SCHEDULING APPROACHES FOR PARAMETER SWEEP APPLICATIONS IN A HETEROGENEOUS DISTRIBUTED ENVIRONMENT

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

GÜLŞAH KARADUMAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER 2010

Approval of the thesis:

SCHEDULING APPROACHES FOR PARAMETER SWEEP APPLICATIONS IN A HETEROGENEOUS DISTRIBUTED ENVIRONMENT

submitted by **GÜLŞAH KARADUMAN** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Adnan Yazıcı
Head of Department, **Computer Engineering** _____

Dr. Cevat Şener
Supervisor, **Computer Engineering Dept., METU** _____

Dr. Nedim Alpdemir
Co-supervisor, **TUBITAK UEKAE ILTAREN** _____

Examining Committee Members:

Assoc. Prof. Dr. Halit Oğuztüzün
Computer Engineering Dept., METU _____

Dr. Cevat Şener
Computer Engineering Dept., METU _____

Asst. Prof. Dr. Pınar Şenkul
Computer Engineering Dept., METU _____

Dr. Mahmut Nedim Alpdemir
TUBITAK UEKAE ILTAREN _____

İbrahim Demir, M.Sc.
TUBITAK UEKAE ILTAREN _____

Date: **17.09.2010**

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: GÜLŞAH KARADUMAN

Signature :

ABSTRACT

SCHEDULING APPROACHES FOR PARAMETER SWEEP APPLICATIONS IN A HETEROGENEOUS DISTRIBUTED ENVIRONMENT

Karaduman, Gülşah

M.Sc., Department of Computer Engineering

Supervisor : Dr. Cevat Şener

Co-Supervisor : Dr. Nedim Alpdemir

September 2010, 80 pages

In this thesis, the focus is on the development of scheduling algorithms for Sim-PETEK which is a framework for parallel and distributed execution of simulations. Since it is especially designed for running parameter sweep applications in a heterogeneous distributed computational environment, multi-round and adaptive scheduling approaches are followed. Five different scheduling algorithms are designed and evaluated for scheduling purposes of Sim-PETEK. Development of these algorithms are arranged in a way that a newly developed algorithm provides extensions over the previously developed and evaluated ones. Evaluation of the scheduling algorithms is handled by running a Wireless Sensor Network (WSN) simulation over Sim-PETEK in a heterogeneous distributed computational system formed in TUBITAK UEKAE ILTAREN. This evaluation not only makes comparisons among the scheduling algorithms but it also and rates them in terms of the optimality principle of divisible load theory which mentions that in order to obtain optimal processing time all the processors used in the computation must stop at the same time. Furthermore, this study adapts a scheduling approach, which uses statistical calibration, from literature to Sim-PETEK and makes an assessment between this approach and the most optimal scheduling approach among the five algorithms that have been previously evaluated. The approach which is found to be the most

efficient is utilized as the Sim-PETEK scheduler.

Keywords: Scheduling, Parameter Sweep Applications, Divisible Load Theory

ÖZ

DAĞITIK HETEROJEN BİR ORTAMDA PARAMETRE TARAMA UYGULAMALARINI ÇİZELGELEME YAKLAŞIMLARI

Karaduman, Gülşah
Yüksek Lisans, Bilgisayar Mühendisliği
Tez Yöneticisi : Dr. Cevat Şener
Ortak Tez Yöneticisi : Dr. Mahmut Nedim Alpdemir
Eylül 2010, 80 sayfa

Bu tez kapsamında, simülasyonların paralel ve dağıtık koşturulması için gerçeklenmiş bir altyapı olan Sim-PETEK yapısı için çizelgeleme algoritmalarının geliştirilmesine yönelik çalışmalar yapılmıştır. Sim-PETEK özellikle parametre tarama uygulamalarının heterojen ve dağıtık hesaplama ortamlarında çalıştırılmasına yönelik bir şekilde geliştirildiği için çizelgeleme sırasında çok turlu ve uyarlanabilir yaklaşımlar izlenmiştir. Bu bağlamda beş farklı algoritma tasarlanmış ve değerlendirilmiştir. Algoritmaların geliştirilmesi sürecinde izlenen yol yeni geliştirilmekte olan bir algoritmanın daha önceden geliştirilmiş ve değerlendirilmiş algoritmalara eklentiler sunması şeklinde olmuştur. Çizelgeleme algoritmalarının değerlendirilmesi için Sim-PETEK altyapısını kullanan bir Kablosuz Algılayıcı Ağ simülasyonu TÜBİTAK UEKAE İLTAREN'de kurulan heterojen ve dağıtık bir hesaplama ortamında koşturulmuştur. Yapılan değerlendirmelerde farklı çizelgeleme algoritmalarının birbirleriyle karşılaştırılarının yanı sıra algoritmaların optimum işleme zamanının ancak bütün işlemcilerin aynı anda durmasıyla elde edilebileceğini belirten bölünebilir yük teorisi açısından da değerlendirilmesi yapılmıştır. Bu çalışmada ayrıca literatürde bulunan istatistiksel çizelgeleme yaklaşımı Sim-PETEK yapısına uyarlanmış ve bu yaklaşımla sunduğumuz en iyi çizelgeleme yaklaşımı karşılaştırılmıştır. Yapılan değerlendirmeler sonucunda en verimli bulunan çizelgeleme yakla-

şımının Sim-PETEK çizelgeleyicisi olarak kullanılması planlanmıştır.

Anahtar Kelimeler: Çizelgeleme, Parametre Tarama Uygulamaları, Bölünebilir Yük Teorisi

To my endless love, to Ayşe and to Çilek

ACKNOWLEDGMENTS

I would like to present my special thanks to my supervisor Dr. Cevat Şener and my co-supervisor Dr. Mahmut Nedim Alpdemir for their guidance, understanding and encouragement throughout the development of this thesis.

I would like to show my gratitude to my thesis jury members Asst. Prof. Dr. Pınar Şenkul, Assoc. Prof. Dr. Halit Oğuztüzün, and İbrahim Demir for reviewing and evaluating my thesis.

My special thanks is to Doruk Bozağaç from TUBITAK UEKAE ILTAREN. His invaluable ideas, suggestions, and help have been essential to my research. I appreciate all the time he has spent for the development of my thesis.

I would like to thank to Kezban Demirtaş Başıbüyük, Filiz Alaca Aygül, and Kevser Sönmez Sunercan for their invaluable friendship and support.

Finally, my deepest thanks are to my parents and my elder sister, Ayşe İlknur Karaduman, who supported me with their never ending love, understanding, encouragement and support throughout my life.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVATIONS

API	Application Programming Interface
GUI	Graphical User Interface
PSA	Parameter Sweep Application
MC	Monte Carlo
HLA	High Level Architecture
CPU	Central Processing Unit
SOA	Service Oriented Architecture
OGSA	Open Grid Services Architecture
URI	Uniform Resource Identifier
WSRF	Web Services Resource Framework
WS	Web Service
SIMA	Simulation Modeling Infrastructure
DEVS	Discrete Events System Specification
TIG	Task Interaction Graph
DLT	Divisible Load Theory

MI Multi Installment

UMR Uniform Multi-Round

MRRS Multi-Round Scheduling with Resource Selection

RUMR Robust Scheduling for Divisible Workloads

TF Task Farm

AMRS Adaptive Multi-Round Synchronous Scheduling Algorithm

AMRA Adaptive Multi-Round Asynchronous Scheduling Algorithm

P-AMRA Probed Adaptive Multi-Round Asynchronous Scheduling Algorithm

SSSE-AMRA Slow Start - Slow End Adaptive Multi-Round Asynchronous Scheduling Algorithm

ISSSE-AMRA Improved Slow Start - Slow End Adaptive Multi-Round Asynchronous Scheduling Algorithm

WSN Wireless Sensor Network

CHAPTER 1

INTRODUCTION

In this first chapter of this thesis, a brief overview of our work and organization of the following chapters are presented.

1.1 Overview

Today it has become a very common approach to simulate real-world objects by modeling them using mathematical formulas. This approach is followed in order to observe the behavior of systems through approximated models of various fidelity and detect possible failures and risks before real life usage. For example, characteristics of a newly designed airplane's flight while it is passing over a river or lake can be simulated by modeling the airplane, lake, and river and errors detected during the simulation can be corrected before the mass production starts [4].

Simulations are very useful in many areas such as financial computations, computational chemistry and physics, genetics and DNA modeling, defense and security modeling, and protein folding. However, such kinds of simulation models require high computational power because of the enormous calculations they have to handle. At this point, super computers with custom architecture can be used but this is an expensive solution. A cheaper solution is the utilization of distributed heterogeneous computational resources over a local or wide area network.

When heterogeneous computational resources are utilized, usage and management of such resources become a problem to be handled. Recently, many researchers have focused on the specification and implementation of software middlewares which simplify the management

and usage of heterogeneous computational resources by providing access to the resources through a standardized programming interface [11]. Figure 1.1 shows a distributed system stack where a software middleware handles resource access and management.

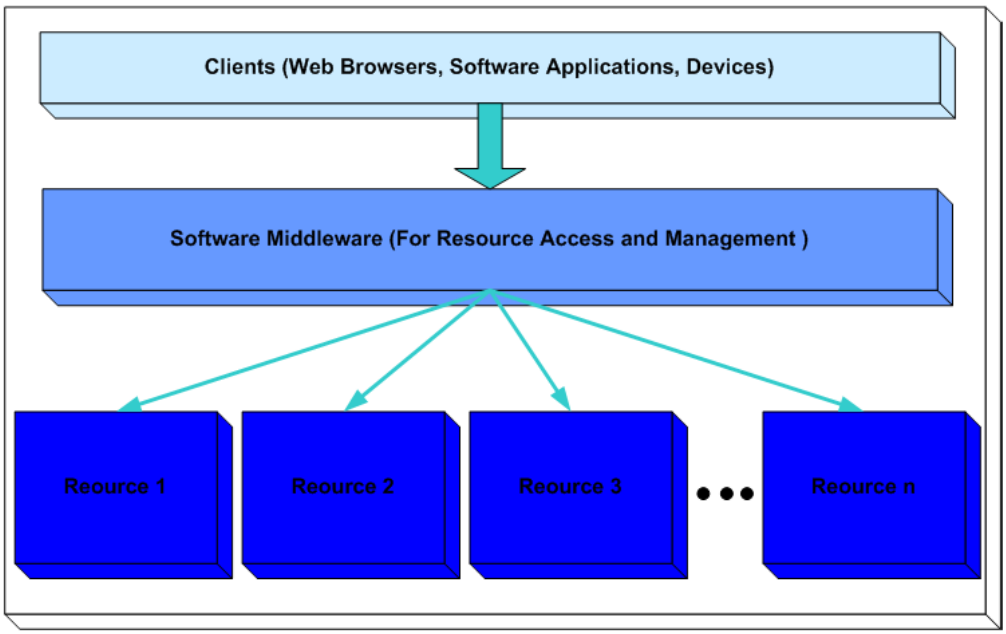


Figure 1.1: Distributed System Stack

More recent studies on software middleware of distributed systems make use of the abstractions defined by the Service-Oriented Architecture (SOA), which is a design principle using a collection of services for simplifying the communication between different systems. Open Grid Services Architecture (OGSA) [2] and its reference implementations such as GT3.x [1] and Web Services Resource Framework (WSRF) [5] are good examples which use SOA for distributed system management especially for managing Grid infrastructures.

A software middleware, which is responsible for the management of a distributed computing environment for running distributed simulation applications, has to consider distinct properties of simulation applications. With this aspect in mind, in TUBITAK UEKAE ILTAREN, we developed a Paralel Simulation Run Framework (Sim-PETEK) which is a WSRF [5] compliant service-oriented software middleware designed for Parameter Sweep Applications (PSA) and stochastic simulations (e.g. Monte Carlo (MC) simulations). Distinct properties of Sim-PETEK which make it a simulation-centric middleware can be described as follows [11]:

- Formalization of a distributable task specification is handled around the notion of a simulation scenario
- Simulation specific aspects are utilized for the scheduling and monitoring of distributed tasks
- Result collection approach meets the simulation applications' requirements

The work presented in this thesis focuses on the scheduling part of Sim-PETEK. In order to design an effective scheduler component, a literature survey has been conducted and existing scheduling approaches has been improved to fit Sim-PETEK requirements. There are basically two different scheduling approaches in literature. Static scheduling approach makes all of the decisions before the application starts running whereas dynamic scheduling approach makes its decisions at run-time.

Before the selection of scheduling approach, characteristics of the load that is going to be processed should be analyzed because different approaches are appropriate for different characteristics. Load characteristics can be grouped as indivisible, modularly divisible, and arbitrarily divisible:

- Indivisible loads can not be divided and have to be processed as a whole.
- Modularly divisible loads can be subdivided into smaller ones but these smaller loads have interactions among.
- Arbitrarily divisible loads can be partitioned into smaller load fractions arbitrarily.

PSAs, for which Sim-PETEK is designed, are the applications consisting of a set of independent "experiments" [15] which show the characteristics of arbitrarily divisible loads. For optimal scheduling of this kind of independent loads, there is a theory in the literature which is called as Divisible Load Theory (DLT). The optimality principle of DLT states that in order to obtain optimal processing time all the processors used in the computation must stop at the same time [10].

Divisible load scheduling approaches found in the literature try to satisfy the optimality principle of DLT by distributing the load fractions either in a single round or in multiple rounds.

For large workloads single round approach is not efficient because of the communication cost and multi-round approach is followed for overlapping communication and computation times.

If a scheduling algorithm is designed for task distribution in a heterogeneous computation environment, it has to consider that the performance is affected by the variations in the system such as network latency. In order to achieve high performance, scheduling approach should adapt itself to these variations by following a load balancing strategy. This means that dynamic scheduling should be applied.

Recently, adaptivity is integrated into divisible load scheduling algorithms for achieving high performance in heterogeneous environments. In [22], adaptivity is provided by estimating processing capacity of the resources by making them to process a small load partition. The method in [23] utilizes statistical calibration techniques for adaptivity purposes.

After inspecting scheduling strategies of literature, Sim-PETEK scheduling algorithms are decided to be designed in multi-round and adaptive manner. This is because Sim-PETEK is developed for heterogeneous computational environments for running PSAs.

Several scheduling algorithms have been developed for Sim-PETEK scheduling purposes. After the implementation, performance tests are carried out for determining the most efficient algorithm. Moreover, the algorithm defined in [23] is implemented and comparison tests have been performed. The reason why the approach in [23] is selected from literature is that, it has been found to be the most appropriate approach for Sim-PETEK architecture.

1.2 Organization

Organization of this thesis can be summarized as follows. In Chapter 2, a background information about parameter sweep applications, stochastic simulations, and Sim-PETEK architecture is given. In Chapter 3, our literature survey on scheduling is presented. This chapter especially focuses on divisible load theory and divisible load scheduling. In Chapter 4, scheduling algorithms developed for Sim-PETEK scheduling purposes are described. Chapter 5 gives some implementation details of scheduling algorithms. In Chapter 6, performance analysis of the scheduling algorithms is made. Chapter 7 provides discussions on our study and lists possible future works. Finally, Chapter 8 presents our conclusions.

CHAPTER 2

BACKGROUND

As mentioned in Chapter 1, the work presented in this thesis aims at developing scheduling algorithms for Sim-PETEK which is a framework designed for running parameter sweep applications that may involve stochastic analysis methods in a distributed computation environment. In this chapter, firstly an introductory information about parameter sweep applications and stochastic simulations is provided. Afterwards, Sim-PETEK architecture and the technologies used in the architectural design are presented.

This chapter also provides a literature survey which is held on scheduling approaches for distributed and heterogeneous systems. This survey especially focuses on divisible load scheduling approaches because parameter sweep applications can be characterized in terms of arbitrarily divisible loads.

2.1 Parameter Sweep Applications

In [15] parameter sweep applications (PSA) are defined as the applications consisting of independent experiments which are held for distinct parameter sets. In the first place, a number of input parameters are selected and then the effects of these selected parameters are analyzed. The analysis is held by defining a minimum and maximum value and a step size for each of the input parameters. Discrete values of the input parameters defined by this method form discrete value sets and the batch parameter set can be defined as the union of such discrete value sets. Simulation is run for each discrete value of the batch parameter set and these different simulation runs are called batch runs. Results of batch runs are collected and analyzed according to the batch parameter value change. Distributed environments such as grid are ideal execution environments for this kind of applications of many scientific and engi-

neering domains such as bioinformatics, operations research, data mining, business model, network simulations, massive searches, ecological modeling, fractals calculations, and image manipulation [20].

