five clusters consist of 64 P-III CPUs each and one has 144 CPUs. We took the run-time information for each job multiplied by a standard benchmark of a P-III 1GHz CPU (1.5GFlops) as the workload for each of the job. Also, the trace has the information on average amount of memory allocated to each jobs, so we took it as the memory footprint size. When a job is migrated, some of the information stored in the filesystem also needs to be carried. We assumed that total amount of data that needs to be carried for each migration is 3 times the average allocated memory for the job.

To evaluate the correlation of per user goodput and SLA capacity obtained from different algorithms, we needed a synthetic workload model where we can control the job arrival rate, but the arrival pattern and job execution-time distribution are modeled after real workload traces. For this purpose we chose the workload model defined by Lublin

et.al. in [64]. To explore the sensitivity of the model over a range of design parameter values, we needed more control and visibility of the workload, so we used a Poisson workload model with varying arrival rates. Each of the jobs had an workload of 5000 CPU-seconds on a dedicated machine, and had 1MB of memory footprint.

7.3.3 Choosing Between Scheduling Schemes

Among different service objectives of the PCU are minimizing the down-time and completion time of the jobs, and enforcing the proportional share of resources according to the SLAs. Replication has been used in order to minimize down-times, and the priority function is carefully designed to minimize the job running time as well as establishing fairness. One major question remains whether we need preemptive migrations to obtain service differentiation, such that the resource capacities delivered to the users are proportional to their SLA defined values. We have compared preemptive and non-preemptive strategies in terms of achieving this service differentiation. To quantify this achievement for different algorithms, we measured the correlation of per user goodput (V) and the SLA maximum allowable workload parameter V_0 (defined in Section 3.3.3).

In the experiment, we fed the PCU with synthetic workload from 100 different users. The V_{0i} (MaxLoad) parameter of the SLA for each user i was chosen randomly. The PCU had 100 dedicated resources, and the total concurrent load on the PCU was on average 5 times higher than the dedicated resource capacity. The workload offered by each user (B_i) was uniform across users and uncorrelated to their V_{0i} parameters. To generate the workload from each user we followed the model described in [64]. Among the two algorithms we compared, the first one (Non-preemptive) assigns the dedicated resources statically to the jobs at arrival time, considering the job from the user with highest SLA deviation (as

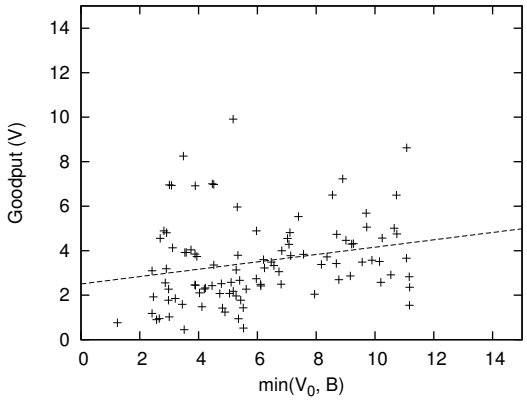

Figure 7.1: Correlation of Goodput with SLA MaxLoad parameter: Static Allocation. Pearson's correlation coefficient, $r = 0.1655306734$

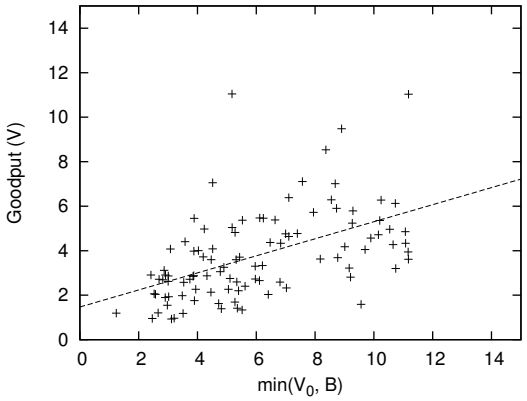

Figure 7.2: Correlation of Goodput with SLA MaxLoad parameter: Dynamic Priority Preemption. Pearson's correlation coefficient, $r = 0.3835632358$

defined in Section 3.3.3) first. Remaining jobs are assigned on available public resources. During the course of execution, a job is never migrated preemptively between dedicated and public resource pools. But, as public resources may fail, replicas of the jobs are reinstantiated on new public resources. In the second algorithm (Priority preemption), instead of allocating the resources statically, the scheduler dynamically computed the priority of each job as described in Section 7.2.3, and migrated the most starving jobs of the most starving users into the dedicated resources.

Figures 7.1 and 7.2 show the scatter-plot of user goodput (V) vs $\min(V_0, B)$ for the two algorithms, respectively. We observe that goodput have much higher correlation when preemptive migration with dynamic priorities is used. Although preemptive migration has high cost (network load and delay) associated with it, this result suggests that we need it in order to achieve service differentiation, especially if the workload has a substantial number of long-running jobs. This happens because the SLA deviation of different users change during the runtime of long running jobs and the best way to deal with this is to dynamically

allocate the scarce dedicated resources. Motivated by these results we have chosen the Priority Preemption scheduling for further performance analysis.

7.3.4 Setting the Design Parameters

In this section, we explore different performance metrics for the chosen Priority Preemption scheduling strategy and the associated trade-offs. In order to focus on the global performance of the PCU, we kept the workload generated by different users similar in this set of experiments, and SLA parameters were also set to be same for all users.

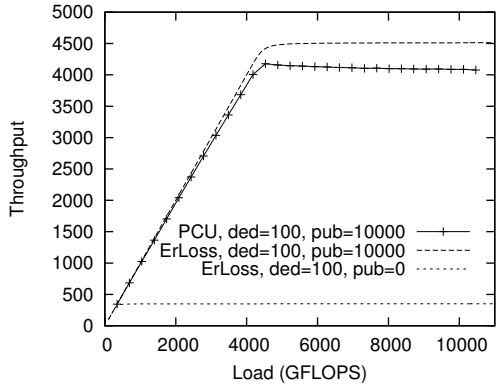

Figure 7.3: Comparing PCU throughput at different loads with Erlang Loss systems with equivalent number of resources. $N_d = 100$, $N_p = 10000$, $r = 2$, $P_{av} = 0.7$, $\frac{T_p}{T_d} = 0.34$

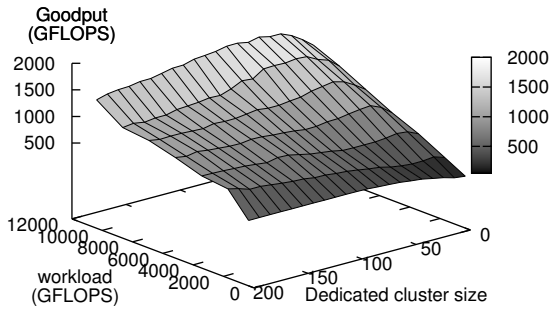

Figure 7.4: How Goodput is affected by number of dedicated resources at different loads

First we set up the experiments to measure the mean overall throughput of the PCU using this algorithm, compared to the theoretically achievable maximum throughput from the same system assuming zero cost for migration. An estimation of the total capacity and maximum throughput of the system can be derived from the analysis in Section 7.2.6.

Figure 7.3 shows the mean overall throughput from PCU for a range of load levels

and comparable theoretically maximum achievable throughput of equivalent systems. The above figure also plots the maximum achievable throughput from a system with only N_d dedicated resources. We observe that despite down-time for preemptive migrations, at moderate loads, PCU can deliver throughput almost equal to the ideal system that ignores migration cost. At high load however, the throughput marginally decreases, due to excessive network overheads for migrations.

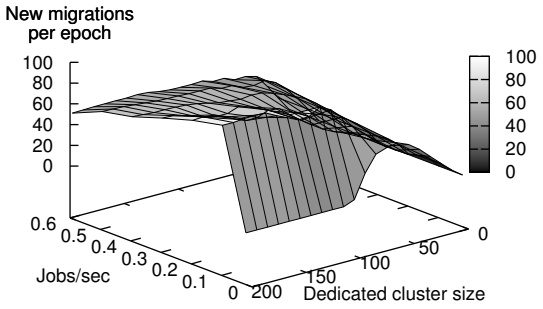

Figure 7.5: Load on the bottleneck network link due to preemptive migrations

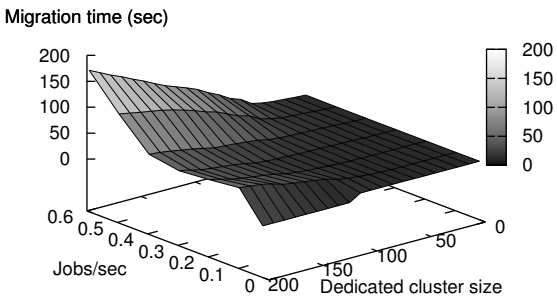

Figure 7.6: Time required for a migration through the bottleneck link

Now we examine the effect of increasing the size of the dedicated cluster. One obvious effect of this is an increased capacity of the PCU. But increasing the number of dedicated resources increases the possible number of migrations per scheduling epoch and hence higher migration cost, both in terms of job execution time and network overhead. Figure 7.4 illustrates the overall goodput of the PCU on the job arrival load vs dedicated cluster size space. It is important to note that network load increases with the number of migration decisions every epoch, and after some limit, the migration time grows towards infinity because the queues in the bottleneck links gets saturated.

Figures 7.5 and 7.6 show the effect of migration on the network. Figure 7.5 plots the number of new preemptive migrations started every epoch and Figure 7.6 plots the average

time taken for each migration, both on the job arrival load vs cluster size space. We observe that network load increase with increased job arrival load and also with increased size of dedicated cluster. Beyond a certain level of network load, the bottleneck link and routers on them gets saturated and each migration takes exponentially longer time to complete. This rise in network load limits the maximal goodput and minimal elongation across a certain band of the arrival load and cluster-size range. However, since the scheduler does not preempt the already ongoing migrations, the increasing network load actually reduces the number of new migrations every epoch acting as a negative feedback. Accordingly, goodput or elongation does not change much at higher load and bigger cluster size.

With the results of above experiments in hand we can now choose the dedicated cluster size and the load on the system from the safe regions of the graphs. As an example settings, we did the latter experiments with 100 dedicated resources and at a moderate workload that is 5 to 10 times higher than the total capacity of the dedicated cluster.

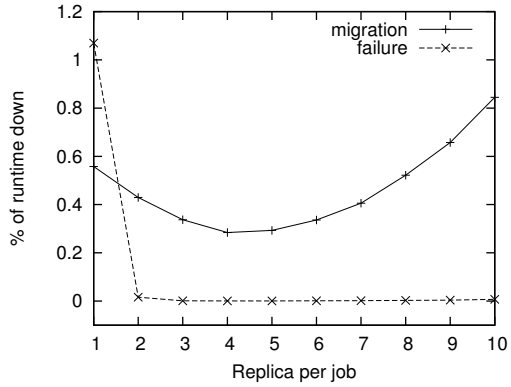

Figure 7.7: Downtime due to failure and preemptive migration, for increasing degree of replication

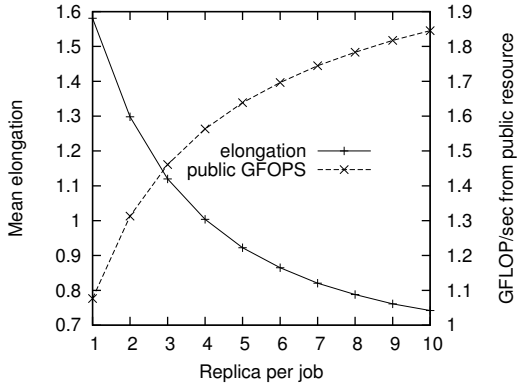

Figure 7.8: Higher degree of replication reduces elongation, because it increases the chance of getting higher capacity public resource

The next parameter we study is the number of replicas that concurrently run when a job is scheduled on public pool. The main rationale behind replication is to mask the failures

of public resources and reduce the down-time of the application due to failures. Here we observe the effect of replication on job's down-time and elongation, using the more versatile simulation model. General observation is that running only 2 concurrent replicas, dramatically reduces the chance of concurrent failures and hence reduces the down-time (Figure 7.7). On the other hand, higher degree of replication increases the network load and therefore, after some point, each migration takes longer to complete and job down-time for migrations actually increases. Another interesting effect of replication is that when we run many replicas on different public resources in parallel, it increases the chance of getting a higher throughput public resource from the global pool, and therefore, runtime of the jobs keeps decreasing (lower elongation) as we increase the degree of replication (Figure 7.8). For the same reason, the need for preemptive migration decreases with higher degree of replication, and we observe a decrease in job down-time caused my migrations, with the first few increases in the degree of replication (Figure 7.7). Although we gain in performance from public resource with increasing replication, it would greatly increase the traffic on the global network and also apparent waste of public resources. Therefore, we suggest that replication beyond order of two may not be desirable. A theoretical estimation of this replication factor is provided in Section 7.2.6.

Another important design parameter is the length of the scheduling epoch i.e., the elapsed time between scheduler invocations. Too short epochs implies very frequent migration decisions and hence, high network load. But as the epoch length increases, the time to detect a failure also increases, because failure is detected by the scheduler from time-outs of the life-pulses from remotely running jobs and this evaluation is done only when the scheduler is invoked. This adversely affects the down time due to failure, as shown in Figure 7.9. Figure 7.10 shows the mean elongation for increasing length of the epoch. We

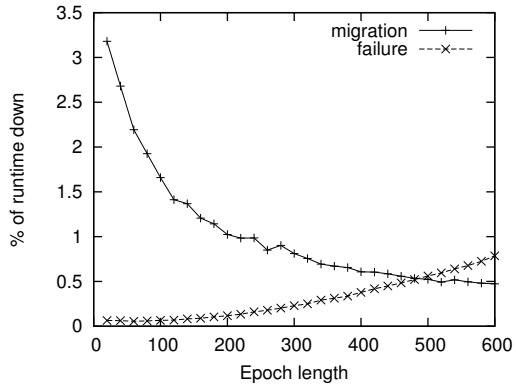

Figure 7.9: Downtime due to failure and preemptive migration at different lengths of scheduling epoch

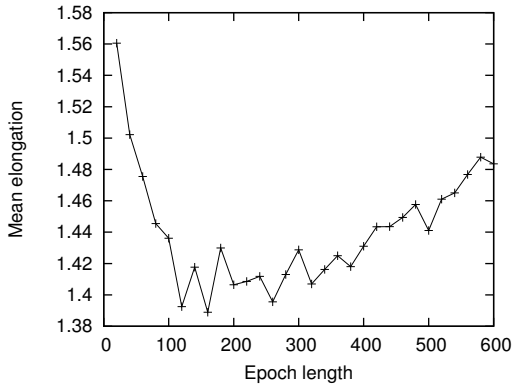

Figure 7.10: How elongation is affected by the length of the scheduling epoch

can observe that there is a optimal epochs length for this particular system settings.

