Divisible load scheduling algorithm in a virtual distributed computing system

Huang, Hao

2016

Huang, H. (2016). Divisible load scheduling algorithm in a virtual distributed computing system. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/68812>

<https://doi.org/10.32657/10356/68812>

Divisible load scheduling algorithm in a virtual distributed computing system

Huang Hao

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Doctor of Philosophy

2015

My prayers to God almighty.

Dedicated to my wife and my parents, who have been there with me throughout.

Acknowledgement

First, I would like to thank my supervisor, Dr. Suresh Sundaram. Words will not be enough to express my gratitude toward him for his constant support, and push at times. This thesis would not have even conceptualized without his help.

I would also like to thank all my family and friends for providing me with a comfortable environment to work. Their support has helped a lot during my research. I would like to thank Badrinarayanan Rangarajan, Dr. Savitha Ramamsamy, Venugopalan Tharuvai Krishnaswamy and Dr. Kartick Subramanaian for helping me revise and discuss the structure of the thesis.

Finally, I would like to thank the School of Computer Engineering at Nanyang Technological University for providing me a supportive environment to conduct my research. I would like to also thank HP Singapore for providing me the financial support necessary for my study and my family.

Contents

Acknowledgements	i
Abstract	viii
List of Figures	xi
List of Tables	xiv
List of Symbols	xvii
List of Abbreviations	xix
1 Introduction	1
1.1 Introduction	1
1.2 General background	1
1.2.1 Cluster computing	2
1.2.2 Grid computing	2
1.2.3 Cloud computing	3
1.3 Motivations	8
1.4 Objectives	11
1.5 Contributions	12
1.6 Thesis organization	14
2 Literature review	17
2.1 Introduction	17
2.2 Divisible load theory review	17
2.2.1 Network topologies	19
2.2.2 Constraints	24
2.2.3 Multi-installment	29
2.2.4 Nonlinear loads	30

2.3	Applications	31
2.3.1	Data processing applications	31
2.3.2	Video processing applications	32
2.3.3	Image processing applications	32
2.3.4	Signal processing applications	33
2.3.5	Scientific applications	33
2.3.6	Real-time applications	34
2.4	Scheduling in cloud environments	34
2.4.1	Workflow scheduling in clouds	36
2.4.2	Rescheduling in clouds	36
2.5	Optimization techniques for scheduling problems	38
2.6	Discussion	42
2.7	Summary	43

3 Scheduling in virtual distributed computing systems with multiple data banks **45**

3.1	Introduction	45
3.2	Mathematical modeling of virtual distributed computing system	46
3.3	Definitions	48
3.4	Problem formulation	49
3.4.1	Constraints imposed by total processing time (T_0):	50
3.4.2	Constraints imposed by release time (R_j):	51
3.4.3	Constraints imposed by continuity	52
3.4.4	Constraints imposed by total load	53
3.4.5	Linear programming formulation	53
3.5	Numerical examples	54
3.5.1	Numerical example 1	54
3.5.2	Numerical example 2	56
3.5.3	Numerical example 3	57
3.5.4	Numerical example 4	59
3.6	Performance evaluation and discussion	59
3.7	Satellite image processing application	65
3.8	Summary	68

4	Dynamic resource handling in virtual distributed computing systems	69
4.1	Introduction	69
4.2	Problem formulation	69
4.3	Numerical examples	78
4.3.1	Numerical example 1	80
4.3.2	Numerical example 2	83
4.3.3	Numerical example 3	86
4.3.4	Numerical example 4	90
4.3.5	Numerical example 5	91
4.4	Summary	94
5	Divisible load scheduling for multiple rounds and multiple data banks	97
5.1	Introduction	97
5.2	Problem formulation	98
5.3	Hybrid genetic algorithm for multi-installment problem	105
5.3.1	Multi-installment hybrid genetic algorithm	107
5.4	Simulation results	113
5.5	Hybrid genetic algorithm for VDCS with multiple data banks problem and numerical example	117
5.6	Summary	119
6	Conclusion and future directions	121
6.1	Thesis Summary	121
6.1.1	DLT-based load scheduling algorithm	121
6.1.2	DLT-based load dynamic rescheduling algorithm	123
6.1.3	HGA-based load scheduling algorithm for multiple installments . .	123
6.1.4	HGA-based load scheduling algorithm for multiple data banks . .	124
6.1.5	Experiment using satellite image processing	124
6.2	Future directions	125
6.2.1	Implementation in future commercial VDCS with multiple data banks	125

6.2.2	Multi-installment in VDCS with multiple data banks and priority	
	task queues	125
	List of Author's Publications	128
	Bibliography	130

Abstract

Divisible Load Theory (DLT) was introduced to resolve scheduling problems in a Distributed Computing System (DCS). A divisible load (job) is one that is arbitrarily divided into a number of fractions among the processors and links in a system, and each load fraction is processed independently. The main objective of DLT is to optimize the scheduling process with a minimum processing time for the entire load. Recent advancements in information technology have lead to a Virtual Distributed Computing System (VDCS) with dynamic resources and multiple data banks with a loosely connected structure. This thesis addresses the problem of load scheduling in a VDCS with dynamic resources and multiple data banks under a DLT framework.

We first propose a DLT solution to solve the problem of optimal load scheduling with fixed resources. The optimal load scheduling with fixed resources is formulated as a linear programming problem with total processing time, release time, continuity and total load constraint equations. The proposed DLT solution for load scheduling in VDCS with fixed resources is studied using four numerical examples. Next, we analyze the effect of processing the load in heterogeneous and homogeneous environments. Finally, we present an experimental study by solving a satellite image processing problem in a compute cloud environment. From the experimental study, we can see that the result is close to the analytical processing time obtained using the proposed load scheduling algorithm.

Based on the proposed DLT solution for a VDCS with fixed resources, we extend the framework to address the dynamic resource allocation problem. The release time of the worker role and the modified load distribution sequence are additional constraints in a dynamic resource handling environment. Therefore, we formulate the problem of dynamic resource handling as a linear programming problem with these additional constraints.

We propose a rescheduling strategy for worker role release time and modified load distribution sequence. Using five numerical examples, we show that the DLT-based scheduling algorithm with the new rescheduling strategy can accomplish the load processing tasks in a dynamic resource environment.

Another important problem with load scheduling is the idle time of resources during each process, which is usually reduced through multiple installments. In this thesis, we propose a method to solve the load scheduling problem with multiple installments using genetic algorithms. First, we present a hybrid genetic algorithm approach for load scheduling in VDCS with a single data bank. Next, we extend it to multiple data banks. The hybrid genetic algorithm integrates the equality constraint during optimization and hence provides better convergence compared to other search-based algorithms. A mean and standard deviation reduction of greater than half is observed using the hybrid genetic algorithm compared with a general real-coded genetic algorithm and a particle swarm optimization algorithm.

In the future, we plan to implement the proposed DLT-based scheduling algorithm as a load manager in a real VDCS with multiple data banks. Further, the DLT-based scheduling algorithm will be extended from the single-installment case to a multi-installment case in VDCS with multiple data banks. Moreover, the task queue will be explored by considering the task with different levels of priorities.

List of Figures

1.1	Overview of cluster computing, reproduced from [1].	2
1.2	Overview of grid computing, reproduced from [77].	3
1.3	Overview of cloud computing logical diagram.	4
1.4	Cloud computing service level architecture, reproduced from [2].	5
1.5	Overview of VDCS architecture.	9
2.1	A model of a cluster computing system, reproduced from [77].	18
2.2	A model of a grid computing system	19
2.3	A model of a cloud computing system, reproduced from [77].	20
2.4	Bus network architecture with m processors attached.	21
2.5	Single-level tree network architecture.	22
2.6	Multi-level tree network architecture.	23
2.7	4-dimensional hypercube network architecture.	25
3.1	The virtual distributed computing system environment	47
3.2	The virtual distributed computing system environment with M worker roles and N data banks	49
3.3	Timing diagram describing the load scheduling process in the virtual dis- tributed computing system environment	50
3.4	Timing diagram of worker role 1 processing by accessing every adjacent data bank pair.	51
3.5	Timing diagram of each adjacent worker role pair processing by accessing each adjacent data bank pair.	53
3.6	Timing diagram for numerical example 1	56
3.7	Timing diagram in 4-worker-role, 2-data-bank case for numerical example 4	60

3.8	Timing diagram in 5-worker-role, 2-data-bank case for numerical example 4	61
3.9	Processing time versus number of heterogeneous worker roles in the virtual distributed computing system environment	61
3.10	Processing time versus number of homogeneous worker roles in the virtual distributed computing system environment	62
3.11	Processing time versus number of homogeneous worker roles for different inverse transmitting speed parameters.	63
3.12	Processing time versus number of homogeneous worker roles for different processing load sizes.	64
3.13	Processing time versus number of homogeneous worker roles in the homogeneous virtual distributed computing system environment for R_j set . .	65
3.14	Processing time versus number of homogeneous worker roles in the virtual distributed computing system environment	66
3.15	Analytical and experimental processing time for satellite image processing problem in the virtual distributed computing environment.	67
4.1	Load rescheduling for the virtual distributed computing system.	73
4.2	Timing diagram describing the load rescheduling process in the virtual distributed computing system environment.	74
4.3	Rescheduling continuity constraints of worker roles with loads.	76
4.4	Timing diagram for original numerical example	81
4.5	Timing diagram for numerical example 1	84
4.6	Timing diagram for numerical example 2	87
4.7	Timing diagram for numerical example 3	89
4.8	Timing diagram for numerical example 4	92
4.9	Timing diagram for numerical example 5.	95
5.1	Two worker roles in the two-installment case with start-up delays	100
5.2	Timing diagram for general m -worker-role, n -installment system	103
5.3	Flow chart for HGA	114
5.4	Timing diagram for numerical example 3	119
5.5	Numerical example 3 HGA converge diagram	119

6.1	Multi-installment load scheduling in a virtual distributed compute system	127
-----	---	-----

List of Tables

2.1	Distributed Computing System Optimization Techniques	38
3.1	Notations and Descriptions	48
3.2	Load Fractions and Processing Time after Scheduling for Numerical Example 1	55
3.3	Load Fractions and Processing Time after Scheduling for Numerical Example 2	57
3.4	Processing Time after Scheduling in the 6 Worker Role Sequence Cases for Numerical Example 3	58
3.5	Processing Time after Scheduling in 2 Data Bank Sequence Cases for Numerical Example 3	58
3.6	Processing Time after Scheduling in 4- and 5-Worker-Role Cases for Numerical Example 4	59
3.7	Virtual Distributed Computing System Simulation Parameters	62
4.1	Rescheduling Notations and Descriptions	72
4.2	The VDCS Original Numerical Example Parameters	79
4.3	Load Fractions and Processing Time after Scheduling for Original Numerical Example	80
4.4	Load Fractions and Processing Time after Rescheduling for Numerical Example 1	83
4.5	Load Fractions and Processing Time after Rescheduling for Numerical Example 2	86
4.6	Load Fractions and Processing Time after Rescheduling for Numerical Example 3	88

4.7	Load Fractions and Processing Times after Rescheduling for Numerical Example 4	93
4.8	Load Fractions and Processing Times after Rescheduling for Numerical Example 5	94
5.1	Motivating Example 1: Distribution Order $\{w_1, w_2\}$	101
5.2	Motivation Example 1: Distribution Order $\{w_2, w_1\}$	101
5.3	Motivating Example 2: Distribution Order $\{w_1, w_2\}$	102
5.4	Processing Time for Numerical Example 1	115
5.5	Numerical Example 1: Optimal Distribution Order	116
5.6	Comparison of Convergence of Different Population-Based Algorithms . .	117
5.7	Processing Time for Numerical Example 2	117
5.8	Processing Time for Numerical Example 3	118

List of Symbols

β_{ij}	The fraction of load retrieved from the data banks (D_j) by a worker role (W_i)
$\beta_{i,j}$	Load fraction assigned to worker role w_i in round j
δ	Number of newly available or failure number of worker roles for this task after T_R
λ_1, λ_2 ...	HGA fitness function constants for VDCS with multiple data banks and single installment
μ_i	The total fraction of processing load assigned to a worker role (W_i)
θ	For the rescheduling round, the number of the first worker role with full load (μ_i)
g_i	Communicate latency (start-up delay)
k	For the rescheduling round, the number of the first worker role with load
l_i	The link connecting the worker role with data w_0 and the i^{th} worker role w_i
w_0	Worker role with data banks
w_i	The i^{th} worker role
A_i	Inverse processing speed to worker role W_i
A'_i	Inverse processing speed to worker role W'_i
C_1	The first solutions selected for crossover operations
C_2	The second solutions selected for crossover operations
D_j	The j^{th} data bank in the virtual distributed computing system
D'_j ..	The j^{th} data bank in the virtual distributed computing system in the rescheduling round
G_i ..	Inverse transmitting speed to link l_i between the worker role with data w_0 and the i^{th} worker role w_i
G_j	Inverse transmitting speed to data bank D_j through a wide area network
G'_j	Inverse transmitting speed to data bank D'_j through a wide area network
H_1, H_2	Two new solutions

J	Size of the processing load
J'	The remaining load after the rescheduling start time (T_{RS})
K_1, K_2	The total load between crossover sites
$K_{i,1}$	The time at which the processing load assigned to worker role w_i in the second round is received at the worker role w_i
M	Number of worker roles present for load scheduling
N	Number of data banks present for load scheduling
N'	Rescheduling round number of data banks
R_j	Release time of data banks D_j
R'_j	Rescheduling data bank release time related to the rescheduling start time
R'_{W_i}	Release time of the i^{th} worker role W'_i in rescheduling round
T	Processing time for total load J
T_0	Total load processing time
T'_0	Rescheduling end time of processing
T_i	Data processing time for the i^{th} worker role (W_i)
T'_i	Data processing time for the i^{th} worker role (W'_i) in rescheduling round
T_R	The time new resource claimed available for load rescheduling
T_{RS}	Rescheduling start time
W_i	The i^{th} worker role in the virtual distributed computing system
W'_i	The i^{th} worker role in the virtual distributed computing system in rescheduling round

List of Abbreviations

AGA	Adaptive Genetic Algorithm
ATM	Asynchronous Transfer Mode
CPU	Central Processing Unit
CRM	Customer Relationship Management
DCS	Distributed Computing System
DLT	Divisible Load Theory
DPSA	Dynamic Priority Scheduling Algorithm
ELM	Extreme Learning Machine
ETF	Earliest Time First
FCFS	First Come First Serve
FIFO	First In First Out
GA	Genetic Algorithm
GA-BT	Genetic Algorithm-Bit-Torrent
GADP	Interating dominate properties with Genetic Algorithm
GrADS	Grid Analysis and Display System
HEFT	Heterogeneous-Earliest-Finish-Time algorithm
HGA	Hybrid Genetic Algorithm
IaaS	Infrastructure as a Service
IBM	International Business Machine
IBS	Incremental Balancing Strategy
ICGA	Integer-coded Genetic Algorithm
IITs	Inserted Idle Times
IT	Information Technology
LAN	Local Area Network

LBACO	Load Balancing Ant Colony Optimization
MOPs	Multi-objective Problems
MQMW	Multiple QoS Constrained scheduling strategy of multi-workflows
MS	Microsoft
NSGAI	Non-dominated Sorting Genetic Algorithm II
OWS	Optimal Workflow-based Scheduling
PaaS.....	Platform as a Service
PDA	Personal Digital Assistant
PPOLD	Pull-push Optimal Load Distribution
PSO	Particle Swarm Optimization
RASA	Resource-Aware Scheduling Algorithm
RCGA	Real-coded Genetic Algorithm
RL	Reinforcement Learning
RUMR	Robus Uniform Multi-round
SaaS.....	Software as a Service
SHC	Stochastic Hill Climbing
SHEFT	Scalable Heterogeneous Earliest-Finish-Time Algorithm
SPSA	Static Priority Scheduling Algorithm
SR	Selective Rescheduling
SRS	Stop-Restart Software
TUF	Time Utility Function
RUMR	Robus Uniform Multi-round
VDCS	Virtual Distributed Computing System
WSN	Wireless Sensor Network

Chapter 1

Introduction

1.1 Introduction

In chapter 1, we begin with a general background review on distributed computing systems such as cluster, grid and cloud computing systems. Then, we review cloud services, cloud system requirements, cloud system features and the different types of cloud systems. After that, we introduce our research motivations, objectives and contributions. Finally, we summarize the organization of this thesis for the remainder of the chapters.

1.2 General background

Clusters, grids and clouds are hot topics of research interest in virtual distributed computing systems (VDCS). These concepts, coupled with advancements in infrastructure, data storage and networks, achieve overall scalability. In general, computing has been transformed from tightly coupled distributed systems (clusters) to loosely connected multi-domain distributed systems (grids) and then to well-managed distributed pools of virtual systems (clouds). Computing systems have also shifted from the delivery of computing power as a product to its delivery as a service, therein promoting the use of computers and other devices as new utilities (such as the electricity grid) over a network (e.g., the Internet). These cloud computing devices mainly focus on Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS), autonomy, scalability and flexibility to provide continuity. For large-scale service-oriented applications [150, 166], the service models represent specific abstraction levels to clouds with different requirements.

1.2.1 Cluster computing

Figure 1.1: Overview of cluster computing, reproduced from [1].

The overview structure of a cluster computing system is depicted in Figure 1.1. Based on this figure, a cluster consists of a number of computers that are linked together and work as a single computer. The component computers are connected to each other through fast Local Area Networks (LANs). The advantages of computing clusters over a single supercomputer are that they usually greatly improve performance and availability while remaining cheaper than a single supercomputer with a comparable speed and size [1].

1.2.2 Grid computing

Figure 1.2 presents the basic structure of grid computing systems. In a distributed computing system, grids consist of virtual systems that contain loosely coupled processors in a multiple-domain network to process high computational loads. Depending on the applications, cloud or grid computing environments have been considered as distributed

Figure 1.2: Overview of grid computing, reproduced from [77].

computing environments. In the case of traditional supercomputers, the processors are connected through a local high-speed communication bus, whereas in distributed computing systems, they rely on a high-speed conventional network interface such as Ethernet.

1.2.3 Cloud computing

Figure 1.3 illustrates the logical diagram of cloud computing with different abstraction layers. From the diagram, cloud computing resources can be accessed by different terminals such as laptops, servers, desktops, tablets and smartphones. Cloud computing has three abstraction layers: the application layer, the platform layer and the infrastructure layer. Now, we present different levels of architectures in clouds, as depicted in Figure 1.4. Software as a service includes Customer Relationship Management (CRM), electronic mail, virtual desktops, web servers, and development tools. The platform as a service layer possesses the execution time, database, network server, development tools, etc. The infrastructure as a service layer contains virtual machines, servers, memory, load balancers, and networks.

Figure 1.3: Overview of cloud computing logical diagram.

Software as a service: SaaS [2] means dedicated software made available through the Internet (e.g., Salesforce). Because the software runs on virtual systems present in a cloud environment, the end user is not expected to manually download and install or run the software on their own systems.

Platform as a service: PaaS provides an environment to share the customized application (e.g., Microsoft Azure and Google App Engine), developed by a group of developers. The service providers provision, maintain, and balance the load of the platform [128].

Figure 1.4: Cloud computing service level architecture, reproduced from [2].

Infrastructure as a service: The resources, such as servers, virtual machines, network bandwidth, storage, and related tools necessary to build an application environment from scratch, are shared in IaaS [172]. Such sharing significantly reduces the set up cost of small- and medium-size companies to enable the initialization of new services, the flexible expansion of existing services etc.

The trend of distributed computing systems is to direct the interconnected, visualized infrastructures, resources that are dynamically provisioned and delivered as an artificial platform for users to move their applications from anywhere in the world [42]. The computing philosophy has been shifted from the use of distributed systems or servers to clouds of virtual distributed resource pools. Currently, the scale of supercomputers is increasing significantly in terms of CPU cores; for example, the Kei supercomputer in Japan, the Argonne Mira supercomputers in the USA, and Livermore's IBM Sequoia further increase the number of CPU cores into the millions. Following this trend, scientific computing cloud centers, cloud computing platforms and enterprise computing systems

are going to consist of an extremely large number of nodes. This results in the challenge of how to optimize geographically distributed resources and users with increased utilization rate of available resources to satisfy users.

Self-service on demand: Computing resources, such as virtual servers and network storage, are provided automatically and unilaterally to the users. Thus, there is no human intervention, and customers can be self-served on demand.

Broad network access: The service delivered by the cloud service provider over the network can be accessed through different types of client platforms such as mobile phones, laptops, and PDAs.

Pooling of resources: The service provider has a pool of computing resources, which is provided to multiple users based on a multi-tenant model. Based on the user requirements, the physical and virtual resources (processor, memory, network, and data storage) are dynamically distributed to the users. The user does not control the exact location of these resources; however, they can provide information on the location of the resources at a higher level of abstraction (e.g., country, state or data center). Therefore, these services are location independent. Examples of resources include storage, processors, memory, network bandwidth, virtual networks and virtual machines.

The seven characteristics of virtual distributed computing system are listed in detail as follows [128]:

1. **Initialization speed and elasticity:** Initialization speed and elasticity are two features of the cloud service provisioning system. From the user's point of view, the resources for provisioning are essentially infinite and can be purchased in any quantity and at any time. Therefore, the initialization speed is very high, and the dynamic assignment of resources makes the system significantly elastic.
2. **Measured service:** Based on the user's requirements, cloud systems are capable of controlling/optimizing the use of resources based on certain abstract measurements and by the type of service requested by the user. Thereby, cloud systems can monitor, control and report on their resources and provide services to users based on his/her requirements.

3. **Ultra large-scale:** The scale of a cloud is large. Google owns more than one million servers for its cloud services. Similarly, Amazon, IBM, MS and Yahoo have hundreds or thousands of servers. Therefore, each enterprise has hundreds of servers, and the cloud enlarges the computing power available to users [128].
4. **Virtualization:** A cloud user can obtain cloud services through their terminal from any location. Instead of obtaining a practical entity, they can receive the cloud resource based on the demand through a laptop or mobile phone. This is an easy way for users to attain or share resources safely, irrespective of location and time. Users can complete a task that cannot be completed on a single computer in a very short time and at low cost.
5. **Reliability:** The cloud system introduces multi-transcript, fault tolerance, computation node isomorphism and exchangeable features, which help to increase the reliability of these services significantly.
6. **Versatility:** A cloud computing system does not have any constraints in terms of virtual space and is not built for specific applications. Such a system is generic and hence customizable to support any number of applications simultaneously.
7. **Affordability:** The central management of cloud helps the user by absorbing the management cost of the data center. Further because it is versatile, the resource can be shared, increasing the rate of resource utilization.

There are five different types of cloud computing models according to deployment and usage: public clouds, community clouds, private clouds, hybrid clouds and compute clouds.

- **Public cloud:** A public cloud service provider provides services, such as storage, to the general public. These services are free or are offered based on a pay-per-use model [2]. Generally, public cloud service providers, such as Microsoft and Google, own and operate the infrastructure and offer access only via the Internet (direct connection is not offered).

- **Community cloud:** The infrastructure is shared between several organizations within a community. Because this type of cloud is shared only by a limited number of users, the cost is also shared only by a circumscribed group of users, thereby increasing the price per user (security, compliance, jurisdiction, etc.). The administration of a community cloud could be done internally or externally by a third party [2].
- **Private cloud:** An individual organization with multiple consumers (e.g., business units) owns and uses the cloud infrastructure. For instance, an internal IT department or a third party may own, manage, and operate the infrastructure. It must also be noted that a pair of organizations can collaborate and operate a cloud.
- **Hybrid cloud:** The hybrid cloud infrastructure is a combination of two or more distinct cloud infrastructures (private, community, or public) with unique entities. However, they are bound together by standardized or proprietary technology that enables data and application portability (cloud bursting for load balancing between clouds) [61].

By utilizing a “hybrid cloud” architecture, companies and individuals enjoy increased fault tolerance and stability independent of Internet connectivity. A hybrid cloud architecture requires both on-site resources and off-site (remote) server-based cloud infrastructure [2].

- **Compute cloud:** Compute cloud is a model for computing and storing resources linked within a ubiquitous network. The main task of a compute cloud is to process the data stored on network storage servers, thereby obtaining meaningful results through distributed compute servers. The popularization of the term can be traced to 2006 when Amazon.com introduced the Elastic Compute Cloud [118]. It is also commonly referred to as a virtual distributed computing system.

1.3 Motivations

The main focus of this thesis is to develop an efficient load scheduling algorithm for Virtual Distributed Computing Systems (VDCS). The architecture of VDCS is shown in

Figure 1.5. In VDCS, there are M participating processors used to compute the load, and they retrieve the load fractions from data sources. Note that these data sources may not be available at the starting time. The virtual Web Role distributes different tasks to different sets of processors and data banks. The major problem is finding the optimal share of the load to be processed in the given processors from a given set of data sources.

Figure 1.5: Overview of VDCS architecture.

In the next chapter, we are going to discuss other optimization techniques in detail for scheduling problems in distributed computing systems, none of which consider multiple data centers in their model for addressing the scheduling problem. Therefore, our motivation is to form a framework that considers large-scale VDCS with multiple data storage centers and is based on a scalable framework for developing the novel load scheduling algorithm to achieve optimal system performance and high resource utilization rates.

This section discusses the major issues and challenges in developing load scheduling algorithms in virtual distributed computing systems with multiple data storage centers. This includes other challenges from VDCS dynamic resource allocation features such as load scheduling algorithms searching for optimized sequences and quantities of resources.

Model of load scheduling for VDCS: VDCS users need a stable, scalable, inexpensive and data-intensive computing system for processing huge amounts of data. Load scheduling in VDCS refers to dividing the load between assigned computational nodes, based on their key features, to achieve an optimal performance. The model should cater to heterogeneous virtual distributed computing systems with variable data processing speeds, data transmission speeds, etc. The load scheduling algorithm becomes complicated because each worker role needs to fetch data from multiple data storage centers in a particular sequence, and the data storage centers that support this task should also be working under high utilization rates.

The framework of load scheduling for VDCS with multiple data centers includes three major roles. All the key features of assigned system resources should be reflected in the framework. The load scheduling algorithm should consider the practical VDCS constraints. The solution that we obtain from the load scheduling algorithm should lead to high utilization rates of assigned resources and superior performances for the task using VDCS with multiple data storage centers.

Model of load dynamic rescheduling for VDCS: In this thesis, dynamic resource load rescheduling for VDCS with multiple data centers is a challenge due to VDCS elasticity features. VDCS need to handle both new resource availability and failure of resources. Under both these scenarios, such systems should ensure optimal scheduling of the current task. By understanding the current VDCS system status at the instant that a new resource is added or dropped from the task's resource pool, the load dynamic rescheduling algorithm should assign the remaining load fractions to each computational node. This should ensure task completion with a high utilization rate and minimum total processing time for the remaining load.

Large-scale load scheduling: The efficiency of load scheduling algorithms has become a challenge in large-scale distributed computing systems. A load scheduling algorithm's calculation time will increase exponentially with the quantity of resources assigned to a task and with the size of the load. The load scheduling problem requires further work in terms of the resource utilization rate. As the scale increases, searching for the optimal quantity of resources would be a challenge for large-scale

load scheduling algorithms. The key system features should also be considered in large-scale load scheduling algorithms to achieve minimum total load processing time and high utilization rates of the assigned resources in VDCS.

Targeting the three major challenges in the literature, Divisible Load Theory (DLT) was developed in 1988 [39]. In [123], DLT is described and shown to offer a tractable approach to scheduling that utilizes the effect of both computation and communication in distributed computing systems. DLT was proposed for solving image processing problems in a distributed workstation cluster [105]. In [89, 75], non-blocking mode communication was introduced into the DLT load scheduling algorithm. In [24, 27], processor release time was introduced into the DLT framework to unify the processor delay caused by occupation. Considering these existing practical aspects of modeling, we intend to solve our problem based on the DLT framework.

1.4 Objectives

In this thesis, a framework for a load scheduling algorithm in VDCS with multiple data storage centers is proposed. The research includes the load scheduling algorithms and an evaluation of their performance. The overall performance of VDCS was improved significantly. The main research objectives could be summarized as follows:

Design of a load scheduling algorithm for VDCS: To formulate a load scheduling framework in VDCS wherein the notations of the framework are adapted based on key system resource features and the system status. Based on this framework, we develop a load scheduling algorithm for VDCS under a multiple-data-storage-center environment. The algorithm is based on a tractable model, scalability, meta-computing accounts, time-varying modeling, extending realism, and experimental results. Apart from these feasible features, we consider multiple data sources (data banks) as well. The novel load scheduling algorithm is employed to solve the load scheduling problem to achieve minimum total load processing time for the task based on the fixed resources that are assigned by the VDCS with multiple data banks. The load scheduling algorithm also attempts to obtain a high resource

utilization rate for the assigned resources. The practical value of the algorithm also must be considered during development. Therefore, we also desire that the load scheduling algorithm satisfy practical constraints of VDCS with multiple data banks.

Design of a rescheduling algorithm for VDCS: We extend the previous load scheduling framework by introducing new definitions and notations representing the current system status and unknowns. This is done to handle dynamic resource availability. In addition, the load dynamic rescheduling algorithm should incorporate practical system constraints. The redundancy of load transmission should be considered, and notations need to be introduced to avoid this issue. Other than the assigned resource utilization rate, the DLT-based dynamic rescheduling algorithm attempts to increase the overall VDCS utilization rate of resources. This way, the waiting time interval for idle resources to join new tasks can be shortened significantly.

Design of a load-scheduling algorithm for large-scale distributed systems: A large-scale load-scheduling algorithm needs to assign tasks among computational nodes to achieve a superior performance for the overall VDCS. Simultaneously, the time consumption of processing the load scheduling algorithm should also remain within a reasonable range. The load scheduling algorithm attempts to find the optimal quantity of resource for minimizing the total processing time. In addition, the solution of the load scheduling algorithm also attempts to fulfill the practical system constraints of VDCS with multiple data banks.

1.5 Contributions

The main contributions of this thesis are given below:

Design of a load-scheduling algorithm for VDCS: A DLT-based load-scheduling algorithm is proposed and used to divide the entire load into fractions for each worker role. This is achieved through resources from multiple data banks and processing in a virtual distributed computing environment to obtain the best performance for a set of tasks with fixed resources. The algorithm is based on Divisible

Load Theory (DLT). To facilitate further discussion, a divisible load-scheduling framework has been formulated. Novel definitions and notations are introduced into the framework to represent key resource features and the system status. Moreover, practical processing constraints are incorporated with recursive equations to solve the load scheduling problem. In this way, the scheduling problem is converted into a linear programming problem. A linear programming technique has two advantages: one advantage is its ability to eventually achieve the best possible utilization of available productive resources, and the other advantage is its ability to solve complex problems by transforming them into solvable parts. Numerical examples and a satellite image classification example are presented, and the experimental simulation results highlight the advantages of the proposed load-scheduling algorithm. Finally, a simulation study to understand the computational complexity of the proposed approach is conducted, and the results clearly highlight that processing a fixed load requires on the order of minutes.