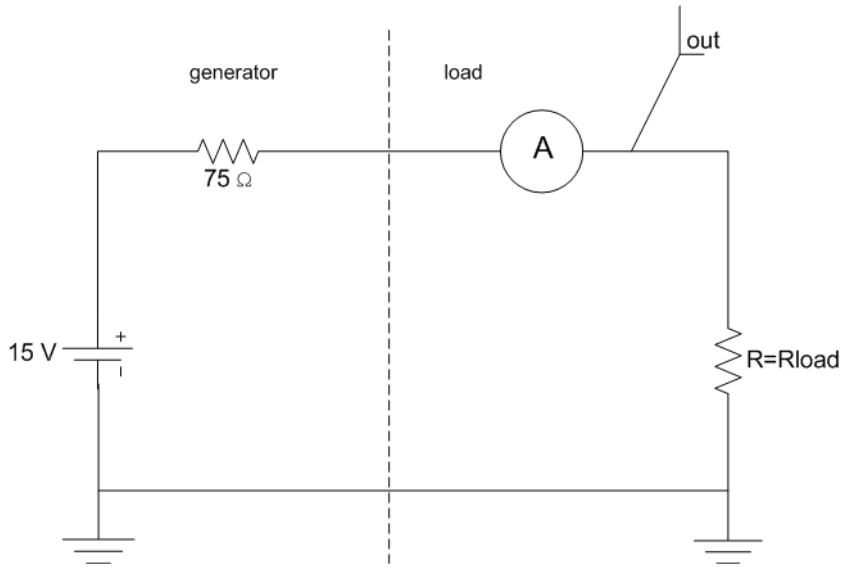


Figure 2.1: A Direct Current Electrical Circuit

As a parameter sweep example, the electrical circuit shown in Figure 2.1 which is taken from [3] can be considered. The circuit contains a generator with an internal resistance of 75 Ω on the left hand side and a resistive load on the right hand side. The current passing through the circuit is measured by the ampermeter. If the load resistance is given as 50 Ω, voltage at the ”out” node, load side power, and the current values will be as shown in Table 2.1.

Table 2.1: Current, Voltage, and Power Values When Rload = 50 Ω

Current	0.12 Amperes
”out” Voltage	6 Volts
Load Side Power	0.72 Watts

If the load resistance is decreased to 25 Ω, current, voltage, and power values will be as shown in Table 2.2.

Table 2.2: Current, Voltage, and Power Values When Rload = 25 Ω

Current	0.15 Amperes
”out” Voltage	3.75 Volts
Load Side Power	0.562 Watts

As it can be seen from the tables, when the load resistance is decreased, a higher current and a lower voltage is produced at the load side. For further observation, a parametric analysis of the circuit can be held by sweeping with load resistance. As an example if load resistance is increased from 5 Ω to 100 Ω in 20 steps, current, voltage, and power changes occur as shown in Figure 2.2 where load resistance is denoted by ”Rload”.

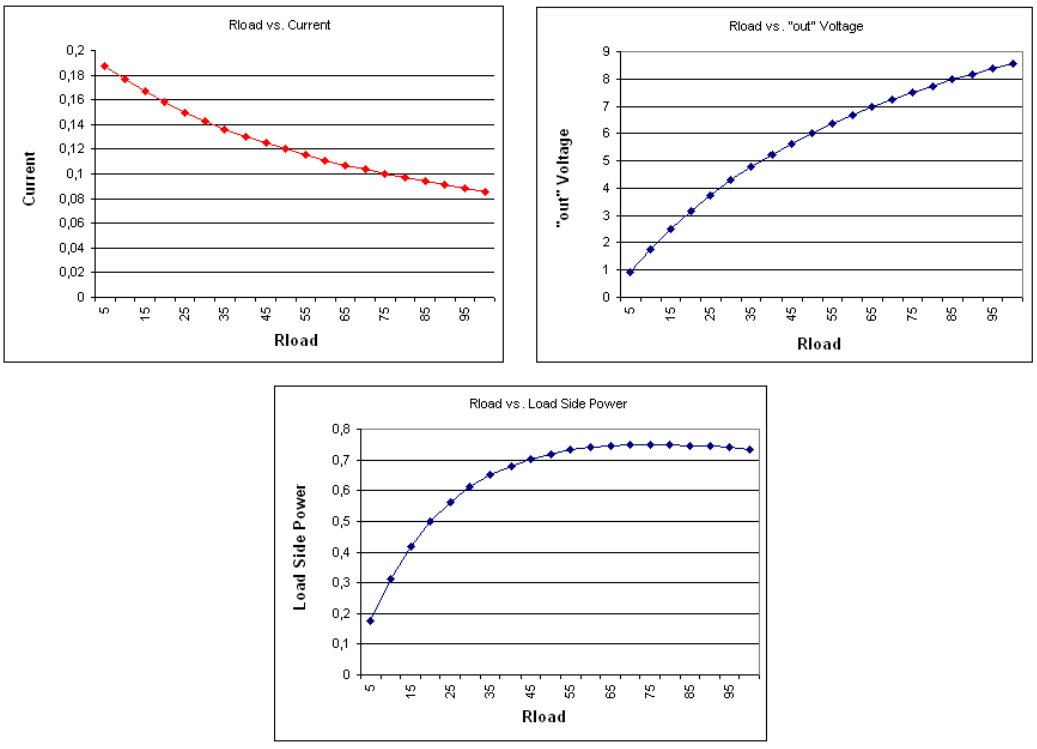


Figure 2.2: Rload vs. Current, Voltage, and Power

2.2 Stochastic Simulations

Many real world problems are solved by making use of computer simulations. However, modeling large scale systems is not an easy task, since large number of variables have effects on the whole system and there are parameters which include some uncertainties. Even in cases where perfect modeling is held, such systems can not get rid of the errors which occur during application or manufacturing and cause system performance changes. In order to be able to make a more realistic model, designers make use of stochastic simulation analysis methodologies. In [24], it is stated that stochastic simulation methods attempt to mimic or replicate the behavior of a system by exploiting randomness to obtain a statistical sample of possible outcomes. It is also stated that because of the randomness involved, simulation methods are also commonly known in some contexts as Monte Carlo (MC) methods. A *Monte Carlo Trial* is a simulation run held by the following approach: Each random variable of a system has a domain space and inputs are generated from each domain space. The simulation runs repetitively for the different input generations and the results of these repetitive runs are analyzed by using histograms and aggregated for a final result summary [11].

Stochastic simulation analysis methods can be included in parameter sweep applications for achieving more confident solutions. For this purpose, a number of i.d.d. (independent and identically distributed) repetitions are held for each parameter in the batch parameter set. The results of the repetitions are used in the expected performance computation and variance exploitation which are then used for finding solution's confidence interval. This approach gives confident results, on the other hand, it causes an exponential increase in the total number of runs. As an example, if the simulation contains two input parameters requiring M and N number of batch runs respectively, and performs T repetitions for each batch, the simulation consists of $M * N * T$ runs in total [11].

2.3 Sim-PETEK as a Simulation Specific Grid MiddleWare

It is previously mentioned that, Sim-PETEK is the software middleware which is used as the sample framework for the study presented in this thesis. In this section, software technologies and design aspects of Sim-PETEK are explained in detail.

2.3.1 Grid Environments and Simulation Applications

In today's world, many applications work on huge data sets and/or require extensive CPU power. This kind of applications have led to the solutions that either use super computers with highly specialized architectures or that utilize distributed heterogeneous resources of a local or wide area network. Recently, researchers of this area have focused on the specification and implementation of software middlewares which are dedicated for the simplification of distributed computational resource management through a standardized programming interface. At this point, grid has been defined as the general name for the common protocols and mechanisms to utilize geographically dispersed computational and data resources for solving CPU intensive problems in distributed, heterogeneous and multi-user environments [11].

Large scale distributed simulations can make use of grid technologies for accessing distributed data sets and computational resources. Recently, HLA (High Level Architecture) simulations are reported to be executed on the Internet by the help of grid technologies [33].

2.3.1.1 Service Oriented Approach

Grid infrastructures are further developed and Service-Oriented approaches are applied in order to be able to make use of the abstractions provided by service-orientation. Service-Oriented Architecture (SOA) is a design principle which focuses on simplifying the communication between the systems running on different platforms. A service is an autonomous system which accepts one or more requests and returns responses via a well-defined interface and SOA is a collection of such services [30].

Introduction of Open Grid Services Architecture (OGSA) has led to the emergence of service-oriented grid technologies. OGSA has defined the "Grid Service" concept which is a unification of the notions of Web Service and object-oriented distributed software architectures [2]. Furthermore, OGSA specifies standard programming interfaces for grid service creation, management and lifetime control [18].

2.3.1.2 Resource Oriented Approach

Resource-oriented approach is a more recent approach defining the ”resource” notion and focusing on resource management. Resource-orientation defines resources as logical addresses, such as URIs, and any operation on a resource is handled by sending an operation request to the resource under consideration [11], [19]. In order to standardize resource management in resource-oriented systems, OASIS defined Web Services Resource Framework (WSRF) in 2004 [5]. WSRF specification defines Web Service Resource expression, management, access, and grouping with the main objective of making web service resources stateful [11], [25]. WSRF consists of four specifications namely, WS-ResourceProperties, WS-ResourceLifetime, WS-ServiceGroup, and WS-ReosurceLifetime.

- WS-ResourceProperties defines how WS-resources are described in the XML-based ”ResourceProperties” document, and how these properties could be modified and queried via this document. This document also includes state information about WS-resources [25].
- WS-ResourceLifetime defines the main mechanisms used for managing resource lifetime.
- WS-ServiceGroup defines the grouping strategies for both services and service resources. This kind of grouping is important in terms of providing a single access point to services and resources of the same group [31].
- WS-BaseFaults defines the standards for error reporting.

2.3.2 Goals of Sim-PETEK

For many scientific application domains which use stochastic simulations for system analysis, number of required batch runs and Monte Carlo trials are so high that efficient and effective usage of available computational resources become crucial in order to obtain simulation results in a permissible timespan. At this point, Service-Oriented Computational environments such as grids provide a viable solution for such CPU-intensive applications by introducing a new paradigm for software deployment, execution and management. However, usage of resources

in a wide area is challenging because of reliability, security, effective management of component deployment, life-cycle, monitoring and disposal, user authentication, access control, auditing and billing issues. Concerning such issues, a Service-Oriented Grid environment is not only an architecture which provides distributed computation but it is also a well-defined programming and execution model consisting of rules, specifications, and APIs. Therefore, Sim-PETEK was developed as a grid middleware which is compliant with Service-Oriented Grid standards such as OGSA and WSRF. Through compliance to relevant standards, the architecture provides a consistent resource access and utilization layer for developers of simulation applications [11].

2.3.3 Architecture of Sim-PETEK

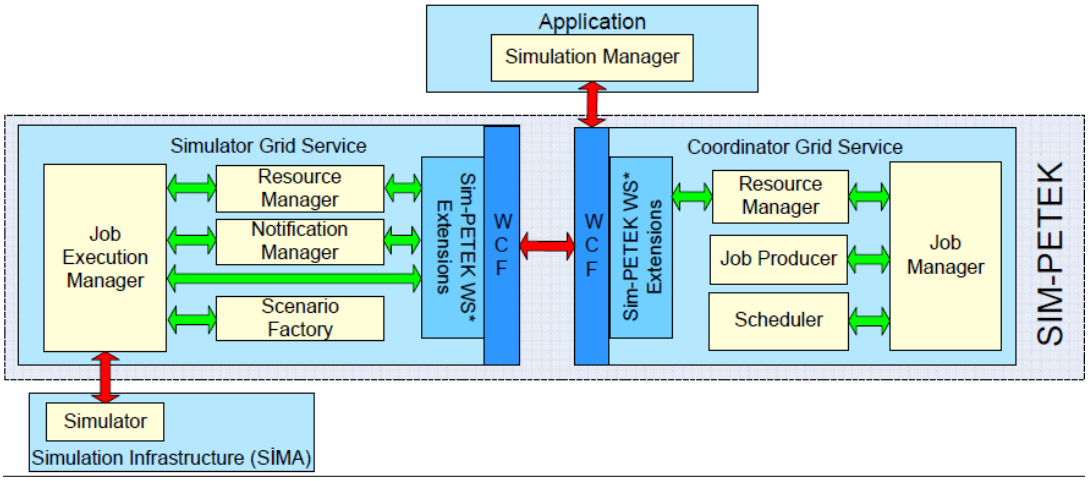


Figure 2.3: Sim-PETEK Architecture

In Figure 2.3 which is directly taken from [11], the general architecture of Sim-PETEK is shown. As the figure indicates, Sim-PETEK has been developed as a Service-Oriented infrastructure. Two main components of Sim-PETEK are the services named Coordinator Grid Service and Simulator Grid Service. As previously mentioned in 2.3.2, Sim-PETEK is compliant with WSRF standards and is implemented by using .NET Windows Communication Foundation (WCF) for the web service layer. WCF is a .NET Framework API which is designed for service-oriented application development. WCF follows service-oriented architecture principles for giving support to distributed computing where a client can interact with one or more services and each service can be called simultaneously by multiple clients [6]. WCF

already implements WS-Addressing WS-RealiableMessaging and WS-Security in its structure. Other standards of WSRF such as WS-ResourceProperties and WS-BaseNotification is implemented by Sim-PETEK by using WCF extension models in order to provide full compliance with WSRF.

For convenience, the following terminology is defined for Sim-PETEK in [11] and architecture definitions are made according to this terminology:

- **Coordinator** stands for Coordinator Grid Service
- **Simulator** stands for Simulator Grid Service
- **Job** is a set of simulation runs, i.e. a single unit of work assigned to a Simulator node by the Coordinator
- **Scenario** is the set of all parameters required by the simulation models for each simulation run
- **Job Execution** is a bunch of simulation runs which contains a number of batch runs and Monte Carlo(MC) trials

When a developer implements a client application using Sim-PETEK for its simulation distribution, he/she should construct a simulation execution order that includes the main simulation scenario, batch parameters and their values, and number of MC trials, and pass that order to the Coordinator. Taking this order, the Coordinator initializes the available Simulator nodes on the network and then distributes jobs to those nodes for execution. Simulator resources send periodic notifications to the Coordinator about their job status and the Coordinator uses this information for monitoring and re-scheduling purposes [11].

The following sections provide a more detailed description of Sim-PETEK's internal structure.

2.3.3.1 Coordinator Grid Service

As it can be seen from Figure 2.3, Coordinator consists of four main components, namely Job Manager, Resource Manager, Scheduler, and Job Producer. Job Manager is the component

which is responsible for establishing Simulator connection and management. This component gathers job status and resource information from the Simulator and provides that information to the Scheduler and Resource Manager components. Sending jobs to the simulator nodes, collecting job results and sending job status and results to the client application are additional responsibilities of the Job Manager. Resource Manager owns the responsibility of Simulator resource creation, query, and lifetime management which are handled in accordance with WS-ResourceProperties and WS-ResourceLifetime standards. Scheduler makes an analysis on simulator nodes' CPU load and available memory for creating an optimized scheduling plan (This component is the one on which the study in this thesis focuses). Prepared schedule is sent to the Job Manager which transmits it to the Job Producer. Job Producer arranges the simulation runs according to the schedule, creates jobs, and returns them to the Job Manager for distribution [11].

2.3.3.2 Simulator Grid Service

Simulator Grid Service of Sim-PETEK is composed of four components which are Resource Manager, Notification Manager, Scenario Factory and Job Execution Manager. Similar to Coordinator's Resource Manager component, Simulator's Resource Manager implements the resource creation, query, and lifetime management mechanisms in compliance with WS-ResurceProperties and WS-ResourceLifetime standards of WSRF. Notification Manager follows WS-BaseNotification standard and is responsible for handling subscription requests for resource information and job execution status and for sending periodic notification messages which includes resource and job status information to the Coordinator. Scenario Factory constructs different scenarios for each run by substituting batch parameter values into the base scenario. These scenarios are used by Job Execution Manager which creates simulation runs. Other responsibilities of the Job Execution Manager are executing the simulation runs and collecting the results to assemble the job result. By default, this result is sent to the Coordinator in the form of a notification message. If the client application provides a service reference to which the results should be sent, Simulator sends the results to the specified service instead of the Coordinator [11].