7.3.5 Design Guidelines

Following the observations from both the simulation experiments and the queueing analysis presented in Section 7.2.6, a set of design principles can be outlined for future design of PCU systems –

- preemptive migration is useful to enforce service differentiation although it incurs overhead.
- The number of dedicated resources that can be deployed in a single cluster should be limited, depending on the capacity of the network link that connects it to the Internet. For larger deployments (e.g. more than 100 machines), multiple clusters need to be setup in different network domains.
- It is useful to keep concurrent replicas of a job, but with more than two replicas, the overheads outweigh the benefits.

- The scheduling epoch should be in range of 1-2 minutes, either longer or shorter epoch length causes performance degradation.

- PCU can tolerate workload much beyond the total capacity of the deployed dedicated resources and hence, PCU provider can oversubscribe users to a certain extent. An estimate of the reachable public resources within certain delay and bandwidth constraints and their failure rates can be used to determine the maximum limit of such overbooking.

7.3.6 Performance Comparison with Grid Systems

Once we got the acceptable ranges for all parameters, we show how the scheduler works with the Grid workload trace from DAS2 [1]. The main argument behind a PCU-like bimodal architecture is that dynamic provisioning of public resources allows much higher volume of workload to be handled by the PCU than having a system of statically provisioned dedicated resources only. Alternatively, offloading much of the work to the public resources, the the volume of dedicated resource provisioning can be cut down by a major fraction without affecting the performance of the service significantly. Both of these facts result in a much higher utilization of the expensive dedicated resources. In our simulation experiments, the workload from the DAS2 trace has been fed to a model PCU system with a range of dedicated resource volumes. From the results of the simulation plotted in Figure 7.11 we observe that a PCU having dedicated resource volume as low as 25% of the DAS2 resources, yields dramatic increase in utilization of dedicated resource as high as 250% with not more than 10% increase in job runtime.

The public resources become a major contributor to the total computational work, especially in times of peak loads, despite their low trust and high failure probabilities. We can

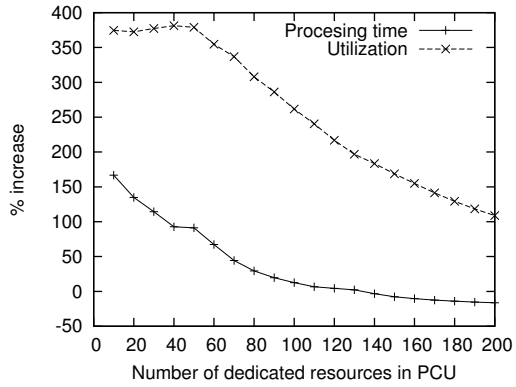

Figure 7.11: Comparing PCU with DAS2 Grid: gain in resource utilization and increase in running time

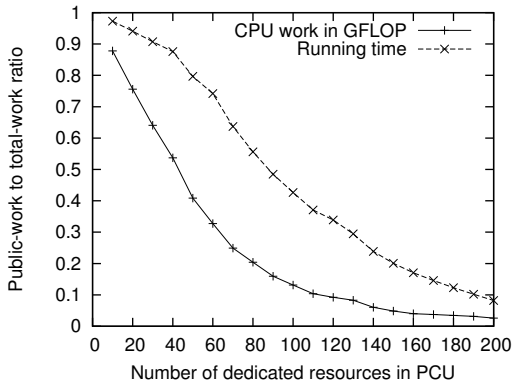

Figure 7.12: Amount of work done on public resources

observe in Figure 7.12 that for a reasonably sized PCU, approximately half of the total computation (30% to 60%) is done on the public resources. If we look into the down-time of the jobs due to migration and failure the general observation is that the although the downtime for migration time is higher than downtime for failure, it is insignificant compared to the gain in the service capacity we achieve.

7.4 Summary

In this chapter, we have analyzed the resource management problem in Public Computing Utility built from a combination of dedicated computing clusters and opportunistically available public copmuters, with elaborate characterization of the available in public resources and exploring the possibility of remote monitoring and live migration of computing processes. We have discussed the trade-offs among different strategies for managing the resources, and based on that, we have devised a scheduling algorithm that allocates resources from both dedicated and public resource pool in order to maximize the goodput of whole utility as well as the compliance with the user SLAs. We analyzed the system model

from the perspective of queueing and reliability theories and derived relationship between different design parameters and system performance. We carried out extensive simulations to establish several design decisions for the architecture and the algorithm. Finally for verifying the performance of the scheduling algorithm on the proposed architecture, we tested it with the compute intensive application workload from a university research Grid [1] and compared the resource utilization and job runtime elongation with the actual trace.

The results from comparison with DAS-2 Grid establishes that the PCU approach dramatically increases the virtual capacity of a computing service platform and enables the platform to substantially overbook for client service agreements. For example, we have seen that with very conservative provisioning of dedicated resources, we have more than 250% increase in average resource utilization compared to the dedicated resource based research Grid, with only minimal increase in execution time of the jobs on average. From the results we have also observed that the data transmission load on the network that connects the dedicated resource pool to the Internet constrains the total size the dedicated resource pool. Therefore, to get the maximum benefit from a given number of dedicated resources, we either need to provide enough network bandwidth, or distribute the resources in different networks. Another major factor that influences the performance is the actual capacity distribution and availability pattern of the public resources. A complete characterization of behavior of the public resources would help better tuning of the parameters for performance.

We have observed that preemptive migration is necessary for assuring service qualities for the clients. Wrapping application programs in virtual machines may be an option for facilitating migrations, however, a more lightweight wrapper that can monitor the application's resource usage as well as implement the migration protocol would be a better option

for implementation of a PCU.

8

Allocation of Network and Node Resources for Stream Processing

8.1 Overview

The series of two chapters, this chapter (Chapter 8) and the following one (Chapter 9), explores the two aspects of managing the server nodes and communication links in a distributed platform that serves data-intensive stream processing applications. Study of this data-intensive platform is performed to extend the concept of bi-modal resource based platforms to include communication resources. Earlier, in Chapter 6 and Chapter 7, aspects of bi-modal management of CPU resources have been explored for high throughput computing applications, where CPU is the most critical resource for performance. Here we explore bi-modal management of network resources in a distributed stream processing platform, where communication resources play the critical role in performance achievement. As described in the architectural details in Chapter 3, the platform attempts to make optimal use of two different types of communication links between its server nodes – the privately installed or leased dedicated links and the overlay links through the public Internet.

Stream processing applications involve a cascade of computational operations or *services* on one or more streams of data originating from some data sources, and delivery of

the data after all processing into one or more specific destination nodes. For example, a video stream can progress through several phases of processing such as encoding, embedding of real-time weather or financial tickers and transcoding into different formats. Each of the individual operations may be served by different servers in distributed locations. After being processed through all the services, the stream may be delivered to a specific user. Because different services are performed by different servers, these services require appropriate amount of computing resources in the servers as well as sufficient communication bandwidth between the servers to maintain the required data delivery rate.

As described in Chapter 4, we divide the resource management problem in our proposed distributed stream processing platform into two phases. The first phase maps or assigns the component services of the stream processing task to appropriate servers, fulfilling the composition order, the processing capacity required for each component and the transmission bandwidth required between each consecutive pair. In this chapter, we analyze the mapping problem in details, develop algorithms to solve it and evaluate the algorithms through simulations.

The servers are installed in distributed locations and interconnected with a communication network of arbitrary topology. Each server may serve multiple service components. The specification of a stream processing task contains the addresses of the data sources and data delivery destinations, name of the set of service components to process the data, and a topology that describes the data-flow through the service components.

We restrict our study to stream processing tasks with linear path flow topologies only. Although, in the general case the flow topology can be any acyclic graph, for many important applications, the composition can be expressed in a linear path topology [61]. We show in Section 8.2.3 that even for a linear path-like flow, finding a mapping of the service

components on the servers and data transmissions on network paths, satisfying the processing capacity and bandwidth constraints, is an NP-complete problem. In this chapter, we develop schemes to solve the problem of mapping linear path-like tasks on a network of servers with arbitrary topology. We present algorithms to find the optimal solutions and derive some heuristics for computing the near-optimal solutions efficiently.

Initially, we present a centralized algorithm using a dynamic programming based extension of the Bellman-Ford scheme [22]. It assumes that the global state of the network is available at a single location prior to the computation of the mapping. This scheme is then extended into a distributed algorithm, where the servers collaboratively compute the mapping only with local knowledge of the network neighborhood and processing capacity. Because the problem is NP-complete, the cost of these exact algorithms grows exponentially with problem size. Therefore we propose some heuristics that reduces the cost of executing the algorithm while producing near-optimal results. Performance of these heuristics are evaluated through simulation.

The rest of this chapter is organized as follows. In Section 8.2 we formally define the resource allocation problem as a constrained graph mapping problem. The Capacity Constrained Path Mapping (CCPM) problem that covers the compositions with path topology is then defined as a special case of the general graph mapping problem. We provide a formal proof of NP-completeness of the CCPM problem in the same section. In Section 8.3, centralized and decentralized algorithms to solve the CCPM problem are developed. A set of heuristics to obtain cost-effective approximate solutions to the problem is provided at the end of the same section. Section 8.4 presents some simulation results that evaluates these heuristics against each other. The algorithms and analysis developed in this chapter are published in [14].

8.2 The Resource Allocation Problem

In this section, we formally define the problem of capacity constrained mapping of stream processing tasks on arbitrary server networks. Any distributed stream processing task can be defined using three types of nodes and interconnections between them. *Source nodes* are the data sources originating the data streams. *Computing nodes* are services where some computational operations on one or more incoming data-stream is performed continually, and an output stream is generated. *Sink nodes* are the servers where the resulting flow from the task is presented. In the general case, a task consists of one or more source nodes, one or more sink nodes and zero or more computing nodes. The topology of data-flow among these nodes is a directed acyclic graph (DAG). Although, theoretically it is possible to have task topology that have loops or cycles, there will be finite number of iterations of the data through the cycles and these iterations can be expanded into finite acyclic graphs. In most common cases however, the data-flow topology is a simple path consisting of a series of computing nodes, or a tree where data-streams from multiple sources merged through several steps and presented at a single sink.

The network of computing and data-forwarding servers over which the distributed stream processing tasks are to be instantiated can be represented by an arbitrary graph. We denote this graph as *resource graph*. The services required by the stream processing tasks may be of different types. Each server may serve a subset of all possible services. In this chapter, we assume that all possible types of services are available in each server, and thus eliminate the constraints of service type matching. Nevertheless, each sever node in the resource graph has a certain computational capacity and each edge (link) of the resource graph has certain data transmission capacity or bandwidth. In addition, each link may have one or more additive cost metric, such as latency or jitter.

8.2.1 Capacity Constrained Graph Mapping Problem

In order to launch the stream processing task on the network of servers, we need to map the dataflow-DAG onto the resource graph such that the computational and transmission requirements are fulfilled. If there is more than one such feasible mapping, one would like to choose the mapping that has minimum end-to-end total cost on the resource network.

More formally, we need to map a dataflow-DAG $G_J = (V_J, E_J)$ on to a resource graph $G_R = (V_R, E_R)$. For each vertex $v_R \in V_R$, an available computational capacity $C_{av}(v_R)$ is given. For each edge $e_R \in E_R$, an available bandwidth $B_{av}(e_R)$ is given. In addition, each edge $e_R \in E_R$ has an additive cost $D(e_R)$. For each vertex $v_J \in V_J$, a computational requirement $C_{req}(v_J)$, and for each edge $e_J \in E_J$, a bandwidth requirement $B_{req}(e_J)$ is defined. There is a set of designated source nodes $S_J \subset V_J = \{s_{1J}, s_{2J}, ..., s_{mJ}\}$ and a set of sink nodes $T_J \subset V_J = \{t_{1J}, t_{2J}, ...t_{nJ}\}$, such that $S_J \cap T_J = \phi$.

The capacity constrained DAG-mapping problem (CCDM) is to find a mapping $M : V_J \rightarrow V_R$. For each source node s_{iJ}, $M(s_{iJ}) = s_{iR}$ and for each sink node t_{iJ}, $M(t_{iJ}) = t_{iR}$ are already given. It is important to note that multiple nodes of the dataflow-DAG can map onto single node of the resource graph and a single edge in the dataflow-DAG can span along a multi-hop path in the resource graph. So, defining the $V_J \rightarrow V_R$ mapping is not sufficient to define the mapping of complete dataflow-DAG. In addition to vertex mapping, another mapping $M_e : E_J \rightarrow P_R$ is needed, where P_R is the set of all possible paths in the resource graphs, including zero length paths. Zero length paths are (v, v) edges with infinite bandwidth and zero latency. Again, it is possible that for two different edges, $e_1, e_2 \in E_J$, the mapped paths $p_1 = M_e(e_1)$ and $p_2 = M_e(e_2)$ may have some common edges.

The mapping should fulfill the following constraints –

$$\forall v_R \in M(V_J)$$

$$\sum_{\{v_J | v_J \in V_J, M(v_J)=v_R\}} C_{req}(v_J) \;\; \leq \;\; C_{av}(v_R)$$

$$\forall e_J = (u, v) \in E_J,$$

$$B(e_J) \leq min[B(e_r), e_r \in M_e(e_J)]$$

When each edge $e_r \in E_r$ in the resource graph has an additive metric $D(e_r)$, such as delay, cost, jitter, etc., we would like to find the feasible mapping that minimizes the total cost

$$D = \sum_{e_J \in E_J} \sum_{e_r \in M_e(e_J)} D(e_r)$$

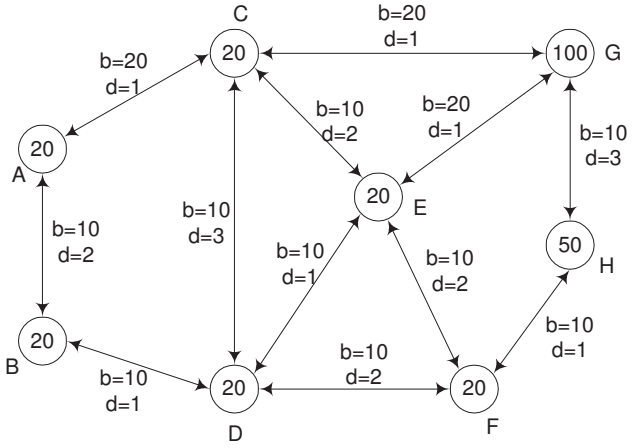

Figure 8.1: An example resource network

Figure 8.1 shows an example resource network of eight interconnected computing nodes. Computational capacity of each node is represented by a number inside the node.