Design of a rescheduling algorithm for VDCS: The DLT-based rescheduling strategy has been constructed to satisfy the given requirements (such as maximizing the utilization rate of system resources and minimizing the task processing time) and to solve dynamic resource allocation and high fault tolerance problems in VDCS with multiple data banks. To minimize the total processing time for rescheduling the remaining load and optimize the load fractions, new notations for describing the current rescheduling VDCS status and worker role release times are introduced in the framework. The worker role release time notation has been employed to minimize redundant load transmission during the rescheduling process. We also altered the formulation of the DLT-based scheduling algorithm based on practical VDCS constraints. The dynamic VDCS scheduling problem is modeled as a linear programming problem. By illustrating with numerical examples, we show that a novel dynamic rescheduling algorithm can guarantee minimum processing time under dynamic resource assignment and provide high fault tolerance.

Design of a load-scheduling algorithm for large-scale distributed systems: A hybrid genetic algorithm is developed to solve the problem of data partitioning using multiple rounds of load distribution with an affine communication model in a

single-level tree network. The algorithm is used to obtain the optimal load fractions for a given distribution order, number of worker roles and number of rounds. In this thesis, we propose modified crossover operators, mutation operators and fitness functions, ensuring that they produce valid solutions that satisfy the total load constraints. Further, the proposed hybrid genetic algorithm uses three different population initialization schemes to obtain better convergence. Using the load fractions, a heuristic approach is presented to determine the optimal distribution order, optimal number of worker roles m^* ($m^* \leq m$), and optimal rounds of load distribution n^* ($n^* \leq n$) such that the processing time of the total processing load is minimized. The numerical simulation results clearly show that the proposed approach finds the best solution for a multi-round load distribution problem. Finally, we discuss the hybrid genetic algorithm used to solve scheduling problems in VDCS with multiple data banks and single installment by updating the fitness function. The new fitness function considers DLT-based scheduling algorithm constraints. The numerical example can obtain a better solution using the new hybrid genetic algorithm. The solution can also obtain the best performance of the VDCS with multiple data banks.

1.6 Thesis organization

The thesis is organized as follows:

In Chapter 2, we review most of the research works related to the current study. The literature on DLT is surveyed based on topologies and different practical constraints such as limited buffer systems, processor release time, start-up delays, non-blocking style of communication and result collection schemes. Then, we discuss DLT multi-installment and nonlinear DLT. Next, DLT-based system applications are reviewed. Then, we move on to scheduling in the cloud, where we categorize previous research works based on different aspects. Finally, we focus on distributed computing system scheduling architectures and algorithms, which we differentiate based on their focus points, advantages and drawbacks. Our proposed scheduling algorithm cannot be compared to existing scheduling algorithms because the framework is different in terms of the number of data banks.

In Chapter 3, we formulate the virtual distributed computing framework and define the notations and definitions. We use these notations and definitions to express the practical constraints in formulating the equations. In this way, the distributed computing system scheduling problem becomes a linear programming (LP) problem. Solving the LP problem, we obtain the optimized load distributions for each processor and the minimized overall load processing time. The scheduling strategy for a compute cloud environment is proposed and demonstrated to perform properly in this chapter with four numerical examples, six simulations and a real-world example of a satellite image processing application. For the first example, we pick a small freight and fewer virtual resources, and then, with the result, we describe it in the timing diagram. The other examples focus on different aspects. The two simulation cases are for heterogeneous and homogeneous cloud computing environments with different numbers of processors and data bank combinations. The other three simulations discuss changing individual key parameters that affect system performance. The last simulation focuses on the proposed load-scheduling algorithm and shows how it can perform efficiently with large quantities of resources. The load-scheduling algorithm obtains the minimum total processing time in the real-world satellite image processing system case.

In Chapter 4, based on the framework in Chapter 3, we propose a heuristic DLT-based rescheduling strategy for resolving VDCS dynamic resource assignment and system fault tolerance in a virtual distributed computing environment with multiple data banks. We also leverage and develop further the notations and definitions used to stop and restart the rescheduling process. These notations and definitions are used to convey the practical constraints for obtaining the equations used to form the linear programming problem with while considering the optimized remaining load processing by applying the worker role release time to our framework. We prove that our rescheduling scheme improves the VDCS service by significantly reducing the processing time for the system user and by increasing the usage of service providers' existing resources.

In Chapter 5, we propose a Hybrid Genetic Algorithm (HGA) to determine the optimal distribution order, optimal number of worker roles $m^*(m^* \leq m)$, and optimal rounds of load distribution $n^*(n^* \leq n)$ such that the processing time of the overall processing load is minimized. HGA employs modified crossover and mutation operators such that

the operators always produce a valid solution. Moreover, we propose various population initialization schemes to improve the convergence. By altering the HGA fitness function by incorporating four practical constraints, we can solve the load scheduling problem for VDCS with multiple data banks in the single-installment case. In conclusion, we provide a comparative study with a simple hybrid genetic algorithm and particle swarm optimization to highlight the advantage of the proposed algorithm. The results clearly show the strength of the proposed HGA in both the VDCS single data bank and multi-installment case. HGA with the new fitness function can also solve scheduling problems in VDCS with multiple data banks in the single installment case. Comparing the results of the numerical examples, we show that the HGA also performs properly when applied to the scheduling problem in VDCS with multiple data banks and for single installments.

In Chapter 6, we draw the conclusions of our study and present the recommendations for future directions.

Chapter 2

Literature review

2.1 Introduction

In the previous chapter, we discussed the motivations, objectives and contributions of our research study. This chapter focuses on providing detailed insight into related research topics in DLT such as network topologies, constraints, multi-installment, nonlinear loads and applications. Then, we discuss existing cloud computing scheduling research works, followed by other optimal scheduling techniques for distributed computing systems.

2.2 Divisible load theory review

In 1988, the multiprocessor scheduling topic began receiving attention [39]. [123] described “the divisible load theory that offers a tractable and realistic approach to scheduling that allows integrated modeling of computation and communication in parallel and distributed computing systems.” DLT considers the computation speed of processors, communication speed of the links, architecture of the networks and other constraints of the system. Hence, the framework is practical and easy to implement.

Divisible Load Theory (DLT) for a general distributed computing system was introduced in [123]. Minimization of the processing time of the entire processing load is achieved by dividing the load into load fractions of arbitrary size and processing independently on the processors in the networks. In a Distributed Computing System (DCS), the components of networked computers communicate and coordinate their actions by passing messages. A DCS is usually built to fulfill a task or a related set of tasks as a

service such as in a cluster, grid, or cloud system. Such systems are conceptually similar to [77]. Cluster computing consists of multiple data processors, multiple storage devices and redundant interconnections to form a single, highly available system [147], as shown in Figure2.1. A typical cluster-based system is IBM's Sysplex [3]. The definition of grid computing is the widespread availability of powerful computing resources as low-cost commodity components [87], as shown in Figure2.2. Cloud computing describes a new model for IT services based on the Internet and typically involves the provisioning of dynamically scalable and often virtualized resources over the Internet [8, 167, 45, 94], as shown in Figure2.3.

Figure 2.1: A model of a cluster computing system, reproduced from [77].

Therefore, we discuss DLT in detail based on network topologies, constraints, multi-installment, nonlinear loads and applications in the following.

Figure 2.2: A model of a grid computing system

2.2.1 Network topologies

The network architecture is one of the main concerns in forming DLT-based scheduling algorithms and solving such problems. The pattern of the connections between computation nodes is known as the network topology. In this section, we will focus on different types of network topologies such as a bus network, single-level tree network, multi-level tree network and hypercube network.

Bus Network: A bus network along all the network topologies is the simplest because the bus-type communication medium is equally shared by all the processors. The network model consists of a computerized processor attached to a linear bus, as in Figure 2.4. The root processor should take the role of both a bus control unit and a processing node simultaneously. After a bus control unit obtains a load, the load is split into m fractions $(\beta_1, \beta_2, \dots, \beta_m)$ and distributed to the processors in a sequence $(1, 2, \dots, m)$. The computation process commences immediately after the respective load fractions are obtained. In [13], a shared bus communication medium is used to solve a load-sharing problem involving the optimal allocation of measurement data among and sensor-driven

Figure 2.3: A model of a cloud computing system, reproduced from [77].

processors. In [140], the authors discussed scheduling divisible loads in a bus network topology using DLT to observe the overhead component due to the start-up time, which could degrade the performance of the system. They provided a closed-form expression to this end. In [34], using a bus network performance analysis and experimental simulation of the scheduling of a divisible load, a very large matrix-vector product computation was given. In [28], the test-bed consisted of high-speed Pentium machines on a bus network, and divisible load scheduling applied to an image processing application (edge detection) was theoretically and experimentally studied. In [62], in a bus type environment, a strategy-proof mechanism for scheduling DLT was proposed. Here, the processors reported their true processing power; then, based on the parameter set, the scheduling algorithm assigns loads using the full processing power to achieve the overall optimization of system resources. In [31], the processors belonged to an autonomous self-interested organization. The authors proposed a strategy-proof mechanism for scheduling divisible loads in bus networks without control processors. In [32], the author proposed a strategy-proof mechanism with verification for scheduling divisible loads in a linear network with

boundary load origination. In [33], the authors designed an incentive-compatible mechanism for scheduling non-malleable parallel jobs on a parallel system. In this work, the user increases their welfare selfishly and does not consider optimal system performance.

Figure 2.4: Bus network architecture with m processors attached.

Single-level tree Network: A single-level tree network is depicted in Figure 2.5. According to Figure 2.5, this distributed network topology consists of $(m + 1)$ processors and m links. p_0 is the root processor, and processors $p_1, p_2, \dots, p_m$ are connected to the root processor via communication links $l_1, l_2, \dots, l_m$. l_i is the communication link connecting the i^{th} processor (p_i) to the root processor (p_0). Once the root processor (p_0) obtains a load, the total load is divided into $(m + 1)$ load fractions $(\beta_0, \beta_1, \dots, \beta_m)$ and then transmitted $(\beta_1, \beta_2, \dots, \beta_m)$ to the child processor $(p_1, p_2, \dots, p_m)$, reserving β_0 for the root processor. The child processors start computations immediately after receiving their respective load fractions from the root processor. In [40], the communicating processors of a tree network are examined in two cases: with the front-end processors using communication in non-blocking mode and without the front-end processors using blocking mode communication. The objective is to obtain the minimum processing time by optimally dividing the entire load into small fractions for each processor. In [138], single-level tree network architecture was used to provide the solution for scheduling issues for polynomial time complexity computational loads by applying the divisible load theory framework. Polynomial time complexity computational loads are employed to solve higher order algebraic equations to determine the optimal load fractions assigned to the processors in the network. In [88], load sequencing and processor-link arrangement

Figure 2.5: Single-level tree network architecture.

is provided under the non-blocking mode of communication in a single-level tree network. The results indicate that the performance under non-blocking mode is much better. A buffer size of the processor was considered in a heterogeneous single-level tree network for discussing scheduling in a divisible load problem [75]. In [20], multi-installment is applied to minimize the processing time in a single-level tree network. A closed-form solution for a particular case containing identical processors and identical links was derived using two techniques, one of which is useful when the number of processors is large and the other being useful when the number of installments is large.

Multi-level tree Network: A multi-level tree network architecture is shown in Figure 2.6. The child processors $(p_1, p_2, \dots, p_m)$ at level 1 are connected to the root processor p_0 . In this figure, each level-1 child processor is connected to a set of a fixed number of level-2 child processors. Because the physical communication of the co-central processing unit is fixed, the processors in this architecture utilize non-blocking-mode communication. According to the divisible load scheduling scheme of the multi-level tree network, the root processor p_0 contains the entire data load and is in charge of load scheduling to assign the optimal load fractions to all the child processors in both level 1 and level 2, and the load β_0 is reserved for the root processor to process. After

scheduling the total load into fractions, the root processor sends the individual fractions $\beta_1, \beta_{11}, \beta_{12}, \beta_{13}, \beta_2, \beta_{21}, \beta_{22}, \beta_{23}, \dots, \beta_m, \beta_{m1}, \beta_{m2}, \beta_{m3}$ one by one to the child processors at level 1. The level-1 processor will retain the fractions for itself and pass the remaining load fractions to level-2 processors, which directly connect to it. In this case, the processor (p_i) at level 1 will retain the load fractions (β_i) to process and will disperse the load fractions ($\beta_{i1}, \beta_{i2}, \beta_{i3}$) to the minor processors in sequence p_{i1}, p_{i2}, p_{i3} . Once the loads are received, all the processors start processing immediately.

Figure 2.6: Multi-level tree network architecture.

For a general multi-level tree network architecture, unknown or time-varying computation and communication parameters are discussed to propose new load distribution scheduling strategies [84]. In [58], the proposed algorithm is applied to resolve the issue of users cheating on processor computation rates, which causes optimization failure for divisible load scheduling in multi-level tree network environments.

Hypercube Network: A four-dimensional hypercube network is illustrated in Figure 2.7. In this network architecture, each processor is a node of the multi-dimensional

hypercube. It is assumed that all computational processors and communication links are homogeneous inside the hypercube network and that the computation and communication speed are the same for all the processors. Further, it is assumed that the processors are utilizing the non-blocking mode of communication because they are equipped with front-end processors. This architecture can support simultaneous computation and communication. The total number of processors required depends on the hypercube dimension, i.e., if the hypercube is d -dimensional, the total number of processors will be 2^d . As with other network architectures, the root processor divides the load and distributes the fractions of the load to the other processors. The optimal load distribution is based on the layers or number of hops between the root processor and the child processors. This is indicated as numbers marked next to each node in Figure 2.7.

The root processor (p_0) maintains a load fraction (β_0) for processing and distributes the remaining load to other processors. Based on scheduling algorithm, the root processor divides the load into parts, which are delivered to child processors at a distance of a single hop. The processor will continue assigning the portions based on a divisible load scheduling algorithm and send the remaining load to the idle processors, which are also at one hop. Then, the processor will perform the same operation until the last idle processor ($2^d - 1$) is assigned a load fraction to be processed.

In [106], in the hypercube cluster environment, finite size buffer constraints are brought into the discussion for achieving the minimum processing time through an efficient scheduling scheme. Other aspects of resource constraints in a DLT framework for hypercubes are addressed in [78, 122, 18].

2.2.2 Constraints

The practical system limitations are realistic in nature. Practical systems include processes and characteristics that increase functional redundancy such as limited buffer systems, processor release times, start-up delays, non-blocking modes of communication, and result collection scheme. The constraints are included to make the system more realistic and practical from an academic point of view.

Limited buffer system: DLT has become a practical paradigm applied to a distributed computing system, although it assumes an infinite buffer size for each node.

Figure 2.7: 4-dimensional hypercube network architecture.

However, a real computational node has a limited buffer size. Therefore, in some cases, the computational node will not have an adequate buffer to store the assigned load fraction for processing. For instance, if the scheduling scheme does not consider the buffer size, it may end up assigning an inappropriately large load to a relatively small buffer system because the load processing time is longer. In an earlier work [18], a rule regarding processors with sufficient resources was set such that those processors should be removed from the system to overcome the buffer size limitation in the scheduling algorithm. In [103], the practical elements that limited the buffer size were studied for extending the DLT paradigm. In the paper in [103], an incremental balancing strategy (IBS) for a heuristic algorithm was presented, and the examples proved that the algorithm efficiently balanced the buffer size limitation. Further research was introduced to support both finite buffer size and granularity constraints using an incremental balancing strategy algorithm. This was further used to extend the scheduling divisible load theory in hypercube clusters [106]. In [165, 153], the DLT limited buffer was proved to be an

NP-hard problem.

In [23], a Pull-Push Optimal Load Distribution (PPOLD) algorithm based on the incremental and optimal sequence theorem was proposed for an unbalanced multi-level tree network. The DLT with finite-sized buffers in heterogeneous single-level tree networks was investigated in [102]. The effects of finite-sized buffers on the multi-installment divisible load were investigated in [17]. A good illustration of computational complexity in divisible load scheduling was also presented. An unexpected conclusion drawn from these experiments is that “bandwidth-centric” distribution does not perform as well as a classical scheduling distribution in terms of minimizing the processing time for each task. The research in [48] analyzed the application of a limited buffer on multi-installment divisible loads, and in [53], a heterogeneous divisible load was considered. Later, these research works on the limitation of the buffer were investigated in [80]. The size of DLT scheduling load fractions for processors was decreased due to the introduction of buffer size limitations. Consequently, the overall processing time increased, as observed in [103, 23, 51, 52]. In [52], out-of-core computations were introduced to address the assumption of flat memory systems, which refers to non-hierarchical memory systems. In most modern computer systems, storage is hierarchical. “The higher a certain level of memory hierarchy is, the higher a transmission rate that can be achieved.” Unfortunately, higher memory hierarchies have smaller sizes; therefore, we refer to applications using external memory, such as hard disks or solid-state storage, to drive the out-of-core computations. The performance study in [52] showed that the multi-installment strategy provides a better performance for reasonably selected load sizes compared to out-of-core computations.

Processor release time: For all contemporary computer systems with operating systems installed, there are certain processes that are needed to support the system for proper operation, such as firmware updates, system software maintenance, and system hardware maintenance, that are occupied by other loads. These processes will inevitably cause delays for processors initiating computations on load fractions assigned to the processors.

In [24, 27], various other computation processes (processing locally) involve the processors in the network. The delays are unified as a release time, which causes the assigned

load to the cloud to not start processing immediately. In [24, 17], homogeneous processors were discussed under two cases: a case using identical release times and a case using non-identical release times based on a divisible load theory multi-installment scheduling strategy on a bus network. In both scheduling schemes, a numerical condition is derived to find a maximum bound on the number of installments that are required to process the load continuously. In [17], the combined results for release times under limited storage and processor conditions is discussed.

Start-up delays: Overhead elements (the start-up delay) of communication and computation certainly cause the scheduling of divisible loads to degrade the overall performance, resulting in an increase in the total processing time. However, start-up delays (protocol processing delays, unavailability of certain communication resources, queuing delays, etc.) are considered. On the other hand, computational overhead is caused by extracting the data delay, processor initialization etc. Because communication and computation models are based on affine functions of the load fractions, introducing start-up delay into the divisible load scheduling scheme is a very challenging task.

In [79], for the first time, start-up delay in communication was presented and lead to an increased total task processing time for a linear network. Here, we found that, once the number of processors increases beyond a certain threshold, the load assigned to partial processors can be zero. In [26, 29], the effect of the start-up delay in computation and communication processes for a bus network was presented. An alternative approach to scheduling problems with overhead components on bus networks was presented in [139], and a closed-form expression was illustrated to achieve the optimal size of load fractions. Instead of the iterative procedure given in [26], applying this closed-form expression, we can easily obtain the conditions for the optimal number of processors and the optimal sequence of load distributions. In [46], communication start-up delays were applied to a linear programming formulation of the divisible load scheduling problem. This linear programming model was applied to the divisible load scheduling problem with limited buffer and communication start-up delays. In [50, 46], we found a study on a DLT multi-installment scheduling algorithm considering the affine model of communication conducted with start-up delays. The linear programming approach was given in [18, 100]. The closed-form expression is difficult to derive in the case of dropped-off processor-link

pairs. It is also difficult, in the case of heterogeneous systems, to obtain the closed-form expression for the processing time and optimal load fraction size.

Non-blocking mode of communication: Divisible load theory was applied with the presumption that communication follows a store-and-forward model, which is also known as “blocking mode” communication [18]. A simple explanation is that the processors cannot compute or transmit until they receive the full load of fractions either for themselves or for other processors.

With the introduction of a new communication concept called store-and-bypass SB , the idle time can be significantly reduced. This type of communication has two other names: “non-blocking mode” of communication and “virtual cut-through” switching [89, 75]. In this communication model, while the loads pass by an intermediate processor, the processor stores its own load and forwards the remaining loads to the adjacent processors instead of storing all the loads. The intermediate processor builds a virtual tunnel to link to the root processor with the child processors. The benefit of introducing this communication model is that it supports linear network DLT scheduling models, similar to single-level tree networks. Therefore, it becomes easy to solve by leveraging all prior DLT scheduling research works on single-level tree networks.

Furthermore, in the non-blocking mode, the processor starts the computation and transmission because its front-end processor is receiving the load fraction. Hence, comparing blocking mode with non-blocking mode, the latter significantly reduces the processor idle time. The divisibility property of the processing load is further exploited in the non-blocking mode of communication. In [90], the optimal sequencing and arrangement of processors in a network using the non-blocking mode of communication was studied. In non-blocking communication mode, many effects, such as the optimal sequencing and organization of processors in the network and the effect of start-up delays, can be easily obtained. The communication model helps to unify the linear networks of single-level tree networks.

Result Collection Scheme: In most of the previous work that has been performed on DLT scheduling, the time necessary to gather these results by the root processor is assumed to be zero. This means that either the results are stored in each computation node’s storage or the result is a simple message such that the time needed to transmit

it back to the root processor can be ignored. In practice, there are certain cases that fulfill these assumptions. Nonetheless, in most cases, the results for each load fraction need to be collected and rebuilt from all the processors to the root processor. Therefore, in addition to the communication delay in sending and distributing the load fractions to the child processors, the time taken to gather these results of the computation for each child processor is also significant and adds up to the overall processing time. The collection of the results by the root processor was first considered in [4]. In that paper, they considered that not all the processors stop processing simultaneously.

For devising a scheduling algorithm with start-up delay, the optimal sequence of load distribution and result collection delays was considered in [12]. It is possible that the size of the load fractions communicated to the child processors and the results of the computations collected from the child processors maybe be different for different applications [7]. According to the post-process load fraction size, the computation, closed-form expressions for the optimal number of processors and load fractions were derived for the case of a homogeneous bus network in [7].

2.2.3 Multi-installment

Earlier research work in divisible load scheduling considers single-installment (single-round) processing. In this technique, the load has been divided into as many fractions as the number of processors, and these fractions are distributed to the processors in a single installment. Because the processors can start processing only after receiving the load fractions in full, the idle time of the processors is high in the single-installment strategy. In multi-installment (i.e., multi-round) processing, the load is sent to a processor in more than one chunk. Multi-installment divisible loads are based on multi-installment (i.e., multi-round) processing. Therefore, the processor P_1 will execute its computations earlier, and the total processing time will be shorter. In [47], multi-installment processing was shown to reduce the scheduled length to 0.632 of that under the single-installment strategy. Multi-installment scheduling with communication delay was presented for a linear network in [19] and for a single-level tree network in [20]. In the multi-installment scheduling problem, obtaining closed-form solution for the optimal size of closed-form expressions and for an optimal size of load fractions in heterogeneous systems is difficult.

In the case of homogeneous networks, closed-form expressions for the optimal size of load fractions were obtained using the binomial theorem and the rational expression theorem in [20]. The closed-form expression obtained using the binomial theorem needs high-degree polynomial and rational expression theorem, which demands the evaluation of fractions of fairly large numbers. In [21], the theoretical and practical aspects of the multi-installment scheduling algorithm for multiple loads in a bus network were presented. In [50, 162], a multi-installment technique was proposed for homogeneous tree networks with start-up delays in the computation and communication processes. The effect of result collection in multi-installment scheduling algorithms for homogeneous networks was presented. In that paper, the rational expansion theorem is applied to achieve a closed-form expression for processing.

Recently, a uniform multi-installment (UMI) strategy (the processor receives equally sized portions of a load in each installment) was presented and applied to heterogeneous tree networks with communication start-up delays in [15, 163]. The main advantage of the uniform multi-installment strategy is that we can obtain the near-optimal number of installments required to process the load to minimize the processing time. In [164], a Robust Uniform Multi-Round (RUMR) algorithm was presented. In the RUMR algorithm, the uncertainty in the communication and computation times is caused by the platforms (e.g., when resources are non-dedicated and time shared) or by the application (e.g., when the computation is data dependent) and are handled by increasing the chunk size (load fraction size) during the initial rounds, later decreasing the chunk size toward the end. In [48], the memory limitation of the multi-installment strategy was investigated. Subsequently, the multi-installment strategy was investigated and also found to depend on the proper number of installments [132]. The multi-installment divisible load method was improved on a k -dimensional mesh in [37]. Concerning recent research on the value of the multi-installment divisible load strategy, a number of heuristics for the scheduling of multi-installment divisible loads were proposed in [80].

2.2.4 Nonlinear loads

The first non-linear model was proposed in [124], which discussed many algorithms that use a non-linear function to compute the time proportional to the input load size. Subsequently, in [76], the researchers considered the scheduling model in a tree network, where

the computational time for each node is nonlinear in the size of the assigned load. By applying simple equations, both the optimal load allocation and speedup for simultaneous load distribution for a quadratic nonlinearity was achieved. It was proven that the computational complexity is nonlinear in the size of the assigned loads and that super-linear speedup is achievable. This strategy can be used in aerospace applications such as spectrum computation, radar and sensor data processing and satellite image processing. The problem of scheduling the optimal distribution of polynomial-time-complexity computational loads in a single-level tree network with a collective model was presented in [138]. To resolve this problem, researchers proposed the combination of a DLT framework with computational loads of polynomial-time complexity to formulate a set of higher order algebraic equations. By solving these non-linear recursive equations, they obtained the optimal processing time and load fractions assigned to processors.

2.3 Applications

In the past two decades, the DLT has found a wide variety of applications in the field of parallel processing. The primary motivation for research in the area of divisible loads evolved from the requirements of processing a great mass of information that arrives at distributed intelligent sensor networks in military surveillance systems [105]. In [14], large numbers of independent tasks with low-granularity applications in both science and engineering are capable of performing parallel processing in a master-worker fashion. In this paper, applications are categorized as follows: data processing applications, video processing applications, image processing applications, signal processing applications, scientific applications and real-time applications. We are going to discuss the content under certain categories.

2.3.1 Data processing applications

Data processing applications start from the conventional single-server architecture. However, because the database becomes very large, DLT can be applied to distributed database processing with tight data processing time demands such as in aerospace database applications. In [46, 81], the proposed model of a computational process includes the time

of distribution to remote sites as well as the processing time. Database record searching is an example of an important application that can be performed in parallel by a large number of server simulation environments such as PVM, MPI, Linda, and Express. In [92], DLT was used to solve for the expected time to search for both single and multiple signatures in certain multiple processor database architectures, for example, in aerospace database applications. For DLT as applied to parallel distributed database applications, please refer to [93].

2.3.2 Video processing applications

Video processing applications represent a typical data-independent application. The processing time of the motion estimation is a large portion of the overall processing time of the video encoder. Periodic Write-Read-Compute scheduling for parallel video processing can address both continuity and periodicity of video data in client-server-based systems with point-to-point communication between the host and the processors [6]. In [7], Parallel Recursive and Parallel Interlaced were proposed as optimal data partitioning and scheduling algorithms, respectively for real-time frame-by-frame processing. These two algorithms could be applied to a wide range of data-independent applications. In [67], the researchers attempted to implement a video encoder on a bus network by considering partitioning and balancing of the computational load among the processors. For other video processing applications, refer to [22].

2.3.3 Image processing applications

Image processing is a type of data processing and is an important application. Feature extraction from input images has two levels of processing: a local computation followed by inter-processor communication and computation. An image is first divided arbitrarily into small segments and processed independently to extract the local features. The locally extracted features are exchanged between the processors, and the local features are used to extract the desired feature. Additional details on the application of the divisible load scheduling problem in image processing applications and computer vision and data processing can be found in [104, 28] and [133, 25]. In [91], an efficient parallel image convolution algorithm was shown to speed up image processing in a distributed

system. In [28], an efficient partition strategy was considered for a heterogeneous set of processors in a bus network distributed system, and the results showed that the proposed strategy obtained a much better performance compared to an equal-partitioning strategy. In [105], an efficient partitioning of the image processing load and scheduling of the tasks was shown in a distributed image processing workstation cluster.

2.3.4 Signal processing applications

A simple application of the divisible load scheduling algorithm is the removal of zero-mean noise from measurement data. Each measurement in the raw data requires the same processing elements (algorithm). The raw data have a larger number of measurements that can be arbitrarily partitioned and processed independently without any precedence. Important signal processing applications are feature extraction filtering and the Hough Transform [82]. In [117, 65, 66], an adaptive indexed divisible load theory was proposed for solving high-power transmission energy imbalance issues in Wireless Sensor Networks (WSNs). Such issues were caused by system status variations due to sensor failures. This method was used to redefine the indices of sensors in each sensing round while calculating the assigned workload portions. In [117], an optimal load allocation approach was presented for measurement and data reporting in wireless sensor networks with a single-level tree network topology.

2.3.5 Scientific applications

Large matrix-vector and matrix-matrix products are required in many scientific computational applications. In the case of matrix-vector product problems [125], the vector and row-wise partitioned matrix are communicated to the child processors. Central processors gather and store the results of computations. In matrix-matrix product problems, the first matrix is partitioned in row-wise and a second matrix is partitioned column-wise and distributed to the processors in the network. The processors compute the product independently and communicate the results to the root processors. A performance analysis and experimental implementation of large matrix-vector multiplication in a bus network are illustrated in [59, 34]. Another application involves database processing, where a complete database is partitioned into smaller databases and processed independently

[93]. The details of the application of divisible load scheduling theory in biomedical scientific signal compression and simulations of molecular dynamics were explained in [11] and [5].

2.3.6 Real-time applications

Because the objective of divisible load theory is to minimize the total processing time, it is obvious to have the processing time as the requirement. In [64], load re-tasking and redistribution was implemented at runtime to achieve feasible load allocations and to minimize the job processing time using an interconnected set of heterogeneous processors characterized by their load computing speeds and I/O speeds. In [108], optimal partitioning in utilizing Inserted Idle Times (IITs) was integrated with a previous approach and used to develop an enhanced algorithm that better utilizes IITs. The results illustrated the advantages of the newly proposed approach. In [112, 113], advance reservation was supported in a novel divisible load real-time scheduling algorithm for a cluster. This approach enhanced both real-time agreement and compensated for under-utilization concerns by introducing advance reservation into grid clusters.

2.4 Scheduling in cloud environments

Many scientific and applied science problems, such as satellite image processing, radar and sensor data processing, and spectrum computation, are data driven and computationally intensive. Therefore, some of the previously mentioned applications yield vast quantities of information. The objective of these data-driven computations is to minimize the processing time of computing the loads. Because the compute cloud [61] provides virtual computing services, these loads can be processed using a compute cloud environment to reduce the total processing time. Currently, many commercial companies provide compute cloud environments (i.e., Amazon EC2 and MS Windows Azure) that allow users to scale their content based on their resource demands. A compute cloud environment provides heterogeneous computation platforms and multiple data storage units for the applications to run. The important challenge in a compute cloud environment is to design a scheduling strategy to handle tasks and to process them in a heterogeneous environment with shared data centers.