2.3.3.3 Workflow in Sim-PETEK

Sim-PETEK provides a software infrastructure which is designed to be used for running stochastic parameter sweep applications. The sequence of the main calls through the layers of the infrastructure can be described in terms of several stages which are given below:

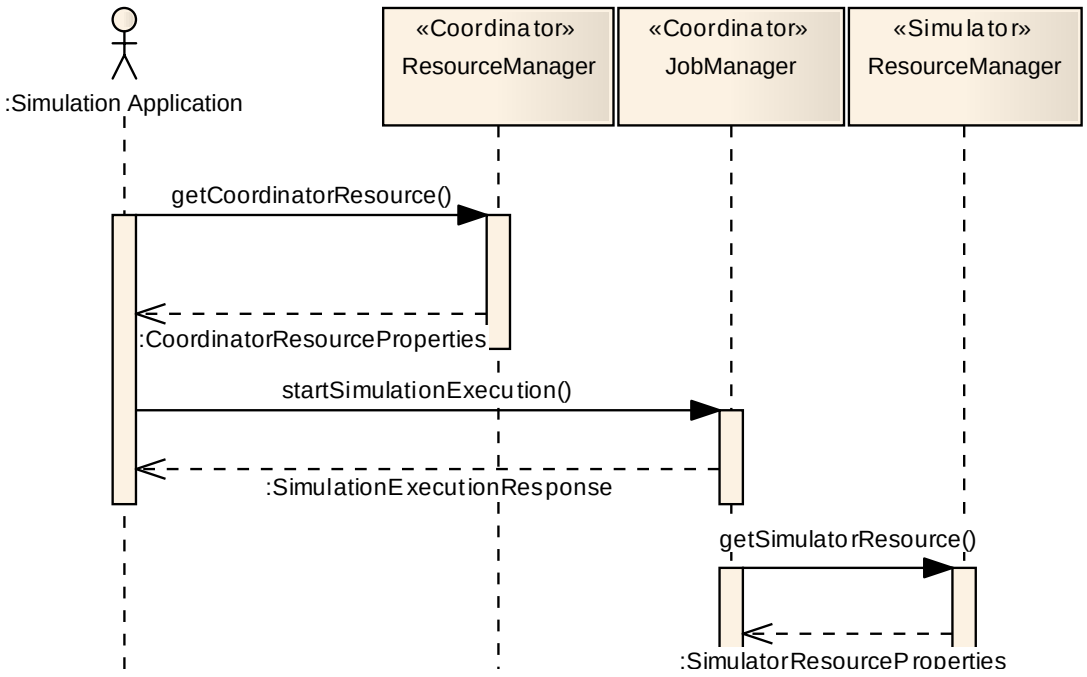


Figure 2.4: Initialization Stage of Sim-PETEK Workflow

- **Initialization** is the first stage, in which Simulator nodes are initialized and a parallel execution environment is formed by the Coordinator by making connections with the Simulator. This stage is explained in detail in Figure 2.4 which is taken from [11]. The flow in the system starts with the Simulation Application's *Coordinator Resource Request* via *getCoordinatorResource* operation. As a response to this request Coordinator sends *CoordinatorResourceProperties* over its Resource Manager. As a result, the connection between Simulation Application and Coordinator is establihed. Subsequently, Simulation Application requests the simulation execution to be started and gets the response as *SimulationExecutionResponse* which includes simulation execution identifiers. From this point on, Coordinator connects to the Simulator, subscribes for notifications and asks for resource information via *getSimulatorResource* operation.

Resource information is retrieved from the Simulator in the form of *SimulatorResourceProperties*.

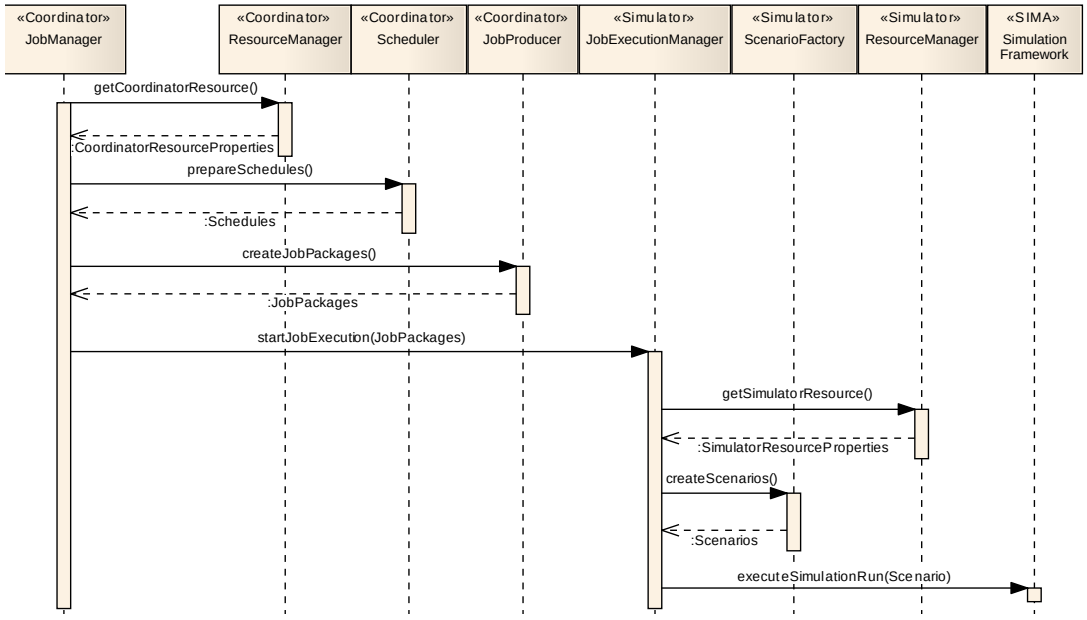


Figure 2.5: Job Distribution Stage of Sim-PETEK Workflow

- **Job Distribution** is the second stage of Sim-PETEK workflow. The detailed description of this stage is provided in Figure 2.5 [11]. After the initialization stage, Job Manager sends the simulator resource information to the Scheduler which produces the scheduling plan. This plan is transmitted to Job Producer which creates job packages. At this point, Coordinator requests the Simulator to start job execution. This request is passed to Simulator's Job Execution Manager which first retrieves resource information and then asks the Scenario Factory to produce simulation run scenarios according to the resource properties. Consequently, a request is sent to Simulation Modeling Infrastructure (SIMA) [26], which is a DEVS based simulation infrastructure [11].

- **Stochastic Analysis** is the third stage in which stochastic analysis methods such as repetitions by Monte-Carlo simulations are applied to simulation runs [11].

- **Result Aggregation** is the forth stage of the workflow. During this stage, simulators apply the filtering logic provided by the simulation application to the simulation results and the filtered results are sent to the Coordinator. This filtering logic prevents huge simulation result data to be sent over the network channels (only the required parts of

the results are transmitted).

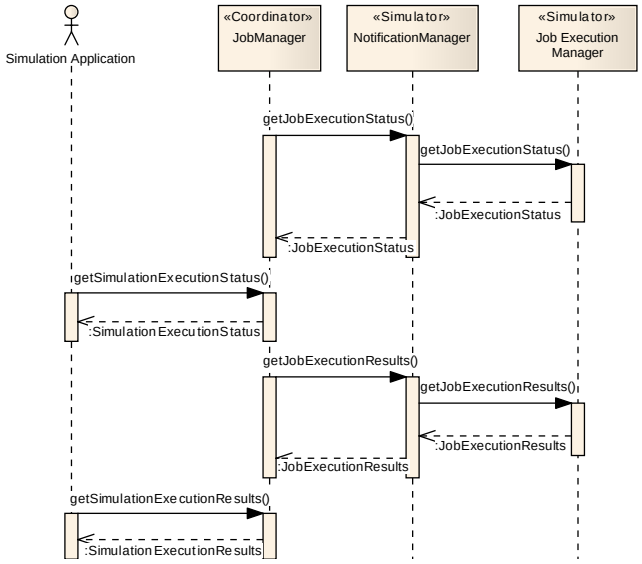


Figure 2.6: Result Collection and Status Monitoring Stage of Sim-PETEK Workflow

- **Result Collection and Status Monitoring** is the final stage of Sim-PETEK workflow. Figure 2.6 shows this stage, in which components can ask for current job status and job results.

2.4 Load Scheduling Approaches

There are two types of scheduling approaches that can be found in literature, namely static scheduling and dynamic scheduling. In static scheduling approach, all decisions are made before the application starts running. This means that static scheduling is appropriate for the cases where future behavior is predictable. On the other hand, dynamic scheduling approach involves making decisions at run-time either by following a predefined strategy or as a function of the current state of the system.

The simplest method of static scheduling is distributing the load as subtasks to computational resources according to a rule such as assigning task t_{ij} to resource $r_{i(j\%N(R_i))}$. Since this task distribution rule does not consider neither the computational power of resources nor the complexity of the subtasks, there is a possibility of inefficient distributions.

The most common mechanism used for dynamic scheduling purposes is the ”Master-Slave Model” which can work efficiently for different types of scenarios. In this model, scheduler is defined as the master and all other resources are slaves. Initially, master collects the whole load as subtasks in a queue and then starts by assigning ”n” tasks at the front of the queue to ”n” resources, i.e. slaves. When a slave finishes its task, it informs the master which assigns the next task from the front of the task queue. The algorithm goes on in the same way until all tasks of the task queue are processed [29].

2.5 Divisible Loads

The behavior of the scheduling algorithms depend on the characteristics of the load that is being processed. This is because some loads cannot be subdivided and have to be processed on a single processor as a whole whereas some other kinds of loads can be subdivided arbitrarily and can be independently processed on different processors. Load classification in [9] is provided in Figure 2.7.

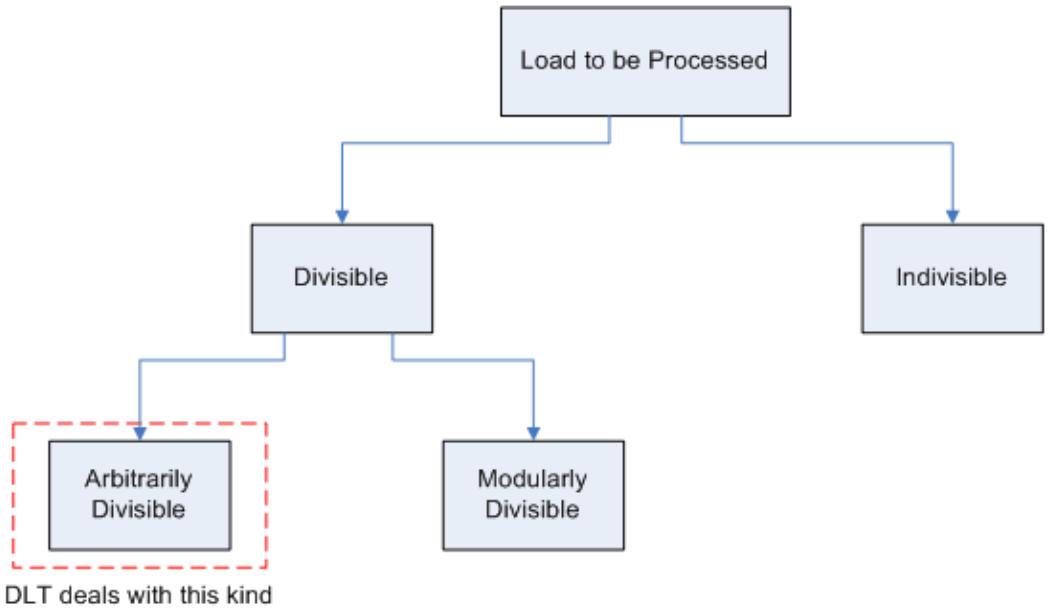


Figure 2.7: Load Classification

According to [9], indivisible loads are indivisible and independent as the name implies. This means that further subdivision of this kind of loads is impossible. If static scheduling approach

is applied for indivisible loads, the problem can be considered as bin-packing problems found in the literature and can be solved by adding some heuristics to the scheduling algorithm. Otherwise, if dynamic scheduling approach is applied, scheduling deals with the computational resource speed and availability and current system state.

Modularly divisible loads are defined in [9] as the loads that are subdivided a priori into smaller loads which have interactions among themselves. This interaction can be represented by a Task Interaction Graph (TIG) vertices of which correspond to load modules and edges of which correspond to module interactions.

Arbitrarily divisible loads are the ones which can be partitioned into smaller load fractions arbitrarily. These fractions can either have precedence relations or not [9].

When the load type of parameter sweep applications which are the focus of the study in this thesis is examined, they fall into the arbitrarily divisible load category. This is because independent experiments of a parameter sweep applications can be divided into arbitrary partitions that do not have any precedences among themselves.

2.5.1 Divisible Load Theory (DLT)

Divisible load theory (DLT) is a methodology defined in [10]. This methodology tries to develop linear and continuous models for partitionable computation and communication loads for parallel processing. The load that is considered by DLT is massive and requires an enormous amount of time to process. In the DLT model, there is one master processor and the other processors are defined as slaves. The master processor partitions the massive load into smaller partitions, keeps one of the load partitions for itself to process and sends the rest to the slaves in the network for processing. An important problem here is to decide how to achieve a balance in the load distribution between processors so that the computation is completed in the shortest possible time. This load balancing can be done at the beginning or dynamically during the computation.

Divisible load theory is interested in the load partitions that do not have any dependency relations (i.e. each load partition can be independently processed, and the results obtained after the process of one load partition does not have an effect on another one). Scheduling of these kinds of loads is nontrivial because the designed algorithms should focus on efficient

utilization of the available resources in terms of computational power and communication channel bandwidth.

Divisible load distribution, in general, follows the following steps. Firstly, the load to be processed arrives at the master node. If the computational node network has a linear topology, the master node pumps all the load in a pipelined manner, and every slave node receiving the load from its predecessor keeps the portion assigned for it and passes the rest to its successor. The problem is then reduced to the decision of the size of load portions in a way that total processing time is kept minimum. Because of the fact that this load partitioning is for a heterogeneous system of processors and network links, dividing the whole load equally results in a poor performance.

One point that is mentioned by the divisible load theory is that in order to obtain optimal processing time all the processors used in the computation must stop at the same time. This is the basic optimality principle for divisible load scheduling problems. The optimal time can only be achieved by an intelligent selection of the proper subset of the available processors. Thus, using a larger number of nodes may cause a poor performance compared to the performance of an optimal subset of nodes among which the load is dispatched according to the optimality principle.

Another point is that the divisible nature of the load provides the opportunity to divide and distribute the load in a repetitive sequence. By this strategy, the idle time of the processors at the farthest end of the load distribution sequence is reduced. In addition to this reduction in time, the finish time of the computation can be controlled by the selection of number of installments (i.e. repetitions) [10].

2.5.2 Divisible Load Scheduling

Load scheduling focuses on minimizing the overall execution time by finding an optimal strategy for both splitting the whole load into chunks and distributing these chunks to the available resources in the right order. This scheduling problem was tried to be solved by single-round and multi-round approaches. In the single-round approach, master processor distributes the task chunks to the workers in a single round. This means that there is a single communication between the master and each worker [8]. However, for large workloads single-round approach

is not efficient because of the idle time incurred by the last processors to receive their chunks. In order to solve this inefficiency problem multi-round scheduling approach has come to the scene. In multi-round scheduling, master processor sends the task chunks to the worker processors in multiple rounds which provides shorter and pipelined communications. Moreover, communication and computation times are overlapped by this approach.

The first multi-round scheduling algorithm is named as Multi Installment (MI) which starts with small chunks and increases the chunk sizes throughout application execution to achieve effective overlap of communication and computation [36]. Some other kinds of multi-round scheduling algorithms start with large chunks and decrease chunk sizes instead of increasing throughout application execution. The major disadvantage of such algorithms is poor overlap of computation with communication [36]. This is because, for most of the applications the amount of data to be sent for a chunk is proportional to the chunk size and starting by sending a large chunk to the first worker would cause all the remaining workers to be idle during that potentially long data transfer [35].

UMR (Uniform Multi-Round) is another scheduling algorithm which distributes work to computational resources in multiple rounds. UMR is an extension of MI algorithm and is developed for addressing the limitations of MI. These limitations are that MI does not model latencies associated with resource utilization and also does not provide any way to determine the optimal number of rounds. UMR handles these limitations by imposing the restriction that rounds must be ”uniform”, i.e. within each round the master assigns identical chunks to all workers [34]. By this restriction, the UMR algorithm makes it possible to compute optimal number of rounds while modeling resource latencies [35].

The MRRS (Multi-round Scheduling with Resource Selection) algorithm [28] extends the UMR by considering the network bandwidth and latency in addition to the computational capacity of workers. Furthermore, the MRRS is featured with a resource selection policy that finds the best subset of available computational resources [27].

Investigations on multi-round algorithms have revealed that [34]:

- dividing the workload into large chunks reduces overhead, and thereby application makespan;
- the use of small chunks at the onset of application execution makes it possible to overlap

overhead with useful work more efficiently; and

- the use of small chunks at the end of the execution leads to better robustness to performance prediction errors

With these investigations in mind, Robust Scheduling for Divisible Workloads (RUMR) [34] algorithm was developed. What RUMR tries to do is to combine ideas from multi-round divisible workload scheduling, for performance, and from factoring-based scheduling, for robustness. Factoring-based approach of this algorithm provides dynamicity.