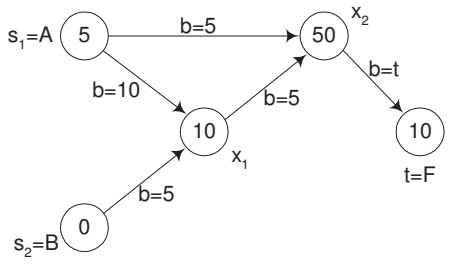

Figure 8.2: An example data-flow computation with a DAG topology

The link bandwidth and latency are mentioned on each edge. Figure 8.2 shows a dataflow-DAG containing 2 source nodes s_1 and s_2, 2 computing nodes x_1 and x_2, and one sink node t. s_1, s_2, and t must be mapped on resource node A, B, and F, respectively. Each node in the dataflow-DAG has some processing capacity requirement which is mentioned inside the node. Each link is also annotated with a bandwidth requirement. A feasible mapping of this dataflow-DAG on the resource graph is –

$$
\begin{array}{rcl|rcl}
M(s_1) & = & A & M_e(s_1, x_1) & = & (A, C, E) \\
M(s_2) & = & B & M_e(s_2, x_1) & = & (B, D, E) \\
M(x_1) & = & E & M_e(x_1, x_2) & = & (E, G) \\
M(x_2) & = & G & M_e(s_1, x_2) & = & (A, C, G) \\
M(t) & = & H & M_e(x_2, t) & = & (G, H, F)
\end{array}
$$

8.2.2 Constrained Path Mapping Problem

Although in very general terms, the data flow topology of a stream processing task resembles a DAG topology, in most practical cases the topology is a simple path. Given that the

mapping of a DAG efficiently on the resource network with all the constraints satisfied is hard to solve, it is useful to tackle the simpler problem of capacity constrained path mapping (CCPM) first. In CCPM, the task topology is restricted to a directed loop-free path, with a single source and a single sink.

Precisely, we are given a dataflow path $P_J = (V_J, E_J)$, $V_J = v_0 = s, v_1, v_2, ..., v_m = t$ and $E_J = \{e_i = (v_i, v_{i+1}) | 0 \leq i < m\}$ to map on the resource graph $G_R = (V_R, E_R)$ defined in the previous section. Each node $v_i, 0 \leq i \leq m$ of the path has a computational capacity requirement $C_{req}(v_i)$, and each edge $e_i = (v_i, v_{i+1}), 0 \leq i < m$ has a bandwidth requirement $B_{req}(e_i)$. We need to find the mappings $M : V_J \rightarrow V_R$ and $M_e : E_J \rightarrow E_R$ that satisfies the constraints. Mapping of s and t is already given.

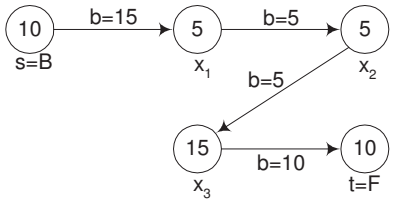

Figure 8.3: An example data-flow computation with a path topology

An example stream processing task with one source s, one sink t and three computational nodes x_1, x_2, x_3 is shown in Figure 8.3, with the node capacity and bandwidth requirements. s and t must be mapped on B and F, respectively. There can be many feasible mappings of this task on the resource graph in Figure 8.1. One of them is –

$$\begin{array}{rcl|rcl} M(s) & = & B & M_e(s, x_1) & = & (B, B) \\ M(x_1) & = & B & M_e(x_1, x_2) & = & (B, B) \\ M(x_2) & = & B & M_e(x_2, x_3) & = & (B, D) \\ M(x_3) & = & D & M_e(x_3, t) & = & (D, F) \\ M(t) & = & F & & & \end{array}$$

which is also optimal in terms of total end-to-end latency of the resource nodes $M(s)$ and $M(t)$.

8.2.3 Computational Complexity of the Problem

We will now prove that CCPM problem is NP-complete. Since, CCPM is a special case of CCDM, NP-completeness of CCPM implies that CCDM is an NP-hard problem. The NP-completeness proof of the CCPM problem is constructed by transformation of the Longest Path problem [46]. Definition of the decision version of the Longest Path problem is as follows -

Instance: A graph $G = (V, E)$, a length function $l : E \rightarrow Z^+$, specified vertices $s, t \in V$ and a positive integer K. *Question*: Is there an $(s \rightsquigarrow t)$ simple path $P \subseteq G$ such that $\sum_{e \in P} l(e) \geq K$?

It is known that the Longest Path problem is NP-complete, even for a special case, where $\forall_{e \in E} l(e) = 1$ [46]. We will show that any instance of this special Longest Path problem can be polynomially transformed into an instance of CCPM.

Longest Path $\propto$ CCPM

We construct an instance of CCPM as follows -

We take $G_R(V_R, E_R) = G(V, E)$, $\forall_{v \in V_R} C_{av}(v) = 1$, $\forall_{e \in E_R} B_{av}(e) = 1$. Take a simple path $P_J = (V_J, E_J)$ such that $|V_J| = K$, $\forall_{v \in V_J} C_{req}(v) = 1$ and $\forall_{e \in E_J} B_{req}(e) = 1$.

Now, if there is a simple $(s \rightsquigarrow t)$ path of length $\geq K$ in G, then that path must have K hops, since $\forall_{e \in E} l(e) = 1$. Therefore, we can map P_J along the corresponding path $P_J\prime$ in G_R. If $|P_J\prime| > K$, then we can map first $K - 1$ nodes of P_J on $P_J\prime$ and map the remaining edge u_{K-1}, u_K on the $v \rightsquigarrow t$ subpath of $P_J\prime$, where u_{K-1} is mapped on v.

Given a mapping of the path P_J on a path $P_J\prime \subseteq G_R$ that satisfies the capacity and bandwidth requirement constraints, $|P_J\prime|$ must be $>= K$, because no two vertices of P_J can be mapped on a single vertex of $|P_J\prime|$ given the abovementioned capacity constraints.

CCPM $\in$ NP

Given an arbitrary mapping $M : V_J \rightarrow V_R$ one can polynomially verify -

- Whether $C_{req}(v) \leq C_{av}(M(v))$, for all $v \in V_J$.
- For each edge $(u, v) \in P_J$, whether there is a $(M(u) \rightsquigarrow M(v))$ path in G_R that satisfies the bandwidth constraint of (u, v) (Similar to bandwidth constrained shortest path problem [93]).

This completes the proof that $CCPM \in NP\text{-}C$.

8.3 Algorithm for Path Mapping Problem

To solve the CCPM problem, we developed an algorithm using the Bellman-Ford relaxation scheme. First, we present the centralized version of the algorithm, where the whole

mapping is computed by a single node that has knowledge of the state of the whole network of nodes. Later, we explain the development of the distributed algorithm based on this centralized one.

This algorithm works by relaxing along each edge of the resource graph $N-1$ times, where $N = |V_R|$, the number of nodes in the resource graph. For each node u of the resource graph, a set of feasible mappings of different length prefixes of the dataflow-path on any resource path from the source node s to the current node, is maintained. In each relaxation along an (u, v) edge, any new feasible map on $(s \rightsquigarrow u)$ is extended in all possible ways, to complete the list of feasible maps of dataflow path-prefixes on the resource path $(s \rightsquigarrow u, v)$ and these new partial mappings are added to the set maintained for node v. After $N-1$ iterations of relaxation of all edges, the map set maintained for terminal node t contains all the feasible mappings of the dataflow-path on any $(s \rightsquigarrow t)$ resource path. The algorithm is presented in Algorithm 2, 3 and 4. A formal proof of the correctness of the algorithm is presented in the following sub-section. Lines 10-12 of the subroutine Relax is added to terminate the algorithm as soon as one feasible $(s \rightsquigarrow t)$ mapping is found. These lines should be omitted when optimal mapping is sought.

We have computed the computational complexity of the algorithm in Section 8.3.2. The complexity is bounded by polynomial of the size of the partial map set S, although the set size is exponential. The problem being NP-hard, it is impossible to have a polynomially bounded optimal algorithm. However, heuristics may be applied to produce sub-optimal solutions within a tractable amount of complexity. A good way of designing such heuristics is to restrict the size of the map-set in some way. In Section 8.3.4 we have discussed several possible heuristics to solve the CCPM problem. Note that because the set of partial map is stored in each node, the memory complexity of the algorithm becomes exponential too.

This can be avoided by omitting the storage of partial maps. Each partial map need to be stored for one iteration of relaxation only. If partial maps are deleted after relaxation, the set size never grows beyond $O(dp)$, where, d is the average in-degree of a node in resource graph and $p = |P_J|$ is the number of nodes in the dataflow path.

Algorithm 2 *Pathmap(P_J, G_R)*

1: **for** $x = 0$ to $|P_J| - 1$ **do**
2: **if** $\sum_{0 \leq k \leq x} C_{req}(k) \leq C_{av}(s)$ **then**
3: $M(s, x) = \{m | m$ maps initial x nodes of P_J on $s\}$
4: **else**
5: **break**
6: **end if**
7: **end for**
8: **for** each vertex $v \in V_R - s$ **do**
9: **for** $i = 0$ to $|P_J|$ **do**
10: $M(v, i) = \phi$
11: **end for**
12: **end for**
13: **for** $i = 1$ to $|V_R| - 1$ **do**
14: **for** each edge $e = (u, v) \in E_R$ **do**
15: Relax(u,v)
16: **end for**
17: **end for**

8.3.1 Correctness of the CCPM Algorithm

In this section we give a formal proof that when the CCPM algorithm terminates, $M(t, |P_J|)$ always contains a feasible mapping of P_J on G_R if and only if such a feasible mapping exists.

Lemma 8.3.1. *If $M(u) = \bigcup_{\forall j} M(u, j)$ contains all feasible mappings of different length prefixes of P_J on an path $(s \rightsquigarrow u) \in G_R$, then after computing **Relax**(u, v), $M(v)$ includes all feasible mappings of different length prefixes of P_J on the path $(s \rightsquigarrow u, v) \in G_R$.*

Algorithm 3 subroutine *Relax(u,v)*

```
1:  for j = 0 to |P_J| do
2:      M_tmp(j) = null
3:  end for
4:  for j = 0 to |P_J| − 1 do
5:      if B_req(j, j+1) ≤ B_av(u, v) then
6:          for each new mapping m ∈ M(u, j) in the last iteration do
7:              if v == t then
8:                  m_x = Extend(m, j, |P_J| − j, v)
9:                  M(v, |P_J|) = M(v, |P_J|) ∪ m_x
10:                 if M(v, |P_J|) ≠ φ then
11:                     terminate the algorithm with M(v, |P|) as result
12:                 end if
13:             else
14:                 for x = 0 to |P_J| − j − 1 do
15:                     m_x = Extend(m, j, x, v)
16:                     if m_x ≠ null then
17:                         M(v, j + x) = M(v, j + x) ∪ m_x
18:                     else
19:                         break
20:                     end if
21:                 end for
22:             end if
23:             mark m as old
24:         end for
25:     end if
26: end for
```

Proof. By the construction of the **Relax**(u, v) subroutine, each mapping $m \in M(u, j)$, of a j-length prefix of P_J on a $(s \rightsquigarrow u)$ path, is extended over the (u, v) edge exactly once. Any possible mapping of a k-length prefix of P_J on the $(s \rightsquigarrow u, v)$ path can be divided into 2 sub-mappings: a mapping of j-length prefix $(j \leq k)$ of P_J on $(s \rightsquigarrow u)$ path and a mapping of the following $k - j$ vertices of the k-length prefix on v. Since all feasible sub-mappings of the first kind is included in $M(u)$ and all the extensions of the second kind is considered in lines 8 to 14 and 15 to 22 of **Relax**(u, v), $M(v)$ contains all feasible mappings of any

Algorithm 4 subroutine *Extend(m, j, x, v)*

1: **if** $\sum_{1 \leq k \leq x} C_{req}(j+k) \leq C_{av}(v)$ **then**
2: extend m by putting computations $\{j+1, j+2, ..., j+x\}$ in node v
3: let m_x be the extended mapping
4: **else**
5: $m_x = null$
6: **end if**
7: **return** m_x

prefix of P_J on $(s \rightsquigarrow u, v)$ paths. □

Lemma 8.3.2. *For any node $v \in V_R$ if there is a $s \rightsquigarrow v$ path $(v_0 = s, v_1, v_2, ..., v_k = v)$ of length k, after kth iteration of the outer for loop in line 7 of the $PathMap$ algorithm, all feasible mappings of different length prefixes of P_J on the $(v_0 \rightsquigarrow v_k)$ path has been recorded in $M(v)$.*

Proof. We will prove by induction on k. When $k = 0$, i.e. after the initialization phase, $M(v_0, i)$ or $M(s, i), 0 \leq i \leq |P_J|$ contains the feasible i-length prefix with first i vertices of P mapped on s. So the basis is true.

Now let us assume that after $i-1$ iterations, $0 < i \leq k$, $M(v_{i-1})$ contains all feasible mappings of different lengths on the $(s \rightsquigarrow v_{i-1})$ portion of the $(s \rightsquigarrow v_k)$ path. Since each edge in E_R is considered once in each iteration, $Relax(v_{i-1}, v_i)$ must be called in the ith iteration too. So, by Lemma 8.3.1, we can conclude that all feasible prefix mappings of P_J on the $(s \rightsquigarrow v_i)$ path is included in $M(v_i)$. □

Theorem 8.3.3. *After $|V_R| - 1$ iterations of the outer loop in line 7 algorithm $Pathmap$, for each node $v \in V_R$, $M(v)$ contains all feasible mappings of different length prefixes of P_J on all possible $s \rightsquigarrow v$ paths.*

Proof. Since there is no simple path longer than $|V_R| - 1$, according to Lemma 8.3.2, all such paths will be covered by the $Relax$ procedure after $|V_R| - 1$ iterations. □

The fact that after termination of $Pathmap$, $M(t)$ contains all the feasible maps of P_J on possible $(s \rightsquigarrow t)$ paths, follows directly from Theorem 8.3.3 with inclusion of lines 7 to 12 in the *Relax* procedure.