Most of the research works on scheduling strategies in a cloud environment are based on service level agreements [42, 101]. Recently, in [151], task scheduling in a cloud environment was solved using a computationally intensive game theory approach with time and cost constraints. Research has been conducted on homogeneous cloud environments based on quality of service, without considering communication costs and queuing theory [74, 119, 73]. Genetic-algorithm-based task scheduling was implemented in the MapReduce dynamic cloud computing environment in [57]. In addition, a swarm-intelligence-based scheduling strategy has been developed to initialize the processing time in compute cloud systems, as proposed in [121]. Both genetic-algorithm-based and swarm-intelligence-based approaches are search-based approaches and are computationally intensive [57, 121].

Recently, in [56], a large-scale polynomial multiplication was presented for computing clouds using the divisible load theory paradigm, where the virtual computer interface handles the data distribution and also determines the optimal number of resources. The method assumes that the load to process arrives at a virtual computation interface, and the load is distributed to cloud resources. In [154], multiple sources fetching the load fractions to the child processors employ simultaneous communication to facilitate load distribution. Here, a closed-form solution to load fractions using DLT is derived under the assumption that all processors start receiving data from all the sources simultaneously and that the processors have the ability to store the completion fraction of assigned data to it.

Currently, in terms of computing nodes, CPU cores in distributed computing systems, such as the Amazon Web Services and Elastic compute cloud (EC^2), Elastic Computing Platform (ECP) from Enomaly, Web-based storage from GoGrid, Azure from Microsoft, and Rackspace cloud from Rackspace, are becoming increasing very fast. Following this trend, scientific distributed computing centers, distributed computing platforms and enterprise computing systems are going to own vast numbers of nodes as well. With these increased scales, issues concerning how to optimize geographically distributed resources and users with an increase in the usage of available resources in terms of time are being raised. For more information on cloud computing, please refer to [9, 83].

A reschedule scheme is a key element of performance that guarantees the proper operation of distributed applications in large-scale heterogeneous dynamic distributed

computing environments. From distributed computing service providers' point of view, it is important to increase the usage of their existing resource pools in a timely manner. In other words, a distributed computing service provider attempts to maximize their profit with their existing infrastructure. For users, the aim is to decrease the processing time at minimal cost. Therefore, it is very challenging to allow the rescheduling scheme to address the static distributed computing resource geographical distributions and dynamic resource pool variations, therein considering the minimum processing time while increasing resource utilization.

2.4.1 Workflow scheduling in clouds

Unlike our work, which concentrates on load rescheduling, a large number of existing works focus on workflow rescheduling. However, all these works focus on minimizing the make span of the task. In [158, 157], batch mode methods were used for allocating tasks to resources by considering the characteristics of distributed and highly heterogeneous systems such as make-span, flow-time, resource usage and matching proximity. The scheduling method was widely applied in many grid-based applications, especially real-time applications. Jin Xu et al. implemented Chemical Reaction Optimization. This meta-heuristic technique was used for task scheduling in a dynamic grid environment. They considered the workload and reliability of resources to minimize the make span and flow time [159]. Tasgetiren et al. proved that a Particle Swarm Optimization (PSO) algorithm can be introduced to solve the permutation flow shop sequencing problem to obtain the minimized makespan and total flow time of tasks. The algorithms were compared with the three best makespan, flow time and both makespan and flow time algorithms. The results indicated that the proposed PSO algorithm obtains the best performance [143]. In [110], the work focused on rescheduling certain jobs belonging to a workflow that avoids temporal violations that affect the overall performance. In [169], the main objective was to shorten the makespans by adopting workflow scheduling over time.

2.4.2 Rescheduling in clouds

In [38], the most profitable requests are prioritized and rescheduled dynamically without compromising the requirements of the requests for cloud service applications. The

newly proposed algorithm managed to achieve a minimum change in the existing service assignments from a graph coloring perspective to resolve such requests. In [16], a dynamic, generic and adaptive rescheduling algorithm was proposed to overcome the issues caused by dynamic distributed systems. The environment, such as availability, reliability, and load balancing, were considered. In [55], a dynamic scheduling algorithm called the Hybrid Re-mapper PS (Hybrid Re-mapper Minimum Partial Completion Time Static Priority) was proposed for heterogeneous environments. In [144], Theresa et al. addressed fault tolerance in terms of resource failure using periodic check pointing, which periodically saves the job state. An inappropriate check-pointing interval leads to delays in job execution and reduces the throughput. They proposed a strategy to achieve fault tolerance by dynamically adapting the checkpoints based on the current status and history of failure as well as information of the resource, which is considered maintenance in the information server. In [114], the proposed Elastic site manager is a resource manager that is able to dynamically provision resources required to run scientific applications from both public clouds and private clouds.

Third, some of the research works have concentrated on improving job delay. In [95], Zomaya et al. investigated the effectiveness of rescheduling using cloud resources to increase the reliability of job completion. Specifically, schedules are initially created using grid resources, and cloud resources (relatively costlier) are applied only for rescheduling to address a delay in job completion. In [111], a self-adaptive network-aware virtual machine rescheduling algorithm was proposed to maintain an optimal system-wide status. By considering unbalanced resource utilization and system communication performance declines, an algorithm is proposed to balance the load processing intercommunication traffic via rescheduling. The real test bed-based experiments demonstrated that the algorithm performs well in terms of decreasing the finish time of map-reduce tasks and reducing the time delay for network applications. In [145], the authors considered not only deadlines but also start time constraints.

Finally, some other focus points are as follows. In [41], the Grid Analysis and Display System (GrADS) implemented rescheduling in their research work on rescheduling. They implemented a user-point check-pointing library called SRS (Stop Restart Software). Stop / migration / restart and a single-processor swapping are feasible and

require minimal additional programming. This technique is also very flexible, and the overhead for processor swapping is quite low. Two mechanisms are helpful in taking corrective actions when used to track the execution once launched: a simple stop / migrate / restart approach to rescheduling grid applications and a process-swapping approach to rescheduling. An assumption is made that a stop and restart technique may not fit all task rescheduling cases. In [152], HEFT was cross-compared with a genetic and simple “myopic” and compared with incremental workflow partitioning against the full graph scheduling strategy in an ASKALON grid environment. The results show that HEFT obtains a better performance compared to the other algorithms. In [127], a few points were selected during executions for rescheduling in grid computing environments using a low-cost rescheduling policy selective rescheduling (SR) technique. Most evaluation and analysis studies of various heuristics surprisingly showed that similar values are obtained for the quality of results, therein identifying the same strengths and weaknesses as found in previous approaches, the difference being only a few percent. In [85], the earliest time first algorithm is based on ensuring that the processor is as busy as possible. It attempts to compute, at each step, the earliest start times of all ready nodes and selects the node that has the shortest start time. In [86], the Highest Level First with Estimated Times algorithm uses a hybrid of the list-based and level-based strategy. This algorithm schedules a task to the processor that provides the earliest start time.

2.5 Optimization techniques for scheduling problems

Optimization algorithms are also introduced for improving the performance of a distributed computing system, with compared aspects listed in Table 2.1.

Table 2.1: Distributed Computing System Optimization Techniques

Ref	Environment	Focus Aspects	Approach	Type of Task	Optimal Improvement
[126]	Grid	Makespan	Resource-Aware-Scheduling Algorithm	Grouped tasks	Reducing makespan
[63]	Cloud	QoS	Genetic Simulated Annealing Algorithm	Various tasks	High QoS
[146]	Cloud	Makespan	Heterogenous-Earliest-Finish-Time Algorithm	Workflows	Reducing makespan
[107]	Cloud	Makespan	Scalable Heterogenous Earliest-Finish-Time Algorithm	Workflows	Reducing makespan
[96]	Cloud	Dynamic priority	Dynamic Priority Scheduling Algorithm	Composite services	High QoS and high throughput
[130]	Cloud	Cost	Improved Cost-based Algorithm	Various tasks	Low cost
[161]	Cloud	Categorizing tasks	Utility-Based Job Scheduling Algorithm	Various tasks	Long-term optimization
[160]	Cloud	Multiple QoS	Multiple QoS Constrained Scheduling Strategy	Multiple workflows	High QoS and Reduced Makespan
[99]	Cloud	Load balancing	Load Balancing Ant Colony Optimization	Various tasks	Systematic load balance
[149]	Cloud	Cost	Pareto optimality theory	Various tasks	Minimized cost
[98]	Cloud	Energy saving	Green Scheduling Algorithm	Various tasks	Energy saving
[109]	Cloud	Cost and makespan	Innovative Transaction Intensive Cost-constraint Scheduling Algorithm	Workflows	Reduced cost
[156]	Cloud	Market oriented	Market-Oriented-Hierarchical Scheduling	Workflows	Reduced makespan and minimized cost
[148]	Cluster	QoS	Optimal Workflow-based Scheduling Algorithm	Workflows	High resource utilization

Resource-Aware-Scheduling Algorithm (RASA):

Saeed et al. [126] recommended an RASA that consists of two conventional scheduling strategies: Max-Min and Min-Min. RASA uses the advantages of the Max-Min and Min-Min strategies to cover each others' risk. However, they did not introduce a deadline for each workflow, the arriving rate of the workflows, the cost of the workflow execution on an individual resource and the cost of the communication. The outcome of the simulation demonstrated that RASA outperforms existing scheduling algorithms in large-scale distributed systems.

Genetic Simulated Annealing Algorithm:

In [63], Guoning et al. illustrated a genetic simulated annealing algorithm for cloud computing and its implementation. This algorithm considers the QoS requirements of different types of tasks for dimensionless QoS parameters. The algorithm efficiently completes task scheduling in a cloud computing environment.

Heterogeneous-Earliest-Finish-Time Algorithm(HEFT):

Topcuoglu et al., in [146], proposed the HEFT algorithm, which calculates the average execution time for each task and the average communication time between resources of two successive tasks. The jobs in the workflow are ordered along a rank function. The task with the higher social status value is given higher precedence. In the resource selection phase, tasks are assigned to the resource that can complete the task at the earliest time.

Scalable Heterogeneous Earliest-Finish-Time Algorithm (SHEFT):

In [107], Cui et al., proposed a SHEFT workflow scheduling algorithm to schedule a workflow elastically on a cloud computing environment. The results show that SHEFT not only enables several representative workflow scheduling algorithms for optimizing the workflow execution time but also allows resources to scale elastically at runtime.

Dynamic Priority Scheduling Algorithm (DPSA):

In [96], Zhongyuan et al. proposed a new strategy by introducing a dynamic priority to solve the scheduling of composite service applications in a cloud computing environment. The algorithm is based on the Static Priority Scheduling Algorithm (SPSA) and possesses a dynamic feature. They conducted a simulation to provide a comparison of DPSA with First Come, First Served (FCFS) and SPSA algorithms. The results illustrated that

the DPSA method provides greater efficiency than does FCFS and is more feasible than SPSA.

Improved Cost-based Algorithm for Task Scheduling:

In [130], Selvarani et al. proposed an improved cost-based scheduling algorithm for providing the efficient mapping of tasks to available resources in a cloud computing environment. The improvisation of traditional activity-based costing was proposed based on a new task scheduling strategy for cloud environments in which there is no relationship between the overhead of the application base and the different tasks that cause them. This scheduling algorithm divides all user tasks depending on the priority of each task into different lists. This scheduling algorithm measures both resource cost and computation performance. It also improves the computation/communication ratio.

Utility-based Job Scheduling Algorithm:

In [161] Bo et al. proposed a solution based on the utility computing framework, which characterizes each job using an economic attribute called utility and configures this utility following a Time Utility Function (TUF). Meanwhile, the failure rate and recovery rate of the computing entity are introduced into the original model. The algorithm is based on the Reinforcement Learning (RL) framework, which considers the long-term optimization of the scheduling algorithm. It is fault aware as well as productive. The experimental results demonstrated that the RL-based scheduling algorithm is more productive in terms of its utility when compared with the resource-constrained Utility Accrual Algorithm, Utility Accrual Packet Scheduling Algorithm and Locke's Best Effort Scheduling Algorithm (LBESA).

Multiple QoS Constrained Scheduling Strategy of Multi-Workflows (MQMW):

In [160], Meng et al. worked on multiple workflows and multiple QoSs. They delivered a scheme implemented for multiple workflow management systems with multiple QoSs. The scheduling access rate is increased by utilizing this strategy. This strategy minimizes the makespan cost of workflows for a cloud computing platform.

Load Balancing Ant Colony Optimization (LBACO):

In [99], Kun et al. applied a LBACO algorithm to facilitate load balancing for task scheduling. They conducted various experiments, with the number of tasks varying from 100 to 500. The results demonstrated that the LBACO effectively balances the load of

the entire system, irrespective of the task size, and that LBACO handles all conditions while outperforming the FCFS and ACO algorithms in a cloud computing environment.

Pareto optimality theory:

In [149], Hao et al. proposed a scheduling strategy for cloud computing based on the Pareto Optimality $M \times N$ production model. This model considers price models, usage policies and changes in the load. The resource providers and resource consumers have different requirements. Because the availability of resources and the load on them dynamically vary over time, an economic method is presented to allocate the cloud resources. This algorithm can avoid wastage of resources and can achieve equilibrium between maximizing resource providers' incomes and minimizing the cost to consumers. The model in this work is a cloud bank model.

Green Scheduling Algorithm:

In [98], Duy et al. developed an energy-oriented service in a cloud computing environment. They proposed a green scheduling algorithm that obtains the number of the server for each service first. In other words, a dynamic programming strategy that defines how to assign the users to different servers is discussed. A random dynamic scheduling scheme is applied to address the demand uncertainty based on Monte Carlo sampling. Because the user demands are known, the proposed random dynamic scheduling scheme possesses a desirable property for sample estimation.

Innovative Transaction Intensive Cost-constraint Scheduling Algorithm:

In [109], Liu et al. proposed a scheduling algorithm that takes the cost and time for instance-intensive cost-constrained cloud workflows with a large number of workflow instances (i.e., instance-intensive) bounded by a certain budget for execution (i.e., cost constrained) in a cloud computing environment. A simulation demonstrated that this algorithm can achieve lower costs compared to other methods while satisfying user-designated deadlines. The work presented a novel compromised-time-cost scheduling algorithm that considers the characteristics of cloud computing to accommodate instance-intensive cost-constrained workflows by providing a compromise between execution time and cost with user input provided on the fly. It was shown that the mean execution time is reduced by 20% of the user-designated execution cost.

Market-Oriented-Hierarchical Scheduling:

In [156], Zhangjun et al. proposed a market-oriented hierarchical scheduling strategy

for cloud workflow systems. This is specific to service-level scheduling, where the tasks of individual workflow instances are mapped to cloud services in global cloud markets based on their functional and non-functional QoS requirements. Task-level scheduling addresses the optimization of the Task-to-Virtual Machine (VM) assignment in local cloud data centers, where the overall running cost of the cloud workflow systems are to be minimized.

Optimal Workflow based Scheduling (OWS) Algorithm:

In [148], Varalakshmi et al. proposed an OWS algorithm that concentrates on satisfying user-preferred QoS parameters such as execution time, reliability and monetary cost. OWS uses the resource discovery algorithm and indexes all the resources, which helps to locate free resources. The scheduling algorithm considers specified QoS parameters as the major factors when scheduling workflows. Utilizing a special metric called the QoS heuristic, the sub-task cluster is routed to its optimal resource. In cases where resources are not available to be assigned a task, compaction is applied to increase CPU availability substantially.

2.6 Discussion

The research works on divisible load theory for distributed computing systems as well as for cloud computing environments address the problem of load scheduling with single sources for facilitating load distribution. In a practical scenario, a virtual distributed computing system has more data sources for load distribution. In addition, the number of processors participating in load processing will change dynamically. Hence, there is a need to develop a new load distribution algorithm under the divisible load theory framework with multiple resources. In addition, the proposed algorithm needs to handle dynamic resources. Similar to current solutions, the bottleneck in minimizing the load processing time is the time to distribute the load. Here, we also address this problem by increasing the number of installments.

The other research works in the literature on load scheduling address the problem of selecting appropriate resources for load scheduling. To select the resources, one needs to know how optimally we can distribute the processing load such that the total load

processing time is minimized. This thesis addresses the problem of load distribution for a given resource from multiple data sources. Further, it also addresses the problem of dynamic load rescheduling based on the availability of processors.

2.7 Summary

In this chapter, we have reviewed a few earlier works on DLT-based scheduling for scheduling in a cloud. First, we introduced the DLT research works on DLT scheduling algorithms based on various network topologies. We discussed research on bus networks, single-level tree networks, multi-level tree networks and hypercube networks. Then, we presented DLT papers that discussed different constraints, such as limited buffers, processor release times, start-up delays and the non-blocking mode of communication, and collection schemes, followed by DLT multi-installment and non-linear DLT methods. We then introduced DLT research works concerning cloud computing system environments. Then, we summarized DLT-based applications such as image processing, video processing, data processing, signal processing, real-time applications and scientific applications. Finally, we introduced some existing optimal scheduling algorithms; however, we cannot compare these algorithms with the proposed algorithms because the difference between the frameworks is in terms of the number of data banks.

Chapter 3

Scheduling in virtual distributed computing systems with multiple data banks

3.1 Introduction

In the previous chapter, we reviewed research works from different categories. In this chapter, we define the virtual distributed computing framework notations and definitions and the proposed DLT-based scheduling algorithm based on the framework. We use these notations and definitions to express the practical constraints used to determine recursive equations. In this way, we transform the load scheduling problem in Virtual Distributed Computing Systems (VDCS) with multiple data banks into a linear programming problem. Then, we demonstrate the proposed scheduling algorithm using four numerical examples, six simulations and one practical set up for a satellite image processing example. The results illustrate that the load scheduling algorithm is accurate when applied to VDCS with multiple data banks.

In a practical virtual distributed computing System environment, the loads and tasks to be processed are available in multiple data centers rather than in a virtual computer interface. The design of a scheduling strategy in a VDCS with heterogeneous, dynamic resources and multiple data centers turns out to be a challenging problem. It should be noted that multiple sources may not be available at the time of load distribution, which might lead to imbalances in resource utilization. In this chapter, we propose a DLT-based load scheduling problem in a VDCS with multiple sources and release times.

In this chapter, we design a load distribution strategy for data-driven computational loads on a heterogeneous VDCS environment, which consists of a virtual resource allocator (“Web role”), a set of virtual computers (“worker roles”), and a set of data storage centers (“data banks”). The worker role in the VDCS is equipped with front-end processors for simultaneous computation and communication (i.e., nonblocking mode of communication) with data banks. The approach described in [54] is used to synchronize the processing data between the data banks without disturbing the load scheduling process. Note that the data banks are not available at the time of load distribution; hence, this will introduce a delay (referred to as the release time) in the load scheduling. The objective of the scheduling strategy is to find an optimal fraction of loads assigned to each processor from multiple data centers to achieve minimum total load processing time.

First, we present the model of VDCS. Next, we present the LP formulation for divisible load scheduling followed by numerical examples and simulation results for a satellite image classification problem in a VDCS environment. The results of this chapter have been published in [137].

3.2 Mathematical modeling of virtual distributed computing system

A VDCS environment, shown in Figure 3.1, consists of heterogeneous or homogeneous virtual computer interfaces (worker roles) and multiple data storage centers (data banks). To better utilize a compute cloud platform, the user submits the request for load scheduling at a virtual resource allocator (Web role). Resource allocation assumes that the data-driven computational load is stored in multiple data banks and that the data can be synchronized effectively without disturbing the load distribution process [54]. Note that the number of data banks is substantially smaller than the number of worker roles available for computation. Once a request is received from a user or an application, the web role divides the computational load and assigns the load fractions to worker roles. Each worker role retrieves the load fractions from the available data banks in installments so that the processing time is minimized. The load size of each installment depends on the mode of communication, the communication link speed, the release time

Table 3.1: Notations and Descriptions

Notations	Descriptions
A_i	Inverse processing speed for the i^{th} worker role W_i .
G_j	Inverse transmitting speed for data bank D_j through a wide area network.
J	Total processing load.
N	Number of data banks present for load scheduling.
M	Number of worker roles present for load scheduling.
R_j	Release time of data bank D_j .
T_i	Data processing time for the i^{th} worker role (W_i).
T_0	Total load processing time.
β_{ij}	The fraction of a load retrieved from the data bank (D_j) by a worker role (W_i).
μ_i	The total fraction of the processing load assigned to a worker role ($W_i, i = 1, 2, \dots, M$).

transmitting speed parameter (G_j) influence the size of the load fractions retrieved from the data bank. The web role partitions the total computational load (J) into M fractions ($\mu_1, \mu_2, \dots, \mu_M$) and assigns them to the worker roles ($W_1, W_2, \dots, W_M$). The worker role (W_i) retrieves its own fraction (μ_i) from N data banks ($\beta_{i1}, \beta_{i2}, \dots, \beta_{iN}$).

3.3 Definitions

The **load distribution** is defined as a M -tuple of the total processing load ($\mu_1, \mu_2, \dots, \mu_M$) assigned to the worker roles ($W_1, W_2, \dots, W_M$) in the virtual distributed computing system. The total processing load is given by Equation 3.1:

$$J = \sum_{i=1}^M \mu_i = \sum_{i=1}^M \sum_{j=1}^N \beta_{ij} \quad (3.1)$$

The **finish time** of the worker role (T_i) is the time interval between the time instant at which the worker role (W_i) stops computing and the time instant at which the worker role initiates the task.

The **processing time** (T_0) is defined as the total time taken by the web role to compute the entire processing load in a virtual distributed computing system, i.e., $T_0 = \max\{T_1, T_2, \dots, T_M\}$. The notations and descriptions used in this chapter are given in Table 3.1.

Figure 3.2: The virtual distributed computing system environment with M worker roles and N data banks

3.4 Problem formulation

The scheduling problem in a distributed computing system environment is presented in Figure 3.2. J represents the total load that arrives at the web role. The web role divides the computing load into smaller fractions $(\mu_1, \mu_2, \dots, \mu_M)$ and assigns the tasks to the worker roles. The worker role retrieves the data from data banks $(D_1, D_2, \dots, D_N)$ and completes the computation process. The main focus points for the load distribution using DLT and all the practical constraints are also based on the worker role computational speed and the bandwidth between worker roles and data banks. Then, for each of the worker roles (W_i) , the data $(\beta_{i1}, \beta_{i2}, \dots, \beta_{iN})$ retrieved are in sequence with the data banks. Now, we formulate the scheduling problem in the VDCS environment.

The load distribution processes performed by the web role and the retrieval sequence of the load fractions used by the worker role are illustrated in the timing diagram shown in Figure 3.3. From the diagram, we can see that worker roles process the data contin-

Figure 3.3: Timing diagram describing the load scheduling process in the virtual distributed computing system environment

uously, and the data are retrieved one by one from different data banks. The worker roles may have time intervals in between and are limited by practical constraints. We can also observe from the diagram that all the worker roles start processing the load simultaneously, and the first bit of the load fraction is received. As we discussed, for very large load scheduling problems, the worker role startup delay compared to the overall processing time is relatively small and can be ignored.

3.4.1 Constraints imposed by total processing time (T_0):

Let T_0 describe the optimized total load processing time. For a minimal computational time, all worker roles stop computing simultaneously, as stated in divisible load scheduling theory [18]. The meaning of this equation is that the release time for data retrieval plus the communication time and computation time should be equal to or less than the total processing time. Hence, from the timing diagram, the total processing time constraint gives us Equation 3.2. This is the sum of the transmission time needed by all the previous worker roles to fetch their respective loads from data banks and the time

taken to compute the load fractions by the worker role. This equation is derived based on the timing diagram given in Figure 3.3.

$$T_0 - R_1 - \sum_{n=1}^{i-1} \beta_{n1} \cdot G_1 - \sum_{j=1}^N \beta_{ij} \cdot A_i \geq 0; i = 1, 2, \dots, M \quad (3.2)$$

3.4.2 Constraints imposed by release time (R_j):

Note that the first worker role retrieves its fractions of the load from the data banks. The release time of the data bank will affect the retrieval of the load by the first worker role. Various factors, such as storage, maintenance, consistency, retrieval by other hosts, and fault recovery by overheating, may affect the availability of a data bank for load retrieval. Processing under this framework, we consider all these items as one group, and the release time will be the reflection of all these items in the framework. This is the constraint necessary for us to solve this problem. If the release time R_{j+1} of data

Figure 3.4: Timing diagram of worker role 1 processing by accessing every adjacent data bank pair.

bank D_{j+1} is higher than the computation of load β_{1j} by worker role W_1 , then the worker role will be idle for some time. In addition to the release time of the data banks, the size of the load retrieval from adjacent data banks (D_j, D_{j+1}) to adjacent worker roles (W_i, W_{i+1}) may result in idle time. To minimize the idle time or maximize the resource

utility, the data bank D_j allocates the work load to the worker role (β_{ij}). To provide a understanding of the load scheduling results, the timing diagram is depicted in Figure 3.4. It shows the distribution process of D_j, D_{j+1} for W_1 with the release time of two data banks. Here, it can be noted that the difference in the release times of two adjacent data banks (D_j, D_{j+1}) influences the component load retrieval by the worker role for computation in Equation 3.3, i.e.,

$$\beta_{1j} \cdot A_1 \geq R_{j+1} - R_j; j = 1, 2, \dots, N - 1; \quad (3.3)$$

3.4.3 Constraints imposed by continuity

The timing diagram depicted in Figure 3.5 describes the continuity constraint on the worker role. The figure shows the time interval of data retrieval and data processing between the adjacent data banks D_j and D_{j+1} and adjacent worker roles W_i and W_{i+1} . From this, we can derive Equation 3.4 below. The figure illustrates the two adjacent worker roles and the two adjacent data banks that are recursive for all pairs of adjacent worker roles and pairs of adjacent data banks. Equation 3.4 is the sum of the processing time of the first worker role load fraction fetched from the first data bank and the transmission time of the first worker role load fraction fetched from the second data bank. This should be equal to or less than the sum of the transmission time for the first worker role to fetch a load fraction from the first data bank and the load processing time of the second worker role fetching a load fraction from the first data bank.

$$\beta_{ij} \cdot A_i + \beta_{i(j+1)} \cdot G_{j+1} \leq \beta_{ij} \cdot G_j + \beta_{(i+1)j} \cdot A_{i+1}; i = 1, 2, \dots, M - 1, j = 1, 2, \dots, N - 1 \quad (3.4)$$

Equation 3.4 applies for two adjacent worker roles and two adjacent data banks and results in $(M - 1) \cdot (N - 1)$ constraint equations. Obtaining a closed-form expression for the load fractions β_{ij} in the divisible load scheduling problem is difficult. Hence, we propose a load scheduling problem with multiple data banks as an optimization problem. The optimization problem is defined as follows: Given the number of worker roles (M), the number of data banks (N), and each of the load retrievals ($W_1, W_2, \dots, W_M$) from the data banks ($D_1, D_2, \dots, D_N$), find the load fractions assigned to each worker role from each data bank that minimize the total load processing time.

Figure 3.5: Timing diagram of each adjacent worker role pair processing by accessing each adjacent data bank pair.

3.4.4 Constraints imposed by total load

For the load fractions $(\mu_1, \mu_2, \dots, \mu_M)$ assigned to all the worker roles for processing and the sum of the load fractions $(\beta_{11}, \beta_{12}, \dots, \beta_{1N})$, ..., $(\beta_{M1}, \beta_{M2}, \dots, \beta_{MN})$ for all the worker roles retrieved from each data bank should be equal to the total load size. The scheduling task is to partition the load assigned to each worker role properly and finally achieve the minimum total processing time. The total processing load is given by Equation 3.1.

3.4.5 Linear programming formulation

For the load fractions $(\mu_1, \mu_2, \dots, \mu_M)$ assigned to all the worker roles for processing, the sum of load fractions $(\beta_{11}, \beta_{12}, \dots, \beta_{1N})$, ..., $(\beta_{M1}, \beta_{M2}, \dots, \beta_{MN})$ for all the worker roles retrieved from each data bank should be equal to the total load size. The scheduling task is to partition the load properly assigned to each worker role and finally achieve the minimum total processing time. The total processing load is given by Equation 3.1.

Minimize T_0 such that

Total processing time constraint:

$$T_0 - R_1 - \sum_{n=1}^{i-1} \beta_{n1} \cdot G_1 - \sum_{j=1}^N \beta_{ij} \cdot A_i \geq 0; i = 1, 2, \dots, M$$

Release time constraint:

$$\beta_{1j} \cdot A_1 \geq R_{j+1} - R_j; j = 1, 2, \dots, N - 1$$

Total load constraint:

$$J = \sum_{i=1}^M \mu_i = \sum_{i=1}^M \sum_{j=1}^N \beta_{ij}; i = 1, 2, \dots, M, j = 1, 2, \dots, N$$

Continuity constraint:

$$\beta_{ij} \cdot A_i + \beta_{i(j+1)} \cdot G_{j+1} \leq \beta_{ij} \cdot G_j + \beta_{(i+1)j} \cdot A_{i+1}; i = 1, 2, \dots, M - 1, j = 1, 2, \dots, N - 1$$

$$\beta_{ij} \geq 0; i = 1, 2, \dots, M, j = 1, 2, \dots, N$$

This linear programming problem has $(M \times N + 1)$ variables and $(M \times N + 1)$ constraints. The variables in our problem are the processing time (T_0) and the load fractions $(\beta_{11}, \beta_{12}, \dots, \beta_{1N}) \dots, (\beta_{M1}, \beta_{M2}, \dots, \beta_{MN})$. The solution to this problem is found in an $(M \times N + 1)$ -dimensional space. We use the Matlab *v7* `linprog()` function to solve the LP problem. For the details of the function, refer to [43, 115, 170].

3.5 Numerical examples

We provide four numerical examples to provide a better understanding of the proposed scheduling strategy for a VDCS environment using a linear programming formulation.