2.5.3 Adaptive Divisible Load Scheduling

In large scale applications running in a heterogeneous environment such as grid, performance is affected by the variances in workload, processors, network latencies, and other system related factors. Adapting to the variance of these factors requires dynamic task assignment, and therefore, dynamic scheduling algorithms are powerful tools for the performance improvement via the load balancing strategies they follow. Dynamic loop scheduling schemes such as Factoring, Fractiling, Weighted Factoring, and Adaptive Weighted Factoring are examples of such strategies. In the factoring method, a probablistic analysis is held, and factoring rules according to which loop iterations are executed are formulated. Fractiling is a method based on factoring. It is a combined scheduling technique that balances processor loads and maintains locality by exploiting self-similarity properties of fractals. Since the heterogeneity in processor performance could lead to severe load imbalance, a Weighted Factoring approach was proposed, where the chunks sizes are proportional to the relative processor speeds. There are computing environments where processor workloads vary dynamically. If a scientific application that requires a number of iterations over the computation space is running in such a dynamic environment then a better performance can be achieved by adjusting the weights dynamically after each iteration. This aspect is addressed by the Adaptive Weighted Factoring technique [7].

Except from RUMR, described algorithms for divisible load scheduling are static because they work under the assumption that the full computational capacity of resources is constantly available and can be readily used, which makes them impractical for dynamic environments such as the Grid [27]. RUMR has a dynamic nature, however all of its parameters are fixed

before it starts, which makes RUMR a non-adaptive scheduling algorithm.

Recently, adaptive approaches in divisible load scheduling have emerged. In [22], a two phase adaptive load distribution strategy is followed. In the first phase (probe phase), a small part of the load is partitioned and communicated to individual processing resources. When a resource completes its load, the average bandwidth and average processing capacity of the processing resource is estimated. Then the optimal load distribution phase starts and distributes the load by computing the optimal load fractions to be dispatched to the individual processor resources. Computations of the optimal load fractions are made according to the estimations of the first phase.

In [23], skeletal task farm is used for scheduling divisible workloads for enhancing the performance. A task farm (TF) consists of a farmer process which administers a set of independent worker processes to concurrently execute a large number of independent tasks, collectively comprising a divisible workload. The work in [23], provides a dynamic framework which tries to make an automatic scheduling of divisible workloads based on the dispersion of the participating computational resources and size of the workload. The core of this work consists of a calibration phase and an execution phase. In the calibration phase, computational nodes are calibrated by making them to execute one element from the task set. When all workers finish their tasks, the execution times are taken and used for resource quantification by means of a fitness index, F. There are two different calibration methods, namely times-only calibration and statistical calibration. If times-only calibration is followed, fitness index, F is defined as a normalized decreasing function based on the inverse of the execution times. For a worker node $node_i$,

$$F_i = \frac{\frac{1}{t_i}}{\sum_{j=1}^{N} \frac{1}{t_j}} \tag{2.1}$$

where t_i denotes the execution time value for $node_i$. When statistical calibration method is used F is computed by using a curve fitting method over execution time. Univariate linear regression fitting method considers that execution time depends only on processor availability whereas multivariate linear regression considers that both the network latency and the processor availability affect the execution time. Univariate linear regression defines its regression function as

$$t = c_0 + c_1 a' \tag{2.2}$$

where a' is a vector of size N (number of workers) denoting scaled processor availability and c_0 and c_1 are constants. The main objective of this approach is to assign fewer tasks to the workers which work more slowly. Calculation of fitness index in equation 2.1 is held by using fitted values of t in 2.2.

In multivariate case, t is expressed as in 2.3 and used for fitness index calculation:

$$t = c_0 + c_1 l + c_2 a^{'2} \tag{2.3}$$

where l is a vector with size N which keeps network latencies for the worker nodes.

In the execution phase for the TF, task assignment is held according to:

$$\alpha_l = \left\lfloor \frac{S}{k} \times F_l + 0.5 \right\rfloor \quad \forall l = 1, ..., N \tag{2.4}$$

where S stands for total number of tasks, and k is the installment factor which dynamically quantifies the number of rounds by making use of the number of tasks in the workload and the system circumstances. If single round scheduling is to be applied, then k = 1 is substituted in 2.4. Otherwise, for multi-round scheduling k is calculated according to node dispersion. This dispersion is estimated by the coefficient of variation(CV):

$$\bar{t} = \frac{1}{N} \sum_{i=1}^{N} t_i \quad and \quad \sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (t_i - \bar{t})^2}$$

$$CV = \frac{\sigma}{\bar{t}} \tag{2.5}$$

$$k = ln(S)^{CV} \tag{2.6}$$

CHAPTER 3

Sim-PETEK SCHEDULING ALGORITHMS

Sim-PETEK is a software infrastructure designed for executing parameter sweep applications on distributed and heterogeneous computational environments such as grid. With these aspects in mind, job scheduling algorithms of Sim-PETEK should be practically used for divisible loads in dynamic computational environments. As mentioned in Section 2.5.3, adaptive approaches are more appropriate for dynamic environments and multi-round scheduling approaches should be followed for divisible loads such as parameter sweeps. For this reason, Sim-PETEK scheduling algorithms were developed as multi-round and in an adaptive manner.

Before a detailed description of the scheduling algorithms, definitions, parameters and notations used in the scheduling algorithms are presented below:

- **Run:** Single execution of a simulation run consisting of one member of the batch parameter set. This single execution includes repetitions from stochastic analysis, since repetitions are not distributed among nodes.
- **Job:** A load sent to the nodes consisting of several runs.
- **ActualJobExecutionTime:** Execution time of the last load sent to a node.
- **NodeAvgRunExecutionTime:** Average execution time of a run from the last load sent to a node.
- **NodeProcessPower:** Average processing power of a node calculated from the last load sent to that node.
- **NPP:** Abbreviation for NodeProcessPower

- **NodeProcessRatio:** Ratio of node process power to the sum of all nodes process power.
- **ENPR:** Expected Node Processing Ratio. Expected ratio of the processing power of the node to the whole processing power in service of the coordinator. Sum of these values add up to 1.
- **NumRounds:** Number of rounds in the scheduling algorithm.
- **TotalNumNodes:** Total number of nodes in service to the coordinator.
- **TotalNumRuns:** Total number of runs to be executed for completing the batch parameter set.
- **numRunsPerRound:** Number of runs to be dispacthed in a round.

The general flow of all scheduling algorithms developed for Sim-PETEK are the same and it is presented as an activity diagram in Figure 3.1. The algorithms start with an initialization phase in which number of rounds and ENPR values of the computational nodes are initialized. ENPR initialization is done according to the number of CPU cores of the nodes. After the initialization phase, job dispatching rounds start. In each round, number of runs to be assigned to the idle simulator nodes are computed and jobs with associated runs are sent to the nodes for processing. When job completion messages are received, ENPR values of the nodes are updated and the next round starts. The rounding phase ends when all of the runs are finished and simulation results are received.

3.1 AMRS (Adaptive Multi-Round Synchronous Scheduling Algorithm)

Adaptive Multi-Round Synchronous Scheduling Algorithm is the first scheduling algorithm that was implemented for scheduling purposes of Sim-PETEK.

The algorithm starts with assigning an expected execution processing ratio to the available computational nodes. This assignment considers number of CPU cores of the nodes and assigns a ratio to each node between 0 and 1 where sum of all values are 1. This value can be represented as the following ratio for a node $node_i$:

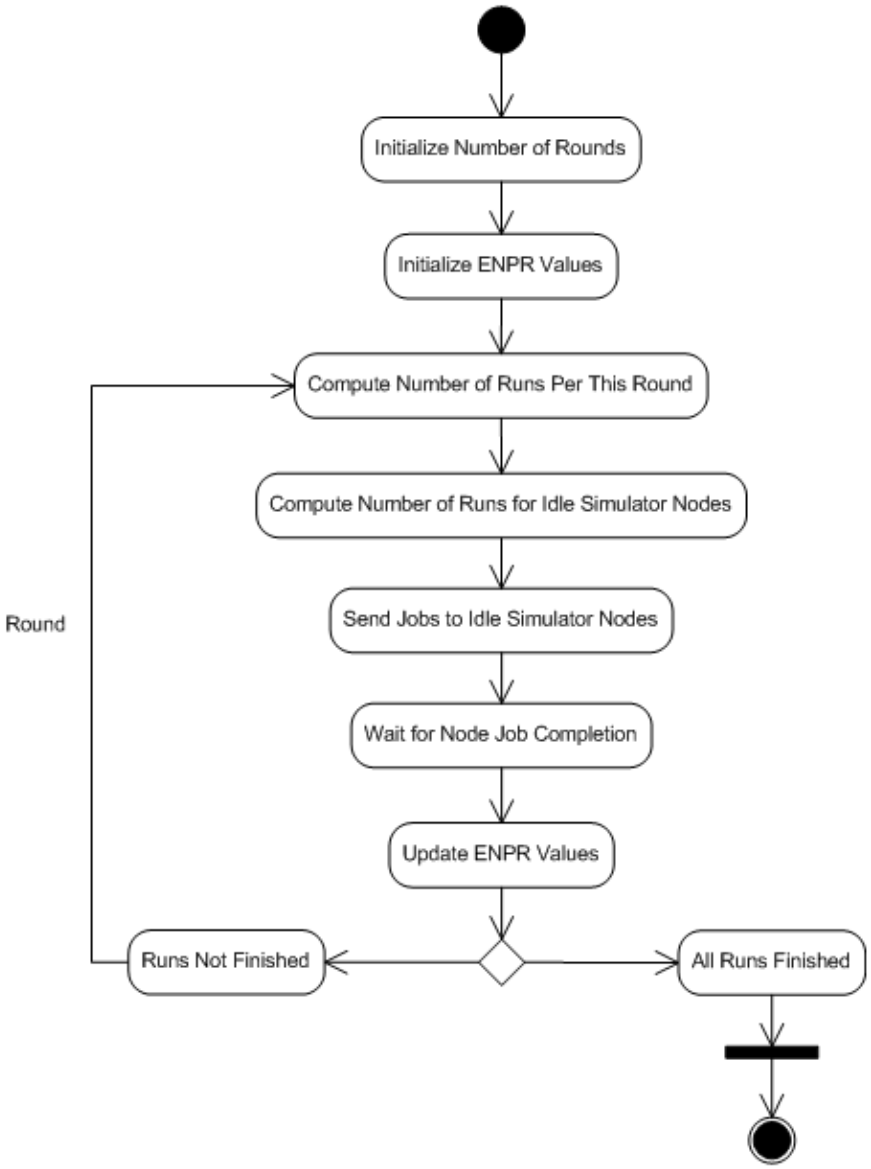


Figure 3.1: Sim-PETEK Scheduling Activity Diagram

$$\frac{Number\ of\ CPU\ Cores\ of\ node_i}{\sum_{j=1}^{N} Number\ of\ CPU\ Cores\ of\ node_j}$$

where N denotes the number of available computational nodes.

After this initial step, first round starts by dispatching runs to the computational nodes in accordance with their ENPR values. The round ends when all of the nodes finish their jobs and send the results. Before the next round starts, ENPR values of the nodes are updated and job assignments are made according to the newly computed processing ratios. This procedure goes on till all the jobs are completed.

AMRS contains ”synchronous” in its naming since it waits for all nodes to finish their jobs in a round (i.e. the next round starts when all of the nodes inform that they have finished their jobs).

```
(1)      Func Schedule
(2)           numRounds = A
(3)           Initialize ENPR[numNodes]
(4)           Initialize Jobs[numNodes]
(5)           For (i = 0; i < numIterations; i++)
(6)               numRunsPerRound = TotalNumRuns / numRounds
(7)               For (j = 0; j < TotalNumNodes; j++)
(8)                   Jobs[j] = numRunsPerRound * ENPR[j]
(9)               End For
(10)              Run jobs
(11)              Wait all nodes to complete their jobs
(12)              roundNumber = i
(13)              RecomputeENPR
(14)           End For
(15)     End Schedule
```

Figure 3.2: Pseudocode for AMRS Schedule Function

Figure 3.2 shows the pseudocode of the AMRS algorithm. The pseudocode briefly follows the steps that was previously explained in Figure 3.1. In the AMRS algorithm number of rounds mentioned as ”A” on the pseudocode line (2) is an optimal value that is determined by experimentation. In lines (3) and (4) initialization of ENPR and Jobs arrays are held. ENPR array keeps the expected node processing ratios of the nodes and Jobs array keeps the number of runs to be sent to the nodes (i.e. Jobs[i] = number of runs to be sent to node i). At line (5), the main loop of the AMRS scheduling algorithm starts. Firstly, number of runs of that round is computed at line (6) and then the number of runs in each node’s job is computed at lines (7), (8), and (9). At line (10) jobs are sent to the nodes and at line (11) all of them are waited to finish. Before the next round begins, ENPR values are recomputed at line (12).

```
(1)      Func InitializeENPR
(2)           For (i = 0; i < numNodes; i++)
(3)               ENPR[i] = Cores[i] / TotalNumberOfCores
(4)               NPPTable[i] = Cores[i]
(5)           End For
(6)      End InitializeENPR
```

Figure 3.3: Pseudocode for AMRS InitializeENPR Function

Detailed information about ENPR initialization is given by the pseudocode in Figure 3.3. For each node, ENPR value is set to a value which is a ratio of the number of CPU cores that the node under consideration contains, to the total number of CPU cores of the all available nodes. NPPTable at line (4) of the pseudocode is the table keeping processing powers of the nodes. In the initialization step, the procesing power of a node is thought to be proportional to its number of CPU cores.

```
(1)      Func RecomputeENPR
(2)           For (i = 0; i < numNodes; i++)
(3)               actualExecutionTime = jobExecutionTimes[i]
(4)               nodeRunTime = actualExecutionTime / Jobs[i]
(5)               nodePocessPower = 1/nodeRunTime
(6)               NPPTable[i] = nodePocessPower;
(7)           End For
(8)           sumNodeProcessPower = Sum(NPPTable.Values)
(9)           For (i = 0; i < numNodes; i++)
(10)              prevENPRValue = ENPR[i];
(11)              alpha = 1.0 / roundNumber;
(12)              nodeProcessRatio = NPPTable[i] / sumNodeProcessPower;
(13)              nextENPRValue = ((1 - alpha) * prevENPRValue +
                                   alpha * nodeProcessRatio)
(14)          End For
(15)      End RecomputeENPR
```

Figure 3.4: Pseudocode for AMRS RecomputeENPR Function

Furthermore, ENPR recomputation is mentioned in detail in figure 3.4. For this recomputation, node process power values are updated firstly. This update is done according to the execution time of the last job completed by the node. At line (4) the average time needed by the node for completing one run is computed and node process power is set accordingly at lines (5) and (6). At line (8) sum of the process powers of all nodes are computed. This sum is used afterwards in the node process ratio computation.

ENPR recomputation is held according to the following estimation formula which is defined in [21]:

$$T^1_{m,E}(t+1) = (1 - \alpha^1_m(t))T^1_{m,E}(t) + \alpha^1_m(t)T^1_{m,R}(t) \tag{3.1}$$

This formula uses an estimation technique which predicts the next value in a series as a weighted average of the currently observed value and the previous estimations. This is known as aging in literature [32]. For adapting this technique, we use historical measures of job execution times of the nodes. In the above formula, $T^1_{m,E}$(t+1) is the predicted execution time of a new run on node m, t is the number of times that the jobs are executed on m, $T^1_{m,R}$(t) is the actual execution time of the job on the same node, and α^1_m(t)(< 1) is the learning rate. Learning rate is kept as 1/2 in order to simplify the implementation of aging [32]. The values of $T^1_{m,E}$(t) and $T^1_{m,R}$(t) are provided by the previous executions.

Lines (9) - (14) of the RecomputeENPR function in Figure 3.4 applies the prediction formula and estimates the expected execution powers of the nodes for the next round. As we have mentioned before, number of runs to be sent to a node is proportional with this estimated value.

For further understanding of AMRS, we should work with a numeric example. In this example, a simulation containing 60 runs and applying 10 MC trials in each run is traced. AMRS applies 3 rounds for dispatching the jobs to 5 computational resource nodes shown in Table 3.1.