8.3.2 Complexity of the Algorithm

The problem size parameters are $|V_R| \equiv n$, $|E_R| \equiv e$ and $|P_J| \equiv p$. The outer loop of *Pathmap* is iterated $n-1$ times and each iteration considers each of the e edges exactly once. So, the *Relax* procedure is called ne times. In each relaxation over an edge (u, v), each of the p prefix mappings from $M(u)$ is tried for relaxation into some of the p mappings in $M(v)$. A j length prefix in $M(u, j)$ is tried for relaxation into $p-j$ of the $M(u, i), j \leq i \leq p$, and each trial requires $(i-j)$ computations of constant complexity for the extension. Let S be the maximum number of entries in the set of mappings $M(u, j), u \in V_R, 0 \leq j \leq p$. Note that only the new entries are relaxed in each iteration. However, the upper bound on the number of entries relaxed per $M(u, j)$ will be S. So, the complexity of *Relax(u,v)* is –

$$\begin{aligned} S\sum_{j=0}^{p-1}\left(\sum_{x=1}^{p-j-1} x+1\right) &= S\left(\frac{5}{12}p^3+\frac{1}{4}p^2+\frac{2}{3}p\right) \\ &= O\left(\frac{5}{12}p^3S\right) \end{aligned}$$

So, the overall time complexity of the algorithm becomes $O(nep^3S)$. Note that, in the worst case, when the nodes and edges of the resource graph has much higher capacity than the requirement, S may be the number of all possible $s \rightsquigarrow t$ paths of length j in G_R, which grows exponentially with j. We see that the sets $M(u, j)$ are creating the major load on both time and memory complexity of the algorithm. Therefore, restricting the growth

of S within polynomial limit would possibly result in a polynomial time approximation algorithm.

8.3.3 Distributed Version of the Algorithm

The centralized algorithm can be easily extended to a distributed version, where each node u in the resource network G_R will maintain the data structure $M(u)$ of partially computed mappings. Also, node u will be responsible for computing the relaxation to each of its neighbors v in G_R. The extended mappings are then transmitted to v. The relaxation procedure is invoked by a node u when any new mapping arrives from any of its incoming neighbors. The algorithm is formally laid out in Algorithm 5. Upon arrival of a map message m, a node u process the message using the algorithm *ProcessMap(u, m)*. It follows from the correctness of the centralized algorithm that the distributed mapping completes after at most $N - 1$ *ProcessMap* invocation by each node in the graph. The algorithm terminates after all the outstanding *ProcessMap* have been completed. Since cycles are avoided during extension, an initial mapping may be extended to at most $N - 1$ hops. Thus there will be a finite number of *ProcessMap* invocation and the algorithm will terminate after a finite amount of time.

8.3.4 Heuristic Approaches to Reduce Complexity

Computational complexity of both the centralized and the distributed path mapping algorithm grows exponentially with the problem size. Therefore, for practical deployment, we need some heuristic that produces good approximation to the optimal result. Here we discuss three possible heuristics that modify the original algorithm to reduce computational, messaging and memory complexity.

Algorithm 5 *ProcessMap(u, m)*

Map message contains the mapping of computation nodes 0,1,2, ... , j on resource nodes. The first message to a node contains the requirement definition of the computation too
$j = |m|$
if $u == t$ **then**
  $m_x =$ Extend(m, j, $|P_J| - j$, u)
else
  for $x = 0$ to $|P_J| - j - 1$ **do**
    $m_x =$ Extend(m, j, x, u)
    if $m_x \neq null$ **then**
      for each neighbor v of u that is not already in m **do**
        if $B_{req}(j + x, j + x + 1) \leq B_{av}(u, v)$ **then**
          extend m_x to $m_x x$ by appending a map of 0 computations on node v
          send $m_x x$ to v
        end if
      end for
    end if
  end for
end if

LeastCostMap

One major source of growth in complexity of the algorithm is the exponential growth of the set of partial maps maintained for each node. In the *LeastCostMap* heuristic, only one partial map of each prefix-length is maintained for each node. If a new map is generated, the cost of the new map in terms of the additive quality metric is compared with that of the already stored one, and the map with higher cost is discarded. This policy reduces the complexity to $O(p^3)$.

Similar policy can be applied to the distributed version of the algorithm. However, in the distributed case, a map message is expanded to its neighbors as soon as the message is received. So, if a higher cost map message is arrived before a lower cost one, the processing of the higher cost message cannot be pruned. However, in most cases, higher cost messages

arrive later, so they are pruned.

AnnealedLeastCostMap

One way of trading off between optimality and complexity of the *LeastCostMap* heuristic is to apply a simulated annealing approach to decide whether to discard a higher cost partial map from the set in presence of a lower cost map. As the temperature of the process anneals, i.e. at the later iterations, the probability of keeping a non-minimal partial solution will decrease. Definitely this approach increases the computation and message complexity. However, this allows some of the non-minimal partial solutions to grow and possibly lead to a better complete solution.

RandomNeighbor

Another way of restricting the message complexity is to extend any partial map to a randomly chosen subset of k neighbors instead of expanding to all of them. Higher values of k increases the chance of getting the optimal solution. The *RandomNeighbor* heuristic with $k = 1$ did not produce results as good as LeastCostMap, although the number of messages was reduced dramatically. Further investigation need to be done to determine a suitable value of k.

8.4 Performance of the Heuristics

In this section we present some simulation results that compare the performance of the heuristics described in the previous section. We implemented the heuristics on a network of servers having a static configuration. Each of the servers is assumed of have a certain

processing capacity and each of the network link has certain bandwidth. The server capacities were randomly sampled from the distribution of node capacities on large scale volunteer computing project [5]. The network topology was generated by BRITE Internet topology generator [69], using the Barabasi-Albert algorithm [18]. This generates a power-law graph and the link bandwidths were sampled from a truncated power-law distribution having min=10Mbps and max=1Gbps. The mapping heuristics are coded in Java as mapping agents for each of the nodes. The agents respond to messages received from their neighbors and propagate the expansion of the partial mappings. Each agent contains only the local knowledge of the processing capacity of its own node and the link bandwidths to its direct neighbor. The network is emulated on a set of computers on a LAN, where agents representing nodes in the network runs on different computers and communicate among them using UDP packets. The LAN settings does not affect the test results because the agents calculate the mappings based on the capacities and bandwidths assigned from the models, instead of the actual capacities of the computers and the LAN.

In the tests, we tried to evaluated the quality of the solutions obtained by the heuristics, as well as the cost of computing the solutions using each heuristic. To measure the quality of the solutions, we used two metrics. First, the heuristic may not be able to find a solution to the mapping problem even when a solution exists. So, the first metric counts the percentage of such false negative cases for the tasks mapped by the heuristics. The second metric measures the closeness of the solutions generated by the heuristics to the optimal solution. The closeness is measured as the ratio of the cost of heuristic derived solution to the cost of the optimal solution.

Because the messaging and computational cost of the optimal algorithm described in Algorithm 5 grows exponentially, it is not feasible to find the solution for all ranges of

network sizes. Therefore, we devised an algorithm that computes a lower bound of the optimal solution. We relaxed the bandwidth constraints and mapped the problem into finding a optimal cost path in a multi-stage graph. The first and last stages resemble the source and the terminal nodes. Each of the internal stages have n nodes, resembling the choice of any of the n servers for the processing components of the tasks. Then we compute the lowest cost path from source to the terminal nodes, subject to node capacity constraints only. Ignoring the bandwidth constraints in the relaxed problem allows lower cost solutions that are not feasible in the actual problem. All the feasible solution for the actual problem will be feasible in the relaxed problem. So, the optimal solution of the relaxed problem will be a lower bound on the optimal cost of the actual problem. We computed the ratio of the cost of heuristic generated solutions to this lower bound cost.

To assess the cost of executing the heuristics, we counted the total number of map-extension messages exchanged among the nodes. Because arrival of each map message invokes the processing algorithm on the receiving node, the total computational cost is proportional to the number of map messages. Although we did not evaluate the message complexity of the exact algorithm, we have compared the complexities of the heuristics, which have helped us to choose one heuristic over the others.

We evaluated each of the metrics across ranges of three system parameters – number of nodes in the network, number of components in the stream processing tasks and average processing load of a task component compared to average processing capacity of the servers.

In Figures 8.4, 8.5 and 8.6 show the closeness of the heuristic derived solutions to

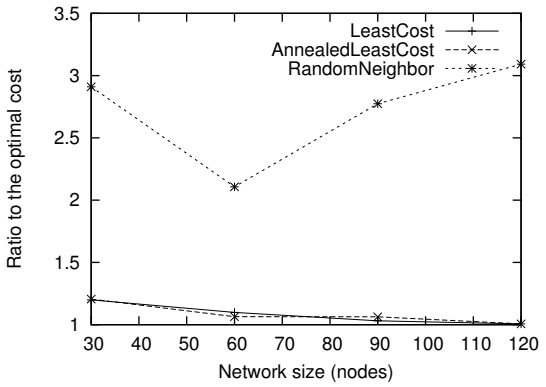

Figure 8.4: The ratio of the cost of heuristic derived solutions to the lower-bound cost of the optimal solution, across different sizes of networks.

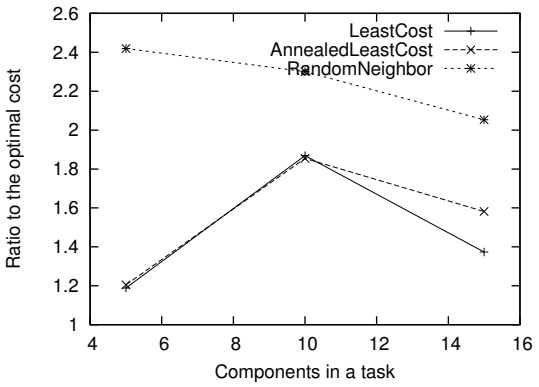

Figure 8.5: The ratio of the cost of heuristic derived solutions to the lower-bound cost of the optimal solution, across different number of components in the stream processing tasks.

the lower bound of optimal solutions. We can observe that the LeastCost and the AnnealedLeastCost heuristics perform fairly well and get solutions that are very close to optimal solutions, especially in cases where the task compositions are shorter in terms of number of components. The AnnealedLeastCost generates slightly better solutions than the LeastCost, because it allows more partial mapping to be expanded. The ratio to the optimal solution increases when the length of the composition path increases, because the more the number of hops in the composition, the more is the proportion of partial maps being discarded, in all three heuristics. We can see that in all cases, the the RandomNeighbor heuristic does not produce very good solutions, because number of feasible of ways to expand the partial maps narrows down very quickly here.

Another measure of the quality of the heuristic derived solutions is the proportion of false negative results. As shown in Figures 8.7, 8.8 and 8.9, we observe that both the LeastCost and AnnealedLeastCost generates much less number of false negative results than the random neighbor heuristic. The heuristics scale well in terms of finding the solution where

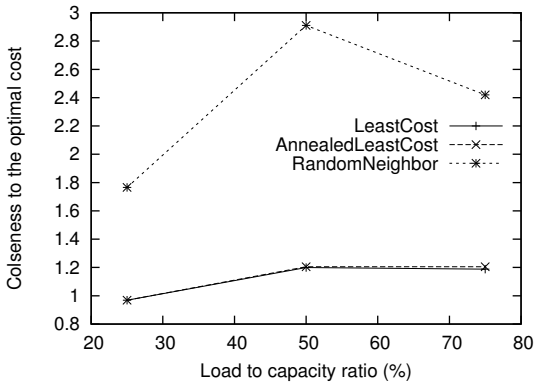

Figure 8.6: The ratio of the cost of heuristic derived solutions to the lower-bound cost of the optimal solution, across different load to capacity ratios.

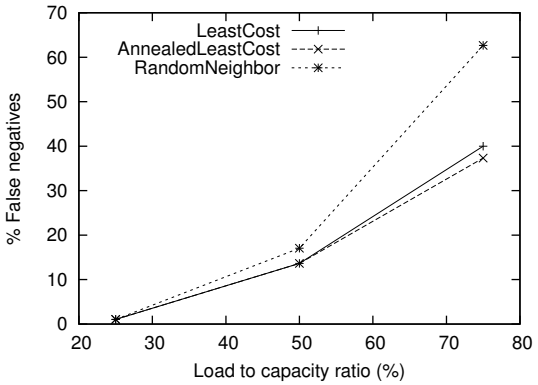

Figure 8.7: Percentage of false negative results, across different load to capacity ratios.

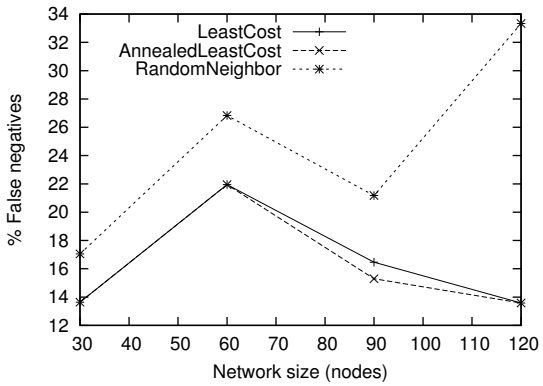

Figure 8.8: Percentage of false negative results, across different sizes of the server network.

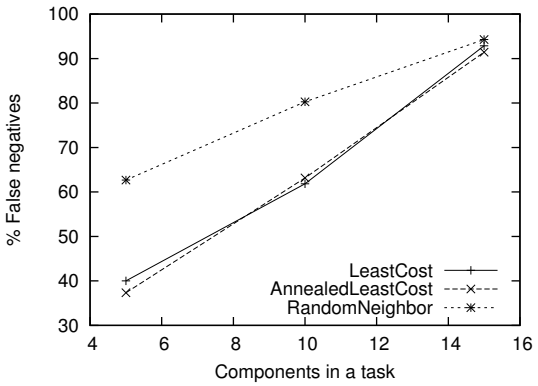

Figure 8.9: Percentage of false negative results, across different number of components in the stream processing tasks.

it exists, across a good range of network size. However, when the length of composition increases, the proportion of false-negatives become as high as 90%. Part of this high ratio stems from the fact that the total number of actual positive cases, i.e. compositions where at least a solution exists, becomes very small when the compositions are very long. Similar situation occurs, when the load-ratio of the compositions are very high (Figure 8.7).

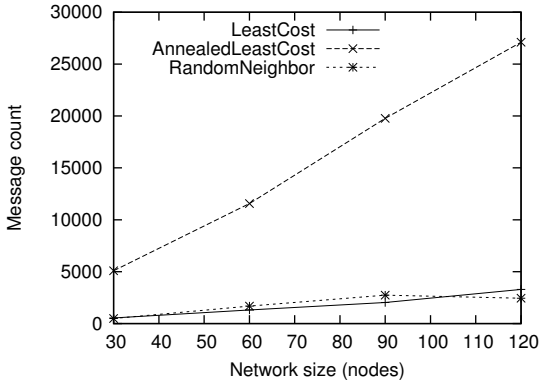

Figure 8.10: Total number of map-extension messages exchanged, across different sizes of the server network.