3.5.1 Numerical example 1

In numerical example 1, we consider a virtual distributed computing environment with three worker roles and two data banks. The inverse processing speed parameters of the worker roles are $A_1 = 4$, $A_2 = 5$, and $A_3 = 6$; the inverse transmitting speed parameters of the data banks are $G_1 = 0.2$ and $G_2 = 0.4$; and the release times of the data banks are $R_1 = 10$ and $R_2 = 50$. Without loss of generality, we assume that the entire processing load ($J = 100$) is stored in the data banks before the start of the computation process. All worker roles that stop computing simultaneously impose three inequality constraints,

and the release time of the data banks imposes one inequality constraint. The total load processing and continuity of the worker role computation imposes one equality constraint and two inequality constraints. The objective is to find the load fractions assigned to each worker role from the data banks that ensure that the total load processing time is minimized. Hence, the load scheduling problem in a VDCS can be formulated as a linear programming problem, as given in the following:

$$\begin{aligned}
 &\text{Minimize } T_0 \\
 &\text{Such that} \\
 &\text{Total processing time constraint:} \\
 &T_0 - R_1 - (\beta_{11} + \beta_{12}) \cdot A_1 \geq 0 \\
 &T_0 - R_1 - \beta_{11} \cdot G_1 - (\beta_{21} + \beta_{22}) \cdot A_2 \geq 0 \\
 &T_0 - R_1 - (\beta_{11} + \beta_{21}) \cdot G_1 - (\beta_{31} + \beta_{32}) \cdot A_3 \geq 0 \\
 &\text{Release time constraint:} \\
 &\beta_{11} \cdot A_1 \geq R_2 - R_1 \\
 &\text{Total load constraint:} \\
 &\beta_{11} + \beta_{12} + \beta_{21} + \beta_{22} + \beta_{31} + \beta_{32} = J \\
 &\text{Continuity constraint:} \\
 &\beta_{11} \cdot A_1 + \beta_{12} \cdot G_2 \leq \beta_{11} \cdot G_1 + \beta_{21} \cdot A_2 \\
 &\beta_{21} \cdot A_2 + \beta_{22} \cdot G_2 \leq \beta_{21} \cdot G_1 + \beta_{31} \cdot A_3
 \end{aligned}$$

Table 3.2: Load Fractions and Processing Time after Scheduling for Numerical Example 1

Total processing time: $T_0 = 173.9$			
D_j, W_i	W_1	W_2	W_3
D_1	$\beta_{11} = 10$	$\beta_{21} = 10.078$	$\beta_{31} = 9.5491$
D_2	$\beta_{12} = 30.974$	$\beta_{22} = 22.301$	$\beta_{32} = 17.098$

Based on the previous formulas, the load fractions assigned to each worker role from each data bank and the total load processing time are given in Table 3.2. To provide a better understanding of the results, we illustrate the process of load retrieval from each

Figure 3.6: Timing diagram for numerical example 1

data bank to three worker roles and the corresponding computation time in the timing diagram in Figure 3.6. From the figure, we can see that the worker role computation is continuous and that all worker roles stop computing simultaneously.

3.5.2 Numerical example 2

To study the effect of the release time, we consider a virtual distributed computing environment with three homogeneous worker roles and two data banks. The inverse processing speed parameters of the worker roles are $A_1 = 1.1$, $A_2 = 1.1$, and $A_3 = 1.1$; the inverse transmitting speed parameters of the data banks are $G_1 = 0.2$ and $G_2 = 0.4$; and the release times of the data banks are $R_1 = 10$ and $R_2 = 10$. For this problem, we also assume that the entire processing load ($J = 100$) is stored in data banks before the start of the computation process. Note that the release time constraint given in Equation 3.3 reduces to $\beta_{11} = 0$. Based on the formulas, the load fractions assigned to each worker role from each data bank and the total load processing time are $\beta_{11} = 0$, $\beta_{12} = 19.8198$, $\beta_{21} = 19.8198$, $\beta_{22} = 36.3364$, $\beta_{31} = 16.5165$, and $\beta_{32} = 7.5075$. The total load processing time is 43.97. In Table 3.3, based on the results, we can see that the first worker role does not receive any load fractions from the first data bank.

Table 3.3: Load Fractions and Processing Time after Scheduling for Numerical Example 2

Total processing time : $T_0 = 43.97$			
D_j, W_i	W_1	W_2	W_3
D_1	$\beta_{11} = 0$	$\beta_{21} = 19.8198$	$\beta_{31} = 16.5165$
D_2	$\beta_{12} = 19.8198$	$\beta_{22} = 36.3364$	$\beta_{32} = 7.5075$

3.5.3 Numerical example 3

To study the sequence for worker roles and the sequence for data banks in heterogeneous VDCS, we consider a compute cloud environment with three heterogeneous worker roles and two data banks. The parameters are the same as in numerical example 1. The inverse processing speeds of the worker roles are $A_1 = 4$, $A_2 = 5$, and $A_3 = 6$; the inverse transmitting speed parameters of the data banks are $G_1 = 0.2$ and $G_2 = 0.4$; and the release times of the data banks are $R_1 = 10$ and $R_2 = 50$. Without loss of generality, we assume that the entire processing load ($J = 100$) is stored in data banks before the start of the computation process. We present the total processing times for 6 different combination sequences of worker roles in Table 3.4. We also show 2 different combination sequences of data banks in this case in Table 3.5. It is proved in the DLT literature that the sequence of the distribution influences the processing time significantly and is a solution for load scheduling in a single installment for tree, bus, and linear networks [18]. The presence of both multiple fractions of load retrieval from different data banks and release times of data banks in a cloud environment complicates the optimal sequencing problem. To study the effect of the sequence of the load retrieval from data banks, we conducted experiments (case 7) in which the worker roles start retrieving the load from data bank 2 and then retrieve the load fraction from data bank 1. The order of the load distribution remains the same as in case 1. Note that the release time of the data banks is modified as $R_1 = 50$ and $R_2 = 10$. The total load processing time increases to 175.45. Note that the release time of the data banks also influences the load retrieval sequence. One should use data banks with smaller release times first; if two data banks have similar release times, then the data bank with the higher communication speed is used first for load retrieval. The problem of finding an optimal load retrieval sequence in the presence of multiple data banks with release times is NP-hard, and in [165], the mentioned multi-installment DLT scheduling is also NP-hard.

- Case 1: $(W_1, W_2, W_3), (D_1, D_2)$
- Case 2: $(W_1, W_3, W_2), (D_1, D_2)$
- Case 3: $(W_2, W_1, W_3), (D_1, D_2)$
- Case 4: $(W_2, W_3, W_1), (D_1, D_2)$
- Case 5: $(W_3, W_1, W_2), (D_1, D_2)$
- Case 6: $(W_3, W_2, W_1), (D_1, D_2)$
- Case 7: $(W_1, W_3, W_2), (D_2, D_1)$

Table 3.4: Processing Time after Scheduling in the 6 Worker Role Sequence Cases for Numerical Example 3

Case Number	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6
Total Processing time(T_0)	173.9	173.9	173.9	173.9	173.9	173.9

Table 3.5: Processing Time after Scheduling in 2 Data Bank Sequence Cases for Numerical Example 3

Case Number	Case 1	Case 7
Total Processing time(T_0)	173.9	175.45

Table 3.6: Processing Time after Scheduling in 4- and 5-Worker-Role Cases for Numerical Example 4

Case Number	Case 1	Case 2
Total Processing time(T_0)	184.3	156.23

3.5.4 Numerical example 4

To study the number of worker roles and the total processing time in heterogeneous virtual distributed systems, we consider a VDCS environment with four heterogeneous worker roles and two data banks. The inverse processing speeds of the worker roles are $A_1 = 6$, $A_2 = 7$, $A_3 = 8$, $A_4 = 9$ and $A_5 = 10$; the inverse transmitting speed parameters of the data banks are $G_1 = 0.2$ and $G_2 = 0.4$; and the release times of the data banks are $R_1 = 0$ and $R_2 = 20$. Without loss of generality, we assume that the entire processing load ($J = 100$) is stored in data banks before the start of the computation process. The results are presented in Table 3.6, and the timing diagrams for these two cases are given in Figure 3.7 and Figure 3.8. From the results of this example, we can say that an increasing number of heterogeneous worker roles helps decrease the total processing time.

- Case 1: $(W_1, W_2, W_3, W_4), (D_1, D_2)$
- Case 2: $(W_1, W_2, W_3, W_4, W_5), (D_1, D_2)$

3.6 Performance evaluation and discussion

In this section, we use the Matlab environment to run all the simulations to verify the proposed DLT-based load scheduling algorithm. In our simulation, we consider a VDCS with 10 worker roles and five different data banks for a satellite image processing application scenario. We generate random values for the worker roles for the inverse processing speeds, the inverse transmitting speeds of the data banks, and the release times of the data banks to guarantee that the computation time is shorter than the communication

Figure 3.7: Timing diagram in 4-worker-role, 2-data-bank case for numerical example 4

time. These parameters are given in Table 3.7. The computational load arrives at the web role, and the web role identifies the worker roles and the data banks for load scheduling. The worker roles receive data from each data bank and stop computing simultaneously.

The performance of the VDCS with different numbers of data banks is evaluated for a heterogeneous and homogeneous worker role. In all our studies, we use the total load processing time as a performance metric. First, we present the performance using heterogeneous worker roles in the VDCS environment.

Figure 3.9 presents the processing time versus the number of worker roles for different numbers of data banks in a VDCS. From the figure, we can see that the processing time decreases with increasing number of worker roles for a given number of data banks. Beyond a certain number of worker roles, the total load processing time does not decrease further. From Figure 3.9, we can also see that the processing time decreases significantly when we increase the number of data banks under a fixed number of worker roles. For example, three worker roles with two data banks require 203.13 time units to process the total load, whereas three worker roles with three data banks require only 186.79 time

Figure 3.8: Timing diagram in 5-worker-role, 2-data-bank case for numerical example 4

Figure 3.9: Processing time versus number of heterogeneous worker roles in the virtual distributed computing system environment

Figure 3.10: Processing time versus number of homogeneous worker roles in the virtual distributed computing system environment

units. We can also observe from the figure that, above a certain number of data banks, the processing time does not decrease significantly.

Table 3.7: Virtual Distributed Computing System Simulation Parameters

Data Banks		Worker Roles
Release Time (R_j)	Inverse Transmitting Speed (G_j)	Inverse Processing Speed (A_i)
(4, 5, 6, 7, 8)	(0.5, 0.6, 0.7, 0.8, 0.9)	(1, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9)

Now, we present the processing time for different numbers of homogeneous worker roles and heterogeneous data banks. For our simulation study, we used worker roles with the computation speed parameter as 1.1, and the remainder of the parameters remain as given in Table 3.7. The total load processing time versus the number of worker roles for different numbers of data banks is shown in Figure 3.10. From the figure, we can observe that the total processing time decreases with increasing number of worker roles and data banks. The reduction in the total load processing time becomes smaller above a certain number of worker roles and data banks.

Further, we conducted experiments by varying the inverse transmitting speed parameter (G) of the homogeneous data banks as 0.5, 0.6, 0.7, and 0.8. The remainder of the parameters are held constant as in the earlier homogeneous scenario, and the number of

Figure 3.11: Processing time versus number of homogeneous worker roles for different inverse transmitting speed parameters.

data banks is kept at two. The processing times for different numbers of worker roles for different inverse transmitting speed parameters are shown in Figure 3.11. From the figure, we can see that the reduction in processing time decreases with increasing data bank inverse transmitting speed parameter. In addition, the rate of decrease in the processing time with increasing number of processors decreases with increasing inverse transmitting speed parameter. To study the effect of the total processing load (J), we conduct a study by varying the processing load in a homogeneous environment with two data banks. The parameters are set as $G = 0.5$, $A = 1.1$, $R_1 = 3$, and $R_2 = 4$, respectively. The processing time variation for various numbers of worker roles with different total processing loads is shown in Figure 3.12. From the figure, we can see that increasing the total processing load increases the processing time. To discuss further, the proposed scheme in each key parameter simulation has been applied in a homogeneous distributed computing system environment. For different R_j parameters, the values are given as

Figure 3.12: Processing time versus number of homogeneous worker roles for different processing load sizes.

$R_1 = 5, R_2 = 6, 10, 15, 20$; the number of worker roles is $M = 2, 3, 4, 5, 6, 7, 8, 9, 10$; and the number of data banks is $N = 2$. The total load is 100 units. From the simulation results shown in the Figure 3.13, it can be inferred that increasing the release time interval increases the total processing time, and larger intervals between release times increase the total processing time for the minimum number of worker roles significantly.

Now, we conduct a study on the computational complexity of the proposed approach. To this end, we consider a homogeneous VDCS environment with an inverse transmitting speed parameter (G) for all links of 0.1 and an inverse processing speed parameter (A) for all worker roles of 1.1. The total load (J) is 100. The number of worker roles (M) varies between 100 and 500, and the number of data banks (N) is kept as 20 and 30, respectively. The release time of the data banks is initialized appropriately (in steps of 10). The time taken to compute the respective load fraction from each data bank to each worker role is plotted against the number of worker roles in Figure 3.14. From the figure,

Figure 3.13: Processing time versus number of homogeneous worker roles in the homogeneous virtual distributed computing system environment for R_j set

we can see that the total processing time increases with increasing number of worker roles and data banks; however, the execution time is on the order of a few seconds.

3.7 Satellite image processing application

For the experimental study, we developed a virtual cloud in a laboratory environment with two servers acting as a data bank and six work stations acting as a worker role for data processing. The HP servers and workstations are running in an *HPUX11v3* operational environment connected through 10 Gbps Ethernet links. For the experimental study, we consider a satellite image classification problem [141]. A high-resolution multi-spectral Landsat 7 Thematic Mapper image portion covering $50 \times 50.75 km^2$ is considered for the study. The spatial resolution of the images is 30 m, and each pixel contains seven spectral frequencies and is used as an input feature to online learning neural classifiers.

Figure 3.14: Processing time versus number of homogeneous worker roles in the virtual distributed computing system environment

The worker roles contain a classifier model developed using an online sequential learning classifier [10] and are expected to estimate the class label for a given pixel in a portion of the satellite image. For additional details about the data set, one should refer to [141].

The communication parameters are estimated by averaging the time taken to send 1000000 pixels (i.e., 7000000 floating point values) between the data banks and workstations 100 times. Similarly, the computation parameter is estimated by averaging the computational time taken by the workstations to process the unit load 100 times. The inverse transmitting speed (G_j) and inverse processing speed (A_i) parameters are 0.7 and 4.9, respectively. The release times for the data banks (D_1 and D_2) are assumed to be 10 and 20 s, respectively. The total number of pixels in the image region is 84583000, and one unit is 106 pixels. These parameters are used in our numerical example and experimental study. All the pixels are divided into smaller fractions, as obtained from our formulation, and the load fractions are rounded to the nearest integer. The workstations retrieve these

pixels from the data banks and classify them into respective classes. The experimental processing and analytical times taken to classify the satellite image with two data banks and three worker roles are 174.18 and 144.85 s, respectively. The load fractions assigned to the worker role from each data bank are 2.1277, 5.7672, 8.2753, 26.564, 22.608, 19.241. The difference between the experimental and analytical times is due to overhead in communication, data reformatting, synchronization, etc.

Figure 3.15: Analytical and experimental processing time for satellite image processing problem in the virtual distributed computing environment.

The analytical and experimental times taken to process the satellite image for different numbers of worker roles are shown in Figure 3.15. From the figure, we can see that the processing time decreases with increasing number of worker roles. As mentioned earlier, the analytical time is smaller than the experimental time due to communication and computation components excluded in the formulation.

3.8 Summary

In this chapter, we presented a scheduling algorithm to compute in a VDCS environment with multiple data banks. First, the problem of load scheduling in a VDCS was formulated under a divisible load scheduling framework. Next, we converted the recursive equations and continuity of processing into constraints and formulated the scheduling problem as a linear programming problem. A linear programming technique has two advantages: it can make the best possible use of available productive resources, and it can solve complex problems by breaking them down into solvable parts. Finally, we presented a numerical satellite image processing example, and the experimental simulation results highlighted the advantage of the proposed solution. In addition, we conducted a simulation study to understand the computational complexity of the proposed approach, and the results clearly highlight that the complexity is on the order of minutes.

In the next chapter, we focus on the DLT-based rescheduling algorithm.

Chapter 4

Dynamic resource handling in virtual distributed computing systems

4.1 Introduction

In the previous chapter, we proposed a DLT-based scheduling algorithm for Virtual Distributed Computing Systems (VDCS) with multiple data banks. This chapter focuses on the development of the DLT-based rescheduling algorithm for VDCS with multiple data banks. We will introduce the new definitions and notations for representing the rescheduling start system status and unknowns, followed by analysis of the timing diagram and formulation of new equations based on the new linear programming problem for VDCS dynamic rescheduling. Considering the rescheduling of the remaining load as the worker role's release time, the equations are formed based on three individual cases. We use 5 numerical examples to demonstrate that our rescheduling scheme performs properly in the different cases for the assigned resources as a task undergoes a change.

4.2 Problem formulation

A VDCS environment consists of heterogeneous or homogeneous virtual computer interfaces ("worker roles") and multiple data storage centers ("data banks"). To achieve better utilization of a compute cloud platform's dynamic resource allocation feature, relative rescheduling of the mathematical model is to be extended based on our previous work to a compute cloud model with multiple data banks [137]. The assumptions for the previous model available for the computation can be considered as using a small number of data

banks in comparison to the number of worker roles, and the number of worker roles and data banks can be equal; however, the number of worker roles cannot be less than the number of data banks.

Once a request is received from a user or application, which is the first round of scheduling, the virtual resource allocator (“Web role”) divides the computational load and assigns the load fractions to worker roles. Each worker role retrieves the load fractions from the available data banks in multiple installments such that the processing time is minimized. The installment size depends on the mode of communication, the communication link speed, the release time of the data banks, and the computing power of the worker roles. The worker role in the VDCS is equipped with front-end processors for simultaneous computation and communication (i.e., non-blocking mode of communication) with data banks.

Rescheduling is triggered when new resources are available or if there is a failure of an assigned resource for a given task. The resources can be worker roles and data banks. In the case of resource addition, the scheduling algorithm needs to ensure optimal processing time. In the other case of resource failure, the algorithm needs to guarantee the completion of the task. However, the frequency of rescheduling depends on the remaining load fraction and the reduction in processing time when we add a new resource. The web role computes the performance increment and allows for the addition of new resources. However, rescheduling is necessary when a processor drops out of the computation process.

In our analysis, the distributed computing system is considered to have M heterogeneous computational worker roles ($W_1, W_2, \dots, W_M$) and N data banks ($D_1, D_2, \dots, D_N$). Note that $N \ll M$ in practice. The worker role accesses the data banks to retrieve their respective load fractions one by one. Note that the data banks may not be available at the beginning of the load distribution. The delay in data bank availability for load retrieval is called the release time. To improve the utilization rate, we assume that the data banks do not occupy the whole processing time; they are going to be released after the last worker role completely fetches the last fraction of load from the data banks. In our formulation, we assume that the web role assigns the sequence of the load distribution to $(W_1, W_2, \dots, W_M)$, and each worker role retrieves their portion of the loads from

the data banks in the order of $(D_1, D_2, \dots, D_N)$. The release time of the data banks (R_j) and the inverse transmitting speed parameter (G_j) influences the size of the load fractions retrieved from the data bank (D_j) . The web role partitions the total computational load (J) into M fractions $(\mu_1, \mu_2, \dots, \mu_M)$ and assigns them to the worker roles $(W_1, W_2, \dots, W_M)$. The worker role (W_i) retrieves its own fractions $(\beta_{i1}, \beta_{i2}, \dots, \beta_{iN})$ from the N data banks. For additional details, refer to Chapter 3.

During the processing of a task, a newly available resource set is assigned to the task. The web role must stop processing until the current transmission is completed and gather the current system status again. Following this, based on the current status, the web role must complete the sorting of resources and reschedule the remaining load to each worker role. The load segment size that is available for the task needs to be retrieved from individual data banks. The assumption here is that every data bank is made available as soon as the last load segment is transmitted to the worker role. For rescheduling few resources, both worker roles and data banks can collaborate on the task. Once the resource completes its part of the work, it leaves the task resource pool. This approach is highly practical for a VDCS environment. Further discussion on the current system status, new notations and definitions involved are as follows:

For the rescheduling model, the time at which new resources are assigned to a web role for the current running task is T_R . The rescheduling starts at this point in time. From T_R , if there is an ongoing transmission, the current transmission task needs to be completed. From T_R , if there is no newly started transmission, then the next transmission start time is called T_{RS} for the first round of scheduling, which represents the rescheduling round redistribution. The start times R'_j and T'_0 respectively represent the rescheduling round's release time for each data bank and the total processing time, which counts from the rescheduling start time (T_{RS}) . On the other hand, the difference in the total load for the task and the load that has been processed is denoted as the remaining load (J') for rescheduling. Sometimes, at the point of the rescheduling start time, a given worker role processes the load assigned from the last round, and the worker role release time (R'_{wi}) has been introduced into our formulae. The worker role release time can be zero for those worker roles without a load assignment. δ represents the number of newly joined or dropped worker roles. θ refers to the number of first worker roles with full load (μ_i) fetched

Table 4.1: Rescheduling Notations and Descriptions

Notations	Descriptions
k	The rescheduling round number of first worker roles with loads after sorting.
θ	The rescheduling round number of first worker roles with full loads (μ_i).
T_{RS}	Rescheduling start time.
T_R	The time a new resource is claimed as available for the task.
R'_i	The rescheduling round release time related to the rescheduling start time.
T'_0	Rescheduling total processing time.
J'	The remaining load requiring rescheduling distributed to the updated set of resources.
R'_{wi}	Worker role load rescheduling release time based on new rescheduling start time.
N'	Rescheduling round number of data banks.
δ	Number of newly available or failed worker roles for this task after T_R .
A'_i	Inverse processing speed of worker role W'_i .
G'_j	Inverse transmitting speed to the data bank D'_j through a wide area network.
β'_{ij}	The fraction of loads retrieved from the data bank (D'_j) by a worker role (W'_i).
T'_i	Data processing time for the i -th worker role (W'_i).

completely from the data banks in the rescheduling round. N' is the number of current data banks assigned for rescheduling. The assumption concerning data banks is that, once the load fractions have been sent from the data bank to the last worker role, the data bank resource is released from the task immediately. The VDCS is considered to have $M + \delta$ heterogeneous computational worker roles $\{W'_1, W'_2, W'_3, \dots, W'_M, W'_{M+1}, \dots, W'_{M+\delta}\}$, and each worker role retrieves their portion of loads from the data banks in the order of $\{D'_1, D'_2, \dots, D'_{N'}\}$. The notations for the release time of the data banks (R'_j), the inverse transmitting speed (G'_i) and the work role inverse processing speed (A'_i) are used. Similar to the first round of scheduling, the web role partitions the total remaining load (J') into $M + \delta$ fractions $\{\mu'_1, \mu'_2, \mu'_3, \dots, \mu'_M, \mu'_{M+1}, \dots, \mu'_{M+\delta}\}$, and the worker role (W'_i) retrieves the load fraction (μ'_i) from N' data banks $\{\beta'_{i1}, \beta'_{i2}, \dots, \beta'_{iN'}\}$. The rescheduling notations and descriptions are listed in Table 4.1.

The rescheduling problem in a VDCS environment is captured in Figure 4.1. The web role obtains the new resources for currently running tasks; then, the web role collects the current rescheduling task resources pool and process status. According to the current status, the web role will calculate the rescheduling start time and sort the worker roles based on the remaining load that needs to be processed. The newly joined worker role or the roles without a remaining load have a high priority in the rescheduling round because they can perform the process at the rescheduling start time. The worker roles

with remaining loads are sorted from the bottom based on the worker role release time. The longer the worker role release time, the further along the queue it is for rescheduling. From the data banks' point of view, some of the data banks may be released from the task according to the rescheduling restart time (T_{RS}), and there might be new data banks joining the task for rescheduling as well. Leveraging our previous scheduling scheme [137], the rescheduling problem in the VDCS environment is as follows.

Figure 4.1: Load rescheduling for the virtual distributed computing system.

After the web role collects the current status, it will reschedule the total remaining compute load into smaller fractions and assign the loads to the worker roles by considering the remaining load as a worker role release time (R'_{wi}). It is to be noted that worker roles retrieve the load from data banks and complete the computation process at the same

time (T'_0). Therefore, we obtain Eqn. 4.1.

$$T'_0 - R'_{wi} - \sum_{j=1}^{N'} \beta'_{ij} \cdot A'_i \geq 0; i = k, \dots, M + \delta \quad (4.1)$$

This formula illustrates that the worker role release time is considered. The reason for performing a rescheduling is to assign the remaining load to the newly available resource set to decrease the total processing time in an additional resource scenario. In a resource failure scenario, this method guarantees task completion and minimizes total processing time. The total processing time for this worker role is the worker role release time plus the time for processing the load assigned during rescheduling. An overview of the rescheduling timing diagram is depicted in Figure 4.2.

Figure 4.2: Timing diagram describing the load rescheduling process in the virtual distributed computing system environment.

Form Figure 4.2, the load transmission for previous rounds of the scheduling will be cut exactly at the rescheduling start time (T_{RS}). This is illustrated in the worker role

(W_1) for the transmission load from data bank 2 ($\beta_{12} \times G_2$), with the previous round marked in red color. In addition, Figure 4.2 illustrates that the worker role without a load will be put at the front of the queue with the highest priority for rescheduling.

For a worker role with a load, the continuity constraint for the load fraction that they fetch from the first data bank (D'_1) is given by the following equations:

For $k = 1$, Equation 4.2:

$$(R'_{W(i+1)} - R'_{Wi}) + \beta'_{(i+1)1} \cdot A'_{i+1} \leq \beta'_{i1} \cdot A'_i + \beta'_{i2} \cdot G'_2; i = k, \dots, M + \delta - 1, k = 1 \quad (4.2)$$

For $k \neq 1$, Equation 4.3:

$$(max(R'_{W(i+1)}, (R'_1 + \sum_{n=1}^{k-1} \beta'_{n1} \cdot G'_1)) - max(R'_{Wi}, (R'_1 + \sum_{n=1}^{k-2} \beta'_{n1} \cdot G'_1))) + \beta'_{(i+1)1} \cdot A'_{i+1} \leq \beta'_{i1} \cdot A'_i + \beta'_{i2} \cdot G'_2; i = k - 1, \dots, (M + \delta - 1), j = 1, k \neq 1 \quad (4.3)$$

For the worker role drop case and for all the worker roles obtaining load fractions, Equation 4.2 is used. This is specific to the worker roles that fetch data from the data bank (D'_1). In the worker role failure case with worker roles without loads and for the case of adding a new worker role, Equation 4.3 should be applied. However, after we obtain the result, because $(R'_1 + \sum_{n=1}^{k-1} \beta'_{n1} \cdot G'_1)$ includes an unknown ($\beta'_{n1}; n = 1, \dots, k - 1$), we need to check if $(R'_1 + \sum_{n=1}^{k-1} \beta'_{n1} \cdot G'_1 \geq R'_{W(i+1)})$. If so, we replace $R'_{W(i+1)}$ with $(R'_1 + \sum_{n=1}^{k-1} \beta'_{n1} \cdot G'_1)$ in Equation 4.2. The same rule applies to R'_{Wi} . In both cases, namely, the worker role ($W'_{k-1}, k \neq 1$) without load in the rescheduling round and the first worker role ($W'_k, k \neq 1$) with load, we should apply Equation 4.3. Here, we consider $(R'_{Wi} = 0, i = k - 1, k \neq 1)$ and directly apply $(R'_1 + \sum_{n=1}^{k-2} \beta'_{n1} \cdot G'_1 \geq R'_{W(i+1)})$ in Equation 4.3.

Although we attempt to achieve continuous transmission because of the need to prevent redundant load transmission, the continuous fetching of loads from data bank D'_1 is not guaranteed in the rescheduling process.

For the rescheduling round, the worker role completely fetches the entire load and assigns load fractions to the other worker roles. This will guarantee that all worker roles stop processing at the same time. We then have Equation 4.4 as the total processing time constraint as follows:

$$T'_0 = T_0 - T_{RS} + \mu'_i \cdot A'_i; i = \theta, \dots, M + \delta \quad (4.4)$$

Figure 4.3: Rescheduling continuity constraints of worker roles with loads.

This formula shows that, in the previous round, the full load fraction is already transferred to the worker roles. Then, in the rescheduling round, a certain load should be dropped according to the new total processing time, and the spare load should be given to other worker roles.

The rescheduling load distribution is defined as a $(M + \delta)$ tuple of the total remaining load $(\mu'_1, \mu'_2, \dots, \mu'_{(M+\delta)})$ assigned to the worker roles $(W'_1, W'_2, \dots, W'_{(M+\delta)})$ in the VDCS. The total remaining load is given by Equation 4.5:

$$J' = \sum_{i=1}^{M+\delta} \mu'_i = \sum_{i=1}^{M+\delta} \sum_{j=1}^{N'} \beta'_{ij}; i = 1, 2, \dots, (M + \delta), j = 1, 2, \dots, N' \quad (4.5)$$

In the $k = 1$ case, all the worker roles in the rescheduling round have a load. The relationship between the data bank release time and the worker role 1 processing time must consider the worker role 1 (W'_1) release time (R'_{w1}). Equation 3.3 is modified into Equation 4.6 as follows:

$$\beta'_{1j} \cdot A'_1 + R'_{w1} \geq R'_{(j+1)} - R'_j; j = 1, 2, \dots, N' - 1, k = 1 \quad (4.6)$$

LP formulation:

Because it is difficult to establish similar coordination in a VDCS environment with multiple data banks for load rescheduling, a linear programming formulation is proposed

in this chapter. The formulas are separated into two cases. $\theta = 0$ refers to the case where no worker role fetched the full load (μ_i) before the start of the rescheduling round; in simple words, no worker role started to fetch the last load fraction from data bank N (D_N). In the $k \leq \theta \leq M + \delta$ case, some of the worker roles started to fetch the last portion of the load from data bank N (D_N). In the case wherein a new worker role is assigned to a task with no load or the failure case when some worker roles are not assigned a load, we obtain $k \neq 1$. The case $k = 1$ refers to worker role failure with all the worker roles having a load. In some cases, the worker role release time is extremely short, and many worker roles do not have any loads in the rescheduling round. Under this condition, after obtaining the result, we need to further check $(R'_1 + \sum_{n=1}^{i-1} \beta'_{n1} \cdot G'_1) \leq R'_{wi}; i = k, \dots, M + \delta$. If $(R'_1 + \sum_{n=1}^{i-1} \beta'_{n1} \cdot G'_1) > R'_{wi}, i = k, \dots, M + \delta$, we need to replace the total processing time equation from chapter 3 with Equation 4.1 for that worker role.