Table 3.1: Computational Resource Nodes for the Example

Node_A	**Node_B**	**Node_C**	**Node_D**	**Node_E**
8-Core	4-Core	4-Core	2-Core	2-Core

In the first place, ENPR initialization of the nodes are made and they will be set as follows:

$$\begin{aligned}
Node_A &\Rightarrow \quad 8/(8+4+4+2+2) = 0.4 \\
Node_B &\Rightarrow \quad 4/(8+4+4+2+2) = 0.2 \\
Node_C &\Rightarrow \quad 4/(8+4+4+2+2) = 0.2 \\
Node_D &\Rightarrow \quad 2/(8+4+4+2+2) = 0.1 \\
Node_E &\Rightarrow \quad 2/(8+4+4+2+2) = 0.1
\end{aligned}$$

Since AMRS will be dispatching 60 runs in 3 rounds, 20 rounds will be dispatched in a

round. In the first round, distribution will be in a manner shown in Table 3.2. As mentioned previously, this distribution is made according to ENPR values. For example,

$$NumberOfRunsForNode\_A = ENPR_{Node\_A} * NumberOfRunsPerRound = 0.4 * 20 = 8$$

Table 3.2: AMRS First Round Job Distribution

Node_A	**Node_B**	**Node_C**	**Node_D**	**Node_E**
8 Runs	4 Runs	4 Runs	2 Runs	2 Runs

In our example we assume that, Node_A finishes 8 runs in 10 seconds, Node_B finishes 4 runs in 25 seconds, Node_C finishes 4 runs in 10 seconds, Node_D finishes 2 runs in 20 seconds, and Node_E finishes 2 runs in 10 seconds. Our further assumption is that this performance of the nodes does not change during the whole simulation execution.

When all nodes finish their jobs, first round finishes and ENPR values are recomputed. For this recomputation, node process power values are computed in the first place. Table 3.3 shows node process power values of our nodes with computation details.

Table 3.3: AMRS Node Process Power Values After First Round

Node_A	1 / (10/8) = 0.8
Node_B	1 / (25/4) = 0.16
Node_C	1 / (10/4) = 0.4
Node_D	1 / (20/2) = 0.1
Node_E	1 / (10/2) = 0.2

ENPR values are then recomputed as follows according to Equation 3.1 where the learning rate $\alpha_m^1(t)$ is taken as 1/2:

$$ENPR_{Node\_A} = \frac{1}{2} * PreviousENPR_{Node\_A} + \frac{1}{2} * \frac{NPP_{Node\_A}}{\sum NPPValuesofNodes} = \frac{1}{2} * 0.4 + \frac{1}{2} * \frac{0.8}{1.66} \approx 0.44$$

$$ENPR_{Node\_B} = \frac{1}{2} * PreviousENPR_{Node\_B} + \frac{1}{2} * \frac{NPP_{Node\_B}}{\sum NPPValuesofNodes} = \frac{1}{2} * 0.2 + \frac{1}{2} * \frac{0.16}{1.66} \approx 0.15$$

$$ENPR_{Node\_C} = \frac{1}{2} * PreviousENPR_{Node\_C} + \frac{1}{2} * \frac{NPP_{Node\_C}}{\sum NPPValuesofNodes} = \frac{1}{2} * 0.2 + \frac{1}{2} * \frac{0.4}{1.66} \approx 0.22$$

$$ENPR_{Node\_D} = \frac{1}{2} * PreviousENPR_{Node\_D} + \frac{1}{2} * \frac{NPP_{Node\_D}}{\sum NPPValuesofNodes} = \frac{1}{2} * 0.1 + \frac{1}{2} * \frac{0.1}{1.66} \approx 0.08$$

$$ENPR_{Node\_E} = \frac{1}{2} * PreviousENPR_{Node\_E} + \frac{1}{2} * \frac{NPP_{Node\_E}}{\sum NPPValuesofNodes} = \frac{1}{2} * 0.1 + \frac{1}{2} * \frac{0.2}{1.66} \approx 0.11$$

After ENPR recomputation, second round starts and runs are distributed according to the newly computed ENPRs as shown in Table 3.4;

Table 3.4: AMRS Second Round Job Distribution

Node_A	$0.44 * 20 \approx 9$ Runs
Node_B	$0.15 * 20 = 3$ Runs
Node_C	$0.22 * 20 \approx 5$ Runs
Node_D	$0.08 * 20 \approx 2$ Runs
Node_E	$0.11 * 20 \approx 3$ Runs

When jobs of the second round are finished, ENPR recomputation is held once more and runs of last round are distributed accordingly. Table 3.5 shows node process power values of our nodes after finishing the second round.

Table 3.5: AMRS Node Process Power Values After Second Round

Node_A	1 / (12/9) = 0.75
Node_B	1 / (20/3) = 0.15
Node_C	1 / (13/5) = 0.38
Node_D	1 / (20/2) = 0.1
Node_E	1 / (15/3) = 0.2

At this point, we should mention how we compute actual execution times. Sim-PETEK architecture is designed such that a job sent to a simulator includes a number of runs, say N, and an indicator of how many Monte Carlo trials should be held, say M. This means that the simulator which has been assigned for a job will be running N*M simulations. Number of

simulations running in parallel on a simulator is proportional with the number of CPU cores that the simulator has. For example, a dual core simulator can not run simulations in parallel, a quad core one can run 4 simulations, a simulator with 8 cores can run 8 simulations in parallel, and a simulator with 16 cores can run 15 simulations.

As an example Node_A is computed to finish 9 runs in 12 seconds. For this simple example, we assume that 10 MC trials are applied for each run meaning that when we send 8 runs to Node_A, this means 8*10 = 80 different simulations. Since there are 8 CPU cores which can run in parallel, each core will be handling 10 simulations and finishing the whole job in 10 seconds. For the 9 runs case, there are 9*10 = 90 different simulations. 6 of the cores will be handling 11 simulations whereas 2 of them will handle 12. As a result, 12 seconds will be spent for finishing the whole job.

Recomputed ENPR values and distributions of the third round are presented in Table 3.6.

Table 3.6: AMRS Third Round ENPR Values and Job Distribution

Node_A	0.46	10 Runs
Node_B	0.12	3 Runs
Node_C	0.23	5 Runs
Node_D	0.07	2 Runs (Since all runs are distributed, these runs are not sent)
Node_E	0.12	3 Runs (Since all runs are distributed, these runs are not sent)

Figure 3.5 shows job distributions of AMRS with a timeline. It can be easily seen that most of the nodes wait idle when they are waiting for the other nodes to finish.

3.2 AMRA (Adaptive Multi-Round Asynchronous Scheduling Algorithm)

Asynchronous scheduling algorithm is an improved version of synchronous scheduling algorithm. Instead of waiting for all of the nodes to finish their tasks in each round, asynchronous scheduling algorithm dispatches new jobs to the nodes immediately after they finish their runs. Before the job assignments expected execution power values of the nodes are updated and assignments are made accordingly.

Figure 3.6 shows the pseudocode of AMRA Schedule function. The difference of this function from AMRS algorithm is at lines between (12) and (15). AMRS algorithm waits for all nodes

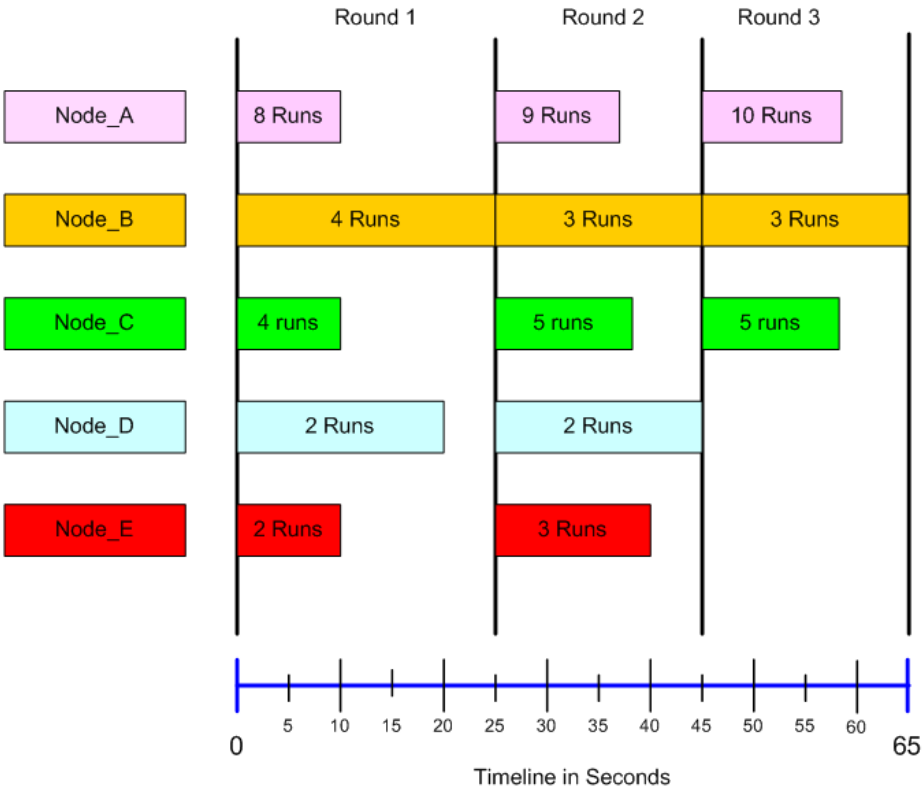


Figure 3.5: AMRS Job Distributions

to complete their jobs whereas AMRA assigns new job to a node immediately after it finishes its runs. With this approach AMRA tries to reduce the idle time spent by AMRS for all nodes to finish their jobs. ENPR initialization and recomputation parts of AMRS and AMRA works in the same manner. However, for AMRA case, first ENPR recomputation is held when all nodes complete their first jobs. Otherwise, ENPR value of a node which has not completed any job would be computed according to its number of CPU cores only which would yield incorrect ENPR values for further rounds.

In order to see the effects of the improvement made by AMRA, same numeric example of AMRS is traced. ENPR computation and job assignments are handled in the same way as AMRS. The only difference is that when a node finishes its job, a new job is assigned to it without waiting for the other nodes.

```
(1)      Func Schedule
(2)           numRounds = A
(3)           Initialize ENPR[numNodes]
(4)           Initialize Jobs[numNodes]
(5)           While (remainingRunCount != 0)
(6)               numRunsPerRound = TotalNumRuns / numRounds
(7)               For (j = 0; j < TotalNumNodes; j++)
(8)                   Jobs[j] = numRunsPerRound * ENPR[j]
(9)                   remainingRunCount -= Jobs[j]
(10)              End For
(11)              Run jobs
(12)              Wait any node to complete its job
(13)              If(allNodesCompletedFirstJobs)
(14)                  RecomputeENPR
(15)              End If
(16)          End While
(17)     End Schedule
```

Figure 3.6: Pseudocode for AMRA Schedule Function

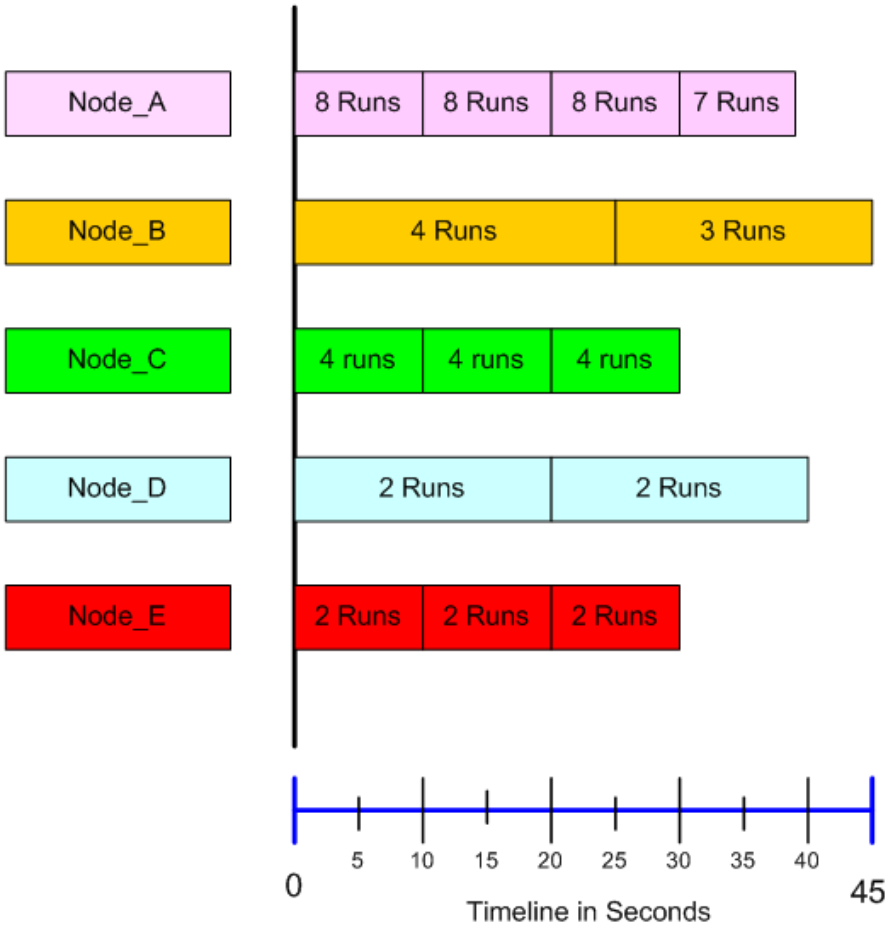


Figure 3.7: AMRA Job Distributions

Figure 3.7 presents the timeline and job distributions for AMRA. This figure depicts that idle wait times between the rounds are prevented. By this approach, simulation execution time is reduced from 65 seconds (shown in Figure 3.5) to 45 seconds.

3.3 Improved Adaptive Multi-Round Asynchronous Scheduling Algorithms

Asynchronous scheduling algorithm is further improved and the following improved adaptive multi-round asynchronous scheduling algorithms are defined and implemented for Sim-PETEK scheduling purposes.

3.3.1 SAMRA (Smart Adaptive Multi-Round Asynchronous Scheduling Algorithm)

In this improved version of AMRA a probing phase is added before the first round. In the probing phase a sample of tasks are assigned to the simulator nodes and by this way the computation powers are approximated. Furthermore, this algorithm is designed in a way which is more compatible with Sim-PETEK design.

As mentioned previously in Section 3.1, Sim-PETEK architecture is designed in a way such that a dual core simulator can not run simulations in parallel, a quad core one can run 4 simulations, and a simulator with 16 cores can run 15 simulations in parallel. With this aspect in mind, SAMRA tries to arrange jobs as multiples of 1, 4, and 15 for dual core, quad core, and 16-core simulators respectively. The aim of this approach is to prevent idle times for some of the CPU cores. Since, number of Monte Carlo trials are provided as multiples of 10 for performance analysis tests of this study, 4 and 15 can be reduced to 2 and 3. In the pseudocode this reduced values are used for simplicity. Another approach for providing simplicity in the pseudocodes is that only dual core, quad core, and 16-core computational resource cases are included. That kind of parts are extended for different number of cores during implementation.

In Figure 3.8 the pseudocode SAMRA is provided. Lines (12)-(26) is the probing part. In the probing part, sample runs are assigned to the resources as multiples of 1,2,3, and etc. for resources having different number of CPU cores. Similarly, jobs are assigned as multiples of 1,2, and 3 between lines (27) and (45) during the rounds.

```
(1)     Func Schedule
(2)          numRounds = A
(3)          Initialize ENPR[numNodes]
(4)          Initialize Jobs[numNodes]
(5)          iterationCount == 1
(6)          While (remainingRunCount != 0)
(7)              numRunsPerRound = TotalNumRuns / numRounds
(8)              For (j = 0; j < TotalNumNodes; j++)
(9)                  If ( ENPR[j] < T )
(10)                      continue
(11)                 End If
(12)                 If (iterationCount == 1)
(13)                      Switch (numberOfCores[j])
(14)                           case 2:
(15)                                Jobs[j] = 1
(16)                                break
(17)                           case 4:
(18)                                Jobs[j] = 2
(19)                                break
(20)                           case 16:
(21)                                Jobs[j] = 3
(22)                                break
(23)                           default:
(24)                                Jobs[j] = 1
(25)                                break
(26)                 End If
(27)                 Else
(28)                      Jobs[j] = numRunsPerRound * ENPR[j]
(29)                     Switch (numberOfCores[j])
(30)                          case 2:
(31)                               break
(32)                          case 4:
(33)                               If (Jobs[j] > 2)
(34)                                   Jobs[j] -= Jobs[j] % 2;
(35)                               End If
(36)                               break
(37)                          case 16:
(38)                               If (Jobs[j] > 3)
(39)                                   Jobs[j] -= Jobs[j] % 3;
(40)                               End If
(41)                               break
(42)                          default:
(43)                               break
(44)                 End Else
(45)                 remainingRunCount -= Jobs[j]
(46)             End For
(47)             iterationCount++
(48)             Run jobs
(49)             Wait any node to complete its job
(50)             If(allNodesCompletedFirstJobs)
(51)                 RecomputeENPR
(52)             End If
(53)         End While
(54)    End Schedule
```

Figure 3.8: Pseudocode for SAMRA Schedule Function

For further understanding of the algorithm and observing the improvement, the numeric example of AMRS and AMRA is also traced with SAMRA. Firstly ENPR initialization of the nodes are made according to their CPU cores:

$$
\begin{aligned}
Node\_A &\Rightarrow \quad 8/20 = 0.4 \\
Node\_B &\Rightarrow \quad 4/20 = 0.2 \\
Node\_C &\Rightarrow \quad 4/20 = 0.2 \\
Node\_D &\Rightarrow \quad 2/20 = 0.1 \\
Node\_E &\Rightarrow \quad 2/20 = 0.1
\end{aligned}
$$

where 20 is the total number of CPU cores (i.e. 8 + 4 + 4 + 2 + 2).