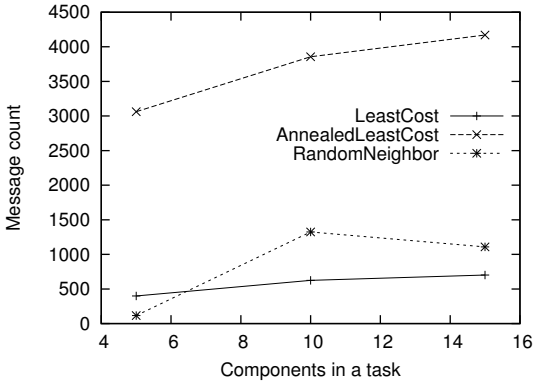

Figure 8.11: Total number of map-extension message exchanged, across different number of components in the stream processing tasks.

In terms of cost of computation of the heuristics, we can observe in Figures 8.10 and 8.11 that number of map-extension messages to complete mapping of a single composition is much higher in the AnnealedLeastCost heuristic than the other two heuristics. Moreover, the cost grows very quickly as the size of the network grows. Given the marginal benefits in terms of solution quality we get from the AnnealedLeastCost heuristic, compared to LeastCost, its cost of computation is unacceptably high. For this reason, we concluded that the LeastCost heuristic as the best among the three proposed heuristics. Another thing to observe is that the messaging cost of the heuristics does not vary that much with the length of the composition (Figure 8.11), but the cost grows with the size of the network.

This is because regardless of the number of components in the composition, the heuristics search the whole graph for feasible mappings from the source to the terminal node.

8.5 Summary

In this chapter we presented an in-depth analysis of the mapping problem that maps the processing and transmission requirements of a stream processing task to a network of servers. We have developed both centralized and distributed algorithms to compute the optimal mapping of computational capacity and network bandwidth requirement of a stream processing task on the resource network. Many high-throughput scientific research platforms need to support applications that resemble stream processing. Finding the exact solutions being an NP-complete problem, we presented several heuristics that cuts down complexity and derives near-optimal solutions. A comparative study of these heuristics is performed to find the best one that has all the desirable properties.

In this work, we developed algorithms to map stream processing tasks with path-topology only. Several interesting applications such as complex continual queries on data streams originating from multiple sites, resemble a tree topology. A possible extension of this work is to modify the algorithm such that it can map the tasks with different data flow topologies.

9

Dynamic Management of Bi-modal Network for Stream Processing

9.1 Overview

This chapter is the second one of the two chapters that explore the problems of bi-modal management of communication links in a distributed stream processing platform. In Chapter 8, we have investigated the problem of mapping the host and link resource requirements of stream processing tasks on a network computing platform from a theoretical perspective. Using techniques from Chapter 8, we can statically allocate the resources required for a stream processing task. However, to effectively manage concurrent resource allocation requests and the variability in the flow-rate of the public network links, we need a mechanism to adapt with the changing network conditions. In this chapter we present the architecture and functional details of the resource management system (RMS) that is capable of dynamic remapping in the bi-modal stream processing platform we introduced in Chapter 3.

The RMS is completely distributed, with an RMS agent on each of the server nodes. The agent on each node has three components – i) map manager ii) reservation manager iii) dynamic scheduler. The map manager implements a version of the distributed mapping

heuristics described in Chapter 8. The reservation manager participates in committing the resource allocation or rolling back the partial allocations in case some map is found infeasible while committing. Details of the mapping and reservation protocols are described in Section 9.2. The rate at which the data stream is processed and delivered to the target node does not remain constant even after acquiring the required amount of node and link resources successfully, because of the variability of the public network links. To obtain maximum possible compliance with the target rate specified in service agreements in presence of such variability, the dedicated links and the public links need to be dynamically re-allocated based on the needs of the tasks. The dynamic scheduling manager handles this periodic re-allocation of the different types of links. The detailed scheduling protocol is described in Section 9.3. We evaluated the described protocols on a simulation testbed implemented in Java and JiST [19]. The assumptions made in the simulation model and the test results are presented in Section 9.4. Results from this chapter are under review for publication in [15].

9.2 Mapping and Reservation Protocols

A user of the stream-processing platform injects her tasks onto one of the server nodes, which becomes the home node for the user. The user submits the specifications of a task to its home node. The specification contains the address of data stream source and the names of the series of service components that should process the data stream. By default the delivery point (destination) of the stream is the user's home node, but any other node can be specified as well. The specification also includes the required rate of data delivery, time window for monitoring the rate and pricing for each byte of data delivered.

After receiving the specification from user, the home node engages the map manager

component of its RMS agent to initiate the mapping of the specified requirements on the network. In the map manager, we implemented the LeastCostMap heuristic described in Section 8.3. In the LeastCostMap heuristic, each node expands the partial maps it has received from a neighboring node. Partial maps are grouped according to their *prefix-length*, which is the number of required service components already mapped in the partial map. In each prefix-length group, only the least cost partial map discovered so far is expanded.

As a cost metric, the map manager uses a function of the parameters – load balance across the servers, number of direct dedicated links used and number of public network links used in the map. When two maps of the same prefix-length is compared for cost the one with the lower load balance metric is preferred. If they are almost same (do not differ by more than 0.1 or 10%), then the one in which the number of hops (links connecting the processing components) assigned to dedicated links is higher, is preferred. If that is also same, the map in which less number of hops are assigned to public network link is preferred. The load balance metric is the average of the processing load on the servers associated with the map. The processing load is computed as the ratio of allocated processing resources to total processing capacity of the node, and is always a number between 0 and 1.

The distributed mapping algorithm possibly generates multiple feasible maps if such maps exist and these maps are stored at the source node of the data stream. In the source node, the maps are stored in a priority queue that prioritizes the complete maps according to the cost metric explained above. After waiting for a certain amount of time expecting for several feasible solution to arrive, the reservation manager in the source node initiates the reservation protocol for the best map received so far.

The reservation protocol basically sends the reservation probe along the actual network

path found in the map. The reservation manager in each server node along the path tries to allocate the node and link resources prescribed by the map. If the allocation is successful, it forwards reservation probe to the next server node in the map. In case the node finds that the required amount of resource is no longer available, it sends a reservation rollback message to the previous node in the path. Receiving a rollback message, the reservation manager releases the resources reserved for that particular task and forwards the rollback to the previous node in the map. Once the rollback message is received by the source node, it re-initiates the reservation on the next map in the priority queue. Once a successful reservation probe reaches the destination node, a confirmation is sent back to the source node along the path and the data streaming begins.

9.3 Dynamic Scheduling of Links

Once the reservation of the link and node resources along a feasible map is successfully committed, the distributed execution of the stream processing task begins. The server nodes involved in the task along the path of the stream may perform one of the two different roles. Some of the nodes execute one or more service components that process the data stream, providing the required CPU and memory resources. We denote such nodes as processing nodes. A server node may concurrently run service components for several stream processing tasks. Some nodes along the path of the data stream may not run any processing service, they only act as a stream forwarder or router. We denote them as *forwarding nodes*. As mentioned earlier, the RMS agent in each node has a scheduler component. At regular intervals, the scheduler evaluates the tasks being processed in the node and allocates necessary bandwidths to the outgoing flows of the competing tasks according to a priority scheme. The scheduling algorithm is presented in pseudocode form in Algorithm 6.

In the stream processing platform, the servers are dedicated machines, and hence, have stable and controlled processing capacities. Therefore, once we allocate the node resources for the processing steps in a task through the mapping and reservation protocols, there is no need to re-allocate them until the task execution completes. However, the links that carry the stream between two processing nodes can be of three different types – i) a direct dedicated link, ii) a multi-hop dedicated link through one or more forwarding nodes iii) an overlay link on the public network. A mapping of a task may contain any combination of these three types of links between the processing nodes. Among them, the direct dedicated links are the most preferred ones, because they provide controlled and stable data rate. A multi-hop dedicated link provides similar control and stability, but it costs more because every forwarding node will charge the sender node for their forwarding task. This in turn reduces the revenue earned by the processing node for its work. Therefore, a direct dedicated link is always preferred over a multi-hop dedicated link. The third possibility is having an overlay link through the public IP network. Because the sending node does not have any direct control over the packet routing in the public network, the flow rate is variable over such links. However, there is no additional per-byte cost for sending data through the overlay links. So, the nodes try to opportunistically use these links when dedicated links are overloaded or not available.

Periodically, the scheduler running in each node evaluates the fulfillment of processing rate requirement of each of the tasks being processed at the node, and re-allocates the available links of three types between this node and the node serving the next component, among the outgoing flows of the tasks, based on their outstanding needs. When a stream processing task is submitted, the price for processing each byte of the stream is given by the user. This price is apportioned to each byte of data processed by each processing step,

Algorithm 6 Link re-allocation algorithm

Invoked for each node u periodically
Group the flows that are being processed in u by their next component server v
for Each group v **do**
  Compute the priority of each flow competing for a (u,v) link
  priority = budget per byte of processed data * bandwidth required to comply with the target rate
  if any dedicated link (u,v) exists **then**
    Assign the dedicated link to top priority flows until all capacity is used
  end if
  Collect all the unassigned flows
end for
for All the remaining flows **do**
  if The budget permits k-hop (u,v) dedicated link, $k > 1$ **then**
    Launch a probe search and reserve multi-hop dedicated path for the flow with maximum k hops
    Assign public network bandwidth for the flow temporarily
  else
    Assign public network bandwidth for the flow
  end if
end for

based on the number of steps and the CPU requirement for each step. Accordingly, the node where the processing step is eventually mapped, earn revenue at this rate for carrying out the processing task. Naturally, the scheduler running on a particular node will try to give higher preference to the streaming tasks that are marked with higher price per unit of processing.

On the other hand, the servers get penalized on the revenue, if they do not deliver the processed stream at the agreed upon rate. Therefore each server tries to fulfill the rate requirements of each task as much as possible. The task that requires more bandwidth to comply with its target gets higher preference. Hence the scheduler computes the priority of each task being processed on the node as a product of the apportioned price and the data rate required in next scheduling epoch.

Every time the scheduler is run, it re-evaluates the priorities of the tasks processed by the node, group them by the node that processes the next step in the path and put them in a priority queue for each group. Now for each group, it allocates bandwidth from the direct dedicated link for the outgoing flows of the highest priority tasks, if such a link is available. The next prior tasks are assigned multi-hop dedicated links. The maximum possible hops in such multi-hop links are restricted by the apportioned price for that task, because there is additional cost of forwarding at each hop and the processing node would not like to spend for forwarding cost beyond the amount of revenue it earns. The flows of the remaining tasks from all the groups are allocated bandwidth from the public overlay links.

9.4 Simulation Study

We evaluated different aspects of the stream processing platform using a simulation testbed that implements the aforementioned mapping, reservation and scheduling protocols. We

developed the simulation testbed using the JiST discrete event simulation platform [19], which is an extension of the Java language for efficient coding and execution of general purpose discrete event simulations. In Section 9.4.1, we describe the assumptions made in the simulation model and explain the rationale behind them. Results of the simulation based experiments are discussed in Section 9.4.2.

9.4.1 Simulation Model

The stream processing platform consists of several interconnected servers. When a service component processes a data stream, it consumes different server resources such as CPU and memory. We assume that the RMS agent allocates server resources in terms of an aggregate unit that includes all necessary resources to process the data stream. Also, we assume that number of units of resources required by a service component to process a stream is proportional to the rate of processing. The resource consumption also depends on particular type of service component being executed.

The amount of available resource units vary from server to server. We assumed that there is a power law distribution of server capacities, because this follows the distribution of financial capacities of the organizations that install the servers. Also, each server may serve multiple service components. We assume that each service is equally available across the network, i.e. total number of nodes where a particular service is available is uniform for all types of services. This implies that the higher capacity servers would naturally serve more variety of services than lower capacity servers.

The servers are connected to the public network or Internet with certain uplink and downlink bandwidth. Since the stream processing involves symmetric data load, we assumed symmetric network links (i.e. equal uplink and downlink capacities). The capacity

varies across the servers and was randomly chosen between 1 Mbps and $2Mbps$ in our simulations, with uniform distribution. Since all the servers are connected to the public network, they are connected by all-to-all overlay links.

Because the routing and flow control inside the public Internet is not controllable by the server nodes, they do not have direct control over the stream flow rate. However the maximum data injection rate by a server is determined by its uplink bandwidth. Due to the variability of the end-to-end path capacity, the network may not always be able carry the data to the destination according to the injection rate. This temporal variability of the data flow through the overlay links on public network is modeled after the statistics recently presented by Wallerich and Feldmann [91]. According to their data, the logarithm of the ratio of the observed transient flow rate to the mean rate of the flow over long period has almost a Normal probability distribution. In our simulation testbed, all the flows on the public overlay links are perturbed every 10 milliseconds according to the model. The allocated bandwidth was assumed to be the mean rate and the standard deviation of the log-ratio is assumed to be 1. Therefore in 95% cases the observed bandwidth remains between one fourth ($2^{-2\sigma}$) and four time ($2^{2\sigma}$) of the assigned or mean bandwidth.

In addition to the connection through the public network, the server nodes develop a dedicated interconnection network among themselves, using fully controlled direct point to point connections. Naturally, the powerful servers will be more interested to interconnect in such way, so that they have more data streams to process and have better utilization of their server resources. An incremental network growth model with preferential connectivity towards the higher capacity nodes is natural for such scenario. In our simulation, we assume a network growth model similar to the one proposed by Barabasi et al [18]. We assume that the servers take the initiatives to establish dedicated links in descending order of their

capacity. When a server establishes a connection, it chooses the other end of the link according to a probability distribution biased towards higher server capacity. If there are total N nodes, and node i tries to install a dedicated link, probability of choosing node j as the other end of the link is $\frac{C_j}{\sum_{k=1}^{N} C_k}$, where C_k is the capacity of node k.

The available bandwidth of the dedicated links may be much higher than the uplink or downlink connections to the public network. We assumed that these dedicated links are symmetric, and their bandwidths were randomly assigned between 1 Mbps and 10 Mbps. The propagation delays ($\frac{1}{2}$RTT) for the direct dedicated links were assumed to be between 1 and 10 milliseconds. The propagation delay through the overlay links are much higher and assumed to be between 10 and 100 milliseconds.

Most of the simulation runs involved 100 server nodes. Total number of dedicated links interconnecting these servers was varied in several experiments. Unless otherwise mentioned, the default number of dedicated links was 99, the minimum number of links to bring all the servers into a connected network. A total of 25 different types of service components were assumed. Each of the stream processing tasks was assumed to be processed through 10 services. The type of service for each step was randomly chosen from the 25 different services. Each task is assumed to process on average 100MB of data from the data source. The target rate for delivery of the processed stream was $1Mbps$ on average. Note that from the data source to the delivery node, the stream may expand or shrink depending on the particular service component that processes the stream.