Minimize T'_0

Such that

Total processing time constraint:

$\theta = 0$:

$$T'_0 - R'_1 - \sum_{n=1}^{i-1} \beta'_{n1} \cdot G'_1 - \sum_{j=1}^{N'} \beta'_{ij} \cdot A'_i \geq 0; i = 1, \dots, k-1, k \neq 1$$

$$T'_0 - R'_{wi} - \sum_{j=1}^{N'} \beta'_{ij} \cdot A'_i \geq 0; i = k, \dots, M + \delta$$

$k \leq \theta \leq M + \delta$:

$$T'_0 - R'_1 - \sum_{n=1}^{i-1} \beta'_{n1} \cdot G'_1 - \sum_{j=1}^{N'} \beta'_{ij} \cdot A'_i \geq 0; i = 1, \dots, k-1, k \neq 1$$

$$T'_0 - R'_{wi} - \sum_{j=1}^{N'} \beta'_{ij} \cdot A'_i \geq 0; i = k, \dots, \theta - 1$$

$$T'_0 = T_0 - T_{RS} + \mu'_i \cdot A'_i; i = \theta, \dots, M + \delta, k \leq \theta \leq M + \delta$$

Release time constraint:

$$\beta'_{1j} \cdot A'_1 \geq R'_{j+1} - R'_j; j = 1, 2, \dots, N' - 1, k \neq 1$$

$$\beta'_{1j} \cdot A'_1 + R'_{W1} \geq R'_{(j+1)} - R'_j; j = 1, 2, \dots, N' - 1, k = 1$$

Total load constraint:

$$J' = \sum_{i=1}^{M+\delta} \mu'_i = \sum_{i=1}^{M+\delta} \sum_{j=1}^{N'} \beta'_{ij}; i = 1, 2, \dots, (M + \delta), j = 1, 2, \dots, N'$$

Continuity constraint:

$k \neq 1$:

$$\beta'_{ij} \cdot A'_i + \beta'_{i(j+1)} \cdot G'_{j+1} \leq \beta'_{ij} \cdot G'_j + \beta'_{(i+1)j} \cdot A'_{i+1}; i = 1, \dots, k-2, j = 1, 2, \dots, N' - 1, k \neq 1$$

$$\begin{aligned}
 & (max(R'_{W(i+1)}, (R'_1 + \sum_{n=1}^{k-1} \beta'_{n1} \cdot G'_1)) - max(R'_{Wi}, (R'_1 + \sum_{n=1}^{k-2} \beta'_{n1} \cdot G'_1))) + \beta'_{(i+1)1} \cdot A'_{i+1} \leq \\
 & \beta'_{i1} \cdot A'_i + \beta'_{i2} \cdot G'_2; i = k-1, \dots, (M+\delta-1), j = 1, k \neq 1 \\
 & \beta'_{ij} \cdot A'_i + \beta'_{i(j+1)} \cdot G'_{j+1} \leq \beta'_{ij} \cdot G'_j + \beta'_{(i+1)j} \cdot A'_{i+1}; i = k-1, \dots, M+\delta-1, j = \\
 & 2, \dots, N'-1, k \neq 1 \\
 & k = 1 : \\
 & (R'_{W(i+1)} - R'_{Wi}) + \beta'_{(i+1)1} \cdot A'_{i+1} \leq \beta'_{i1} \cdot A'_i + \beta'_{i2} \cdot G'_2; i = k, \dots, M+\delta-1, k = 1 \\
 & \beta'_{ij} \cdot A'_i + \beta'_{i(j+1)} \cdot G'_{j+1} \leq \beta'_{ij} \cdot G'_j + \beta'_{(i+1)j} \cdot A'_{i+1}; i = k, \dots, M+\delta-1, j = \\
 & 2, \dots, N'-1, k = 1
 \end{aligned}$$

This linear programming problem has $((M+\delta) \times N' + 1)$ variables and $((M+\delta) \times N' + 1)$ constraints. The variables in our problem are the processing time (T'_0) and the load fractions $(\beta'_{11}, \beta'_{12}, \dots, \beta'_{1N'})$, $\dots$, $(\beta'_{(M+\delta)1}, \beta'_{(M+\delta)2}, \dots, \beta'_{(M+\delta)N'})$. The solution to this problem is a point in a $((M+\delta) \times N' + 1)$ -dimensional space. We applied the `linpro()` function from Matlab *v7* to solve our LP problems.

4.3 Numerical examples

In this section, we provide five numerical examples to facilitate a better understanding of the proposed rescheduling strategy for VDCS environments with multiple data banks using a linear programming formulation. The original VDCS for the numerical examples contains four worker roles and three data banks. The inverse processing speed parameters of the worker roles are $A_1 = 4, A_2 = 5, A_3 = 6$, and $A_4 = 7$; the inverse transmitting speed parameters of the data banks are $G_1 = 0.2, G_2 = 0.4$, and $G_3 = 0.6$; and the release times of the data banks are $R_1 = 10, R_2 = 30$, and $R_3 = 70$. In this example, these parameters are randomly chosen without loss of generality. We assume that the entire load ($J = 100$) to be processed is stored in data banks at the start of the computation process. The parameters for the worker roles and data banks are mentioned in Table 4.2. We will be discussing five dynamic rescheduling cases: Three cases that include adding an additional worker role at $T_R = 12/35/81$, one case that includes the addition of one

Table 4.2: The VDCS Original Numerical Example Parameters

Data Banks		Worker Roles
Release Time (R_j)	Inverse Transmitting Speed (G_j)	Inverse Processing Speed (A_i)
(10, 30, 70)	(0.2, 0.4, 0.6)	(4, 5, 6, 7)

worker role and one data bank at $T_R = 50$, and one case wherein one worker role is dropped from the resource pool at $T_R = 35$.

All worker roles stop computing at the same time, which imposes four inequality constraints, and the release time constraint of the data banks imposes two inequality constraints. The continuity constraint imposes six inequality constraints and one total load constraint. The objective is to find the load fractions designed for each worker role from data banks needed to fulfill the criteria that the total load processing time is minimized. Hence, the load scheduling problem in VDCS can be formulated as a linear programming problem as given below. The results are shown in Table 4.3.

Minimize T_0

Such that

Total processing time constraint:

$$T_0 - R_1 - (\beta_{11} + \beta_{12} + \beta_{13}) \cdot A_1 \geq 0$$

$$T_0 - R_1 - \beta_{11} \cdot G_1 - (\beta_{21} + \beta_{22} + \beta_{23}) \cdot A_2 \geq 0$$

$$T_0 - R_1 - (\beta_{11} + \beta_{21}) \cdot G_1 - (\beta_{31} + \beta_{32} + \beta_{33}) \cdot A_3 \geq 0$$

$$T_0 - R_1 - (\beta_{11} + \beta_{21} + \beta_{31}) \cdot G_1 - (\beta_{41} + \beta_{42} + \beta_{43}) \cdot A_4 \geq 0$$

Release time constraint:

$$\beta_{11} \cdot A_1 \geq R_2 - R_1$$

$$\beta_{12} \cdot A_1 \geq R_3 - R_2$$

Total load constraint:

$$\beta_{11} + \beta_{12} + \beta_{13} + \beta_{21} + \beta_{22} + \beta_{23} + \beta_{31} + \beta_{32} + \beta_{33} + \beta_{41} + \beta_{42} + \beta_{43} = J$$

Continuity constraint:

$$\beta_{11} \cdot A_1 + \beta_{12} \cdot G_2 \leq \beta_{11} \cdot G_1 + \beta_{21} \cdot A_2$$

$$\beta_{12} \cdot A_1 + \beta_{13} \cdot G_3 \leq \beta_{12} \cdot G_2 + \beta_{22} \cdot A_2$$

$$\beta_{21} \cdot A_2 + \beta_{22} \cdot G_2 \leq \beta_{21} \cdot G_1 + \beta_{31} \cdot A_3$$

Table 4.3: Load Fractions and Processing Time after Scheduling for Original Numerical Example

Total processing time: $T_0 = 142.87$				
D_j, W_i	W_1	W_2	W_3	W_4
D_1	$\beta_{11} = 5$	$\beta_{21} = 4.6$	$\beta_{31} = 4.3057$	$\beta_{41} = 4.0496$
D_2	$\beta_{12} = 10$	$\beta_{22} = 9.3861$	$\beta_{32} = 8.4348$	$\beta_{42} = 7.5265$
D_3	$\beta_{13} = 18.217$	$\beta_{23} = 12.388$	$\beta_{33} = 9.0844$	$\beta_{43} = 7.0079$

$$\begin{aligned} \beta_{22} \cdot A_2 + \beta_{23} \cdot G_3 &\leq \beta_{22} \cdot G_2 + \beta_{32} \cdot A_3 \\ \beta_{31} \cdot A_3 + \beta_{32} \cdot G_2 &\leq \beta_{31} \cdot G_1 + \beta_{41} \cdot A_4 \\ \beta_{32} \cdot A_3 + \beta_{33} \cdot G_3 &\leq \beta_{32} \cdot G_2 + \beta_{42} \cdot A_4 \end{aligned}$$

Based on the above formulas, the load fractions assigned to each worker role from each data bank and the total load processing time are given in Table 4.3. To provide a better understanding of the results, we illustrate the process of load retrieval from each data bank for the three worker roles and their computation time in the timing diagram in Figure 4.4. From the figure, we can see that the worker role computation is continuous and that all worker roles stop computing simultaneously.

4.3.1 Numerical example 1

At $T_R = 12$, one worker role is added to the resource pool, which triggers the web role to call off the former scheduled process, collects the current system status, and performs the rescheduling process based on our rescheduling scheme and current status. To achieve the minimum processing time, the remaining load is reassigned to the current available worker roles, and the load fractions are fetched from the currently available data banks.

The status progresses as follows: $T_{RS} = 12.781$ is the rescheduling start time. The rescheduling load starts to distribute loads to worker roles from the data banks. The following parameters are based on the $T_{RS} = 12.781$ timing denoted with apostrophes. The total remaining load is ($J' = 80.1972$). The rescheduling release times for the data banks are $R'_1 = 0$, $R'_2 = 17.219$ and $R'_3 = 57.219$. By considering the worker roles with the remaining load (W_1, W_2, W_3), the worker roles' release times are $R'_{W_3} = 17.219$, $R'_{W_4} =$

Figure 4.4: Timing diagram for original numerical example

21.219 and $R'_{W_5} = 24.972$. Because no worker role fetches a full load during the scheduling round at this time, $\theta = 0$; in addition, we have two worker roles without loads in the current status, and thus, $k = 3$. Moreover, 1 worker role is added to the system, resulting in $\delta = 1$. According to our assumption, a worker role without a load is going to be placed at the front of the queue for processing; therefore, we obtained a new sequence as $(W'_1 = W_4, W'_2 = W_5, W'_3 = W_1, W'_4 = W_2, W'_5 = W_3)$. In addition, the inverse processing speeds are $A'_1 = 7, A'_2 = 8, A'_3 = 4, A'_4 = 5, A'_5 = 6$. All the data banks

are available, giving $N' = 3$; in addition, the data bank inverse transmitting speeds are $G'_1 = 0.2, G'_2 = 0.4, G'_3 = 0.6$.

All worker roles stop computing simultaneously, which imposes five inequalities as the total processing time constraints. The release times of the data banks impose two inequality constraints. The continuity constraint imposes eight inequality constraints and one total remaining load constraint. The objective is to find the load fractions assigned to each worker role from the data banks to fulfill the criteria of the total remaining load processing time being minimized. Hence, the load rescheduling problem in VDCS can be formulated as a linear programming problem, as given below. The results are shown in Table 4.4.

Minimize T'_0

Such that

Total processing time constraint:

$$T'_0 - R'_1 - (\beta'_{11} + \beta'_{12} + \beta'_{13}) \cdot A'_1 \geq 0$$

$$T'_0 - R'_1 - \beta'_{11} \cdot G'_1 - (\beta'_{21} + \beta'_{22} + \beta'_{23}) \cdot A'_2 \geq 0$$

$$T'_0 - R'_{w3} - (\beta'_{31} + \beta'_{32} + \beta'_{33}) \cdot A'_3 \geq 0$$

$$T'_0 - R'_{w4} - (\beta'_{41} + \beta'_{42} + \beta'_{43}) \cdot A'_4 \geq 0$$

$$T'_0 - R'_{w5} - (\beta'_{51} + \beta'_{52} + \beta'_{53}) \cdot A'_5 \geq 0$$

Release time constraint:

$$\beta'_{11} \cdot A'_1 \geq R'_2 - R'_1$$

$$\beta'_{12} \cdot A'_1 \geq R'_3 - R'_2$$

Total load constraint:

$$\beta'_{11} + \beta'_{12} + \beta'_{13} + \beta'_{21} + \beta'_{22} + \beta'_{23} + \beta'_{31} + \beta'_{32} + \beta'_{33} + \beta'_{41} + \beta'_{42} + \beta'_{43} + \beta'_{51} + \beta'_{52} + \beta'_{53} = J'$$

Continuity constraint:

$$\beta'_{11} \cdot A'_1 + \beta'_{12} \cdot G'_2 \leq \beta'_{11} \cdot G'_1 + \beta'_{21} \cdot A'_2$$

$$\beta'_{12} \cdot A'_1 + \beta'_{13} \cdot G'_3 \leq \beta'_{12} \cdot G'_2 + \beta'_{22} \cdot A'_2$$

$$\beta'_{21} \cdot A'_2 + \beta'_{22} \cdot G'_2 \leq (R'_{w3} - (R'_1 + \beta'_{11} \cdot G'_1)) + \beta'_{31} \cdot A'_3$$

$$\beta'_{22} \cdot A'_2 + \beta'_{23} \cdot G'_3 \leq \beta'_{22} \cdot G'_2 + \beta'_{32} \cdot A'_3$$

$$\beta'_{31} \cdot A'_3 + \beta'_{32} \cdot G'_2 \leq (R'_{w4} - R'_{w3}) + \beta'_{41} \cdot A'_4$$

$$\beta'_{32} \cdot A'_3 + \beta'_{33} \cdot G'_3 \leq \beta'_{32} \cdot G'_2 + \beta'_{42} \cdot A'_4$$

Table 4.4: Load Fractions and Processing Time after Rescheduling for Numerical Example 1

The remaining load processing time: $T'_0 = 105.1$					
D'_j, W'_i	W'_1	W'_2	W'_3	W'_4	W'_5
D'_1	$\beta'_{11} = 2.46$	$\beta'_{21} = 2.38$	$\beta'_{31} = 1.09$	$\beta'_{41} = 0.93$	$\beta'_{51} = 0.75$
D'_2	$\beta'_{12} = 5.71$	$\beta'_{22} = 5.23$	$\beta'_{32} = 10.75$	$\beta'_{42} = 8.96$	$\beta'_{52} = 7.55$
D'_3	$\beta'_{13} = 5.71$	$\beta'_{23} = 5.47$	$\beta'_{33} = 10.12$	$\beta'_{43} = 6.88$	$\beta'_{53} = 5.04$

$$\begin{aligned}\beta'_{41} \cdot A'_4 + \beta'_{42} \cdot G'_2 &\leq (R'_{w5} - R'_{w4}) + \beta'_{51} \cdot A'_5 \\ \beta'_{42} \cdot A'_4 + \beta'_{43} \cdot G'_3 &\leq \beta'_{42} \cdot G'_2 + \beta'_{52} \cdot A'_5\end{aligned}$$

Based on the above formulas, load fractions assigned to each worker role from each data bank, and the total load processing times are given in Table 4.4. To provide a better understanding of the results, the process of load retrieval from each data bank for the five worker roles and their computation times are illustrated in the timing diagram in Figure 4.5. From the figure, we can see that the worker role computation is continuous even though the computation contains the remaining load, and all worker roles stop computing simultaneously. When rescheduling with a newly added resource, there is a decrease in the total load processing time.

4.3.2 Numerical example 2

At $T_R = 35$, one worker role is added to the resource pool, which triggers the web role to call off the former scheduled process, collect the current system status, and perform a rescheduling process based on our rescheduling scheme and the current status. We achieve the minimum processing time by reassigning the remaining load to the currently available worker roles and by fetching the load fractions from the currently available data banks.

The status changes as follows: $T_{RS} = 37.754$ is the rescheduling start time at which the rescheduling load starts to distribute loads to the worker roles from the data banks. The following parameters are based on the $T_{RS} = 37.754$ timing denoted by apostrophes. The total remaining load is ($J' = 55.8189$). The rescheduling release times for two

Figure 4.5: Timing diagram for numerical example 1

data banks are $R'_1 = 0$, $R'_2 = 32.246$. Considering the worker roles with remaining loads (W_1, W_2, W_4), the worker roles' release times are $R'_{W_3} = 3.374$, $R'_{W_4} = 32.25$ and $R'_{W_5} = 43.176$. No worker role fetches a full load during the scheduling round at this time, and thus, $\theta = 0$. We have two worker roles without loads in the current status, and thus, $k = 3$. In addition, 1 new worker role is added to the system, resulting in $\delta = 1$. According to our assumption, a worker role without a load is going to be placed in front of the queue for processing; therefore, we obtain the new sequence as $(W'_1 = W_3, W'_2 = W_5, W'_3 = W_4, W'_4 = W_1, W'_5 = W_2)$. In addition, the inverse processing

speeds are $A'_1 = 6, A'_2 = 8, A'_3 = 7, A'_4 = 4, A'_5 = 5$. Two of the data banks are available, and thus, $N' = 2$. The data bank inverse transmitting speeds are $G'_1 = 0.4, G'_2 = 0.6$.

All worker roles stop computing simultaneously, which imposes five inequalities as total processing time constraints. The release time of the data banks imposes only one inequality constraint. The continuity constraint imposes four inequality constraints and one total remaining load constraint. The objective is to find the load fractions assigned to each worker role from the data banks such that the total remaining load processing time is minimized. Hence, the load rescheduling problem in VDCS can be formulated as a linear programming problem, as given below. In addition, the results are presented in Table 4.5.

Minimize T'_0

Such that

Total processing time constraint:

$$T'_0 - R'_1 - (\beta'_{11} + \beta'_{12}) \cdot A'_1 \geq 0$$

$$T'_0 - R'_1 - \beta'_{11} \cdot G'_1 - (\beta'_{21} + \beta'_{22}) \cdot A'_2 \geq 0$$

$$T'_0 - R'_{w3} - (\beta'_{31} + \beta'_{32}) \cdot A'_3 \geq 0$$

$$T'_0 - R'_{w4} - (\beta'_{41} + \beta'_{42}) \cdot A'_4 \geq 0$$

$$T'_0 - R'_{w5} - (\beta'_{51} + \beta'_{52}) \cdot A'_5 \geq 0$$

Release time constraint:

$$\beta'_{11} \cdot A'_1 \geq R'_2 - R'_1$$

Total load constraint:

$$\beta'_{11} + \beta'_{12} + \beta'_{21} + \beta'_{22} + \beta'_{31} + \beta'_{32} + \beta'_{41} + \beta'_{42} + \beta'_{51} + \beta'_{52} = J'$$

Continuity constraint:

$$\beta'_{11} \cdot A'_1 + \beta'_{12} \cdot G'_2 \leq \beta'_{11} \cdot G'_1 + \beta'_{21} \cdot A'_2$$

$$\beta'_{21} \cdot A'_2 + \beta'_{22} \cdot G'_2 \leq (R'_{w3} - (R'_1 + \beta'_{11} \cdot G'_1)) + \beta'_{31} \cdot A'_3$$

$$\beta'_{31} \cdot A'_3 + \beta'_{32} \cdot G'_2 \leq (R'_{w4} - R'_{w3}) + \beta'_{41} \cdot A'_4$$

$$\beta'_{41} \cdot A'_4 + \beta'_{42} \cdot G'_2 \leq (R'_{w5} - R'_{w4}) + \beta'_{51} \cdot A'_5$$

Based on the above formulas, the load fractions assigned to each worker role from each data bank and the total load processing time are given in Table 4.5. To provide a better

Table 4.5: Load Fractions and Processing Time after Rescheduling for Numerical Example 2

The remaining load processing time: $T'_0 = 82.9188$					
D'_j, W'_i	W'_1	W'_2	W'_3	W'_4	W'_5
D'_1	$\beta'_{11} = 5.3743$	$\beta'_{21} = 4.3954$	$\beta'_{31} = 5.214$	$\beta'_{41} = 2.8165$	$\beta'_{51} = 1.2501$
D'_2	$\beta'_{12} = 8.4455$	$\beta'_{22} = 5.7007$	$\beta'_{32} = 6.0733$	$\beta'_{42} = 9.8507$	$\beta'_{52} = 6.6985$

understanding of the results, we illustrate the process of load retrieval from each data bank to five worker roles and their computation times in the timing diagram in Figure 4.6. From the figure, we can see that the worker role computation is continuous and includes the remaining load. Further, all worker roles stop computing simultaneously. With a newly added resource, as a result of rescheduling, the total load processing time is decreased.

4.3.3 Numerical example 3

At $T_R = 81$, one worker role is added to the resource pool, which triggers the web role. The worker role stops the former scheduled process, collects the current system status, and performs a rescheduling process based on our rescheduling scheme and current status. We achieve the minimum processing time by reassigning the remaining load to the currently available worker roles and by fetching the load fractions from the currently available data banks.

The status progresses as follows: $T_{RS} = 88.363$ is the rescheduling start time for when the rescheduling load distributes loads to the worker roles from the data banks. The following parameters are based on the $T_{RS} = 88.363$ timing indicated with apostrophes. The total remaining load is ($J' = 14.0047$). For rescheduling, the release time for the data banks is $R'_1 = 0$. Considering the worker roles with the remaining loads (W_1, W_2, W_4), the worker roles' release times are $R'_{W_3} = 5.4506$, $R'_{W_4} = 54.505$ and $R'_{W_5} = 54.507$. We have two worker roles without loads in the current status, and thus, $k = 3$. In addition, 1 worker role is added to the system, resulting in $\delta = 1$. According to our assumption, a worker role without a load is going to be placed at the front of the queue for processing. Thus, we obtain the new sequence as ($W'_1 = W_3, W'_2 = W_5, W'_3 = W_4, W'_4 = W_1, W'_5 = W_2$). Because two worker roles fetched full load fractions during the scheduling round,

Figure 4.6: Timing diagram for numerical example 2

at this time, $\theta = 4$. In addition, the inverse processing speeds are $A'_1 = 6$, $A'_2 = 8$, $A'_3 = 7$, $A'_4 = 4$, $A'_5 = 5$. All the data banks are available, and thus, $N' = 1$. In addition, the data bank inverse transmitting speed is $G'_1 = 0.6$.

All worker roles stop computing simultaneously, which imposes three inequalities and total processing time constraints. Equation 4.4 imposes two inequality constraints and one total remaining load constraint. The objective is to find the load fractions assigned to each worker role from data banks such that the total remaining load processing time

Table 4.6: Load Fractions and Processing Time after Rescheduling for Numerical Example 3

The remaining load processing time: $T'_0 = 45.4346$					
D'_j, W'_i	W'_1	W'_2	W'_3	W'_4	W'_5
D'_1	$\beta'_{11} = 7.5724$	$\beta'_{21} = 5.1114$	$\beta'_{31} = 5.4035$	$\beta'_{41} = -2.2681$	$\beta'_{51} = -1.8145$

is minimized. Hence, the load rescheduling problem in VDCS can be formulated as a linear programming problem, as given below. Here, for $i = 3$, because we are not involving continuity constraints, we need to check $(R'_1 + \sum_{n=1}^{i-1} \beta'_{n1} \cdot G'_1) \leq R'_{wi}$. In this case, the data bank 1 release time plus transmission time for the $i - 1$ load fractions of the worker roles is greater than the release time of worker role W'_3 ; therefore, here, for worker role 3 (W'_3), we use the total processing time equation from chapter 3. In addition, the results are presented in Table 4.6.

Minimize T'_0

Such that

Rescheduling load reassignment constraint:

$$\beta'_{41} = -(T_0 - T_{RS} - T'_0) \times \frac{1}{A'_4}$$

$$\beta'_{51} = -(T_0 - T_{RS} - T'_0) \times \frac{1}{A'_5}$$

Total load constraint:

$$\beta'_{11} + \beta'_{21} + \beta'_{31} + \beta'_{41} + \beta'_{51} = J'$$

Total processing time constraint:

$$T'_0 - A'_1 \cdot \beta'_{11} = 0$$

$$T'_0 - G'_1 \cdot \beta'_{11} - A'_2 \cdot \beta'_{21} = 0$$

$$T'_0 - G'_1 \cdot (\beta'_{11} - \beta'_{21}) - A'_3 \cdot \beta'_{31} = 0$$

Based on the above formulas, load fractions assigned to each worker role from each data bank, and the total load processing times are given in Table 4.6. To provide a better understanding of the results, we illustrate the process of load retrieval from each data bank to the three worker roles and their computation times in the timing diagram in

Figure 4.7: Timing diagram for numerical example 3

Figure 4.7. From the figure, we can see that the worker role computation is continuous even though the computation includes the remaining loads, and all the worker roles stop computing simultaneously. Further, using rescheduling, the total load processing time is decreased.

4.3.4 Numerical example 4

At $T_R = 50$, one worker role and one data bank are added to the resource pool. This triggers the web role to call off the former scheduled process, collect the current system status, and perform a rescheduling process based on our rescheduling scheme and current status. We achieve the minimum processing time by reassigning the remaining load to the currently available worker roles and by fetching the load fractions from the currently available data banks.

The status progresses as follows: $T_{RS} = 70$ is the rescheduling start time, where the rescheduling load starts to distribute loads to the worker roles from the data banks. The following parameters are based on the $T_{RS} = 70$ timing indicated by apostrophes. The total remaining load is ($J' = 40.6397$). For rescheduling, the release times for the data banks are $R'_1 = 0, R'_2 = 0$. Considering the worker roles with remaining loads (W_2, W_3, W_4), the worker roles' release times are $R'_{W3} = 80.9302, R'_{W4} = 88.363$ and $R'_{W5} = 92.56774$. No worker role fetched a full load during the scheduling round, and at this time, $\theta = 0$. Because we have two worker roles without loads in the current status, $k = 3$, and 1 worker role is added to the system; therefore, $\delta = 1$. According to our assumption, a worker role without a load is going to be placed in the front of the queue for processing; thus, we obtain a new sequence as ($W'_1 = W_5, W'_2 = W_1, W'_3 = W_2, W'_4 = W_3, W'_5 = W_4$). The inverse processing speeds are $A'_1 = 8, A'_2 = 4, A'_3 = 5, A'_4 = 6, A'_5 = 7$. The two data banks are available; thus, $N' = 2$. In addition, the data bank inverse transmitting speeds are $G'_1 = 0.6, G'_2 = 0.8$.

All worker roles stop computing simultaneously, which imposes five inequalities and total processing time constraints. The release time of the data banks imposes one inequality constraint. The continuity constraint imposes four inequality constraints and one total remaining load constraint. The objective is to find the load fractions assigned to each worker role from the data banks such that the total remaining load processing time is minimized. Hence, the load rescheduling problem in a distributed computing system can be formulated as a linear programming problem, as given below. In addition, the results are presented in Table 4.7.

Minimize T'_0

Such that

Total processing time constraint:

$$T'_0 - R'_1 - (\beta'_{11} + \beta'_{12}) \cdot A'_1 \geq 0$$

$$T'_0 - R'_1 - \beta'_{11} \cdot G'_1 - (\beta'_{21} + \beta'_{22}) \cdot A'_2 \geq 0$$

$$T'_0 - R'_{w3} - (\beta'_{31} + \beta'_{32}) \cdot A'_3 \geq 0$$

$$T'_0 - R'_{w4} - (\beta'_{41} + \beta'_{42}) \cdot A'_4 \geq 0$$

$$T'_0 - R'_{w5} - (\beta'_{51} + \beta'_{52}) \cdot A'_5 \geq 0$$

Release time constraint:

$$\beta'_{11} \cdot A'_1 \geq R'_2 - R'_1$$

Total load constraint:

$$\beta'_{11} + \beta'_{12} + \beta'_{21} + \beta'_{22} + \beta'_{31} + \beta'_{32} + \beta'_{41} + \beta'_{42} + \beta'_{51} + \beta'_{52} = J'$$

Continuity constraint:

$$\beta'_{11} \cdot A'_1 + \beta'_{12} \cdot G'_2 \leq \beta'_{11} \cdot G'_1 + \beta'_{21} \cdot A'_2$$

$$\beta'_{21} \cdot A'_2 + \beta'_{22} \cdot G'_2 \leq (R'_{w3} - (R'_1 + \beta'_{11} \cdot G'_1)) + \beta'_{31} \cdot A'_3$$

$$\beta'_{31} \cdot A'_3 + \beta'_{32} \cdot G'_2 \leq (R'_{w4} - R'_{w3}) + \beta'_{41} \cdot A'_4$$

$$\beta'_{41} \cdot A'_4 + \beta'_{42} \cdot G'_2 \leq (R'_{w5} - R'_{w4}) + \beta'_{51} \cdot A'_5$$

Based on the above formulas, load fractions assigned to each worker role from each data bank, and the total load processing times are presented in Table 4.7. To provide a better understanding of the results, we illustrate the process of load retrieval from each data bank to the five worker roles and their computation times in the timing diagram in Figure 4.8. From the figure, we can see that the worker role computation is continuous even though the computation contains the remaining loads, and all worker roles stop computing simultaneously. By rescheduling using newly added resources, the total load processing time is decreased.

4.3.5 Numerical example 5

At $T_R = 35$, one worker role fails and drops off from the resource pool. This triggers the web role to call off the former scheduled process, collect the current system status,

Figure 4.8: Timing diagram for numerical example 4

and conduct a rescheduling process based on our rescheduling scheme current status. We achieve task completion by reassigning the remaining load to the currently available worker roles and by fetching the load fractions from the currently available data banks.

The status progresses as follows: $T_{RS} = 35$ is the rescheduling start time, where the rescheduling load starts to distribute loads to the worker roles from the data banks. The following parameters are based on the $T_{RS} = 35$ timing indicated by apostrophes. The total remaining load is ($J' = 71.8453$). For rescheduling, the release times for the

Table 4.7: Load Fractions and Processing Times after Rescheduling for Numerical Example 4

The remaining load processing time : $T'_0 = 125.723$					
D'_j, W'_i	W'_1	W'_2	W'_3	W'_4	W'_5
D'_1	$\beta'_{11} = 0$	$\beta'_{21} = 1.3931$	$\beta'_{31} = 0.93444$	$\beta'_{41} = 0.42071$	$\beta'_{51} = 0.24548$
D'_2	$\beta'_{12} = 6.9654$	$\beta'_{22} = 12.538$	$\beta'_{32} = 8.0241$	$\beta'_{42} = 5.8059$	$\beta'_{52} = 4.313$

data banks are $R'_1 = 0, R'_2 = 35$. Considering the worker roles with the remaining load (W_3, W_4, W_1) , the worker roles' release times are $R'_{w1} = 2.754, R'_{w2} = 6.1284$ and $R'_{w3} = 35$. No worker role fetched a full load during the scheduling round; thus, at this time, $\theta = 0$. Because we have three worker roles with loads in the current status, $k = 1$; additionally, 1 worker role drops off from the system, resulting in $\delta = -1$. According to our assumption, a worker role without a load will be placed in the front of the queue for processing; therefore, we obtain a new sequence as $(W'_1 = W_3, W'_2 = W_4, W'_3 = W_1)$. In addition, the inverse processing speeds are $A'_1 = 6, A'_2 = 7, A'_3 = 4$. The two data banks are available; thus, $N' = 2$. The data bank inverse transmitting speeds are $G'_1 = 0.4, G'_2 = 0.6$.

All worker roles stop computing simultaneously, which imposes three inequalities and total processing time constraints. The release time of the data banks imposes one inequality constraint using Equation 4.6. The continuity constraint imposes two inequality constraints and one total remaining load constraint. The objective is to find the load fraction assignment to each worker role from the data banks to achieve task completion with a minimum total remaining load processing time. Hence, the load rescheduling problem in the distributed computing system can be formulated as a linear programming problem, as given below. The results are presented in Table 4.8.