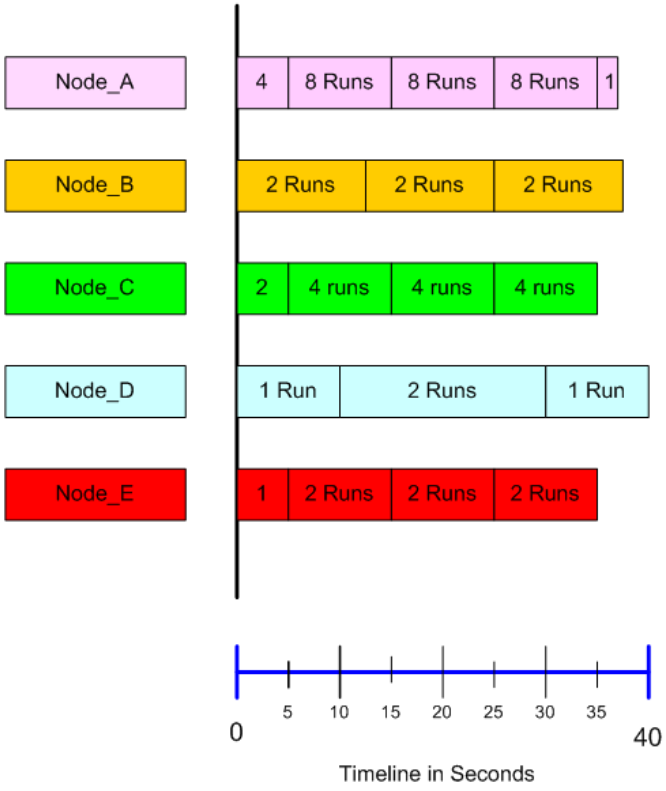


Figure 3.9: SAMRA Job Distributions

After the ENPR initialization job dispatching starts. Timeline and distributions for SAMRA are presented in Figure 3.9. As it can be seen from the figure, small number of runs are sent to the nodes at the beginning for probing purposes. Node_A is probed with 4 runs, Node_B

is probed with 2 runs, and Node_C, Node_D, and Node_E are probed with 2, 1, and 1 runs respectively. In the 5^{th} second, Node_A, Node_C, and Node_E finish their runs and new jobs are assigned to them. Since all nodes have not completed their runs yet, ENPR values are not updated and job assignments of Node_A, Node_C, and Node_E are made according to their initial ENPRs:

$$Node\_A \Rightarrow \ 0.4 * 20 = 8 \Rightarrow \ 8 - (8\%8) = 8 \ Runs$$

$$Node\_C \Rightarrow \ 0.2 * 20 = 4 \Rightarrow \ 4 - (4\%2) = 4 \ Runs$$

$$Node\_E \Rightarrow \ 0.1 * 20 = 2 \Rightarrow \ 2 - (2\%1) = 2 \ Runs$$

where 20 is the number of runs to be dispatched in a round.

In the 10^{th} second, Node_D finishes its run and assigned with 2 new runs (0.1 * 20 = 2). When Node_E finishes its run in 12.5^{th} second ENPR values of the nodes are updated. Table 3.7 shows Node Process Power values and Table 3.8 shows ENPR computations.

Table 3.7: SAMRA Node Process Power Values After Probing

Node_A	1 / (5/4) = 0.8
Node_B	1 / (12.5/2) = 0.16
Node_C	1 / (5/2) = 0.4
Node_D	1 / (10/1) = 0.1
Node_E	1 / (5/1) = 0.2

Table 3.8: SAMRA ENPR Values After Probing

Node_A	$1/2 * 0.4 + 1/2 * 0.8/1.66 \approx 0.44$
Node_B	$1/2 * 0.2 + 1/2 * 0.16/1.66 \approx 0.15$
Node_C	$1/2 * 0.2 + 1/2 * 0.4/1.66 \approx 0.22$
Node_D	$1/2 * 0.1 + 1/2 * 0.1/1.66 \approx 0.08$
Node_E	$1/2 * 0.1 + 1/2 * 0.2/1.66 \approx 0.11$

In 15^{th} second, Node_A, Node_C, and Node_E again finish their runs and their new job assignments are made after ENPR update. Details of ENPR update is shown in Table 3.9 and Table 3.10.

Table 3.9: SAMRA Node Process Power Values After 15 Seconds

Node_A	1 / (10/8) = 0.8
Node_B	1 / (12.5/2) = 0.16
Node_C	1 / (10/4) = 0.4
Node_D	1 / (10/1) = 0.1
Node_E	1 / (10/2) = 0.2

Table 3.10: SAMRA ENPR Values After 15 Seconds

Node_A	$1/2 * 0.44 + 1/2 * 0.8/1.66 \approx 0.46$
Node_B	$1/2 * 0.15 + 1/2 * 0.16/1.66 \approx 0.12$
Node_C	$1/2 * 0.22 + 1/2 * 0.4/1.66 \approx 0.23$
Node_D	$1/2 * 0.08 + 1/2 * 0.1/1.66 \approx 0.07$
Node_E	$1/2 * 0.11 + 1/2 * 0.2/1.66 \approx 0.12$

Job assignments of Node_A, Node_C, and Node_E are as follows:

$$Node\_A \Rightarrow \ 0.46 * 20 = 9.2 \Rightarrow \ 9.2 - (9.2\%8) = 8 \ Runs$$
$$Node\_C \Rightarrow \ 0.23 * 20 = 4.6 \Rightarrow \ 4.6 - (4.6\%2) = 4 \ Runs$$
$$Node\_E \Rightarrow \ 0.12 * 20 = 2.4 \Rightarrow \ 2.4 - (2.4\%1) = 2 \ Runs$$

Further tracing of the algorithm goes on in the same manner until all runs are distributed. SAMRA finishes the same amount of work in 40 seconds (shown in Figure 3.9) which is a less amount of time than AMRA and AMRS which spend 45 and 65 seconds respectively.

3.3.2 SSSE-AMRA (Slow Start - Slow End Adaptive Multi-Round Asynchronous Scheduling Algorithm)

All of the algorithms described in the previous sections handle job assignments in a fixed number of rounds which is determined by experimentation and the number of runs to be assigned in a round is fixed. Our experiments with these algorithms have shown that there is so much waiting time at the last round for all of the resources to finish their jobs. This idle waiting condition is tried to be solved by a slow start-slow end scheduling algorithm where the number of runs to be assigned in a round is determined by a sinusoidal function meaning

that the algorithm starts with small number of runs in a round, increases that number to some extent, and then decreases.

Run count to be dispatched in a round is determined by the following formula:

$$RoundNumber < peak \Rightarrow \lfloor TotalRunCount * \frac{\sin(\frac{RoundNumber*\pi}{2*peak})}{mValue} \rfloor$$

$$RoundNumber \geq peak \Rightarrow \lfloor TotalRunCount * \frac{\sin(\frac{(RoundNumber+kValue-peak)*\pi}{2*kValue})}{mValue} \rfloor$$

The first part of this formula, i.e. when $RoundNumber < peak$, increases the number of runs in a round to some extent, and the second part decreases from that point on. Beginning by dispatching small number of runs helps collecting several historical metrics about nodes' execution power values. By this way better estimates can be done by the estimation formula used for ENPR computation. Furthermore, assigning small number of runs at the beginning is a precaution for preventing bottleneck of slow nodes. The decreasing part of this algorithm is completed in more number of rounds. This is for preventing wait conditions for slow nodes to complete their jobs at the last rounds by dispatching less number of runs. In this formula, peak represents round number at which maximum number of runs to be dispatched in a round is achieved, kValue is the total number of rounds to be applied, and mValue is the normalization factor. This normalization is necessary for obtaining the total area under the sinusodial curve as 1 which means in our case that 100% of the runs are dispatched when all rounds are finished. A sample number of runs versus rounds graphic can be seen in Figure 3.10 which is drawn for 1500 runs in total. In this sample graphic, peak is 5, kValue is 50 and mValue is 33. The optimal values for peak, kValue, and mValue are determined by experimentation.

If communication costs of Sim-PETEK were not negligible, this slow start-slow end scheduling approach would not be working efficient because this approach increases total number of rounds and there is a communication cost at each round. For many architectures, there is a huge data transfer which increases the communication cost. However, Sim-PETEK architecture is designed in a way that transfered data is really small so that communication costs are negligible.

Pseudocode of SSSE-AMRA Schedule function is shown in Figure 3.11. The main difference of the algorithm can be seen on lines (8)-(12) of the pseudocode where number of runs for

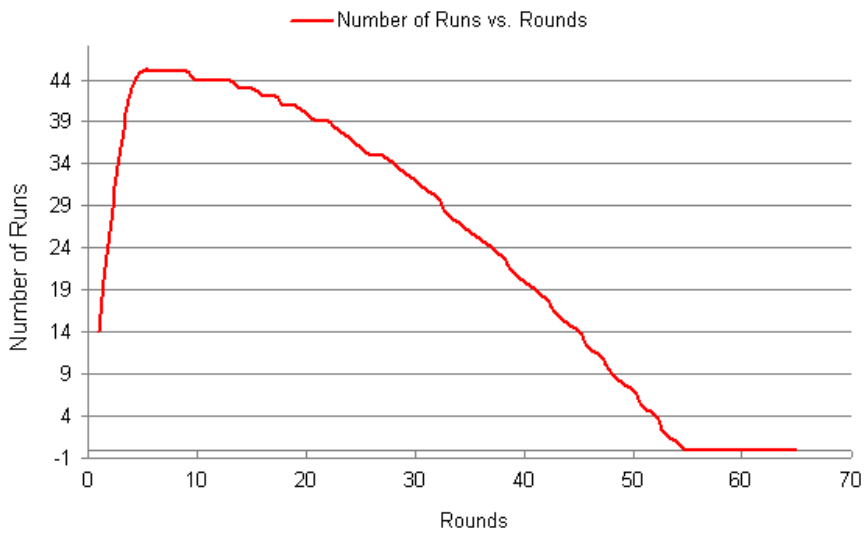


Figure 3.10: Number of Runs vs. Rounds

the associated round is calculated. The round number counter namely ”roundNumber” in the pseudocode is incremented when all of the runs in a round are dispatched (lines (41)-(44)). This action means that the next round starts and number of runs of that round is determined accordingly.

When we work with the same numeric example by using SSSE-AMRA with peak = 1, kValue = 3, and mValue = 2.3, number of runs that are distributed in rounds of SSSE-AMRA are computed as follows:

$$Round\ 1 \Rightarrow \lfloor \frac{TotalRunCount}{mValue} * \sin(\frac{RoundNumber*\pi}{2*peak}) \rfloor \Rightarrow \lfloor \frac{60}{2.3} * \sin(\frac{1*\pi}{2*1}) \approx 26$$

$$Round\ 2 \Rightarrow \lfloor \frac{TotalRunCount}{mValue} * \sin(\frac{(RoundNumber+kValue-peak)*\pi}{2*kValue}) \rfloor \Rightarrow \lfloor \frac{60}{2.3} * \sin(\frac{(2+3-1)*\pi}{2*3}) \rfloor \approx 22$$

$$Round\ 3 \Rightarrow \lfloor \frac{TotalRunCount}{mValue} * \sin(\frac{(RoundNumber+kValue-peak)*\pi}{2*kValue}) \rfloor \Rightarrow \lfloor \frac{60}{2.3} * \sin(\frac{(3+3-1)*\pi}{2*3}) \rfloor \approx 12$$

This algorithm initializes ENPR values in the same way as the previously traced algorithms, AMRS, AMRA, and SAMRA. After the initialization phase first round starts and runs are dispatched as presented below:

$$Node\_A \Rightarrow \ 0.4 * 26 = 10.4 \Rightarrow \ 10.4 - (10.4\%8) = 8\ Runs$$

$$Node\_B \Rightarrow \ 0.2 * 26 = 5.2 \Rightarrow \ 5.2 - (5.2\%2) = 4\ Runs$$

```
(1)      Func Schedule
(2)           Initialize ENPR[numNodes]
(3)           Initialize Jobs[numNodes]
(4)           roundNumber = 1
(5)           numOfDispacthedRunsOfARound = 0
(6)           While (remainingRunCount != 0)
(7)               If (roundNumber < peak)
(8)                    numRunsForRound = (TotalNumberOfRuns / mValue) *
                                         sin((roundNumber * PI) / (2*peak));
(9)               End If
(10)              Else
(11)                   numRunsForRound = (TotalNumberOfRuns / mValue) *
                                         sin((roundNumber + kValue - peak) *
                                                    Math.PI) / (2 * kValue));
(12)              End Else
(13)              For (j = 0; j < TotalNumNodes; j++)
(14)                  If ( ENPR[j] < T )
(15)                       continue
(16)                  End If
(17)                  Jobs[j] = numRunsForRound * ENPR[j]
(18)                 Switch (numberOfCores[j])
(19)                      case 2:
(20)                           break
(21)                      case 4:
(22)                           If (Jobs[j] > 2)
(23)                                Jobs[j] -= Jobs[j] % 2;
(24)                           End If
(25)                           break
(26)                      case 16:
(27)                           If (Jobs[j] > 3)
(28)                                Jobs[j] -= Jobs[j] % 3;
(29)                           End If
(30)                           break
(31)                      default:
(32)                           break
(33)                  remainingRunCount -= Jobs[j]
(34)                  numOfDispacthedRunsOfARound += Jobs[j]
(35)              End For
(36)              Run jobs
(37)              Wait any node to complete its job
(38)              If(allNodesCompletedFirstJobs)
(39)                  RecomputeENPR
(40)              End If
(41)              If (numOfDispacthedRunsOfARound == numRunsForRound)
(42)                  numRounds++
(43)                  numOfDispacthedRunsOfARound = 0
(44)              End If
(45)          End While
(46)     End Schedule
```

Figure 3.11: Pseudocode for SSSE-AMRA Schedule Function

$$Node\_C \Rightarrow \ 0.2 * 26 = 5.2 \Rightarrow \ 5.2 - (5.2\%2) = 4\ Runs$$
$$Node\_D \Rightarrow \ 0.1 * 26 = 2.6 \Rightarrow \ 2.6 - (2.6\%1) = 2\ Runs$$
$$Node\_E \Rightarrow \ 0.1 * 26 = 2.6 \Rightarrow \ 2.6 - (2.6\%1) = 2\ Runs$$

Figure 3.12 shows all distributions of SSSE-AMRA on a timeline. As the figure shows, 3 of the nodes finish their runs at the 10^{th} second and start with their new jobs. At that time, 20 runs of first round are distributed and 6 runs are left. So, number of runs of the new jobs are computed as follows:

$$Node\_A \Rightarrow \ 0.4 * 26 = 10.4 \Rightarrow \ 10.4 - (10.4\%8) = 8\ Runs$$
$$Node\_C \Rightarrow \ 0.2 * 22 = 4.4 \Rightarrow \ 4.4 - (4.4\%2) = 4\ Runs$$
$$Node\_E \Rightarrow \ 0.1 * 22 = 2.2 \Rightarrow \ 2.2 - (2.2\%1) = 2\ Runs$$

When 20 seconds pass, Node_A, Node_C, and Node_E finish their second jobs and Node_D finishes the first one. Then, it is time for a new distribution which is held as:

$$Node\_A \Rightarrow \ 0.4 * 22 = 8.8 \Rightarrow \ 8.8 - (8.8\%8) = 8\ Runs$$
$$Node\_C \Rightarrow \ 0.2 * 22 = 4.4 \Rightarrow \ 4.4 - (4.4\%2) = 4\ Runs$$
$$Node\_E \Rightarrow \ 0.1 * 22 = 2.2 \Rightarrow \ 2.2 - (2.2\%1) = 2\ Runs$$
$$Node\_D \Rightarrow \ 0.1 * 12 = 1.2 \Rightarrow \ 1.2 - (1.2\%1) = 1\ Run$$

Here, when runs of Node_A, Node_C, and Node_E arranged, number of distributed runs is equal to 48 meaning that third round should start. For this reason, runs of Node_D is computed according to the third round.