For each of the experiments, the results were averaged over 100 different randomly generated networks with the specified parameters. For each sample network, a synthetic workload trace containing 500 stream processing tasks were generated, filtering out the

obviously infeasible tasks, such as those having capacity requirement for the source or destination nodes exceeding the total capacities of the nodes, or, those requiring more bandwidth between two steps that is not available between any two servers serving those two steps. The task arrival process is assumed to be a Poisson process, with different arrival rates depending on the experiments. The default arrival rate was 60 tasks per hour, if not mentioned otherwise.

9.4.2 Results

We performed several sets of experiments to evaluate the benefits of using bi-modal network in the stream processing platform. In the experiments, we compare three possible settings – i) a network with the dedicated links only, ii) a network containing public links only and iii) a network that combines both types of links.

Our first argument in favor of bi-modal architecture is that such augmentation of dedicated and public resources allows high work throughput and high service quality with very low investment on dedicated resources. To examine this, we fed similar workload traces at same arrival rates to two system set-ups, one with only dedicate link based networks and the other using the combination of dedicated links and public network. In the first result, shown in Figure 9.1 acceptance ratio of the offered jobs is used as a relative measure of the overall capacity of the system. We observe that, for the same workload, if the platform uses dedicated links only, it needs more than 120 links to get 50% acceptance ratio, whereas the same acceptance ratio can be obtained with 50 dedicated links only, if the public network is utilized in conjunction. Similar evidence in Figure 9.2 shows that inclusion of the public network helps to achieve the same overall system throughput at much lower number of dedicated link installations.

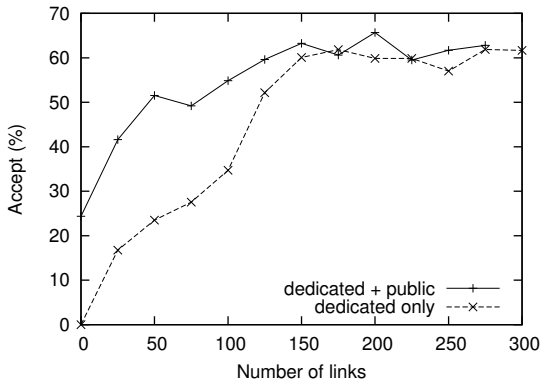

Figure 9.1: Proportion of offered jobs accepted (arrival rate = 60 tasks/hr)

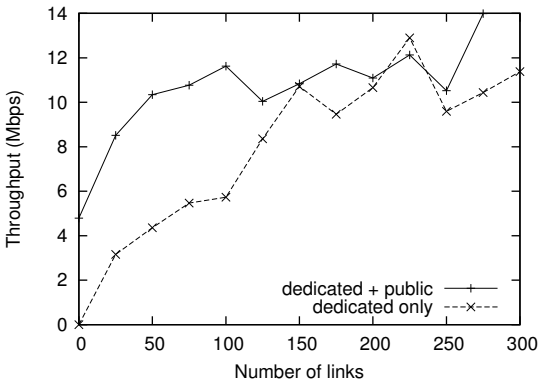

Figure 9.2: Overall system throughput in bi-modal and uni-modal systems (arrival rate = 60 tasks/hr)

The next argument is that utilization of the privately deployed expensive dedicated resources is increased, if inexpensive public resources are augmented to the system. In the stream processing platform, we assumed that all the servers are privately installed. Therefore higher utilization of these servers will imply higher return on investment for these resources. In Figure 9.3, utilization of the server resources is plotted for three different resource combinations for a range of task arrival rates. We observe that when a combination of dedicated links and public network is used, the utilization is much higher than the sum of utilizations of both the other cases using a single type of network links.

In Figures 9.5 and 9.6, we observe another evidence of higher return on investment on dedicated resource installations. In Figure 9.5, we observe that for the same number of dedicated link installations, the utilization of these links becomes consistently higher across a wide range of loading scenarios if the public network is used in combination. The lower utilization in case of a dedicated link only network results from the fact that the platform has to reject many task requests that would have been feasible by the augmentation of the public resources. Figure 9.6 shows the utilization of the dedicated links in uni-modal and

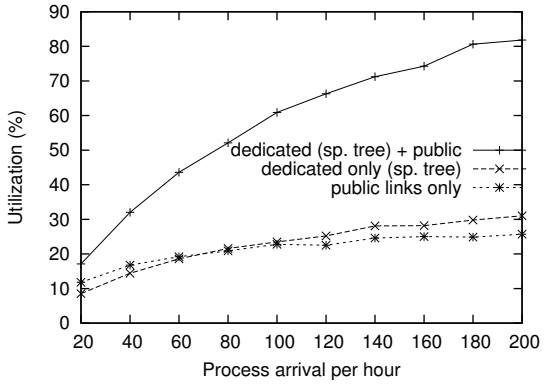

Figure 9.3: Server utilization in three different cases of resource combinations (with 99 dedicated links)

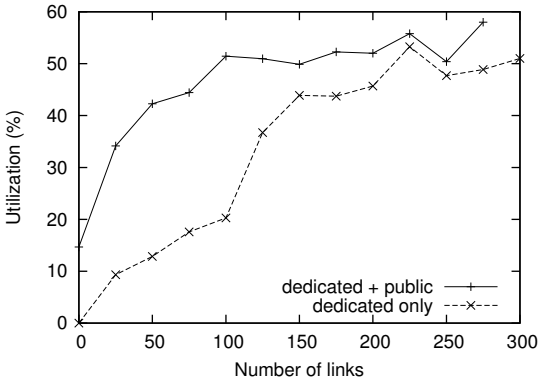

Figure 9.4: Server utilization in bi-modal and uni-modal systems with increasing installation of dedicated links (arrival rate = 60 tasks/hr)

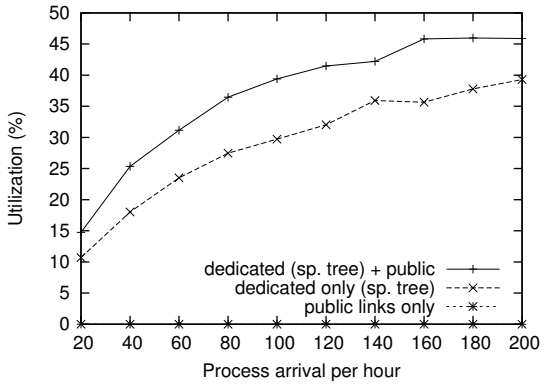

Figure 9.5: Utilization of dedicated links at three different cases of resource combinations (with 99 dedicated links)

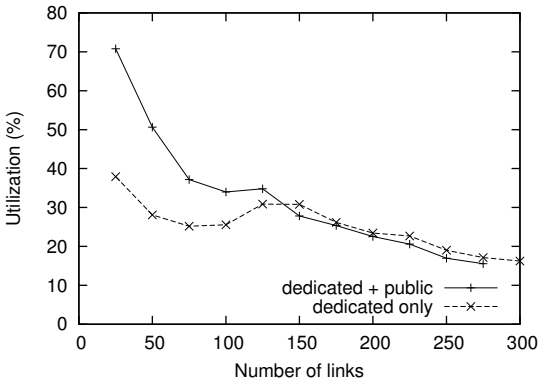

Figure 9.6: Utilization of the dedicated links in bi-modal and uni-modal systems, with increasing installation of dedicated links (arrival rate = 60 tasks/hr)

bi-modal systems, with increasing number of links installed. We observe that the difference in utilization between bi-modal and uni-modal system diminishes as the number of link installation increases. This is because when there is sufficient number of dedicated links to carry the required traffic of all the tasks, the public resources are not used at all, and the bi-modal system becomes equivalent to a dedicated link only system. In both cases, utilization of the links keeps decreasing when more and more links are added.

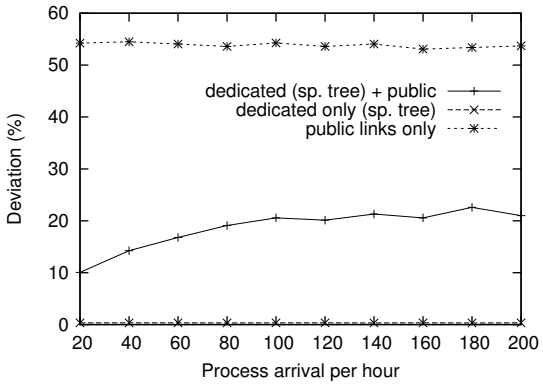

Figure 9.7: Deviation from the contracts of the accepted jobs for three different cases of resource combinations (with 99 dedicated links)

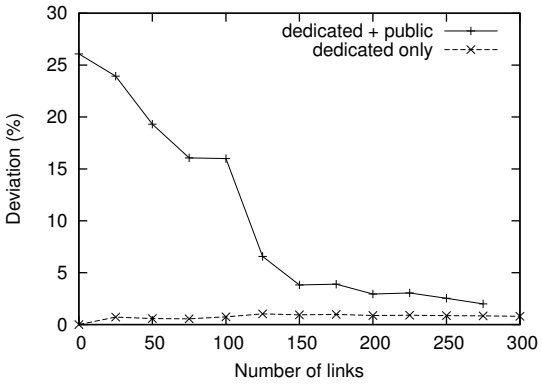

Figure 9.8: Deviation from the contracts of the accepted jobs with increasing installation of dedicated links (arrival rate = 60 tasks/hr)

The discussion above highlighted the benefits of using public network links towards improving the utilization of dedicated server and link resources. The augmentation can be viewed from another perspective, where dedicated links are added to an already existing public network. We investigate the benefits by measuring the improvement in terms of compliance with the data delivery rate requirements. We assumed that each stream processing task specifies a desired rate of data delivery. The platform accepts the task request only if it finds it feasible to process and deliver at the required rate. Because the public link capacities are inherently variable, it is impossible to guarantee the delivery rate solely

based on them. One solution is of course to install an all-private network with sufficient capacity where links can be fully reserved for particular tasks without any variability in the traffic flow, but that is an expensive solution. We attempted to measure how well a bi-modal system can achieve the rate guarantees, if not up to 100% compliance with the requirements. Each task request specifies a time window T that is used to monitor the delivery rate. We measured the deviation from the required rate as $\sum_{\text{over all windows}} \frac{B-\hat{B}}{B}$, where B is the desired rate and $\hat{B}$ is the observed rate of delivery. In Figure 9.7, we observe that use of dedicated links brings the percent deviation down to between 10% and 20% from above 50%. In this case the number of installed dedicated links was just enough to make a spanning tree of the nodes, i.e. $N-1$ links for N nodes. Note that deviation is counted on the accepted jobs only. So, even though for a dedicated link only network, the deviation is almost zero, we have seen that such network is unable to accept enough jobs to fully utilize the resources. In Figure 9.8, we observe that the deviation in the bi-modal system gets closer to zero as more and more dedicated links are added to the network. However, beyond certain number of links, (125 in this particular experiment), the improvement is very marginal.

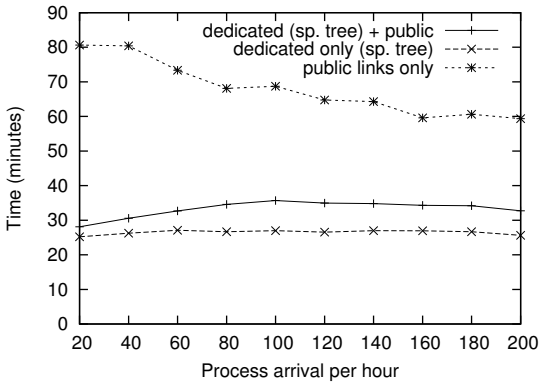

Figure 9.9: Mean task execution time in three different cases of resource combinations, showing the elongation of execution time when using public links (with 99 dedicated links)

When we use a combination of dedicated and public links, it is expected that the completion time of each task will be slightly elongated compared to a system with only dedicated links, due to the variability in the public network. Nevertheless, the combination marginalizes the elongation to a small value, compared to the case where only public network is available. In Figure 9.9, we observe a $10 - 20\%$ increase in the execution time in the bi-modal system, whereas execution time would be $200 - 300\%$ more in case of a public network only system.

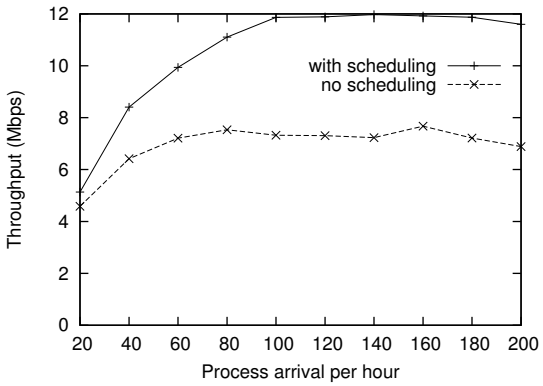

Figure 9.10: Overall throughput of the bi-modal system, with or without dynamic scheduling (with 99 dedicated links)

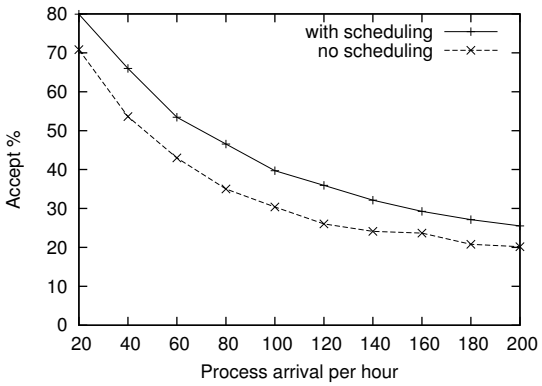

Figure 9.11: Dynamic scheduling increases the capacity of the system and hence the task acceptance ratio (with 99 dedicated links)

One important question in the resource management of the bi-modal stream processing platform is whether we really need the dynamic re-allocation of the network links, or we can simply keep the initial assignment of the nodes and links until the completion of the task. The main intuition behind introducing dynamic re-allocation is that the flows that goes through the public network suffer from the variability and lag from the target rate, whereas the flows that uses dedicated links all-through, do not lag from the target at all. Dynamic scheduling introduces fairness across all tasks. In addition, given a task executes

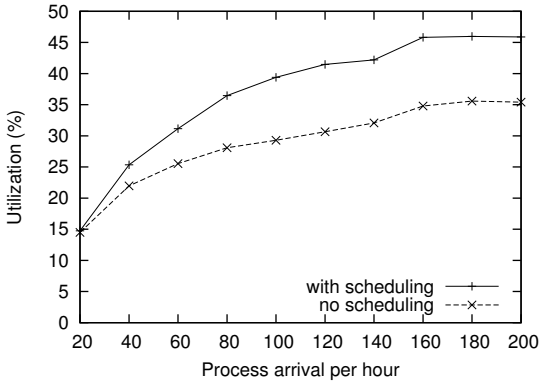

Figure 9.12: Dynamic scheduling increases the utilization of dedicated links (number of dedicated links = 99)

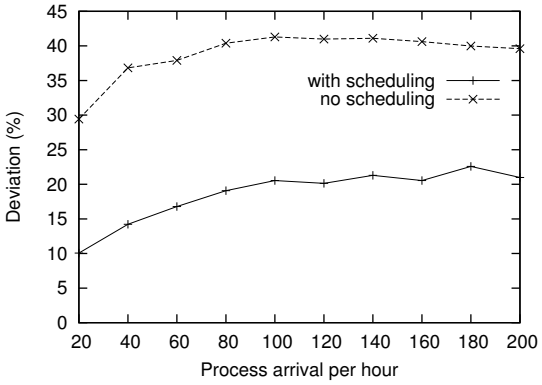

Figure 9.13: Dynamic scheduling reduces the deviation from the target delivery rate (number of dedicated links = 99)

for a long time, availability of the dedicated links change a lot within this period. So if link assignment is done dynamically, it is expected to improve the utilization of the resources and increase the overall capacity of the system.