Minimize T'_0

Such that

Total processing time constraint:

$$T'_0 - R'_{w1} - (\beta'_{11} + \beta'_{12}) \cdot A'_1 \geq 0$$

$$T'_0 - R'_{w2} - (\beta'_{21} + \beta'_{22}) \cdot A'_2 \geq 0$$

$$T'_0 - R'_{w3} - (\beta'_{31} + \beta'_{32}) \cdot A'_3 \geq 0$$

Table 4.8: Load Fractions and Processing Times after Rescheduling for Numerical Example 5

The remaining load processing time: $T'_0 = 146.428$			
D'_j, W'_i	W'_1	W'_2	W'_3
D'_1	$\beta'_{11} = 8.75$	$\beta'_{21} = 8.16$	$\beta'_{31} = 7.8653$
D'_2	$\beta'_{12} = 27.857$	$\beta'_{22} = 15.7856$	$\beta'_{32} = 12.1775$

Release time constraint:

$$\beta'_{11} \cdot A'_1 \geq R'_2 - R'_1 - R'_{w1}$$

Total load constraint:

$$\beta'_{11} + \beta'_{12} + \beta'_{21} + \beta'_{22} + \beta'_{31} + \beta'_{32} = J'$$

Continuity constraint:

$$\beta'_{11} \cdot A'_1 + \beta'_{12} \cdot G'_2 \leq (R'_{w2} - R'_{w1}) + \beta'_{21} \cdot A'_2$$

$$\beta'_{21} \cdot A'_2 + \beta'_{22} \cdot G'_2 \leq (R'_{w3} - R'_{w2}) + \beta'_{31} \cdot A'_3$$

Based on the above formulas, the load fractions assigned to each worker role from each data bank and the total load processing times are presented in Table 4.8. To provide a better understanding of the results, we illustrate the process of load retrieval from each data bank to three worker roles and their computation times in the timing diagram in Figure 4.9. From the figure, we can see that the worker role computation is continuous even though the computation contains the remaining load, and all worker roles stop computing simultaneously. By rescheduling with failure resources, the task is completed with the remaining resources with increased total processing time.

4.4 Summary

In this chapter, a DLT-based rescheduling strategy has been proposed for solving dynamic resource allocation and system fault tolerance for computing node failure. This is performed to fulfill the objective of maximizing the utilization of distributed computing cloud resources and minimizing the task processing time. For optimizing the rescheduling

Figure 4.9: Timing diagram for numerical example 5.

scheme, the overall load has to be minimized. We introduced new notations for incorporating the current rescheduling VDCS status and worker role release time into the framework. Then, we introduced the altered formulae for rescheduling and re-ordering the sequence of executing the worker roles based on this release time. The numerical examples show that the rescheduling scheme can help to guarantee a minimum processing time in a dynamic VDCS environment and assists distributed computing service providers in obtaining the maximum utilization of resources.

Chapter 5

Divisible load scheduling for multiple rounds and multiple data banks

5.1 Introduction

In the previous chapter, we proposed a rescheduling scheme that addresses VDCS dynamic and fault-tolerant features. In this chapter, we are going to propose a heuristic Hybrid Genetic Algorithm (HGA) for solving divisible load scheduling with multiple rounds for an optimization problem for the single-data-bank case. Similar to a virtual distributed computing system, the main objective is to partition and schedule the processing load among the worker roles such that the total processing time is minimized. Therefore, an efficient partitioning and scheduling algorithm is necessary for the system. Being a heterogeneous system, the worker roles and the communication links have different speeds in a VDCS, which represents a challenge. The VDCS with the affine communication model includes m worker roles and n rounds of load distribution. Using HGA, we find the optimal number of worker roles ($m^*, m^* \leq m$), the optimal number of rounds ($n^*, n^* \leq n$) and the optimal distribution order of worker roles. Then, we provide simulation results to demonstrate the performance of our algorithm and also illustrate the convergence rate by comparing the HGA, RCGA and PSO algorithms. This work is published in [136]. Finally, we altered and applied HGA with VDCS with an example using multiple data banks.

5.2 Problem formulation

A VDCS with the affine communication model has $(m + 1)$ worker roles and m links, as shown in Figure 2.5. The processing load originates at the worker role with data (w_0) . The worker role with data distributes the load assigned to each worker role in n rounds (installments). The order in which the processing load is distributed (the sequence of the load distribution) in all rounds is the same and is $\{w_1, w_2, \dots, w_m\}$. The worker roles in the VDCS are equipped with communication co-processors (front-ends) so that communication and computation can be performed concurrently. The objective here is to find the size of the load fractions assigned to each worker role in each installment such that the processing time of the entire processing load is minimized. The computation and communication models used in this paper are as follows:

Computation time is linear: If β_i is the load fraction assigned to worker role w_i , then the time to compute this load fraction by worker role w_i is $\beta_i A_i$, where A_i is the inverse processing speed (reciprocal of the speed) of the worker role w_i .

The communication time includes latencies: If β_i is the load fraction assigned to worker role w_i , then the time to communicate this load fraction to worker role w_i by the worker role with data w_0 is $g_i + \beta_i G_i$. Here, g_i is the communication latency, and G_i is the inverse transmitting speed (reciprocal of bandwidth) of the link l_i to worker role w_i . For communication, a one-port model is used. The worker role with data (w_0) can communicate with only one worker role at a given time.

The **load distribution** is defined as an $(mn + 1)$ -tuple $\{\beta_0, \beta_{1,1}, \beta_{2,1}, \dots, \beta_{m,1}, \beta_{1,2}, \beta_{2,2}, \dots, \beta_{m,2}, \dots, \beta_{1,n}, \beta_{2,n}, \dots, \beta_{m,n}\}$, such that $0 \leq \beta_{i,j} \leq 1$ and $\beta_0 + \sum_{i=1}^m \sum_{j=1}^n \beta_{i,j} = J$. Here, β_0 is the load fraction assigned to the worker role with data (w_0) , and $\beta_{i,j}$ is the load fraction assigned to the i -th worker role (w_i) in round j . J is the total processing load.

The **finish time** (T_i) for worker role w_i is the time difference between the time instant at which the worker role w_i stops computing and the time instant at which the worker role with data w_0 initiates the load distribution process.

The **processing time** (T) is the time at which the entire load (J) is processed and is given by the maximum of the finish times of all worker roles, i.e., $T = \max\{T_i\}$, $i = 0, 1, \dots, m$, where T_i is the finish time of worker role w_i .

We now present some motivating numerical examples to demonstrate the combinatorial nature of this multi-round load distribution strategy with the affine communication model.

Motivation example.1: Consider a system with two worker roles w_1 and w_2 attached to the worker role with data w_0 . The inverse processing speed parameters of the worker roles are $A_0 = A_1 = A_2 = 1.0$; the inverse transmitting speed parameters are $G_1 = 0.6$, $G_2 = 2.0$; and the communication latencies are $g_1 = 2$ and $g_2 = 1$. The processing load is sent to the worker roles w_1 and w_2 in two rounds (installments). The total processing load is J ($J = 100$). The distribution order is $\{w_1, w_2\}$ in both rounds. In this situation, we need to consider the following cases:

- Case 1: Use both the worker roles w_1 and w_2 in two rounds.
- Case 2: Use the worker role w_1 only in the first round and w_2 in both (two) rounds.
- Case 3: Use the worker role w_1 only in the second round and w_2 in both (two) rounds.
- Case 4: Use the worker role w_1 in both (two) rounds and w_2 only in the first round.
- Case 5: Use the worker role w_1 in both (two) rounds and w_2 only in the second round.

The processing time and the load fractions assigned to the worker roles in each round are given in Table 5.1. Note here that $\beta_{i,j}$ is the load fraction assigned to the worker role i (w_i) in the j -th round. Now, we explain how we arrived at the load fractions. In the divisible load scheduling literature, a timing diagram is the usual way of representing the load distribution process. It is shown in the divisible load scheduling literature [18] that,

Figure 5.1: Two worker roles in the two-installment case with start-up delays

to achieve the minimum processing time, all the worker roles participating in the computation process should stop computing at the same time instant. The timing diagram for the two-worker-role system with start-up overhead in two rounds (two installments) is shown in Figure 5.1. From this timing diagram, the recursive load distribution equations for the two rounds are

$$\beta_{1,2}A_1 = g_2 + \beta_{2,2}\{G_2 + A_2\} \quad (5.1)$$

$$\beta_{2,1}A_2 = g_1 + \beta_{1,2}G_1 + g_2 + \beta_{2,2}G_2 \quad (5.2)$$

$$\beta_{1,1}A_1 = g_1 + \beta_{1,2}G_1 + g_2 + \beta_{2,1}G_2 \quad (5.3)$$

$$\beta_0 A_0 = g_1 + \beta_{1,1}G_1 + g_2 + \beta_{2,1}G_2 + g_1 + \beta_{1,2}G_1 + g_2 + \beta_{2,2}G_2 + \beta_{2,2}A_2 \quad (5.4)$$

Table 5.1: Motivating Example 1: Distribution Order $\{w_1, w_2\}$

Inverse Processing Speed Parameters: $A_0 = 1$ $A_1 = 1$ $A_2 = 1$						
Inverse Transmitting Speed Parameters: $G_1 = 0.6$ $G_2 = 2.0$						
Start-up Time: $g_1 = 2$ $g_2 = 1$; Total load (J) = 100						
<i>Case</i>	β_0	$\beta_{1,1}$	$\beta_{2,1}$	$\beta_{1,2}$	$\beta_{2,2}$	T
1	53.2066	27.9569	10.5358	6.4756	1.8252	53.2066
2	54.1509	32.5943	9.1698	0	4.0849	54.1509
3	63.2178	0	19.0990	13.7624	3.9208	63.2178
4	54.0623	28.9318	11.2344	5.7715	0	54.0623
5	52.0856	16.3527	0	23.9212	7.6404	52.0856

Table 5.2: Motivation Example 1: Distribution Order $\{w_2, w_1\}$

Inverse Processing Speed Parameters: $A_0 = 1$ $A_1 = 1$ $A_2 = 1$						
Inverse Transmitting Speed Parameters: $G_1 = 0.6$ $G_2 = 2.0$						
Start-up Time: $g_1 = 2$ $g_2 = 1$; Total load (J) = 100						
<i>Case</i>	β_0	$\beta_{1,1}$	$\beta_{2,1}$	$\beta_{1,2}$	$\beta_{2,2}$	T
1	62.6471	19.1123	12.4866	4.3102	1.4438	62.6471
2	64.0636	21.0212	6.8432	0	8.0720	64.0636
3	54.0623	0	28.9318	11.2344	5.7715	54.0623
4	63.1188	19.3267	13.4158	4.1386	0	63.1188
5	67.4118	19.1176	0	9.0588	4.4118	67.4118

The normalization equation is

$$\beta_0 + \beta_{1,1} + \beta_{1,2} + \beta_{2,1} + \beta_{2,2} = 100 \quad (5.5)$$

By making the appropriate values of $\beta_{i,j}$ zero and removing the unnecessary equation for the cases, from the above equations, the load fractions for all the cases (1-5) are obtained. For example, in Case 3, the load fractions are obtained by setting $\beta_{1,1} = 0$ (also g_1 for that round) and by removing the $\beta_{1,1}$ equation.

For this problem, there are two possible distribution orders for worker roles with data: $\{w_1, w_2\}$ and $\{w_2, w_1\}$. The results for the distribution order $\{w_2, w_1\}$ are given in Table 5.2. From Tables 5.1 and 5.2, we see that the optimal distribution order is $\{w_1, w_2\}$ for the above problem.

Motivation Example 2: Let us consider the same problem in Example 1 with

Table 5.3: Motivating Example 2: Distribution Order $\{w_1, w_2\}$

Inverse Processing Speed Parameters: $A_0 = 1$ $A_1 = 1$ $A_2 = 1$						
Inverse Transmitting Speed Parameters: $G_1 = 0.6$ $G_2 = 2.0$						
Start-up Time: $g_1 = 0$ $g_2 = 0$; Total load (J) = 100						
$Case$	β_0	$\beta_{1,1}$	$\beta_{2,1}$	$\beta_{1,2}$	$\beta_{2,2}$	T
1	51.1918	26.6742	10.7832	8.5131	2.8377	51.1918
2	52.8302	33.0189	9.4339	0	4.7169	52.8302
3	61.3861	0	18.8119	14.8515	4.9505	61.3861
4	52.5223	28.1899	11.8694	7.4184	0	52.5223
5	50.3425	15.4109	0	25.6849	8.5616	50.3425

$g_1 = g_2 = 0$. The results for the above cases are shown in Table 5.3. From Table 5.3, we see that using worker role w_1 in both rounds and worker role w_2 only in the second round produces an optimal processing time. Note here, also from the load fractions, that we can obtain the optimal distribution order for the worker role with data as in Motivation Example 1.

In a recent study [14], it was shown that, in the case of an affine communication model for single-round load distribution, the complexity represents an open problem. The difficulty concerns, for multi-round load distribution, the affine or linear communication model. Here, we need to find the participating worker roles (resource selection [14]) in each round and then find the load fractions assigned to the worker roles in each round.

For a general m -worker-role system with n rounds (installments) of load distribution, the following two important questions arise:

1. *In each round, should one use all the worker roles or only some of the worker roles, and what is the load fraction assigned to each worker role in each round?*
2. *What is the optimal number of worker roles, the optimal number of rounds, and the optimal distribution order for the root worker role in a general m -worker-role, n -round system?*

These questions were raised in a recent study [14] for the first time and is found to be an open problem. Hence, in [14], a uniform multi-round scheduling algorithm is used.

In this chapter, these questions are addressed and solved using a hybrid genetic algorithm. We consider a general m -worker-role system with n rounds of load distribution.

The communication model is affine. The hybrid genetic algorithm gives the load fractions assigned to every worker role in every round. The load fractions assigned to some worker roles in some rounds of load distribution may be zero. For example, if the value of the load fraction $\beta_{4,5}$ is zero, then the worker role w_4 is not participating in round 5 of the load distribution. From the load fractions for every worker role in every round, we obtain the optimal number of rounds, the optimal number of worker roles, and the optimal distribution order for the root worker role for a general m -worker-role system with n rounds of load distribution.

Figure 5.2: Timing diagram for general m -worker-role, n -installment system

The timing diagram for a general m -worker-role system with n -round (installments) load distribution is shown in Figure 5.2. We see that the worker roles and the rounds are numbered in natural order (and not in reverse order, as in [20, 14]). Additionally, note that, for a general m -worker-role, n -round load distribution, it may not be possible to satisfy Rule 2, as mentioned above. A worker role may have *idle* time after completing

one round and waiting for the load assigned to it in the next round. Therefore, we modify the finish time of worker role w_i , $i = 1, \dots, m$ as

$$T_0 = \beta_0 A_0 \quad (5.6)$$

$$T_i = \left\{ \sum_{k=1}^i \{g_k + \beta_{k,1} G_k\} \right\} + \text{Max}\{\beta_{i,1} A_i, K_{i,1}\} + \dots + \text{Max}\{\beta_{i,n} A_i, K_{i,n}\} \quad (5.7)$$

In the above equation, the first term $(\sum_{k=1}^i \{g_k + \beta_{k,1} G_k\})$ gives the time at which the worker role w_i starts computing the first round load fraction. Each Max function ensures that the processing of a load fraction in any round is started only after that load fraction is received by that worker role and addresses if the worker role is idle after complete the processing of the load assigned in the previous round. For example, $\beta_{i,1} A_i$ is the time at which the worker role w_i completes the processing of the load fraction assigned to it in the first round. $K_{i,1}$ is the time at which the processing load assigned to worker role w_i in the second round is received by worker role w_i . Additionally, note in the above equation that if any load fraction $(\beta_{i,j})$ is zero, then the corresponding g_i is made zero for computing the value of T_i . The value of $K_{p,q}$ is

$$K_{p,q} = \sum_{j=p+1}^m \{g_j + \beta_{j,q} G_j\} + \sum_{j=1}^p \{g_j + \beta_{j,q+1} G_j\} \quad (5.8)$$

We now pose this multi-round load distribution problem as an optimization problem.

Given the speed parameters of the network (G_i, A_i) , start-up delays (g_i) , number of rounds of load distribution (n) , and the distribution order for the root worker role.

Find the optimal load fractions assigned to each worker role in each round such that the processing time (T) of the entire processing load is minimized. The processing time (T) is given by

$$T = \max \{T_i, \ i = 0, 1, \dots, m\} \quad (5.9)$$

where T_i is the finish time of worker role w_i . One should note that the solution should satisfy the equality constraint

$$J = \sum_{i=1}^m \sum_{q=1}^n \beta_{i,q} \quad (5.10)$$

where J is total size of the processing load.

In this chapter, we propose an improved hybrid genetic algorithm to obtain the optimal load fractions assigned to each worker role in each round for the above problem. The hybrid genetic algorithm employs real code crossover and mutation operators. The operators are designed such that the solution produced by them always satisfy the total load constraint given by Equation 5.10.

5.3 Hybrid genetic algorithm for multi-installment problem

Genetic Algorithms (GAs) were developed by Holland [71] in an attempt to explain the adaptive process of natural systems and to design artificial systems based upon these natural systems. In earlier studies of genetic algorithms [71, 60], the solutions were coded using binary representations. It was shown in [70] that, for numerical optimization problems, the floating point representation of solutions performs better than a binary representation because they are more precise and more consistent and lead to faster convergence. Genetic algorithms using a floating point representation for solutions are called hybrid genetic algorithms. This type of representation for solutions is also known as the floating-point representation, the real number representation or the continuous representation. A hybrid genetic algorithm with different types of genetic operators at different stages of the evolution process can provide a more effective and better solution to many practical optimization problems. In recent years, many researchers have used hybrid genetic algorithms to solve optimization problems [116, 70].

In [69], Integer-coded Genetic Algorithm (ICGA) and Particle Swarm Optimization (PSO) were combined with a neural-network-based Extreme Learning Machine (ELM). The algorithm performed well and provided accurate gene selection and sparse data classification for microarray data for multiclass cancer classification. The author compared their algorithm with algorithms in the literature. In [120, 35], the ACGA algorithm was applied to solve several scheduling problems. The main characteristic of ACGA is that it alternates the EDAs and genetic operators in each generation. Many extensions of ELM have been developed to improve the performance in sparse high-dimensional applications

such as cancer recognition, prediction, finance and control [135]. In [30], the researchers proposed an Adaptive Genetic Algorithm (AGA) with a dynamic fitness function for Multi-Objective Problems (MOPs) in a dynamic environment. The results demonstrated that the performance of the fuzzy-augmented GA is better than that of a standard GA in terms of improved convergence to solutions of dynamic MOPs. In [155], GA-BT is a genetic-algorithm-based peer selection optimization strategy for efficient content distribution in Bit-Torrent networks. GA-BT employs divisible load theory to dynamically call an optimal fitness value to accelerate the convergence process for producing optimal or near-optimal schedules in peer selection. Integrating dominance properties with a genetic algorithm (GADP) attempted to solve parallel machine scheduling problems while considering sets up to the fourth dimension.

Additional details on how genetic algorithms work for a given problem can be found in the literature [49, 71, 60]. In [57], a genetic algorithm was designed as the optimal method for a new scheduler. The results demonstrated that this scheduler achieves a better performance than FIFO and delay scheduling policies. In [149], Jianfeng et al. proposed a virtual resource scheduling model called the multi-objective optimization model and solved it using an advanced Non-dominated Sorting Genetic Algorithm II (NSGA II). The major focus of this model is load balancing. The virtual resources and physical resources are abstracted using many nodes with attributes based on analyzing the flow of virtual resource scheduling. NSGA II was employed to address this model, and a new tree sorting algorithm was adopted to improve the efficiency of NSGA II. It has been shown that NSGA II is at least 1.06 times and at most 40.25 times more computationally efficient than the Random algorithm, Static algorithm and Rank algorithm. In [44], the researchers proposed a genetic-algorithm-based load balancing strategy for cloud computing. Simulation results for a typical sample application demonstrated that the proposed algorithm outperformed existing approaches such as First Come, First Serve (FCFS), round robin and the local search algorithm Stochastic Hill Climbing (SHC). In [131], a new shadow price guided operator in a GA was used to achieve good measurable evolutions for solving the traveling salesman problem in terms of both performance and speed. A genetic algorithm that injects artificial chromosomes was developed to solve single-machine scheduling problems. The results showed that the algorithm can improve

the solution quality significantly in [36]. In [171], Hai et al. proposed an optimized resource scheduling algorithm that focuses on increasing the utilization rate of resources and speed. This algorithm contributes mainly to open-source IaaS cloud systems by discovering a flexible way to allocate virtual machines to permit the maximum usage of physical resources. The IGA uses the shortest genes and introduces the idea of Dividend policy in Economics to select an optimal or suboptimal allocation for the VM requests. Based on that study, it could be inferred that IGA is twice as fast as TGA.

Particle Swarm Optimization (PSO) is a computational method that optimizes a problem by iteratively attempting to improve a candidate solution with regard to a given measure of quality. PSO optimizes a problem by having a population of candidate solutions, here dubbed particles, and moving these particles around in a search space according to simple mathematical formulas for the particle position and velocity. Each particle movement is influenced by its local best-known position; however, the particle is also guided toward the best-known positions found by other particles. This is expected to move the cloud toward the best resolutions. In [129], PSO was applied to optimize the parameters of the ELM algorithm to obtain higher classification accuracy. In [134], a PSO-based resource allocation mechanism for a cloud resource broker that effectively distributes cloud resources and completes jobs for scientific applications within a user-specified deadline was presented.

In genetic algorithms, a solution to the optimization problem is represented as a string (a coded solution or solution). For our problem, the string (solution) gives the load fractions assigned to each worker role in each round of load distribution. Each string (solution) is evaluated according to a fitness function (related to the objective function), which is problem specific. A survival-of-the-fittest strategy is adopted to identify the best strings (solutions), and subsequent genetic operators are used to create new solutions for the next generation. This process of producing successive generations continues until the termination (convergence) criterion is satisfied.

5.3.1 Multi-installment hybrid genetic algorithm

The construction of a hybrid genetic algorithm for our problem involves the following issues: string representation, population initialization, selection function, design of genetic

operators, fitness function and termination criterion. Now, we will describe these issues involved in applying a genetic algorithm to our multi-round divisible load scheduling problem.

String Representation: The string representation is the process of encoding a solution to the problem. Each string in the population represents a possible solution to the scheduling problem. The string representation scheme depends on the structure of the problem in the GA framework and on the genetic operators used in the algorithms. For our problem, the string consists of an array of real numbers. The value of each element in the array represents the load fraction assigned to each worker role in each round. The length of the string is $mn + 1$, representing load fractions β_0 and $\beta_{i,j}$ $i = 1, 2, \dots, m$ and $j = 1, 2, \dots, n$. A valid string is one in which the sum of the load fractions (sum of the elements in the array) is equal to the total processing load (J).

For example, in the case of a two worker roles ($m = 2$) and two rounds ($n = 2$) in the system, a string will represent the load fractions $\{\beta_0, \beta_{1,1}, \beta_{2,1}, \beta_{1,2}, \beta_{2,2}\}$, where β_0 is the load fraction assigned to the worker role with data (w_0) and $\beta_{i,j}$ is the load fraction assigned to worker role w_i in the j -th round. For example, a string $\{20, 20, 20, 20, 20\}$ is a valid string because the sum of the assigned load fractions is equal to the total processing load J (100).

Population Initialization: One of the advantages of a genetic algorithm is that the genetic algorithm searches from a population of solution points instead of a single solution point. The most frequently used technique for population initialization is random generation or based on the knowledge of the given problem. However, the initial population size and the method of population initialization will affect the rate of convergence to the solution. For our problem, we proposed three different initial schemes such that the solution covers the search space efficiently. The population of solutions is generated in the following manner.

- Equal allocation: The value of the load fractions in the solution is $\frac{J}{mn+1}$.
- Random allocation: Generate $mn + 1$ random numbers. These random numbers are normalized such that the sum is equal to J . This is the value of β_0 and $\beta_{i,j}$ in the solution.

- **Zero allocation:** Select a solution using equal or random allocation. Any one element in the selected solution is assigned a value of zero, and its value is equally allocated to other elements.

Selection Function: We used the normalized geometric ranking method given in [72] for the selection process. In this method, the solutions (population) are arranged in descending order of their fitness value. Let q be the selection probability of selecting the best solution, and let r_j be the rank of the j -th solution in the partially ordered set. The probability of solution j being selected using the normalized geometric ranking method is

$$s_j = q' (1 - q)^{r_j - 1} \quad (5.11)$$

where $q' = \frac{q}{1 - (1 - q)^N}$ and N is the population size. q is the probability of selecting the best individual; r_j is the rank of the individual, where 1 is the best; and s_j is the probability of selecting the j^{th} individual [72].

Genetic Operators: Genetic operators are used to create new solutions based on the current solutions. New solutions are obtained by combining or rearranging parts of the old solutions, and a newly obtained solution may be a better solution to the optimization problem. These genetic operators are analogous to those that occur in nature: reproduction, crossover (or recombination), and mutation. One should note that the solution generated by these operators should satisfy the total load constraint given by Equation 5.10. Hence, we improved the operators to produce a valid solution at any given time.

Crossover Operator: is a primary operator in GA. The role of the crossover operator is to recombine information from the two selected solutions to produce two new solutions. The crossover operator improves the diversity of the solution. Now, we describe four different crossover operators used in our divisible load scheduling problem. Let C_1 and C_2 be the two solutions selected for the crossover operations.

$$C_1 = \{c_1^1, c_2^1, \dots, c_i^1, c_{i+1}^1, \dots, c_j^1, c_{j+1}^1, \dots, c_{mn+1}^1\} \quad (5.12)$$

$$C_2 = \{c_1^2, c_2^2, \dots, c_i^2, c_{i+1}^2, \dots, c_j^2, c_{j+1}^2, \dots, c_{mn+1}^2\} \quad (5.13)$$

Modified Two-Point Crossover (MTPX): This operator first randomly selects two crossover points i and j , where $i < j$. Let $K_1 = c_i^1 + c_{i+1}^1 + \dots + c_j^1$ and $K_2 = c_i^2 + c_{i+1}^2 + \dots + c_j^2$. Here, K_1 and K_2 are the total loads between the crossover sites. In addition, let $x_1 = K_1/K_2$ and $x_2 = K_2/K_1$. Two new solutions H_1 and H_2 are obtained as

$$H_1 = \{c_1^1, c_2^1, \dots, x_1 c_i^2, x_1 c_{i+1}^2, \dots, x_1 c_j^2, c_{j+1}^1, \dots, c_{mn+1}^1\} \quad (5.14)$$

$$H_2 = \{c_1^2, c_2^2, \dots, x_2 c_i^1, x_2 c_{i+1}^1, \dots, x_2 c_j^1, c_{j+1}^2, \dots, c_{mn+1}^2\} \quad (5.15)$$

For example, consider two ($m = 2$) worker roles in a system with two ($n = 2$) rounds. Let the solutions C_1 and C_2 be selected following the crossover operation. The crossover points selected are $i = 2$ and $j = 4$. The load fractions between two crossover sites are represented in bold.

$$C_1 = [40 \textbf{20 10 15} 15]$$

$$C_2 = [30 \textbf{15 25 10} 20]$$

Here, $K_1 = 20 + 10 + 15 = 45$ and $K_2 = 15 + 25 + 10 = 50$. The two new obtained solutions H_1 and H_2 are

$$H_1 = \left[40 \textbf{15} \frac{45}{50} \textbf{25} \frac{45}{50} \textbf{10} \frac{45}{50} 15 \right] = [40 \textbf{13.5 22.5 9} 15]$$

$$H_2 = \left[30 \textbf{20} \frac{50}{45} \textbf{10} \frac{50}{45} \textbf{15} \frac{50}{45} 20 \right] = [30 \textbf{22.2222 11.1111 16.6667} 20]$$

The MTPX ensures that the sum of the load fractions assigned to the worker roles in the network is equal to the total processing load. The operator generates two new solutions only when K_1 and K_2 are greater than zero.

Modified Simple Crossover (MSCX): Here, only one crossover site i is selected at random, and the second crossover site is $mn + 1$. Let $K_1 = c_i^1 + c_{i+1}^1 + \dots + c_{mn+1}^1$, $K_2 = c_i^2 + c_{i+1}^2 + \dots + c_{mn+1}^2$, $x_1 = K_1/K_2$ and $x_2 = K_2/K_1$. Two new solutions H_1 and H_2 are

$$H_1 = \{c_1^1, c_2^1, \dots, x_1 c_i^2, x_1 c_{i+1}^2, \dots, x_1 c_{mn+1}^2\} \quad (5.16)$$

$$H_2 = \{c_1^2, c_2^2, \dots, x_2 c_i^1, x_2 c_{i+1}^1, \dots, x_2 c_{mn+1}^1\} \quad (5.17)$$

Modified Uniform Crossover (MUCX): In this operator, the crossover sites are selected at random. Suppose that i and j are the two points selected at random for the crossover operation; then, the sums of the load fractions selected for the crossover operation are $K_1 = c_i^1 + c_j^1$ and $K_2 = c_i^2 + c_j^2$. Let the factors $x_1 = K_1/K_2$ and $x_2 = K_2/K_1$. Two new solutions H_1 and H_2 are

$$H_1 = \{c_1^1, c_2^1, \dots, x_1 c_i^2, c_{i+1}^1, \dots, x_1 c_j^2, c_{j+1}^1, \dots, c_{mn+1}^1\} \quad (5.18)$$

$$H_2 = \{c_1^2, c_2^2, \dots, x_2 c_i^1, c_{i+1}^2, \dots, x_2 c_j^1, c_{j+1}^2, \dots, c_{mn+1}^2\} \quad (5.19)$$

Modified Averaging Crossover (MACX): Averaging crossover is a commonly used operator in real-coded genetic algorithms that generates new solutions by averaging the two parents. The modified two new solutions H_1 and H_2 are

$$H_1 = C_1 + \beta (C_1 - C_2)$$

$$H_2 = C_2 + \beta (C_2 - C_1)$$

where β is a scalar value in the range of $(0 \leq \beta \leq 1)$.

Hybrid Crossover (HCX): We have presented four types of crossover operators. The performance of these operators in terms of convergence to an optimal solution depends on the given problem. One type of crossover operator that performs well for one problem may not perform well for another problem. Hence, many research works have studied the effect of combining crossover operators in a genetic algorithm [68, 168] for a given problem. Hybrid crossovers are a simple way of combining different crossover operators. The hybrid crossover operator uses different types of crossover operators to produce diverse offspring from the same parents. The hybrid crossover operator presented in this study generates eight offsprings for each pair of parents via the MSCX, MTPX, MUCX and MACX crossover operators. The most promising offsprings of the eight substitute their parents in the population.

Mutation Operator: The mutation operator alters one solution to produce a new solution. The mutation operator is needed to ensure diversity in the population and to overcome the premature convergence and local minima problems. Note that the mutation operators are modified such that they satisfy the equality constraints given in Equation 5.10. We describe different mutation operators used in this study.