At the 25^{th} second, Node_B finishes its first job meaning that ENPR recomputation should be held and a new job should be assigned to Node_B. Since, this computation is same with the previous algorithms, details are not given and computed values are presented in Table 3.11. A job with 2 runs ($0.15 * 12 \approx 2$) is assigned to Node_B after the computation.

Job distribution procedure goes on in a similar manner until all runs are completed. Figure 3.12 depicts that SSSE-AMRA has finished whole work in 39 seconds which is a bit better than SAMRA.

Table 3.11: SSSE-AMRA ENPR Values After First Jobs Completed

Node_A	0.44
Node_B	0.15
Node_C	0.22
Node_D	0.08
Node_E	0.11

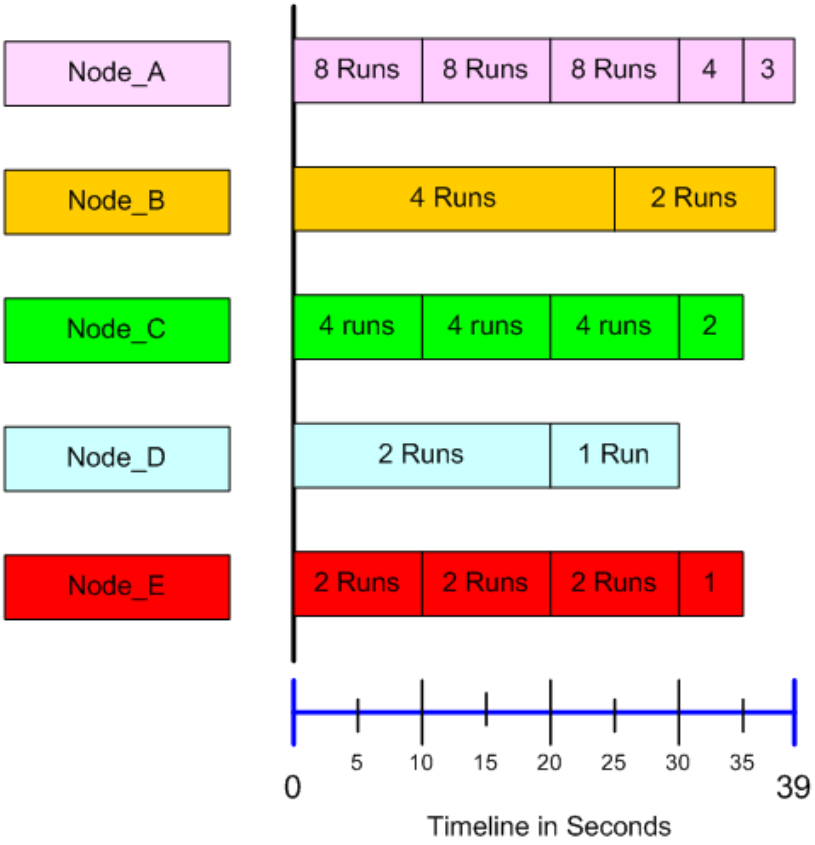


Figure 3.12: SSSE-AMRA Job Distributions

3.3.3 ISSSE-AMRA (Improved Slow Start - Slow End Adaptive Multi-Round Asynchronous Scheduling Algorithm)

ISSSE-AMRA improves SSSE-AMRA in the last iteration by redistributing the jobs of slow nodes to the faster ones which have completed their jobs. This is a small improvement but it prevents redundant waits for the bottlenecked nodes if there are any. Pseudocode for job redistribution part of ISSSE-AMRA schedule function is shown in Figure 3.13. This code part is inserted after line (38) of SSSE-AMRA scheduling function of Figure 3.11. The other parts of the scheduling functions are the same.

```
(1)              If(remainingRunCount == 0)
(2)                   maxFinishTime = 0
(3)                   Foreach running node "node_i"
(4)                       If(finishTime(node_n, Jobs[i]) < finishTime(node_i,Jobs[i])
                            &&
                            finishTime(node_i, Jobs[i]) > maxFinishTime)
(5)                            Jobs[n] = Jobs[i]
(6)                            maxFinishTime = finishTime(node_i, Jobs[i])
(7)                       End If
(8)                   End Foreach
(9)                   Run job
(10)              End If
(11)              If (numOfDispacthedRunsOfARound == numRunsForRound)
(12)                  numRounds++
(13)                  numOfDispacthedRunsOfARound = 0
(14)              End If
```

Figure 3.13: Pseudocode for Task Redistribution Part of ISSSE-AMRA Schedule Function

When we trace the same numeric example with ISSSE-AMRA, an improvement is not possible because as it can be seen from Figure 3.12, Node_A,which is the node with greatest execution power, finishes lastly and job redistribution is not held.

However, if the case were the one shown in Figure 3.14, Node_B's job will be rescheduled to Node_C which reduces time cost from 45 seconds to 40 seconds as shown in Figure 3.15.

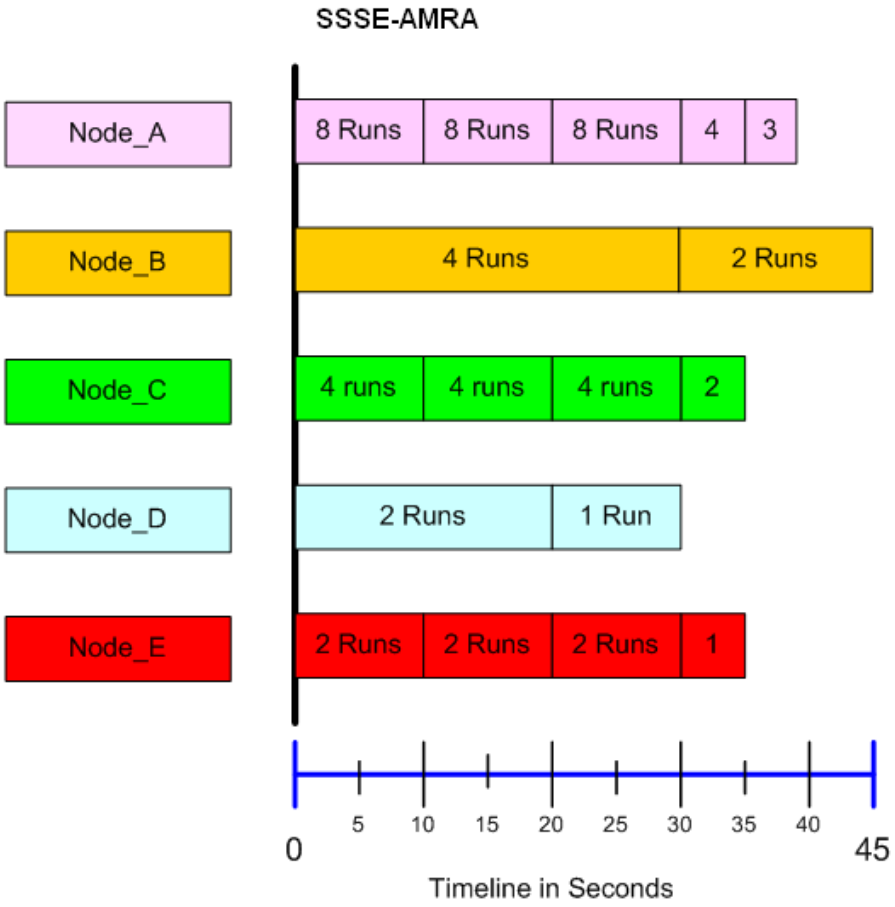


Figure 3.14: SSSE-AMRAJob Distributions: Assumed Case

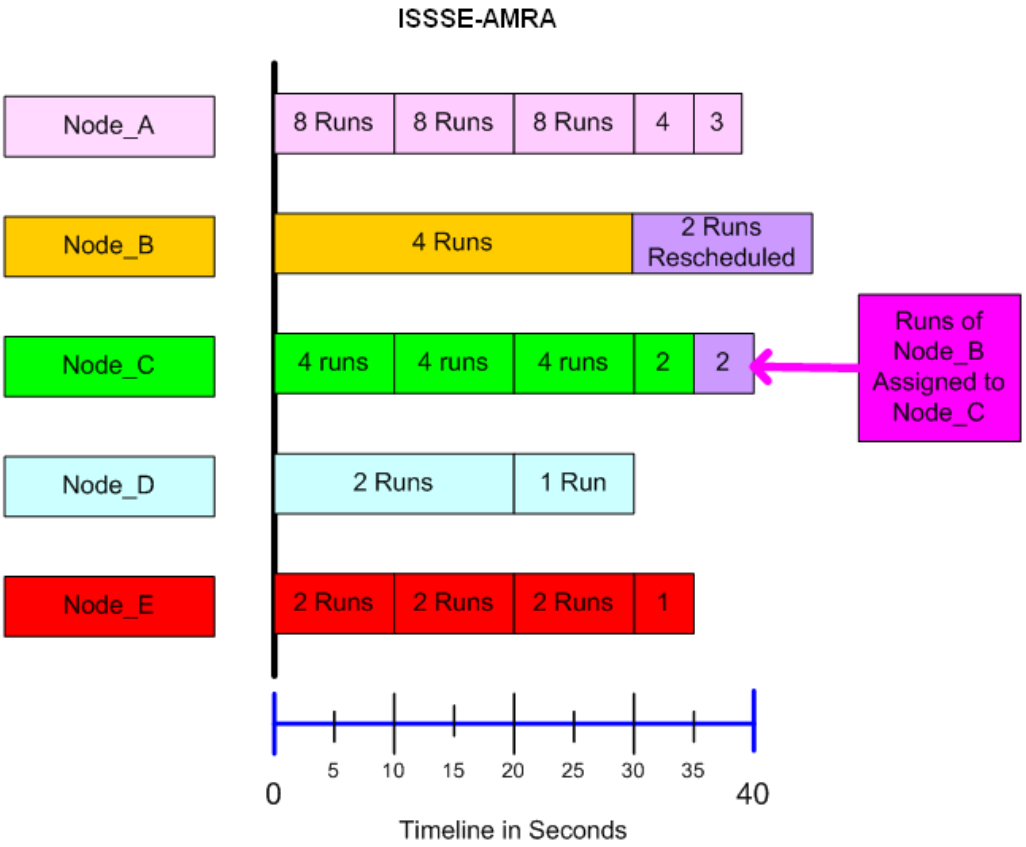


Figure 3.15: ISSSE-AMRA Job Distributions: Assumed Case

CHAPTER 4

IMPLEMENTATION OF SCHEDULING ALGORITMS

Implementation details of Sim-PETEK scheduling algorithms are presented in this chapter. The algorithms are implemented with C# on Microsoft Visual Studio 2008 platform same as Sim-PETEK.

As mentioned in Section 2.3.3.1, Coordinator Grid Service contains a Scheduler component which is responsible for producing optimized job scheduling plans. The study in this thesis has focused on producing and analyzing such scheduling plans by integrating the proposed scheduling algorithms defined in Chapter 3 into the Scheduler component of Sim-PETEK.

4.1 Scheduling Workflow

Figure 4.1 shows the activity diagram of Sim-PETEK scheduling. The flow starts with reading scheduling algorithms' properties from an XML file named as "SchedulerList.xml" and "Scheduling Algorithms List" is populated by these properties (Details of "SchedulerList.xml" is provided in Section 4.2). In the second step of the flow, a simulation execution request is popped from the "Simulation Execution Request Queue" which keeps the requests sent by the "Simulation Application" to the "Coordinator". In the third step, a previously unselected scheduling algorithm is selected from the algorithm list and the corresponding scheduling object is created at run time. From this point on, scheduling algorithm starts running and produces the schedules in an iterative manner. When the schedule is formed, it is sent to the "Job Producer" module which prepares the jobs. Prepared jobs are then sent to the simulators by "Job Manager". This procedure goes on until all runs in the simulation execution request are executed. When the simulation finishes, it is repeated with other scheduling algorithms in the algorithm list in order to make comparisons. The whole process after the third step of the

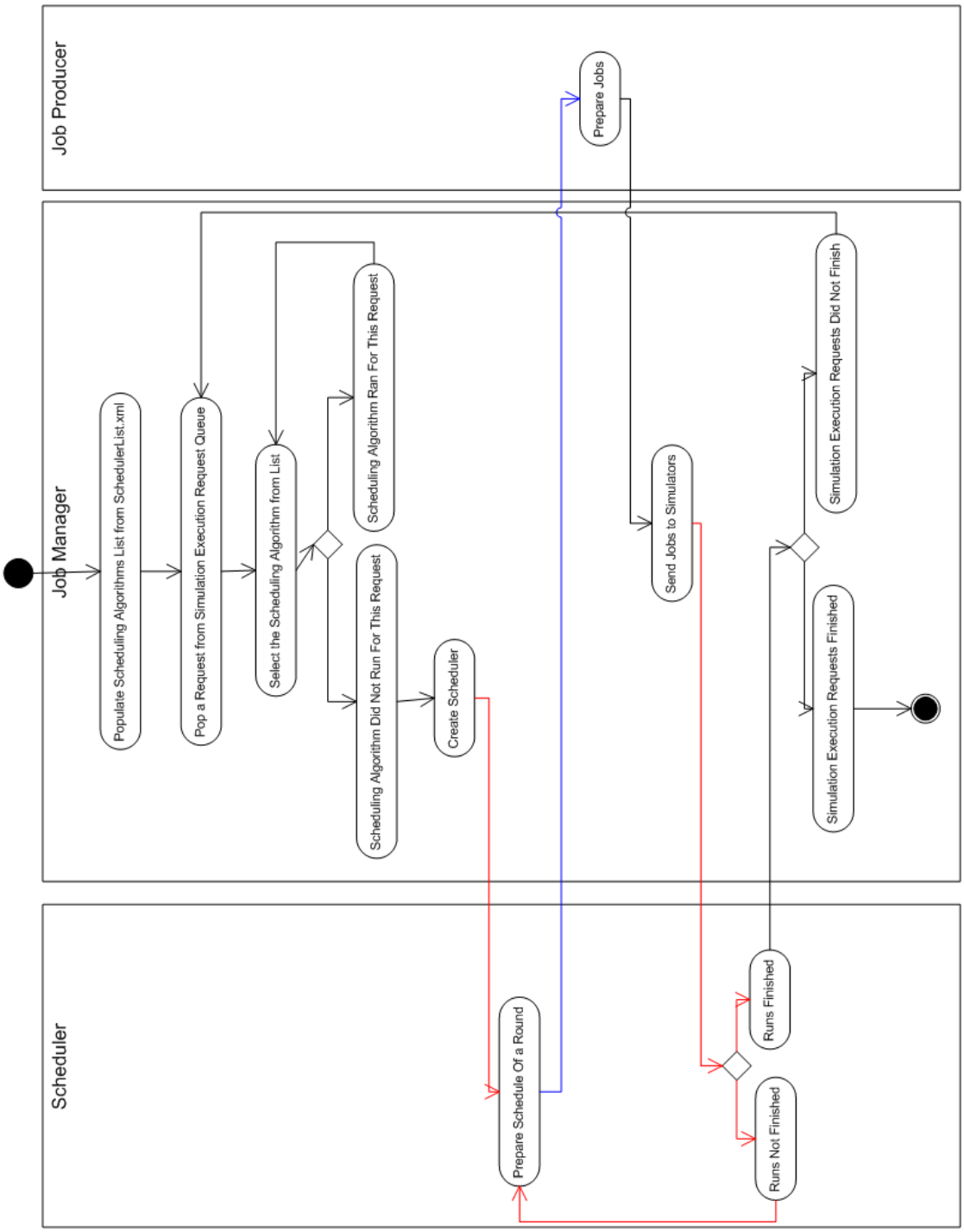


Figure 4.1: Sim-PETEK Scheduling Workflow

flow is replicated for all simulation requests in the ”Simulation Execution Request Queue”.

4.2 Scheduling Algorithm Descriptions

Scheduling algorithms that are intended to be applied for a simulation execution request are described in an XML file which is called ”SchedulerList.xml”. Before the simulation execution these algorithms are read into the memory from this file and corresponding scheduling objects are created at run time.

```
<?xml version="1.0" encoding="utf-8" ?>
<SchedulerList>
  <Scheduler>
    <Name>AMRA</Name>
    <ParameterSet>
      <NumRounds>2</NumRounds>
      <NumRounds>3</NumRounds>
    </ParameterSet>
  </Scheduler>
  <Scheduler>
    <Name>ImprovedSSSEScheduler</Name>
    <ParameterSet>
      <Peak>2</Peak>
      <KValue>15</KValue>
      <MValue>10.5</MValue>
      <Peak>3</Peak>
      <KValue>30</KValue>
      <MValue>20.3</MValue>
    </ParameterSet>
  </Scheduler>
  <Scheduler>
    <Name>CalibratedScheduler</Name>
  </Scheduler>
</SchedulerList>
```

Figure 4.2: Sample SchedulerList.xml

Structure of a sample ”SchedulerList.xml” can be seen in Figure 4.2. In this file, each scheduler is defined between $< Scheduler >< /Scheduler >$ tags. In this tag there are two different tags namely $< Name >$ and $< ParameterSet >$. As the names of the tags imply, name of the scheduler is provided in the $< Name >$ tag and scheduler parameters such as number of iterations and peak value are provided in the $< ParameterSet >$ tag. For AMRS, AMRA, and SAMRA only number of rounds are provided in the parameter set with $< NumRounds >$ tag. For each different number of rounds value, a new scheduling entry is added to the ”Scheduling Algorithms List”. In the SSSE-AMRA and ISSSE-AMRA case, there are 3 different parameters (peak, kValue, mValue) in the parameter set. Values of these parameters are provided in

$< Peak >$, $< KValue >$, and $< MValue >$ tags and for each peak, kValue, and mValue triple a new entry is added to the ”Scheduling Algorithms List”.