We fed the same workload on two system set-ups, both containing combination of dedicated and public network links. In one we disabled the dynamic re-scheduling of the links to the competing tasks, and let the task complete with the initial assignment of links and nodes. In Figure 9.10 and 9.11, we observe that both task acceptance ratio and overall system throughput increases with dynamic scheduling, as an indication of higher utilization of the system resources. Figure 9.12 demonstrates that dynamic scheduling results in much higher utilization of the dedicated links. CPU utilization remains unchanged (not shown), because the dynamic re-allocation does not alter the node assignments. Another rationale behind re-allocations is to increase fairness and improve compliance with the target delivery rate. Figure 9.13 shows that irrespective of workload, dynamic scheduling decreases the deviation from the specified target, having the same number of dedicated links and same public network bandwidth.

9.5 Summary

In this chapter, we have discussed the necessary policies and protocols for allocating and re-allocating resources in a distributed stream processing platform. The resource management for each stream processing task is divided into two phases. The algorithms for the first phase, i.e. initial mapping of the node and link resource requirements of the stream processing task on the network of servers, was developed and evaluated in Chapter 8. To handle concurrent arrival of task request, a reservation and rollback mechanism have been implemented for the mapping phase, which is discussed in this chapter.

In the second phase, the bi-modal link resources have been periodically re-allocated. Detailed evaluation of the re-allocation policies have been presented in this chapter. Evidences have been presented from the simulation based experiments in support of our claim that bi-modal organization of communication links is beneficial from several perspectives. We have observed that the combination allows higher utilization of the dedicated links as well as nodes, and thus increases the total capacity of the platform to accept more tasks. Moreover, we have demonstrated that similar job-acceptance ratio and server utilization can be achieved at much lower investment on dedicated links, if a combination of links is used instead of using a dedicated-only network.

We also demonstrated that although the public network is inexpensive and available in vast quantity, if only the unreliable public network links are used, the deviation from the target rate of data delivery becomes unacceptable. However, if the public network links are used in conjunction with dedicated links, the deviation can be contained within an acceptable limit.

10

Conclusion

10.1 Summary of Contributions

Our major contribution in this thesis was the introduction of a novel bi-modal architecture for distributed service hosting platforms. The key aspect of these platforms is that the most critical resources for a certain type of application served by the platform are organized in two different classes. A small set of privately installed or leased and fully controlled dedicated resources are used in conjunction with widely and opportunistically available public resource. In one hand, such combination allows the platform to scale to support a large volume of workload with a very low deployment cost. On the other hand, the combination allows to ensure some service quality constraints such as response time or data delivery rate, which is not possible using only the unreliable public resources.

Such dedicated versus public resource combination can be attained for several different types of resources to create platforms that support different types of applications. The resource management policies depend on the characteristics of the resources as well as the desirable performance objectives of the applications. We have explored the bi-modal combination of two different types of resources, supporting two different application types. In the first case, we examined how to manage public computing resources in conjunction with

a dedicated privately installed compute-server cluster to serve high-throughput computing applications. In the second case, a distributed set of servers were connected using a combination of overlay links on public IP network and a small number of privately installed dedicated links, to support distributed multi-hop processing of continuous data streams. In Chapter 3, we characterized both the public computing resources and public communication links, and described the architectural details of the two hosting platforms that serves compute-intensive and data-intensive applications, respectively.

In Chapters 6 and 7, we explored different alternative organizations of the resource management system of the bi-modal platform for compute-intensive jobs, and developed scheduling algorithms for each cases. The public processing units are geographically dispersed and interconnected through communication networks. In Chapters 6 and 7, we assumed the availability of a well managed network and concentrated on management of the processing units. We devised a centralized resource management system that schedules the job request on these pools of processing units. Aggregation of state information is a hard problem in such geographically distributed deployment of resources. So, we explored two different scenarios, in one of which the resource manager performs the scheduling activities in a state oblivious manner and in the other, periodic status update is enabled. The trade-off between informed decision and cost of information aggregation is thoroughly investigated. Preemptive migration of jobs from one processing unit to another, although found to be a very desirable feature, potentially costs a huge amount of communication resources. So, presence and absence of migration is also evaluated in those two studies.

In Chapters 8 and 9, we have examined the resource management problem in a distributed stream processing platform, where the communication links are organized bi-modally. For the data-intensive applications like distributed stream processing, the communication links become the most critical resources. In the proposed platform, the server nodes that serve different stream processing services, are installed as dedicated servers. The communication network that interconnects them has two different types of links – some privately installed or leased dedicated links and some overlay paths through the public IP network.

The resource management in this platform is divided into two phases. In the first phase, a requirement specification of a composite stream processing task is mapped on the server nodes and communication links fulfilling the computing and transmission capacity requirements. In the later phase, the assignments of communication links are altered as necessary, due to the inherent variability of the public network links. Finding an optimal mapping of the requirement, subject to the given capacity constraints is a computationally expensive problem. We analyzed the problem in details in Chapter 8, outlined centralized and distributed algorithms to solve the problem and developed some heuristics to find out workable solutions in a cost-effective way.

In Chapter 9, we developed protocols to perform the mapping and allocation of resources in presence of concurrent task requests, as well as a periodic re-allocation mechanism for the link bandwidth to cope with the variability of data flow rate through the public links. We performed detailed simulation based evaluation of the scheduling schemes. The results support that bi-modal organization of communication links is beneficial from several perspectives. We have observed that the combination allows higher utilization of the

dedicated links as well as nodes, and thus increases the total capacity of the platform to accept more workload of tasks. Moreover, we have demonstrated that similar job-acceptance ratio and server utilization can be achieved at much lower investment on dedicated links, if a combination of links is used instead of using a dedicated-only network. We have also demonstrated that although inexpensive and available in vast quantity, if only the unreliable public network links are used, the deviation from the target rate of data delivery becomes unacceptable. Although the public network links cause major deviations from the target rates, if they are used in conjunction with dedicated links, the deviation can be contained within acceptable limits.

In summary, we can articulate the following contributions from this thesis –

- Introduced a bi-modal architecture for service hosting platforms.
- Explored combination of public and dedicated resources for two different types of resources and applications – computing resources for high-throughput computing applications and communication resources for data intensive stream processing applications.
- Explored how the usage of public computing resources can be leveraged to develop a commercially viable hosting platform for compute-intensive applications.
- Examined alternative resource management policies and evaluated the benefits of job-migration and status aggregation in the scheduling process.
- Developed scheduling heuristics and evaluated them through detailed simulation models of the platforms.
- Analyzed the problem of mapping resource requirements of a distributed stream processing task on a network of servers, subject to node and link capacity constraints

- Developed centralized and distributed algorithms to find optimal solution to the mapping problem and proposed several heuristics to minimize the run-time cost of these algorithms.
- Demonstrated the benefits of using a combination of public and dedicated networks links for higher utilization of server and link resources and higher quality assurance for stream processing tasks.
- Developed algorithm for dynamic re-allocation of network links to achieve the benefits of bi-modal network organization.

10.2 Future Extensions

In this section, we discuss some research problems for future investigation motivated by the results of this thesis.

10.2.1 Computing Platform

In Chapters 6 and 7, we have examined different resource management heuristics for a computing platform, assuming the existence of a single platform provider that installs a cluster of compute-servers on a single network location to augment with the available public resources. Although the public computers are available in a large scale across the Internet, there may be contention on them when multiple platform providers try to exploit the idle capacities. How the platform should react to such contention to manage its workload, is an important concern. A contention resolution protocol need to be developed so that each of the contending provider gets a fair share of the public resources.

In Chapter 7 we have observed that the communication overhead limits the size of the dedicated cluster installed in a single network location. So, a single platform provider may install multiple clusters of dedicated computers at different locations in the network, to distribute the workload and overcome the capacity limit. In that case, all the clusters will share the same set of public resources. The core problem here to develop a locality aware scheme to rout the jobs in proper cluster so that load is balanced among the clusters and communication overhead is minimized.

Centralized control of the resources, although a desirable feature for optimal allocation, may become a limiting factor for scalability and reliability of the system. An alternative approach would be to run a monitoring and controlling agent on each of the resources and allow the agents to collaborate in a decentralized way to execute the computing jobs. The question that needs investigation here is how to develop decentralized protocols to ensure the response time guarantees for the jobs and to maximize the utilization of the dedicated resources.

There is a need to secure isolation of the jobs launched by the hosting platform on the public resources, from the local processes running on those machines. We have proposed a virtual machine based approach to achieve such isolation. However, full-blown virtual machines are too heavyweight in terms of memory footprint, and create large communication overhead if a job need to be migrated. Using the recent advancement in the processor supports for virtualization, low overhead process wrappers may be developed that provides secure isolation as well as supports live migration.

10.2.2 Stream Processing Platform

In Chapter 8, we investigated the problem of mapping a stream processing task on a network of servers. We restricted our analysis to service compositions with path topologies only. In some applications the composition resembles more general topologies like tree or DAG. Although several solutions to the generalized DAG mapping problems have been proposed [92, 48], they do not consider the node capacity constraints for the mapping. With minor modifications, our proposed algorithm may be adapted for tree topologies. Further investigation is required to develop efficient heuristics for all possible topologies of service compositions.

We have evaluated a bi-modal architecture in Chapter 9, where two types of network links are utilized to improve server utilization and quality assurance for stream processing applications. However, the services that process the data streams are assumed to run only in dedicated servers. From the benefits of using public computing resources opportunistically, as we have observed in Chapters 6 and 7, it may be interesting to explore the use of public computers to run stream processing services. Such architecture will be very suitable for low cost deployment of dissemination platforms for multimedia streams, where streams need to decoded into different formats for different classes of users.

Bibliography

[1] "The Distributed Advanced School of Computing and Imaging (ASCI) Supercomputer 2 (DAS-2)," http://www.cs.vu.nl/das2/, Vrije Universiteit, Amsterdam.

[2] T. F. Abdelzaher, K. G. Shin, and N. Bhatti, "Performance Guarantees for Web Server End-systems: a Control-Theoretical Approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 1, pp. 80–96, Jan. 2002.

[3] R. J. Al-Ali, K. Amin, G. Laszewski, O. F. Rana, D. W. Walker, M. Hategan, and N. Zaluzec, "Analysis and Provision of QoS for Distributed Grid Applications," *Journal of Grid Computing*, vol. 2, no. 2, pp. 163–182, Jun. 2004.

[4] D. P. Anderson, "BOINC: A System for Public-Resource Computing and Storage," in *5th IEEE/ACM International Workshop on Grid Computing*, Nov. 2004.

[5] D. P. Anderson and G. Fedak, "The Computational and Storage Potential of Volunteer Computing," in *6th IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, May 2006.

[6] D. P. Anderson, E. Korpela, and R. Walton, "High-Performance Task Distribution for Volunteer Computing," in *First IEEE International Conference on e-Science and Grid Technologies*, Dec. 2005.

[7] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "SETI@home: an Experiment in Public-resource Computing," *Communications of the ACM*, vol. 45, no. 11, pp. 56–61, 2002.

[8] N. Andrade, L. Costa, G. Germoglio, and W. Cirne, "Peer-to-peer Grid Computing with the OurGrid Community," in *23rd Brazilian Symposium on Computer Networks - IV Special Tools Session*, May 2005.

[9] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D. Pazel, J. Pershing, and B. Rochwerger, "Oceano – SLA Based Management of a Computing Utility," in *7th IFIP/IEEE International Symposium on Integrated Network Management*, May 2001.

[10] M. Aron, P. Druschel, and W. Zwaenepoel, "Cluster Reserves: A Mechanism for Resource Management in Cluster-Based Network Servers," in *ACM SIGMETRICS*, Jun. 2000.

[11] S. Asaduzzaman and M. Maheswaran, "Leveraging Public Resource Pools to Improve the Service Compliances of Computing Utilities," in *Proceedings of IEEE/ACM International conference on high-performance computing (HiPC)*, Dec. 2004, pp. 242–251.

[12] ——, "Utilizing Unreliable Public Resources for Higher Profit and Better SLA Compliance in Computing Utilities," *Journal of Parallel and Distributed Computing*, vol. 66, no. 6, pp. 796–806, 2006.

[13] ——, "Strategies to Create Platforms for Differentiated Services from Dedicated and Opportunistic Resources," *Journal of Parallel and Distributed Computing*, vol. 67, no. 10, pp. 1119–1134, 2007.

[14] ——, "Towards a decentralized algorithm for mapping network and computational resources for distributed data-flow computations," in *21st IEEE International Symposium on High Performance Computing Systems and Applications*, May 2007, p. 30.

[15] ——, "Distributed Stream Processing on Network Computing Platforms with Dedicated and Opportunistic Resources," in *22nd IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Apr. 2008, submitted for review.

[16] F. Azzeddin, M. Maheswaran, and A. Mitra, "Applying a trust brokering system to resource matchmaking in public-resource grids," *Journal of Grid Computing*, vol. 4, no. 3, pp. 247–263, 2006.

[17] R. Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, B. Park, and H. Song, "Parsec: A Parallel Simulation Environment for Complex Systems," *Computer*, vol. 31, no. 10, pp. 77–85, Oct. 1998.