Swap Mutation (SM): Let C_1 be the solution selected for the mutation operation. This operator first randomly selects two mutation points i and j . The new solution (H_1) is generated by swapping the values at these mutation points.

$$C_1 = \{c_1^1, c_2^1, \dots, c_i^1, c_{i+1}^1, \dots, c_j^1, c_{j+1}^1, \dots, c_{mn+1}^1\} \quad (5.20)$$

$$H_1 = \{c_1^1, c_2^1, \dots, c_j^1, c_{i+1}^1, \dots, c_i^1, c_{j+1}^1, \dots, c_{mn+1}^1\} \quad (5.21)$$

Consider the two-worker-role ($m = 2$), two-round ($n = 2$) system. Let C_1 be the solution selected for the mutation operation, and the mutation sites (i and j) are shown in bold.

$$C_1 = [40 \textbf{20} 10 \textbf{15} 15]$$

The new solution (H_1) is generated by swapping the values at these mutation points.

$$H_1 = [40 \textbf{15} 10 \textbf{20} 15]$$

Random Zero Mutation (RZM): Let C_1 be the solution selected for the mutation operation. This operator first randomly selects one mutation point i . The new solution (H_1) is generated by making the value at this mutation point zero and distributes this value to other elements in the solution equally.

$$C_1 = \{c_1^1, c_2^1, \dots, c_i^1, c_{i+1}^1, \dots, c_j^1, c_{j+1}^1, \dots, c_{mn+1}^1\} \quad (5.22)$$

$$H_1 = \{x + c_1^1, \dots, 0, x + c_{i+1}^1, \dots, x + c_j^1, \dots, x + c_{mn+1}^1\} \quad (5.23)$$

where $x = \frac{c_i^1}{mn+1}$. The random zero mutation operator is useful because one element in the string is made zero. For example, if the element in the string corresponding to $\beta_{3,4}$ is made zero, then the worker role w_3 is not participating in round 4 of load distribution. This operator also increases the rate of convergence of the algorithm.

Fitness Function: The calculation of the fitness function is easy. The string gives the load fractions β_0 and $\beta_{i,j}$ $i = 1, \dots, m$, where $j = 1, \dots, n$. Once the load fractions are known, the finish time of all worker roles (T_i) can be easily obtained. The processing time of the entire processing load T is $\max(T_i, \forall i = 0, 1, \dots, m)$. Because the genetic algorithm maximizes the fitness function, the fitness is defined as the negative of the processing time (T).

$$F = -T \quad (5.24)$$

Termination Criteria: In the genetic algorithm, the evolutionary process continues until a termination criterion is satisfied. The maximum number of generations is the most widely used termination criterion and is used in our simulation studies.

In general, a hybrid genetic algorithm applied to our problem follows the steps shown in Figure 5.3:

5.4 Simulation results

The hybrid genetic algorithm discussed in the previous section was implemented in MATLAB on a 3.0GHz Pentium IV machine with 512MB of RAM. The hybrid genetic algorithm parameters used in our simulations are obtained by trial and error and are fixed for all the simulations. We have conducted many numerical simulations for various values of m , n , G_i , A_i , and g_i . We present numerical examples to demonstrate how this methodology works.

Numerical Example 1. Consider a six-worker-role ($m = 6$) system. The inverse processing speed parameters are $A_0 = 15$, $A_1 = 1.5$, $A_2 = 1.4$, $A_3 = 1.3$, $A_4 = 1.2$, $A_5 = 1.1$ and $A_6 = 1.0$, and the inverse transmitting speed parameters are $G_1 = 0.3$, $G_2 = 0.4$, $G_3 = 0.2$, $G_4 = 0.1$, $G_5 = 0.35$ and $G_6 = 0.1$. The communication latencies are $g_1 = 1$, $g_2 = 2$, $g_3 = 5$, $g_4 = 1.5$, $g_5 = 1.1$ and $g_6 = 3.5$. The distribution order for the worker role with data (w_0) is $\{w_1, w_2, \dots, w_6\}$, and the total processing load (J) is 100.

This example was solved using a hybrid genetic algorithm for 6 rounds with all 6 worker roles. The load fractions assigned to each worker role in each round obtained from the hybrid genetic algorithm are given in Table 5.4. From Table 5.4, we see that the load fractions assigned to worker role w_3 are zero in all rounds. This implies that worker role w_3 is not participating in any of the rounds, and thus, the optimal number of worker roles is 5. Similarly, we see that the load fractions assigned to all the worker roles in round 6 are zero. This implies that this round is not used. Hence, the optimal number of rounds is 5.

Figure 5.3: Flow chart for HGA

Now, we will explain how a genetic algorithm gives the optimal number of rounds and the optimal number of worker roles. The genetic algorithm is initialized with a population of solutions (the load fractions to the worker roles in each round) to the problem. Every solution in the population is used to find the processing time (T). Based

on the processing time, a fitness is assigned to every solution. Based on the fitness, new solutions are obtained using genetic operators. Hence, the genetic algorithm searches all possible solutions to the problem. The solutions other than the solution with the optimal number of rounds and the optimal number of worker roles are discarded because their processing time is greater than the processing time for the optimal number of rounds and worker roles. Therefore, the genetic algorithm converges to the optimal number of rounds and the optimal number of worker roles. This is possible because the only information needed by the genetic algorithm is the objective function value (processing time T). The objective function is easy to calculate once the load fractions are known. It is also easy in a genetic algorithm to make any particular load fraction ($\beta_{i,j}$) zero (by using the random zero mutation) and obtain the processing time. Any idle time for the worker roles in the computation are included in the finish time expression. In addition, we can see that rules 2 and 3 of the load distribution are automatically satisfied. This can be verified from the obtained load fractions in that there is no idle time for worker roles and all worker roles stop computing simultaneously.

Table 5.4: Processing Time for Numerical Example 1

Total Install.	Install. No. (i)	Load Fractions							T
		β_0	$\beta_{1,i}$	$\beta_{2,i}$	$\beta_{3,i}$	$\beta_{4,i}$	$\beta_{5,i}$	$\beta_{6,i}$	
6	1	2.48196	0	0	0	19.13122	0	27.56029	37.22944
	2		10.19143	0	0	0	11.72740	0	
	3		0	7.58627	0	9.04905	0	0	
	4		5.47714	0	0	0	4.90739	0	
	5		0	1.88785	0	0	0	0	
	6		0	0	0	0	0	0	

How does one obtain the optimal distribution order from the load fractions given in Table 5.4 for the distribution order $\{w_1, w_2, \dots, w_6\}$?

Let the optimal distribution order be $\{*, *, *, *, *, *\}$. Here, $*$ denotes that the worker role w_i for this position is currently not known. From Table 5.4, the worker roles participating in round 1 are w_4 and w_6 . Therefore, in the optimal distribution order, the load is first distributed to worker role w_4 and then to worker role w_6 . Now, the optimal distribution order is $\{w_4, w_6, *, *, *, *\}$. The worker roles participating in the second round are w_1 and w_5 . After activating worker role w_6 , worker role w_1 is activated, followed by

Table 5.5: Numerical Example 1: Optimal Distribution Order

Total Install.	Install. No. (i)	Load Fractions							T
		β_0	$\beta_{4,i}$	$\beta_{6,i}$	$\beta_{1,i}$	$\beta_{3,i}$	$\beta_{5,i}$	$\beta_{2,i}$	
2	1	2.48196	19.13122	27.56029	10.19143	0	11.72740	7.58627	37.22944
	2		9.04905	0	5.47714	0	4.90739	1.88785	

worker role w_5 in the optimal distribution order. Now, the optimal distribution order is $\{w_4, w_6, w_1, w_5, *, *\}$. The worker roles participating in the third round are w_2 and w_4 . Of these worker roles, w_4 is already activated. Therefore, after activating worker role w_5 , activate worker role w_2 in the optimal distribution order. Now, the optimal distribution order is $\{w_4, w_6, w_1, w_5, w_2, *\}$. Worker role w_3 is not participating in any of the rounds, and thus, it is considered to be removed from the network. Therefore, the optimal distribution order is $\{w_4, w_6, w_1, w_5, w_2\}$.

We used this optimal distribution order ($\{w_4, w_6, w_1, w_5, w_2\}$) in our hybrid genetic algorithm. The load fractions and the processing time obtained from this optimal distribution order in 2 rounds of load distribution are given in Table 5.5. We can see that the results given in Table 5.5 are the same as those obtained for 6 rounds of load distribution for the distribution order $\{w_1, w_2, w_3, w_4, w_5, w_6\}$ given in Table 5.4. Hence, for this example, the optimal distribution order is $\{w_4, w_6, w_1, w_5, w_2\}$, the optimal number of worker roles is 5, and the optimal number of rounds of load distribution is 2. This example clearly shows that the hybrid genetic algorithm gives the optimal number of worker roles, the optimal number of rounds of load distribution, and the optimal distribution order for a general m -worker-role n -round load distribution system.

The processing time for the entire processing load over 2 rounds of load distribution for the distribution order $\{w_1, w_2, w_3, w_4, w_5, w_6\}$ is 38.911. The processing time for the entire processing load over 2 rounds of load distribution for the optimal distribution order $\{w_4, w_6, w_1, w_5, w_2\}$ is 37.22944.

Convergence Study. We now conduct a convergence study of the proposed Hybrid Genetic Algorithm (HGA) using the optimal distribution order $\{w_4, w_6, w_1, w_5, w_2\}$ and optimal number of rounds (2 rounds) of load distribution. The proposed HGA is called 30 times, and the mean and standard deviation of the converged results with respect to the optimal processing time of 37.22944 is reported in Table 5.6. A similar study

Table 5.6: Comparison of Convergence of Different Population-Based Algorithms

	HGA	RCGA	PSO
Mean	1.5317	8.5721	4.4715
STD	1.231	4.237	3.521

Table 5.7: Processing Time for Numerical Example 2

Total Install Rounds.	Install. No. (i)	Load Fractions					T
		β_0	$\beta_{1,i}$	$\beta_{2,i}$	$\beta_{3,i}$	$\beta_{4,i}$	
2	1	3.1602	4.1050	0	7.8859	10.453	47.40288
	2		21.691	0	26.883	25.823	

has been conducted using a simple Real-coded Genetic Algorithm (RCGA) with random initialization and well-known Particle Swarm Optimization (PSO) [142, 97]. The number of solutions, the number of iterations and the initial populations are kept constant among these three algorithms. The PSO algorithm parameters are selected as suggested in [97]. From the table, we can clearly see that the proposed HGA converges to a closer optimum than do the RCGA and PSO algorithms. HGA obtains better convergence due to the modified operators, which produce valid solutions.

Numerical Example 2. We now consider a case with $A_0 = 15$, $A_1 = 1.8$, $A_2 = 2.4$, $A_3 = 1.3$, $A_4 = 1.2$, $G_1 = 0.2$, $G_2 = 2.05$, $G_3 = 0.15$, $G_4 = 0.15$, $g_1 = 0.15$, $g_2 = 4$, $g_3 = 0.05$, and $g_4 = 0.1$ and total processing load J of 100. The distribution order for the worker role with data (w_0) is $\{w_1, w_2, w_3, w_4\}$, and number of rounds is 2.

The load fractions assigned to the worker roles in all rounds and the total load processing time are given in Table 5.7. From the table, we can observe that the worker role w_2 is not participating in all rounds, and hence, it is removed from the load distribution process.

5.5 Hybrid genetic algorithm for VCDS with multiple data banks problem and numerical example

We apply HGA to solve the load scheduling problem that we mentioned in Chapter 3 for the case of VCDS with multiple data banks and single installment using a DLT scheduling algorithm. To this end, we need to change the fitness function that we used in HGA for

the multi-installment case by introducing the arbitrary constraints λ_1 and λ_2 , where λ_1 is for the release time constraints and λ_2 is for the remaining equality constraints. Therefore, the new fitness function is as in Eq. 5.25:

$$F = -(T + \lambda_1 \cdot \sum_{i=1}^{N-1} \text{release time constraints violation} + \lambda_2 \cdot \sum_{i=1}^{M \cdot N - N + 2} \text{equality constraints violation}) \quad (5.25)$$

This equation means that the valid solution needs to fulfill all the DLT scheduling algorithm constraints in Chapter 3 and obtain the minimum processing time. The remainder of the HGA algorithm remains unchanged. We provide a numerical example for the HGA for multiple data banks using the numerical example 1 parameters from Chapter 3 as follows.

Numerical Example 3. Consider a virtual distributed computing environment with three worker roles $M = 3$ and two data banks $N = 2$. The inverse processing speed parameters of the worker roles are $A_1 = 4$, $A_2 = 5$, and $A_3 = 6$. The inverse transmitting speed parameters of the data banks are $G_1 = 0.2$ and $G_2 = 0.4$. The release times of the data banks are $R_1 = 10$ and $R_2 = 50$. Without loss of generality, we assume that the entire processing load ($J = 100$) is stored in data banks before the start of the computation process. For HGA parameters $\lambda_1 = 50$ and $\lambda_2 = 20$, our simulation runs for 500 generations, where each generation has a population of 25. Using HGA, we obtain the best solution, which is presented in Table 5.8. Then, based on the results, we drawn the processing timing diagram in Figure 5.4. Comparing the diagram with numerical example 1 in Chapter 3 with the DLT scheduling algorithm results and the timing diagram, the difference can be ignored. Finally, we show the diagram of the convergence in Figure 5.5. From this diagram, we can say that HGA with the new fitness function performs well for VCDS in the case of multiple data banks and a single installment.

Table 5.8: Processing Time for Numerical Example 3

Total processing time: $T_0 = 173.9964$			
D_j, W_i	W_1	W_2	W_3
D_1	$\beta_{11} = 9.9999$	$\beta_{21} = 10.0751$	$\beta_{31} = 9.5481$
D_2	$\beta_{12} = 30.9424$	$\beta_{22} = 22.3241$	$\beta_{32} = 17.11$

Figure 5.4: Timing diagram for numerical example 3

Figure 5.5: Numerical example 3 HGA converge diagram

5.6 Summary

In this chapter, the problem of data partitioning over multi-rounds of load distribution with an affine communication model was considered and was demonstrated to be a difficult optimization problem. For the problem of finding optimal load fractions for a given distribution order and number of worker roles, the number of rounds is solved for using the hybrid genetic algorithm. The chapter proposes modified crossover and mutation

operators that produce valid solutions satisfying the equality constraints. Further, the proposed hybrid genetic algorithm uses three different population initialization schemes to achieve better convergence. Using the load fractions, a heuristic approach has been presented to find the optimal distribution order, optimal number of worker roles m^* ($m^* \leq m$), and optimal rounds of load distribution n^* ($n^* \leq n$) such that the processing time of the entire processing load is minimized. The numerical simulation results clearly indicate that the proposed approach finds the best solution for a multi-round load distribution problem. By updating the fitness function, HGA also works well for VDCS with multiple data banks and single installment. The numerical example shows that the best solution has been obtained and that the process converges properly. Concerning the result comparison between HGA and the DLT-based scheduling algorithm in Chapter 3, the differences are very small and can be ignored.

Chapter 6

Conclusion and future directions

6.1 Thesis Summary

The objective of this thesis was to develop a load scheduling algorithm for a virtual distributed computing system (VDCS) with multiple data banks. The major contributions are summarized below.

- Development of a DLT-based load scheduling algorithm in VDCS with multiple data banks wherein the total processing time is minimized.
- Development of dynamic rescheduling algorithm to address the effective utilization of worker roles.
- Hybrid-genetic-algorithm-based load scheduling to address multiple installments with a single data bank.
- Hybrid-genetic-algorithm-based load scheduling to address single installment with multiple data banks.
- Experimental evaluation using a satellite image classification problem.

We shall now describe each of the contributions, in detail, below.

6.1.1 DLT-based load scheduling algorithm

In this thesis, a DLT-based load scheduling algorithm was developed for VDCS with multiple data banks. The algorithm can achieve the minimum total processing time in

VDCS with multiple data banks (N) to guarantee that all the worker roles (M) stop processing assigned load fractions simultaneously. The novel load scheduling algorithm divided the total load into fractions for each worker role to process, and each worker role retrieves the assigned load fractions from multiple data banks. The optimized load fractions for each worker role have been achieved by considering features of assigned resources and realistic constraints such as the computational capacity of worker roles, the communication capacity between worker roles and data banks, data bank release times, total processing time constraints, release time constraints, total load constraints, and continuity constraints.

Total processing time constraint (T_0): For each worker role, the first data bank release time plus the communication time for previous worker roles and the total assigned load computation time for the worker role should be less than or equal to the total processing time. This constraint guarantees that all the worker roles for the task stop computing simultaneously.

Release time constraint (R_j): We introduce the release time constraint by considering data bank availability. The release time constraint is enforced such that the load fraction size for the first worker role fetched from each data bank guarantees a continuous data computing process.

Total load constraint (J): The sum of all the load fractions for each worker role retrieved from each data bank should be equal to the total load. This constraint guarantees that, at the end, the entire load has been completely processed in VDCS.

Continuity constraint: This refers to having a continuous processing and fetching time relationship between adjacent pairs of worker roles and data banks. This constraint is recursively extended to all pairs of worker roles and data banks. Introducing this constraint ensures that the transmission and processing of the load fractions for each worker role is continuous and ensures a high utilization rate of the assigned resources for processing loads.

Employing all these constraints in the algorithm, the optimization of the processing load fraction problem has been converted into a linear programming problem and achieves the minimum total processing time with the highest assigned resource utilization rate.

6.1.2 DLT-based load dynamic rescheduling algorithm

An extended DLT-based algorithm for solving VDCS with multiple data banks, referred to as the DLT-based dynamic rescheduling algorithm, was proposed. This algorithm is mainly based on our VDCS with the multiple-data-bank model for the DLT-based load scheduling algorithm and is used to achieve the minimum total processing time. For our algorithm, new notations were introduced to describe the load processing status at the time a new resource is added or an assigned resource is dropped from the current scheduling resource pool at the rescheduling start time (T_{RS}). To avoid remaining load fraction transmission redundancy, the worker role release time was introduced to the algorithm. As a result, the ongoing load transmission process should be completed before the rescheduling start time. Because it is predictable, it is going to be considered as the unavailable time of the worker role. Then, we separate the condition based on the number of worker roles that fetched load fractions and on the number of worker roles with full load fractions transmitted. The extended DLT-based dynamic rescheduling algorithm is then developed by including the new system's current status parameters in the constraints of the DLT-based load scheduling algorithm. Using this algorithm, the problem of load scheduling for VDCS with multiple data banks with dynamic resource assigning features and individual resource failure can be solved to achieve task completion with minimal total load processing time. The newly released resource can immediately be reassigned to other tasks during processing. From the overall system point of view, this technique can help to achieve a high utilization rate for all resources as well.

6.1.3 HGA-based load scheduling algorithm for multiple installations

In this thesis, a hybrid genetic algorithm was proposed to resolve the issue of data partitioning in multi-rounds of load distribution in single-level tree network environments with the affine communication model. To solve the data partitioning problem efficiently and obtain a minimum total processing time, a Hybrid crossover operator was introduced to the algorithm to achieve fast convergence and satisfy total load constraints. The introduced hybrid crossover operators are the Modified Two-Point Crossover (MTPX), Modified Simple Crossover (MSCX), Modified Uniform Crossover (MUCX) and Hybrid

Crossover (HCX). The convergence performance in obtaining the optimal solution of these four crossover operators depends on the problem. Under the multi-installment load distribution strategy, it is shown that there exists an optimal load distribution sequence, an optimal number of worker roles $m^*(m^* \leq m)$, and an optimal number of installments $n^*(n^* \leq n)$ for obtaining a minimum processing time. This is a difficult optimization problem because of the given load distribution sequence. In our HGA, the worker roles were identified in every installment based on the processing time. Thus, the convergence of the algorithm to an optimal solution ensures minimum total load processing time.

6.1.4 HGA-based load scheduling algorithm for multiple data banks

The HGA was extended to solve the load scheduling problem of VDCS with multiple data banks and single installments. The new fitness function involves two new arbitrary constraints λ_1, λ_2 to consider all four constraints from the DLT-based load scheduling algorithm. By filtering the solutions with the new fitness function, the optimal solution can satisfy all the DLT-based load scheduling constraints. The optimal solution of the evaluation example demonstrates that the HGA with the novel fitness function can solve the problem of multiple-data-bank load scheduling. The minimum total processing time is achievable with optimal load fractions for each worker role.

6.1.5 Experiment using satellite image processing

In evaluating the proposed DLT-based scheduling algorithm, we established a virtual distributed computing laboratory environment with 6 worker roles and 2 data banks. HP servers and workstations were set up as data banks and worker roles. Our study case was a high-resolution multi-spectral Landsat 7 Thematic Mapper image portion covering $50 \times 50.75km^2$. The satellite image classifier model is contained in the worker roles, and the algorithm uses online-learning neural classifiers [10]. The communication and computation parameters are based on actual system unit load tests averaged over 100 repetitions. The experimental results show that the offset between the experimental and analytical times is caused by communication and data reformatting and synchronization.

6.2 Future directions

In the future, our research directions consist of implementing our current algorithm in a real VDCS system with multiple data banks and improving our algorithm to address multiple installments with different priorities for queues for divisible load processing tasks.

6.2.1 Implementation in future commercial VDCS with multiple data banks

With increasing system interconnection bandwidth, we expect traditional large data systems to separate their computation functions with the data storage function from one node. Our VDCS with multiple data banks is going to be the model of a future large data system. On the other hand, current large data system scheduling managers focus on resource scheduling for multiple jobs but not on load scheduling. Once the multiple-data-source large data system is available, our proposed algorithms will represent a potential load scheduling solution. Subsequently, we envision adding our proposed algorithm to future large data systems with multiple data banks such as Hortonworks or Hadoop.

6.2.2 Multi-installment in VDCS with multiple data banks and priority task queues

In our framework, designing the multi-installment scheduling optimization was a challenge due to the presence of multiple data banks in the virtual distributed computing environment. Our discussion thus mainly focused on the single-installment case with multiple data banks and using a hybrid genetic algorithm to optimize the scheduling under a multi-installment case and under a single data bank / multiple data banks with single installment case in VDCS. The scheduling is simply based on a FIFO algorithm for task queuing, and the task is processed individually with the current scheduling algorithm under the assumption that the resource assignment for the single-installment case has been performed properly and that the quantity of the resources is optimized for the individual tasks.

In general, the VDCS environment includes multiple data banks, among which the web role has multiple tasks, and each task has multiple installments. The web roles

execute their processing based on their priority levels and locations, which helps to assign an optimized quantity of resources to each task. There is a need to optimize the load fractions for each installment. We can use queuing theory and finite state machines along with a hybrid genetic algorithm, as discussed earlier, to address multi-installment scheduling to simulate real-world cases with multiple data banks under our framework. Because the number of data banks is limited, compared to single-installment scheduling, task queuing optimization needs to be considered for the multi-installment case. Because the resources are finite, the optimized amount of resources for each round of installment in the multi-installment case will contribute to achieving the overall VDCS resource optimization. These pre-scheduling elements should be added to the current single-installment DLT-based scheduling algorithm. With these scheduling elements, the current framework can be made suitable for actual VDCS. To further optimize the resources assigned to individual tasks, the task locations should be specified to assist in choosing the worker roles and data banks that are physically close to each other along with the fastest interconnection network paths. This will help to reduce the long-distance load transmissions between worker roles and data banks. This will also help to decrease the transmission cost significantly and improve system performance by shortening the total processing time. Based on task priorities, we can apply these data to build different queues for different tasks. In this way, the important tasks will be processed early, and the performance of the system is guaranteed based on the priorities submitted by users and the cost. Figure 6.1 illustrates the case of a multi-installment scheduling model in a realistic virtual distributed computing environment.

Figure 6.1: Multi-installment load scheduling in a virtual distributed compute system

Author's Publications

Journals

1. S. Suresh, Huang Hao, H.J. Kim, "Real-coded genetic algorithm for data partitioning in multi-round load distribution and scheduling in heterogeneous systems", *Applied Soft Computing*, vol 24, Pages 500-510, November 2014.
2. S. Suresh, Huang Hao, H.J. Kim, "Scheduling in Compute Cloud With Multiple Data Banks Using Divisible Load Paradigm", *IEEE Transactions on Aerospace and Electronic systems*, vol 51,no 2, Pages 1288-1297, April 2015.
3. Huang Hao, S. Suresh, "Divisible Load Theory based rescheduling algorithm for compute cloud with multiple data centers", *Journal of Supercomputing*, 2015. (submitted)

Bibliography

- [1] Beginner guide. [online]https://www.hpc2n.umu.se/support/beginners_guide, 2015.
- [2] Cloud computing. [online]http://en.wikipedia.org/wiki/Cloud_computing, 2015.
- [3] Parallel sysplex. [online]<http://www-03.ibm.com/systems/z/advantages/pso>, 2015.
- [4] R. Agrawal and H. Jagadish. Partitioning techniques for large-grained parallelism. *Computers, IEEE Transactions on*, 37(12):1627–1634, Dec 1988.
- [5] W. Alds, W. Dzwinel, J. Kitowski, J. Moscinski, and D. Yuen. Penetration mechanics via molecular dynamics. Technical report, University of Minnesota Supercomputing Institute, 1993. Research Report UMSI 93/58.
- [6] D. T. Altilar and Y. Paker. An optimal scheduling algorithm for parallel video processing. In *Multimedia Computing and Systems, 1998. Proceedings. IEEE International Conference on*, pages 245–248, Jun 1998.
- [7] D. T. Altilar and Y. Paker. Optimal scheduling algorithms for communication constrained parallel processing. In B. Monien and R. Feldmann, editors, *Euro-Par 2002 Parallel Processing*, volume 2400 of *Lecture Notes in Computer Science*, pages 197–206. Springer Berlin Heidelberg, 2002.
- [8] Y. Amir, B. Awerbuch, A. Barak, R. Borgstrom, and A. Keren. An opportunity cost approach for job assignment in a scalable computing cluster. *Parallel and Distributed Systems, IEEE Transactions on*, 11(7):760–768, Jul 2000.

- [9] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: a berkeley view of cloud computing. *University of California, Berkeley, Tech. Rep. No. UCB/EECS-2009028*, 2009.
- [10] G. Babu and S. Suresh. Sequential projection-based metacognitive learning in a radial basis function network for classification problems. *Neural Networks and Learning Systems, IEEE Transactions on*, 24(2):194–206, 2013.
- [11] G. Barlas. *Compression Algorithms for One-Dimensional Semiperiodical Biomedical Signals and Methods for Their Parallelization*. Ph.d. thesis, National Technical University of Athens, 1996.
- [12] G. Barlas. Collection-aware optimum sequencing of operations and closed-form solutions for the distribution of a divisible load on arbitrary processor trees. *Parallel and Distributed Systems, IEEE Transactions on*, 9(5):429–441, May 1998.
- [13] S. Bataineh and T. G. Robertazzi. Bus-oriented load sharing for a network of sensor driven processors. *IEEE Transactions on Systems, Man and Cybernetics*, 21(5):1202–1205, 1991.
- [14] O. Beaumont, H. Casanova, A. Legrand, Y. Robert, and Y. Yang. Scheduling divisible loads on star and tree networks: results and open problems. *Parallel and Distributed Systems, IEEE Transactions on*, 16(3):207–218, March 2005.
- [15] O. Beaumont, A. Legrand, and Y. Robert. Optimal algorithms for scheduling divisible workloads on heterogeneous systems. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, page 14, April 2003.
- [16] A. Benoit, M. Hakem, and Y. Robert. Contention awareness and fault-tolerant scheduling for precedence constrained tasks in heterogeneous systems. *Parallel Comput.*, 35(2):83–108, 2009.
- [17] V. Bharadwaj and G. Barlas. Scheduling divisible loads with processor release times and finite size buffer capacity constraints in bus networks. *Cluster Computing*, 6(1):63–74, 2003.

- [18] V. Bharadwaj, G. Debasish, M. Venkataraman, and T. G. Robertazzi. *Scheduling Divisible Loads in Parallel and Distributed Systems*. Los Alamitos, CA: IEEE Computer Society Press, 1996.
- [19] V. Bharadwaj, D. Ghose, and V. Mani. Multi-installment load distribution strategy in linear networks with communication delays. *1st IWPP. Parallel Processing. Proceedings of the First International Workshop on Parallel Processing (IWPP-94).*, pages 563 – 568, 1994.
- [20] V. Bharadwaj, D. Ghose, and V. Mani. Multi-installment load distribution in tree networks with delays. *Aerospace and Electronic Systems, IEEE Transactions on*, 31(2):555–567, April 1995.
- [21] V. Bharadwaj, D. Ghose, and V. Mani. Theoretical and practical aspects of multi-installment distribution for the processing of multiple divisible loads on bus networks. *Proceedings of ICCI 2000*, 2000.
- [22] V. Bharadwaj, D. Ghose, and T. G. Robertazzi. Divisible load theory: A new paradigm for load scheduling in distributed systems. *Cluster Computing*, 6(1):7–17, Jan. 2003.
- [23] V. Bharadwaj and Y. Jingnan. Divisible load scheduling strategies on distributed multi-level tree networks with communication delays and buffer constraints. *Comput. Commun.*, 27(1):93 – 110, Jan. 2004.
- [24] V. Bharadwaj, H. Li, and T. Radhakrishnan. Scheduling divisible loads in bus networks with arbitrary processor release times. *Computers & Mathematics with Applications*, 32(7):57 – 77, 1996.
- [25] V. Bharadwaj, X. Li, and C. C. Ko. Efficient partitioning and scheduling of computer vision and image processing data on bus networks using divisible load analysis. *Image and Vision Computing*, 18(11):919 – 938, 2000.
- [26] V. Bharadwaj, X. Li, and C. C. Ko. On the influence of start-up costs in scheduling divisible loads on bus networks. *Parallel and Distributed Systems*, 11(12):1288–1305, 2000.

- [27] V. Bharadwaj and W. H. Min. Scheduling divisible loads on heterogeneous linear daisy chain networks with arbitrary processor release times. *Parallel and Distributed Systems, IEEE Transactions on*, 15(3):273 – 288, March 2004.
- [28] V. Bharadwaj and R. Surendra. Theoretical and experimental study on large size image processing applications using divisible load paradigm on distributed bus networks. *Image and Vision Computing*, 20(13-14):917–935, 2002.
- [29] V. Bharadwaj, L. Xiaolin, and C. C. Ko. Design and analysis of load distribution strategies with start-up costs in scheduling divisible loads on distributed networks. *Mathematical and Computer Modelling*, 32(7-8):901–932, 2000.
- [30] Z. Bingul. Adaptive genetic algorithms applied to dynamic multiobjective problems. *Applied Soft Computing*, 7(3):791 – 799, 2007.
- [31] T. E. Carroll and D. Grosu. A strategyproof mechanism for scheduling divisible loads in bus networks without control processors. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 1–8, April 2006.
- [32] T. E. Carroll and D. Grosu. A strategyproof mechanism for scheduling divisible loads in linear networks. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–9, March 2007.
- [33] T. E. Carroll and D. Grosu. An incentive-compatible mechanism for scheduling non-malleable parallel jobs with individual deadlines. In *Parallel Processing, 2008. ICPP '08. 37th International Conference on*, pages 107–114, Sept 2008.
- [34] S. Chan, V. Bharadwaj, and D. Ghose. Large matrix| vector products on distributed bus networks with communication delays using the divisible load paradigm: performance analysis and simulation. *Mathematics and Computers in Simulation*, 58(1):71–92, 2001.
- [35] P.-C. Chang, S.-H. Chen, C.-Y. Fan, and C.-L. Chan. Genetic algorithm integrated with artificial chromosomes for multi-objective flowshop scheduling problems. *Applied Mathematics and Computation*, 205(2):550 – 561, 2008. Special Issue on Advanced Intelligent Computing Theory and Methodology in Applied Mathematics and Computation.