[18] A. Barabasi and R. Albert, "Emergence of Scaling in Random Networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[19] R. Barr, Z. J. Haas, and R. van Renesse, "JiST: An efficient approach to simulation using virtual machines," *Software: Practice and Experience*, vol. 35, no. 6, pp. 539–576, 2005.

[20] P. Bartley, M. Florian, and P. Robillard, "Scheduling with Earliest Start and Due Date Constraints," *Naval Research Logistics Qaurterly*, vol. 18, pp. 511–519, Dec. 1971.

[21] S. A. Baset and H. Schulzrinne, "An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol," Department of Computer Science, Columbia University, Tech. Rep. CUCS-039-04, Sep. 2004.

[22] R. Bellman, "On a Routing Problem," *Quarterly of Applied Mathematics*, vol. 16, no. 1, pp. 87–90, 1958.

[23] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services," IETF, Tech. Rep. RFC 2475, Dec. 1998.

[24] E. Bouillet, D. Mitra, and K. G. Ramakrishnan, "The Structure and Management of Service Level Agreements in Networks," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 4, pp. 691–699, May 2002.

[25] R. Braden, D. Clark, and S. Shenker, "Integrated Services in the Internet Architecture: an Overview," IETF, Tech. Rep. RFC 1633, Jun. 1994.

[26] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification," IETF, Tech. Rep. RFC 2205, Sep. 1997.

[27] R. Brayanrd, D. Kosic, A. Rodriguez, J. Chase, and A. Vahdat, "Opus: An overlay peer utility service," in *15th Int'l Conf. Open Architectures and Network Programming (OPENARCH)*, June 2002.

[28] J. Brevik, D. Nurmi, and R. Wolski, "Automatic Methods for Predicting Machine Availability in Desktop Grid and Peer-to-peer Systems," in *IEEE International Symposium on Cluster Computing and the Grid (CCGrid'04)*, Apr. 2004.

[29] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger, "Economic models for resource management and scheduling in grid computing," *Concurrency and Computation: Practice and Experience*, vol. 14, no. 13–15, pp. 1507–1542, 2003.

[30] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, "Managing Energy and Server Resources in Hosting Centers," in *18th ACM Symposium on Operating Systems Principles*, Oct. 2001.

[31] S. Chen and K. Nahrstedt, "On finding multi-constrained paths," in *Proc. IEEE ICC*, Jun. 1998, pp. 874–879.

[32] B. Chun and A. Vahdat, "Workload and Failure Characterization on a Large-Scale Federated Testbed," Intel Research Berkeley, Tech. Rep. IRB-TR-03-040, Nov. 2003.

[33] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live Migration of Virtual Machines," in *2nd Symposium on Networked Systems Design and Implementation (NSDI'05)*, May 2005.

[34] E. Crawley, R. Nair, B. Jayagopalan, and H. Sandick, "A Framework for QoS-based Routing in the Internet," IETF, Tech. Rep. RFC 2386, Aug. 1998.

[35] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke, "SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems," *Lecture Notes in Computer Science*, vol. 2537, pp. 153–183, 2002.

[36] S. Das, S. Tewari, and L. Kleinrock, "The Case for Servers in a Peer-to-Peer World," in *Proceedings of IEEE International Conference on Communications (ICC '06)*, Jun. 2006, pp. 331–336.

[37] M. L. Dertouzos and A. K. Mok, "Multiprocessor On-line Scheduling of Hard-Real-Time Tasks," *IEEE Transactions on Software Engineering*, vol. 15, no. 12, pp. 1497–1506, 1989.

[38] Y. Drougas and V. Kalogeraki, "Distributed, reliable restoration techniques using wireless sensor devices," in *21th International Parallel and Distributed Processing Symposium (IPDPS)*, Mar. 2007, pp. 1–10.

[39] A. Erlang, "Solution of some Problems in the Theory of Probabilities of Significance in Automatic Telephone Exchanges," *The Post Office Electrical Engineer's Journal*, vol. 10, pp. 189–197, 1918.

[40] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*. San Fransisco, CA: Morgan Kaufmann, 1999.

[41] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The physiology of the grid: An open grid services architecture for distributed systems integration," Open Grid Service Infrastructure WG, Global Grid Forum, Tech. Rep., June 2002.

[42] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International J. Supercomputer Applications*, vol. 15, no. 3, 2001.

[43] A. Fox, S. D. Gribble, Y. Hcawathe, E. Brewer, and P. Gauthier, "Cluster-based scalable network services," in *Proceedings of the sixteenth ACM symposium on Operating systems principles (SOSP)*, Oct. 1997, pp. 78–91.

[44] M. Garey and D. Johnson, "Two-processor Scheduling with Start-times and Deadlines," *SIAM Journal on Computing*, vol. 6, pp. 416–426, 1977.

[45] ——, *Computers and Intractability: A Guide to the theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

[46] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W H Freeman Co., NY, USA, 1979.

[47] X. Gu, K. Nahrstedt, R. N. Chang, and C. Ward, "Qos-assured service composition in managed service overlay networks," in *23rd International Conference on Distributed Computing Systems*, May 2003, pp. 194–203.

[48] X. Gu and K. Nahrstedt, "Distributed multimedia service composition with statistical QoS assurances," *IEEE Trans. Multimedia*, vol. 8, no. 1, pp. 141–151, 2006.

[49] J. G. Hansen and E. Jul, "Self-migration of Operating Systems," in *2004 ACM SIGOPS European Workshop*, Sep. 2004, pp. 241–299.

[50] D. Jackson, Q. Snell, and M. Clement, "Core Algorithms of the Maui Scheduler," *Lecture Notes in Computer Science*, vol. 2221, p. 87, Jan. 2001.

[51] X. Jiang and D. Xu, "SODA: a Service-on-Demand Architecture for Application Service Hosting Utility Platforms," in *Proceedings of 12th IEEE International Symposium on High Performance Distributed Computing*, Jun. 2003, pp. 174–183.

[52] C. Kenyon and G. Cheliotis, "Creating Services with Hard Guarantees from Cycle Harvesting Resources," in *3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'03)*, May 2003.

[53] L. Kleinrock and W. Korfhage, "Collecting Unused Processing Capacity: an Analysis of Transient Distributed Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 5, pp. 535–546, 1993.

[54] E. Kotsovinos, "Global public computing," Ph.D. dissertation, The Computer Laboratory, University of Cambridge, Jan. 2005.

[55] K. Krauter, R. Buyya, and M. Maheswaran, "A taxonomy and survey of grid resource management systems for distributed computing," *Software: Practice and Experience*, vol. 32, no. 2, pp. 135–164, 2001.

[56] V. Kumar, B. F. Cooper, Z. Cai, G. Eisenhauer, and K. Schwan, "Resource aware distributed stream management using dynamic overlays," in *Proc. 25th IEEE ICDCS*, Jun. 2005, pp. 783–792.

[57] M. Larabel, "Linux KVM Virtualization Performance," http://www.phoronix.com, Jan. 2007.

[58] E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys, *Handbooks in Operations Research and Management Science*. Elsevier Science Publishers, 1993, vol. 4, ch. 9, pp. 445–522.

[59] F. Leung, G. Neiger, D. Rodgers, A. Santoni, and R. Uhlig, "Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization," *Intel Technology Journal*, vol. 10, no. 3, 2006.

[60] H. Li, D. Groep, and L. Walters, "Workload Characteristics of a Multi-cluster Supercomputer," *Lectur Notes in Computer Science: Job Scheduling Strategies for Parallel Processing(JSSPP'04)*, vol. 3277, 2005.

[61] J. Liang and K. Nahrstedt, "Service composition for generic service graphs," *Multimedia Systems*, vol. 11, no. 6, pp. 568–581, 2006.

[62] D. A. Lifka, "The ANL/IBM SP Scheduling System," in *Workshop on Job Scheduling Strategies for Parallel Processing(IPPS '95)*. Springer-Verlag, 1995, pp. 295–303.

[63] V. Loa, D. Zhou, D. Zappala, Y. Liu, and S. Zhao, "Cluster Computingon the Fly: P2P Scheduling of Idle Cycles in the Internet," in *3rd International Workshop on Peer-to-Peer Systems (IPTPS'04)*, Feb. 2004.

[64] U. Lublin and D. G. Feitelson, "The Workload on Parallel Supercomputers: Modeling the Characteristics of Rigid Jobs," School of Computer Science and Engineering, The Hebrew University of Jerusalem, Tech. Rep. 2001-12, Oct. 2001.

[65] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems," in *8th Heterogeneous Computing Workshop*, Apr. 1999, pp. 30–46.

[66] M. Maheswaran, B. Maniymaran, S. Asaduzzaman, and A. Mitra, "Towards a Quality of Service Aware Public Computing Utility," in *3rd IEEE Symposium on Network Computing and Applications: Adaptive Grid Computing Workshop*, Aug. 2004, pp. 376–379.

[67] B. Maniymaran and M. Maheswaran, "Bandwidth landmarking: A scalable bandwidth prediction mechanism for distributed systems," in *Proceedings of IEEE GlobeCom 2007*, Nov. 2007, to appear.

[68] B. Maniymaran, M. Maheswaran, and Y. Gao, "Benefits of clustering in landmark-aided positioning algorithms," in *21st IEEE International Symposium on High Performance Computing Systems and Applications*, May 2007, p. 29.

[69] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: an approach to universal topology generation," in *Proc. 9th Intl. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, Aug. 2001, pp. 346–353.

[70] J. Meehean and M. Livny, "A Service Migration Case Study: Migrating the Condor Schedd," in *Midwest Instruction and Computing Symposium*, Apr. 2005.

[71] D. S. Milojicic, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou, "Process Migration," *ACM Computing Surveys*, vol. 13, no. 3, pp. 241–299, Sep. 2000.

[72] A. Mitra, R. Udupa, and M. Maheswaran, "A Secured Hierarchical Trust Management Framework for Public Computing Utilities," in *15th Annual International Conference in the IBM Centers for Advanced Studies (CASCON)*, Oct. 2005.

[73] M. W. Mutka and M. Livny, "The Available Capacity of a Publicly Owned Workstation Environment," *Performance Evaluation*, vol. 12, pp. 269–284, 1991.

[74] J. Nabrzyski, J. Schopf, and J. Weglarz, *Grid Resource Management: State of the Art and Future Trends*. Springer, 2004.

[75] K. Nahrstedt, "QoS-aware Resource Management for Distributed Multimedia Applications," *Journal of High Speed Networks*, vol. 7, no. 3-4, pp. 229–257, 1998.

[76] P. R. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. I. Seltzer, "Network-Aware Operator Placement for Stream-Processing Systems," in *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006*, Apr. 2006, p. 49.

[77] J. A. Pouwelse, P. Garbacki, D. H. J. Epema, and H. J. Sips, "The Bittorrent P2P File-sharing System: Measurements and Analysis," in *4th Int'l Workshop on Peer-to-Peer Systems (IPTPS)*, Feb. 2005, pp. 205–216.

[78] R. Raman, M. Livny, and M. Solomon, "Policy Driven Heterogeneous Resource Co-Allocation with Gangmatching," in *12th IEEE International Symposium on High-Performance Distributed Computing*, Jun. 2003.

[79] S. Ranjan, J. Rolia, and E. Knightly, "QoS Driven Server Migraion for Internet Data Centers," in *IWQoS 2002*, May 2002.

[80] M. Ripeanu, "Peer-to-Peer Architecture Case Study: Gnutella Network," in *1st International Conference on Peer-to-Peer Computing (P2P01)*, Aug. 2001.

[81] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol Label Switching Architecture," Internet draft, Tech. Rep., Mar. 1998.

[82] S. M. Ross, *Introduction to Probability Models*, 5th ed. Academic Press Inc., 1993.

[83] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, November 2001, pp. 329–350.

[84] S. Saroiu, K. P. Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy, "An Analysis of Internet Content Delivery Systems," in *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, Dec. 2002, pp. 315–328.

[85] S. Seshadri, V. Kumar, B. F. Cooper, and L. Liu, "Optimizing Multiple Distributed Stream Queries Using Hierarchical Network Partitions," in *Proceedings of 21th International Parallel and Distributed Processing Symposium (IPDPS 2007)*, Mar. 2007, pp. 1–10.

[86] J. Sgall, "On-Line Scheduling – a Survey," in *Online Algorithms – The State of the Art*. Springer Verlag, 1997, pp. 196–231.

[87] J. Smed, T. Kaukoranta, and H. Hakonen, "A Review on Networking and Multiplayer Computer Games," Turku Centre for Computer Science, Turku, Finland, Tech. Rep. TUCS-454, Apr. 2002.

[88] M. Song and S. Sahni, "Approximation algorithms for multiconstrained quality-of-service routing," *IEEE Trans. Computers*, vol. 55, no. 5, pp. 603–617, 2006.

[89] D. Thain, T. Tannenbaum, and M. Livny, "Distributed Computing in Practice: The Condor Experience," *Concurrency and Computation: Practice and Experience*, vol. 17, no. 2-4, pp. 323–356, 2005.

[90] B. Uragaonkar, P. Shenoy, and T. Roscoe, "Resource Overbooking and Application Profiling in Shared Hosting Platforms," in *Proceedings of the 5th symposium on Operating systems design and implementation*, Dec. 2002, pp. 239–254.

[91] J. Wallerich and A. Feldmann, "Capturing the variability of internet flows across time," in *25th IEEE International Conference on Computer Communications (INFOCOM-2006)*, Apr. 2006.

[92] M. Wang, B. Li, and Z. Li, "sFlow: Towards resource-efficient and agile service federation in service overlay networks," in *Proc. 24th IEEE ICDCS*, Mar. 2004, pp. 628–635.

[93] Z. Wang and J. Crowcroft, "Quality of service routing for supporting multimedia applications," *IEEE J. Selected Areas in Communications*, vol. 14, no. 7, pp. 1228–1234, 1996.

[94] R. Wolski, D. Nurmi, J. Brevik, H. Casanova, and A. Chien, "Models and Modeling Infrastructures for Global Computational Platforms," in *19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, Apr. 2005.

[95] X. Xiao and L. M. Ni, "Internet QoS: a big picture," *IEEE Network*, vol. 13, no. 2, pp. 8–18, 1999.

[96] D. Xuy, M. Hefeeda, S. Hambrusch, and B. Bhargava, "On Peer-to-Peer Media Streaming," in *Proceedings of 22nd International Conference on Distributed Computing Systems (ICDCS)*, Jul. 2002, pp. 363–371.

[97] D. Zhou and V. Lo, "Wave Scheduling: Scheduling for Faster Turnaround Time in Peer-to-peer Cycle Sharing Systems," in *Workshops on Job Scheduling Strategies for Parallel Processing (JSSPP'05)*, Jun. 2005.