- [36] P.-C. Chang, S.-S. Chen, and C.-Y. Fan. Mining gene structures to inject artificial chromosomes for genetic algorithm in single machine scheduling problems. *Applied Soft Computing*, 8(1):767 – 777, 2008.
- [37] Y.-K. Chang, J.-H. Wu, C.-Y. Chen, and C.-P. Chu. Improved methods for divisible load distribution on k-dimensional meshes using multi-installment. *Parallel and Distributed Systems, IEEE Transactions on*, 18(11):1618–1629, Nov 2007.
- [38] Y.-M. Chen and W.-C. Wang. An adaptive rescheduling scheme based heuristic algorithm for cloud services applications. In *Machine Learning and Cybernetics (ICMLC), 2011 International Conference on*, volume 3, pages 961–966, July 2011.
- [39] Y. C. Cheng and T. G. Robertazzi. Distributed computation with communication delay distributed intelligent sensor networks. *IEEE Transactions on Aerospace and Electronic Systems*, 24(6):700–712, 1988.
- [40] Y. C. Cheng and T. G. Robertazzi. Distributed computation for a tree network with communication delays. *IEEE Transactions on Aerospace and Electronic Systems*, 26(3):511–516, May 1990.
- [41] K. Cooper, A. Dasgupta, K. Kennedy, C. Koelbel, A. Mandal, G. Marin, M. Mazina, J. Mellor-Crummey, F. Berman, H. Casganova, A. Chien, H. Dail, X. Liu, A. Olugbile, O. Sievert, H. Xia, L. Johnsson, B. Liu, M. Patel, D. Reed, W. Deng, C. Mendes, Z. Shi, A. YarKhan, and J. Dongarra. New grid scheduling and rescheduling methods in the grads project. In *Proceedings of 18th International on Parallel and Distributed Processing Symposium*, page 199, 2004.
- [42] D. Daniel and S. P. J. Lovesum. A novel approach for scheduling service request in cloud with trust monitor. In *International Conference on Signal Processing, Communication, Computing and Networking Technologies*, pages 509–513, 2011.
- [43] G. B. Dantzig, A. Orden, and P. Wolfe. The generalized simplex method for minimizing a linear form under linear inequality restraints. *Pacific J. Math.*, 5(2):183–195, 1955.

- [44] K. Dasgupta, B. Mandal, P. Dutta, J. K. Mandal, and S. Dam. A genetic algorithm (ga) based load balancing strategy for cloud computing. *Procedia Technology*, 10(0):340–347, 2013. First International Conference on Computational Intelligence: Modeling Techniques and Applications (CIMTA) 2013.
- [45] C. Diaz, M. Guzek, J. Pecero, P. Bouvry, and S. Khan. Scalable and energy-efficient scheduling techniques for large-scale systems. In *Computer and Information Technology (CIT), 2011 IEEE 11th International Conference on*, pages 641–647, 2011.
- [46] M. Drozdowski. Selected problems of scheduling tasks in multiprocessor computer systems. *Wydawnictwa Politechniki Poznanskiej*, 1997.
- [47] M. Drozdowski. *Scheduling for Parallel Processing*. Springer, London, 2009.
- [48] M. Drozdowski and M. Lawenda. Multi-installment divisible load processing in heterogeneous systems with limited memory. In R. Wyrzykowski, J. Dongarra, N. Meyer, and J. Waniewski, editors, *Parallel Processing and Applied Mathematics*, volume 3911 of *Lecture Notes in Computer Science*, pages 847–854. Springer Berlin Heidelberg, 2006.
- [49] M. Drozdowski and M. Lawenda. Scheduling multiple divisible loads in homogeneous star systems. *Journal of Scheduling*, 11(5):347–356, 2008.
- [50] M. Drozdowski and P. Wolniewicz. Multi-installment divisible job processing with communication start-up cost. *Foundations of Computing and Decision Sciences*, 27(1):43 – 57, 2002.
- [51] M. Drozdowski and P. Wolniewicz. Divisible load scheduling in systems with limited memory. *Cluster Computing*, 6(1):19–29, 2003.
- [52] M. Drozdowski and P. Wolniewicz. Out-of-core divisible load processing. *Parallel and Distributed Systems, IEEE Transactions on*, 14(10):1048–1056, Oct 2003.
- [53] M. Drozdowski and P. Wolniewicz. Optimum divisible load scheduling on heterogeneous stars with limited memory. *European Journal of Operational Research*, 172(2):545 – 559, 2006.

- [54] T. E. H. El-Gorashi and J. M. H. Elmirghani. Distributed storage scenario in a wide area wdm mesh architecture under heterogeneous traffic. In *International Conference on Optical Network Design and Modeling*, pages 1–6, 2009.
- [55] A. Gandhi, H. Akkary, R. Rajwar, S. T. Srinivasan, and K. Lai. Scalable load and store processing in latency tolerant processors. In *Proceedings of the 32Nd Annual International Symposium on Computer Architecture, ISCA '05*, pages 446 – 457, Washington, DC, USA, 2005. IEEE Computer Society.
- [56] N. Ganesh, V. Bharadwaj, and G. K. Sakthi. On handling large-scale polynomial multiplications in compute cloud environments using the divisible load paradigm. *IEEE Transactions on Aerospace and Electronics Systems*, 48(1):820 – 831, 2012.
- [57] Y. G. Ge and G. Y. Wei. Ga-based task scheduler for the cloud computing systems. In *International Conference on Web Information Systems and Mining*, pages 181–186, 2010.
- [58] S. Ghanbari, M. Othman, M. R. A. Bakar, and W. J. Leong. Multi-objective method for divisible load scheduling in multi-level tree network. *Future Generation Computer Systems*, (0), 2015.
- [59] D. Ghose and H. J. Kim. Load partitioning and trade-off study for large matrix-vector computations in multicast bus networks with communication delays. *Journal of Parallel and Distributed Computing*, 55(1):32 – 59, 1998.
- [60] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.
- [61] R. L. Grossman, G. Yunhong, M. Sabala, and Z. Wanzhi. The nist definition of cloud computing. *NIST Special Publication 800-145. NIST, US Department of Commerce*, 25(2):179–183, 2009.
- [62] D. Grosu and T. E. Carroll. A strategyproof mechanism for scheduling divisible loads in distributed systems. In *Parallel and Distributed Computing, 2005. ISPDC 2005. The 4th International Symposium on*, pages 83–90, July 2005.

- [63] G. Guo-ning, H. Ting-lei, and G. Shuai. Genetic simulated annealing algorithm for task scheduling based on cloud computing environment. In *Intelligent Computing and Integrated Systems (ICISS), 2010 International Conference on*, pages 60–63, Oct 2010.
- [64] E. Haddad. Runtime reallocation of divisible load under processor execution deadlines. In *Parallel and Distributed Real-Time Systems, 1995. Proceedings of the Third Workshop on*, pages 30–31, Apr 1995.
- [65] S. Haiyan, W. W. Kwok., N. Ming, and C. Shengyong. Adaptive indexed divisible load theory for wireless sensor network workload allocation. *International Journal of Distributed Sensor Networks*, page 18, 2013.
- [66] S. Haiyan., W. Wanliang., and K. Ngaiming. Energy dependent divisible load theory for wireless sensor network workload allocation. *Mathematical Problems in Engineering*, 2012:16, 2012.
- [67] W. Han Min and V. Bharadwaj. Aligning biological sequences on distributed bus networks: a divisible load scheduling approach. *Information Technology in Biomedicine, IEEE Transactions on*, 9(4):489 – 501, Dec 2005.
- [68] F. Herrera and M. Lozano. Gradual distributed hybrid genetic algorithms. *Evolutionary Computation, IEEE Transactions on*, 4(1):43 – 63, Apr 2000.
- [69] F. Herrera, M. Lozano, and A. Snchez. Hybrid crossover operators for real-coded genetic algorithms: an experimental study. *Soft Computing*, 9(4):280 – 298, 2005.
- [70] F. Herrera, M. Lozano, and J. Verdegay. Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artificial Intelligence Review*, 12(4):265 – 319, 1998.
- [71] J. H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, MA, USA, 1992.
- [72] C. R. Houck, J. A. Joines, and M. G. Kay. A genetic algorithm for function optimization: A matlab implementation. Technical report, 1996.

- [73] J. H. Hu, J. H. Gu, G. F. Sun, and T. H. Zhao. A scheduling strategy on load balancing of virtual machine resources in a cloud computing environment. In *the Third International Symposium on Parallel Architectures, Algorithms and Programming*, pages 89–96, 2010.
- [74] Q. Y. Huang and T. L. Huang. An optimistic job scheduling strategy based on qos for cloud computing. In *International Conference on, Intelligent Computing and Integrated Systems*, pages 673–675, 2010.
- [75] J. T. Hung and T. Robertazzi. Switching in sequential tree networks. *Aerospace and Electronic Systems, IEEE Transactions on*, 40(3):968 – 982, 2004.
- [76] J. T. Hung and T. Robertazzi. Scheduling nonlinear computational loads. *Aerospace and Electronic Systems, IEEE Transactions on*, 44(3):1169 – 1182, July 2008.
- [77] H. Hussain, S. U. R. Malik, A. Hameed, S. U. Khan, G. Bickler, N. Min-Allah, M. B. Qureshi, L. Zhang, W. Yongji, N. Ghani, J. Kolodziej, A. Y. Zomaya, C.-Z. Xu, P. Balaji, A. Vishnu, F. Pinel, J. E. Pecero, D. Kliazovich, P. Bouvry, H. Li, L. Wang, D. Chen, and A. Rayes. A survey on resource allocation in high performance distributed computing systems. *Parallel Computing*, 39(11):709 – 736, 2013.
- [78] B. Jacek and D. Maciej. Scheduling divisible jobs on hypercubes. *Parallel Computing*, 21(12):1945 – 1956, 1995.
- [79] B. Jacek and D. Maciej. Distributed processing of divisible jobs with communication startup costs. *Discrete Applied Mathematics*, 76(1-3):21–41, 1997. Second International Colloquium on Graphs and Optimization.
- [80] B. Jacek and D. Maciej. Heuristics for multi-round divisible loads scheduling with limited memory. *Parallel Computing*, 36(4):199 – 211, 2010.
- [81] B. Jacek, D. Maciej, and M. Mariusz. Divisible task scheduling | concept and verification1. *Parallel Computing*, 25(1):87 – 98, 1999.

- [82] J. Jenq and S. Sahni. Reconfigurable mesh algorithms for the hough transform. *Journal of Parallel and Distributed Computing*, 20(1):69 – 77, 1994.
- [83] G. Jeremy. Twenty-one experts define cloud computing. [online], Jan 2009.
- [84] J. Jia, V. Bharadwaj, and D. Ghose. Adaptive load distribution strategies for divisible load processing on resource unaware multilevel tree networks. *Computers, IEEE Transactions on*, 56(7):999 – 1005, July 2007.
- [85] S. Jin, G. Schiavone, and D. Turgut. A performance study of multiprocessor task scheduling algorithms. *The Journal of Supercomputing*, 43(1):77 – 97, 2008.
- [86] S. Jin, G. Schiavone, and D. Turgut. A performance study of multiprocessor task scheduling algorithms. *The Journal of Supercomputing*, 43(1):77 – 97, 2008.
- [87] K. Joanna, K. Samee Ullah, K.-D. S. A. M. Lizhe, Wang. Marek, Y. Z. Ewa. Niewiadomska-Szynkiewicz, Albert, and X. Cheng-Zhong. Security, energy, and performance-aware resource allocation mechanisms for computational grids. *Future Generation Computer Systems*, 31(0):77–92, 2014. Special Section: Advances in Computer Supported Collaboration: Systems and Technologies.
- [88] H. Kim and V. Mani. Divisible load scheduling in single-level tree networks: Optimal sequencing and arrangement in the nonblocking mode of communication. *Computers & Mathematics with Applications*, 46(10-11):1611–1623, 2003.
- [89] H. J. Kim. A novel optimal load distribution algorithm for divisible loads. *Cluster Computing*, 6(1):41–46, 2003.
- [90] H. J. Kim, T. Kim, and V. Mani. Scheduling divisible loads in non-blocking mode of communication: optimal sequencing and arrangement in a single-level tree network. In *Research and Development, 2002. SCORed 2002. Student Conference on*, pages 464–467, 2002.
- [91] C. kin Lee and M. Hamdi. Parallel image processing applications on a network of workstations. *Parallel Computing*, 21(1):137 – 160, 1995.

- [92] K. Ko and T. Robertazzi. Signature search time evaluation in flat file databases. *Aerospace and Electronic Systems, IEEE Transactions on*, 44(2):493 – 502, April 2008.
- [93] K. Ko and T. G. Robertazzi. Record search time evaluation using divisible load analysis. *Tech. Rep. 765, SUNY at Stony Brook College of Engineering and Applied Science*, 2012., 1998.
- [94] J. Kolodziej, S. U. Khan, and E.-G. Talbi. Scalable optimization in grid, cloud, and intelligent network computing | foreword. *Concurrency and Computation: Practice and Experience*, 25(12):1719–1721, 2013.
- [95] Y. C. Lee and A. Y. Zomaya. Rescheduling for reliable job completion with the support of clouds. *Future Generation Computer Systems*, 26(8):1192 – 1199, 2010.
- [96] Z. Lee, Y. Wang, and W. Zhou. A dynamic priority scheduling algorithm on service request scheduling in cloud computing. In *Electronic and Mechanical Engineering and Information Technology (EMEIT), 2011 International Conference on*, volume 9, pages 4665 – 4669, Aug 2011.
- [97] M.-S. Leu and M.-F. Yeh. Grey particle swarm optimization. *Applied Soft Computing*, 12(9):2985 – 2996, 2012.
- [98] H. Li and H. Li. A research of resource scheduling strategy for cloud computing based on pareto optimality m x n production model. In *Management and Service Science (MASS), 2011 International Conference on*, pages 1 – 5, Aug 2011.
- [99] K. Li, G. Xu, G. Zhao, Y. Dong, and D. Wang. Cloud task scheduling based on load balancing ant colony optimization. In *Chinagrid Conference (ChinaGrid), 2011 Sixth Annual*, pages 3 – 9, Aug 2011.
- [100] P. Li, B. Veeravalli, and A. Kassim. Design and implementation of parallel video encoding strategies using divisible load analysis. *Circuits and Systems for Video Technology, IEEE Transactions on*, 15(9):1098 – 1112, Sept 2005.

- [101] Q. Li and Y. K. Guo. Optimization of resource scheduling in cloud computing. In *International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 315–320, 2010.
- [102] X. Li, V. Bharadwaj, and C. Ko. Divisible load scheduling on single-level tree networks with buffer constraints. *Aerospace and Electronic Systems, IEEE Transactions on*, 36(4):1298 – 1308, 2000.
- [103] X. Li, V. Bharadwaj, and C. Ko. Scheduling divisible tasks on heterogeneous single-level tree networks with finite-size buffers. In *Proceedings of the Joint Conference on Information Sciences*, volume 5, pages 285 – 288, 2000.
- [104] X. Li, V. Bharadwaj, and C. Ko. *Experimental study on processing divisible loads for large size image processing applications using PVM clusters*. International Journal of Computers and Applications, ACTA Press, July 2001.
- [105] X. Li, V. Bharadwaj, and C. Ko. Distributed image processing on network of workstations. *International Journal of Computer and Applications*, 25(2):1–10, 2003.
- [106] X. L. Li, V. Bharadwaj, and C. C. Ko. Divisible load scheduling on a hypercube cluster with finite-size buffers and granularity constraints. In *First IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 660–667, 2001.
- [107] C. Lin and S. Lu. Scheduling scientific workflows elastically for cloud computing. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 746 – 747, July 2011.
- [108] X. Lin, Y. Lu, J. Deogun, and S. Goddard. Enhanced real-time divisible load scheduling with different processor available times. In S. Aluru, M. Parashar, R. Badrinath, and V. Prasanna, editors, *High Performance Computing / HiPC 2007*, volume 4873 of *Lecture Notes in Computer Science*, pages 308–319. Springer Berlin Heidelberg, 2007.

- [109] K. Liu, H. Jin, J. Chen, X. Liu, D. Yuan, and Y. Yang. A compromised-time-cost scheduling algorithm in swindow-c for instance-intensive cost-constrained workflows on a cloud computing platform. *International Journal of High Performance Computing Applications*, page 2010.
- [110] X. Liu, J. Chen, Z. Wu, Z. Ni, D. Yuan, and Y. Yang. Handling recoverable temporal violations in scientific workflow systems: A workflow rescheduling based strategy. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 534 – 537, May 2010.
- [111] G. Luo, Z. Qian, M. Dong, K. Ota, and S. Lu. Network-aware re-scheduling: Towards improving network performance of virtual machines in a data center. In X.-h. Sun, W. Qu, I. Stojmenovic, W. Zhou, Z. Li, H. Guo, G. Min, T. Yang, Y. Wu, and L. Liu, editors, *Algorithms and Architectures for Parallel Processing*, volume 8630 of *Lecture Notes in Computer Science*, pages 255 – 269. Springer International Publishing, 2014.
- [112] A. Mamat, Y. Lu, J. Deogun, and S. Goddard. Real-time divisible load scheduling with advance reservation. In *Real-Time Systems, 2008. ECRTS '08. Euromicro Conference on*, pages 37 – 46, July 2008.
- [113] A. Mamat, Y. Lu, J. Deogun, and S. Goddard. Scheduling real-time divisible loads with advance reservations. *Real-Time Systems*, 48(3):264 – 293, 2012.
- [114] P. Marshall, K. Keahey, and T. Freeman. Elastic site: Using clouds to elastically extend site resources. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 43 – 52, May 2010.
- [115] S. Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on Optimization*, 2(4):575–601, 1992.
- [116] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs (3rd Ed.)*. Springer-Verlag, London, UK, UK, 1996.

- [117] M. Moges and T. Robertazzi. Wireless sensor networks: scheduling for measurement and data reporting. *Aerospace and Electronic Systems, IEEE Transactions on*, 42(1):327 – 340, Jan 2006.
- [118] R. Montero. Building iaas clouds and the art of virtual machine management. In *High Performance Computing and Simulation (HPCS), 2012 International Conference on*, pages 573–573, July 2012.
- [119] R. H. M. Murata, Y. Egawa and H. Kobayashi. A history-based job scheduling mechanism for the vector computing cloud. In *IEEE/IPSJ International Symposium on Applications and the Internet*, pages 125–128, 2010.
- [120] M. Murugan, S. Suresh, R. Ganguli, and V. Mani. Target vector optimization of composite box beam using hybrid genetic algorithm: a decomposition approach. *Structural and Multidisciplinary Optimization*, 33(2):131 – 146, 2007.
- [121] S. Pandey, L. Wu, S. Guru, and R. Buyya. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In *IEEE International Conference on Advanced Information Networking and Applications*, pages 400–407, 2010.
- [122] A. Piriya Kumar and C. Murthy. Distributed computation for a hypercube network of sensor-driven processors with communication delays including setup time. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 28(2):245 – 251, Mar 1998.
- [123] T. Robertazzi. Ten reasons to use divisible load theory. *Computer*, 36(5):63 – 68, 2003.
- [124] T. Robertazzi. *Networks and Grids: Technology and Theory*. Springer, 2007.
- [125] a. B. V. S. K. Chan and D. Ghose. Experimental study on large size matrix-vector product computations using divisible load paradigm on distributed bus networks. *Tech. Rep. VB/DLT/003/1999, Department of Electrical and Computer Singapore*, 1999.

- [126] P. Saeed and E. Reza. An approach to optimized resource scheduling algorithm for open-source cloud systems. In *World Applied Sciences Journal 7(Special Issue of Computer & IT)*:, pages 152 – 160, 2009.
- [127] R. Sakellariou and H. Zhao. A low cost rescheduling policy for efficient mapping of workflows on grid systems. In *Proceedings of 18th International on Parallel and Distributed Processing Symposium*, pages 253–262, 2004.
- [128] S. Sakr, A. Liu, D. Batista, and M. Alomari. A survey of large scale data management approaches in cloud environments. *Communications Surveys Tutorials, IEEE*, 13(3):311 – 336, Third 2011.
- [129] S. Saraswathi, S. Sundaram, N. Sundararajan, M. Zimmermann, and M. Nilsen-Hamilton. Icga-pso-elm approach for accurate multiclass cancer classification resulting in reduced gene sets in which genes encoding secreted proteins are highly represented. *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, 8(2):452 – 463, March 2011.
- [130] S. Selvarani and G. Sadhasivam. Improved cost-based algorithm for task scheduling in cloud computing. In *Computational Intelligence and Computing Research (ICCIC), 2010 IEEE International Conference on*, pages 1 – 5, Dec 2010.
- [131] G. Shen and Y.-Q. Zhang. A new evolutionary algorithm using shadow price guided operators. *Applied Soft Computing*, 11(2):1983 – 1992, 2011. The Impact of Soft Computing for the Progress of Artificial Intelligence.
- [132] A. Shokripour, M. Othman, H. Ibrahim, and S. Subramaniam. New method for scheduling heterogeneous multi-installment systems. *Future Generation Computer Systems*, 28(8):1205 – 1216, 2012. Including Special sections SS: Trusting Software Behavior and SS: Economics of Computing Services.
- [133] H. Siegel, J. Armstrong, and D. Watson. Mapping computer-vision-related tasks onto reconfigurable parallel-processing systems. *Computer*, 25(2):54 – 63, Feb 1992.

- [134] T. S. Somasundaram and K. Govindarajan. Cloudb: A framework for scheduling and managing high-performance computing (hpc) applications in science cloud. *Future Generation Computer Systems*, 34(0):47–65, 2014. Special Section: Distributed Solutions for Ubiquitous Computing and Ambient Intelligence.
- [135] S. Suresh, R. V. Babu, and H. Kim. No-reference image quality assessment using modified extreme learning machine classifier. *Applied Soft Computing*, 9(2):541 – 552, 2009.
- [136] S. Suresh, H. Huang, and H. Kim. Hybrid real-coded genetic algorithm for data partitioning in multi-round load distribution and scheduling in heterogeneous systems. *Applied Soft Computing*, 24:500 – 510, 2014.
- [137] S. Suresh, H. Huang, and H. Kim. Scheduling in compute cloud with multiple data banks using divisible load paradigm. *IEEE Transactions on Aerospace and Electronic systems*, 51(2), 2015.
- [138] S. Suresh, H. Kim, C. Run, and T. Robertazzi. Scheduling nonlinear divisible loads in a single level tree network. *The Journal of Supercomputing*, 61(3):1068 – 1088, 2012.
- [139] S. Suresh, V. Mani, and S. Omkar. The effect of start-up delays in scheduling divisible loads on bus networks: An alternate approach. *Computers & Mathematics with Applications*, 46(10-11):1545 – 1557, 2003.
- [140] S. Suresh, V. Mani, and S. N. Omkar. The effect of start-up delays in scheduling divisible loads on bus networks: an alternate approach. *Computers Mathematics with Applications*, 40(10):1545–1557, 2003.
- [141] S. Suresh, V. Mani, S. N. Omkar, and N. Sundararajan. Multi-spectral satellite image classification using an evolving neural network approach. *Journal Aerospace Technology*, 58(4):287–304, 2006.
- [142] S. Suresh, P. Sujit, and A. Rao. Particle swarm optimization approach for multi-objective composite box-beam design. *Composite Structures*, 81(4):598 – 605, 2007.

- [143] M. Tasgetiren, Y. Liang, M. Sevkli, and G. Gencyilmaz. A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *European Journal of Operational Research*, 177(10):1930–1947, 2007.
- [144] A. L. Therasa.S, Sumathi.G, and A. Dalya.S. Article: Dynamic adaptation of checkpoints and rescheduling in grid computing. *International Journal of Computer Applications*, 2(3):95 – 99, May 2010. Published By Foundation of Computer Science.
- [145] L. Tomas, B. Caminero, C. Carrion, and A. Caminero. On the improvement of grid resource utilization: Preventive and reactive rescheduling approaches. *Journal of Grid Computing*, 10(3):475–499, 2012.
- [146] H. Topcuoglu, S. Hariri, and M.-Y. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *Parallel and Distributed Systems, IEEE Transactions on*, 13(3):260 – 274, Mar 2002.
- [147] G. Valentini, W. Lassonde, S. Khan, N. Min-Allah, S. Madani, J. Li, L. Zhang, L. Wang, N. Ghani, J. Kolodziej, H. Li, A. Zomaya, C.-Z. Xu, P. Balaji, A. Vishnu, F. Pinel, J. Pecero, D. Kliazovich, and P. Bouvry. An overview of energy efficiency techniques in cluster computing systems. *Cluster Computing*, 16(1):3 – 15, 2013.
- [148] P. Varalakshmi, A. Ramaswamy, A. Balasubramanian, and P. Vijaykumar. An optimal workflow based scheduling and resource allocation in cloud. In A. Abraham, J. Lloret Mauri, J. Buford, J. Suzuki, and S. Thampi, editors, *Advances in Computing and Communications*, volume 190 of *Communications in Computer and Information Science*, pages 411 – 420. Springer Berlin Heidelberg, 2011.
- [149] T. Vinh, T. Duy, Y. Sato, and Y. Inoguchi. Performance evaluation of a green scheduling algorithm for energy savings in cloud computing. In *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1 – 8, April 2010.

- [150] B. Wang, X. Wang and J. Huang. Cloud computing and its key techniques. In *IEEE International Conference on Computer Science and Automation Engineering*, pages 404–410, 2011.
- [151] G. Y. Wei, A. V. Vasilakos, and N. X. Xiong. Scheduling parallel cloud computing services: an evolutionary game. In *International Conference on Information Science and Engineering*, pages 376–379, 2009.
- [152] M. Wieczorek, R. Prodan, and T. Fahringer. Scheduling of scientific workflows in the askalon grid environment. *SIGMOD Rec.*, 34(3):56 – 62, Sept. 2005.
- [153] P. Wolniewicz and M. Drozdowski. Processing time and memory requirements for multi-instalment divisible job processing. In R. Wyrzykowski, J. Dongarra, M. Paprzycki, and J. Waniewski, editors, *Parallel Processing and Applied Mathematics*, volume 2328 of *Lecture Notes in Computer Science*, pages 125 – 133. Springer Berlin Heidelberg, 2002.
- [154] H. M. Wong, D. Yu, V. Bharadwaj, and T. G. Robertazzi. Data intensive grid scheduling: multiple sources with capacity constraints. In *IASTED International Conference on Parallel and Distributed Computing and Systems*, 2003.
- [155] T. Wu, M. Li, M. Ponraj, and M. Qi. A genetic algorithm based peer selection strategy for bittorrent networks. In *Nature Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on*, pages 336 – 341, Dec 2009.
- [156] Z. Wu, X. Liu, Z. Ni, D. Yuan, and Y. Yang. A market-oriented hierarchical scheduling strategy in cloud workflow systems. *The Journal of Supercomputing*, 63(1):256 – 293, 2013.
- [157] F. Xhafa, L. Barolli, and A. Durresi. Batch mode scheduling in grid systems. *Int. J. Web Grid Serv.*, 3(1):19 – 37, Mar. 2007.
- [158] F. Xhafa, J. Carretero, L. Barolli, and A. Durresi. Immediate mode scheduling in grid systems. *Int. J. Web Grid Serv.*, 3(2):219 – 236, June 2007.

- [159] J. Xu, A. Lam, and V. Li. Chemical reaction optimization for task scheduling in grid computing. *Parallel and Distributed Systems, IEEE Transactions on*, 22(10):1624 – 1631, Oct 2011.
- [160] M. Xu, L. Cui, H. Wang, and Y. Bi. A multiple qos constrained scheduling strategy of multiple workflows for cloud computing. In *Parallel and Distributed Processing with Applications, 2009 IEEE International Symposium on*, pages 629 – 634, Aug 2009.
- [161] B. Yang, X. Xu, F. Tan, and D. H. Park. An utility-based job scheduling algorithm for cloud computing considering reliability factor. In *Cloud and Service Computing (CSC), 2011 International Conference on*, pages 95 – 102, Dec 2011.
- [162] Y. Yang and H. Casanova. A multi-round algorithm for scheduling divisible workload applications: Analysis and experimental evaluation. *Technical Report-CS2002-0721, Dept. of Computer Science, University of California, San Diego*, 2002.
- [163] Y. Yang and H. Casanova. Rumr: robust scheduling for divisible workloads. In *High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on*, pages 114 – 123, June 2003.
- [164] Y. Yang and H. Casanova. Umr: a multi-round algorithm for scheduling divisible workloads. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, page 9, April 2003.
- [165] Y. Yang, H. Casanova, M. Drozdowski, M. Lawenda, and A. Arnaud. On the complexity of multi-round divisible load scheduling. *Institut national de recherche en informatique et en automatique*, Rep. 6096, 2007.
- [166] Y. Yang, J. Choi, K. Choi, P. Gannon, and D. Kim. Biovlab-microarray: microarray data analysis in virtual environment. In *IEEE Fourth International Conference on eScience*, pages 159–165, 2008.
- [167] C. S. Yeo and R. Buyya. A taxonomy of market-based resource management systems for utility-driven cluster computing. *Softw. Pract. Exper.*, 36(13):1381 – 1419, 2006.

- [168] H.-S. Yoon and B.-R. Moon. An empirical study on the synergy of multiple crossover operators. *Evolutionary Computation, IEEE Transactions on*, 6(2):212 – 223, Apr 2002.
- [169] Z. Yu and W. Shi. An adaptive rescheduling strategy for grid workflow applications. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1 – 8, March 2007.
- [170] Y. Zhang. Solving large-scale linear programs by interior-point methods under the matlab environment. Technical report, Optimization Methods and Software, 1996.
- [171] H. Zhong, K. Tao, and X. Zhang. An approach to optimized resource scheduling algorithm for open-source cloud systems. In *ChinaGrid Conference (ChinaGrid), 2010 Fifth Annual*, pages 124 – 129, July 2010.
- [172] M. Zhou, R. Zhang, D. Zeng, and W. Qian. Services in the cloud computing era: A survey. In *Universal Communication Symposium (IUCS), 2010 4th International*, pages 40 – 46, Oct 2010.