Scheduling approach using statistical calibrations and described in [23] is also implemented in this study. In ”SchedulerList.xml”, this algorithm is named as ”CalibratedScheduler”. There is no need for defining a parameter set for this scheduling method.

4.3 Class Hierarchy of Scheduling Algorithms

Scheduling algorithms of Sim-PETEK are implemented in a way that each algorithm resides in a different class. Instances of these classes are created at run time according to the descriptions in ”SchedulerList.xml”.

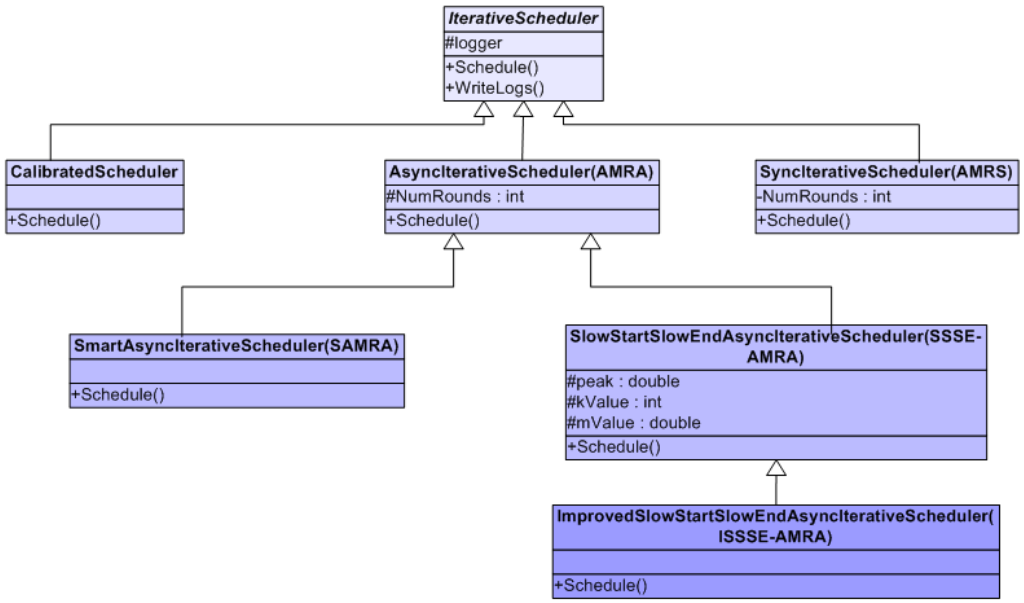


Figure 4.3: Scheduling Class Hierarchy

A hierarchical class structure is formed for the implementation of scheduling algorithms. This structure is presented in Figure 4.3. The hierarchy starts with an abstract class named as ”IterativeScheduler” at the root. The first level under the root contains ”CalibratedScheduler”, ”SyncIterativeScheduler(AMRS)”, and ”AsyncIterativeScheduler(AMRA)”. In the second level, there exist extensions of AMRA namely ”SmartAsyncIterativeScheduler(SAMRA)”

and ”SlowStartSlowEndAsyncIterativeScheduler(SSSE-AMRA)”. At the last level there is only one algorithm, ”ImprovedSlowStartSlowEndAsyncIterativeScheduler(ISSSE-AMRA)” which is the extension of SSSE-AMRA.

As it can be seen from Figure 4.3, each class has a ”Schedule()” function. This function is the coding part of a scheduler class in which corresponding scheduling algorithm is implemented.

IterativeScheduler has a function ”WriteLogs()” which is called by all schedulers for recording schedulers’ performance metrics.

CalibratedScheduler does not contain any attributes. *AMRS* contains ”NumRounds” as a private attribute and *AMRA* contains ”NumRounds” as a protected attribute so that *SAMRA* can reach it. As previously mentioned, ”NumRounds” is the parameter which determines the number of rounds that would be applied by the scheduling algorithm. $SSSE-AMRA$ contains ”peak”, ”kValue”, and ”mValue” attributes as protected so that $ISSSE - AMRA$ can reach them. As also mentioned precviously, ”peak” represents round number at which maximum number of runs to be dispatched in a round is achieved, ”kValue” is the total number of rounds to be applied, and ”mValue” is the normalization factor.

CHAPTER 5

CASE STUDY AND PERFORMANCE ANALYSIS

In this chapter performance analysis of the scheduling algorithms is presented. For this analysis, a stochastic simulation application (Wireless Sensor Network simulation) has run on SimPETEK in a heterogeneous computation environment which is formed in TUBITAK UEKAE ILTAREN. Following sections provide detailed information about the Wireless Sensor Network simulation, our computation environment, and performance tests.

5.1 Wireless Sensor Network Simulation

In this study, Wireless Sensor Network(WSN) simulation developed for the study in [17] is used. This simulation models a system consisting of 5 main components:

1. **Sensors** are the components which sense the movement activities and communicate with other sensors in their range
2. **Main Sensor** is the component which communicates with the sensors in its range and sends activation messages to them
3. **Truck** is the component which follows a predefined path during the simulation
4. **Logger** is a saver component saves the location and data packages created by truck and sensors.
5. **Sensor Adder** adds sensors at runtime.

In the simulation, a wireless sensor network system is constructed by randomly distributing the sensors. When the simulation starts, a truck with a predefined velocity and random path

starts its movement and follows its track. During this movement, sensors seeing the truck in their range detect the truck's location and send an accuracy value between 0 and 1 to their parents. These values are finally collected at the main sensor and observed path of the truck is determined after the analysis.

5.2 Performance Evaluation and Analysis of the Scheduling Algorithms

In this section, firstly a brief description about the testing environment and test cases are presented. Afterwards, the behavior of scheduling algorithms in different test cases are explained via graphics and obtained results are analyzed.

5.2.1 Testing Environment and Test Cases

Testing environment for Sim-PETEK scheduling algorithms has been formed in TUBITAK UEKAE ILTAREN. It is a heterogeneous computing environment consisting of 17 computational resources. One of these resources was determined as the Coordinator and the remaining 16 ones were Simulators. Resources own the following configurations:

- 1 coordinator resource: Quad-core with Windows XP Professional x64 Edition
- 5 simulator resources: 16-core with Windows Server 2003 x64 Edition
- 6 simulator resources: Quad-core with Windows XP Professional x64 Edition
- 3 simulator resources: Dual-core with Windows XP Professional x32 Edition
- 1 simulator resource: Dual-core with Windows Server 2003 x32 Edition

Various tests has been made in this environment with WSN Simulation. For a stochastic parameter sweep approach, some input parameters were determined to be batch parameters and simulations were held for their different values.

Test cases included in this study can be grouped into three:
In the **first group of tests**, AMRS defined in 3.1, AMRA defined in 3.2, and improved versions of AMRA (defined in sections 3.3.1, 3.3.2, and 3.3.3) are used as scheduling approaches

for deciding optimal number of rounds and determining the most efficient algorithm.
The **second group of tests** are applied for making a comparison between the most effective scheduling algorithm among AMRS, AMRA, SAMRA, SSSE-AMRA, and ISSSE-AMRA and the scheduling approach using calibration which is described in [23].

Scheduling approach of [23] has been chosen from literature because it is one of the most recent studies for divisible load scheduling. Similar with Sim-PETEK scheduling algorithms, this calibrated approach is adaptive and multi-round. Furthermore, it is evaluated for a parameter-sweep in a heterogeneous environment which is very similar to the case of this study.

The **third group of** tests are organized for proving the effect of adaptivity. A nonadaptive version of ISSSE-AMRA is developed by distributing the runs according to number of CPU cores of the computational resources in all rounds. Afterwards, execution times achieved by this nonadaptive version is compared with the execution times of adaptive ISSSE-AMRA.

First Group of Tests

The first group of tests which uses 16 of the computational resources can be described as follows:

There are 6 test cases each of which is repeated for different scheduling algorithms or for the same algorithm with different number of rounds. These test cases are:

- 20 Monte Carlo trials for 100, 400, and 800 runs
- 50 Monte Carlo trials for 100, 400, and 800 runs

Different numbers of Monte Carlo trials means shorter or longer simulations, i.e. a simulation with 20 Monte Carlo trials is shorter than the one with 50 trials. Number of sensors, truck's step size, and truck's velocity are determined as batch parameters of WSN simulation and used for task size arrangement (for our case number of runs). Number of sensors is provided as 150 as minimum and 169 as maximum where the increasing step is 1, meaning 20 different values. Truck's step size is provided as 0.020 as minimum and 0.024 as maximum where the increasing step is 0.001, meaning 5 different values. For 100 runs, truck velocity is kept constant since different values of number of sensors and truck's step size result in 100 runs

(i.e. $20 \times 5 = 100$). For 400 runs, truck velocity is provided as 0.40 as minimum and 0.43 as maximum where stepping is 0.01, i.e. 4 different values ($20 \times 5 \times 4 = 400$). Similarly, for 800 runs, truck velocity is provided as 0.40 as minimum and 0.47 as maximum where stepping is 0.01, i.e. 8 different values ($20 \times 5 \times 8 = 800$).

As mentioned in 3.1 optimal number of rounds for the scheduling algorithms are determined by experimentation. AMRS scheduling algorithm tests are made for 2, 3, 5, 8, and 10 rounds. AMRA and SAMRA tests are made for 2, 3, 5, 8, 10, 15, 20, 30, 40, 50, 60, 80 and 100 rounds. SSSE-AMRA and ISSSE-AMRA test are repeated with peak = 1, kValue = 3, mValue = 2; peak = 1, kValue = 5, mValue = 3.3; peak = 2, kValue = 15, mValue = 10.5; peak = 3, kValue = 30, mValue = 20.3; and peak = 5, kValue = 50, mValue = 33 where 50%, 30%, 15%, 10%, and 5% of the total runs respectively are distributed until reaching the peak.

Second Group of Tests

The second group of tests, which use all of the resources in the resource set, determined four different batch parameters for WSN simulation which are number of sensors, truck's step size, truck's velocity X component, and truck's velocity Y component. There are 15 different test cases in this group:

- 10 Monte Carlo trials for 500, 1000, and 1500 runs
- 30 Monte Carlo trials for 500, 1000, and 1500 runs
- 50 Monte Carlo trials for 500, 1000, and 1500 runs
- 80 Monte Carlo trials for 500, 1000, and 1500 runs
- 100 Monte Carlo trials for 500, 1000, and 1500 runs

As many tests as possible have been applied in order to make a better analysis on the behavior of the algorithms in different situations. For the arrangement of number of runs in this second group of tests, minimum and maximum values and stepping size for number of sensors and truck's step size are same as the first group. Truck velocity's X component is provided as 0.40 as minimum and 0.44 as maximum where the increasing step is 0.01, meaning 5 different values. For 500 runs, truck velocity's Y component is kept constant (i.e. $20 \times 5 \times 5 = 500$). For 1000 runs, it is provided as 0.40 as minimum and 0.41 as maximum with stepping size

0.01 meaning 2 different values so that 20×5×5×2 = 1000. Similarly, for 1500 runs velocity's Y component is provided as 0.40 as minimum and 0.42as maximum with the same stepping size meaning 3 values (i.e. $20 \times 5 \times 5 \times 3 = 1500$).

Third Group of Tests

Third group of tests uses 6 of the computational resources. 3 of such resources consists of 16 cores, 2 of them have 4 cores, and 1 of them is a dual-core. For a better analysis of the effect of adaptivity, quad-core and dual-core resources are loaded with another CPU intensive application for 30 seconds in each 40 seconds period. There are 12 different test cases in this group:

- 20 Monte Carlo trials for 100, 400, 1000, and 2000 runs
- 50 Monte Carlo trials for 100, 400, 1000, and 2000 runs
- 100 Monte Carlo trials for 100, 400, 1000, and 2000 runs

Different number of runs in the above test cases are arranged by playing with the values of four different batch parameters of WSN simulation which are number of sensors, truck's step size, truck's velocity X component, and truck's velocity Y component.

5.2.2 Test Results

This section consists of three subsections. The first subsection presents the results of first group of tests, the second one presents the results of second group, and the third one presents the results of third group. Evaluations and comparisons of different schedulers are made according to the total execution costs of the schedulers for the same test case.

Test results are shown via figures which present them visually. In the graphics number of Monte Carlo trials and number of runs are represented as MC***X***NumberOfRuns***Y***. For example, MC20NumberOfRuns100 stands for 20 Monte Carlo trials for 100 runs.

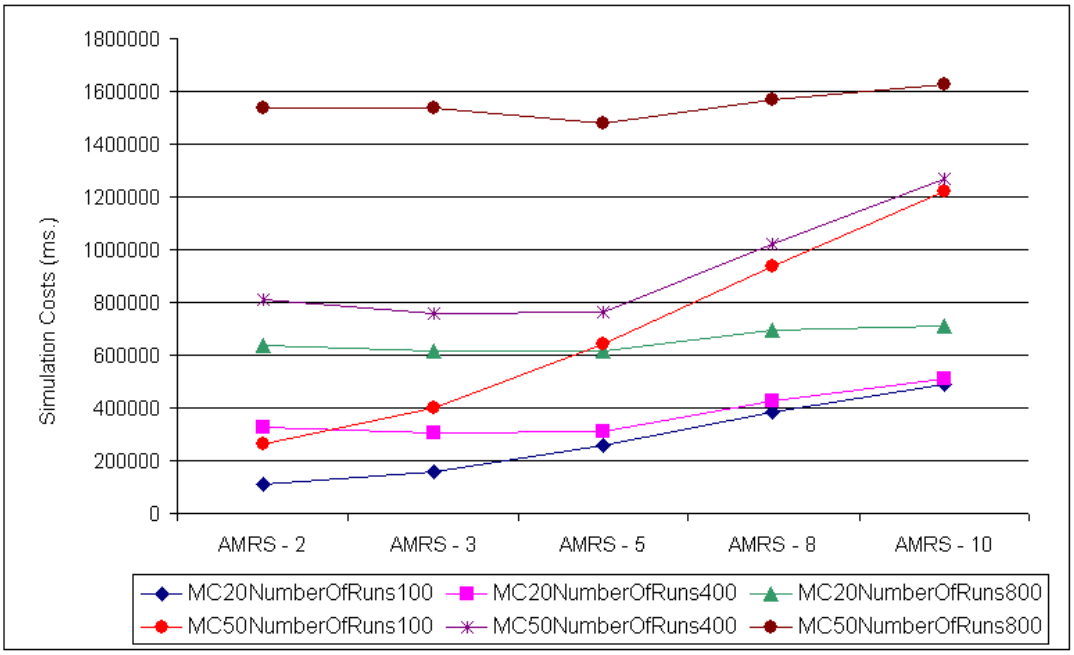


Figure 5.1: AMRS Scheduling Algorithm Execution Times

5.2.2.1 Results of First Group of Tests

In Figure 5.1, execution times obtained by AMRS Scheduling Algorithm are presented. In this figure AMRS-2 represents that the algorithm has applied 2 rounds. In the same manner AMRS-3 has applied 3 rounds, AMRS-5 has applied 5 rounds, AMRS-8 has applied 8 rounds, and AMRS-10 has applied 10 rounds.

This graphic shows that for the cases where number of runs is 100, optimal number of rounds is 2, for the ones where number of runs is 400 or 800, optimal number of rounds is 5. This result means that when task size is increased, number of rounds that the algorithm applies should also be increased. However, if number of rounds is further increased then performance of the algorithm gets worse. This is because, when number of rounds is increased, runs to be distributed in a round decreases and some of the nodes can not get any jobs. For a better understanding of this situation, let's take the numeric example traced in Section 3.1 but this time the assumption is that 10 rounds are applied. With this assumption, number of runs to be dispatched in a round would be 6 (60/10 = 6). Then, job distributions would be in a manner as shown in Figure 5.2.

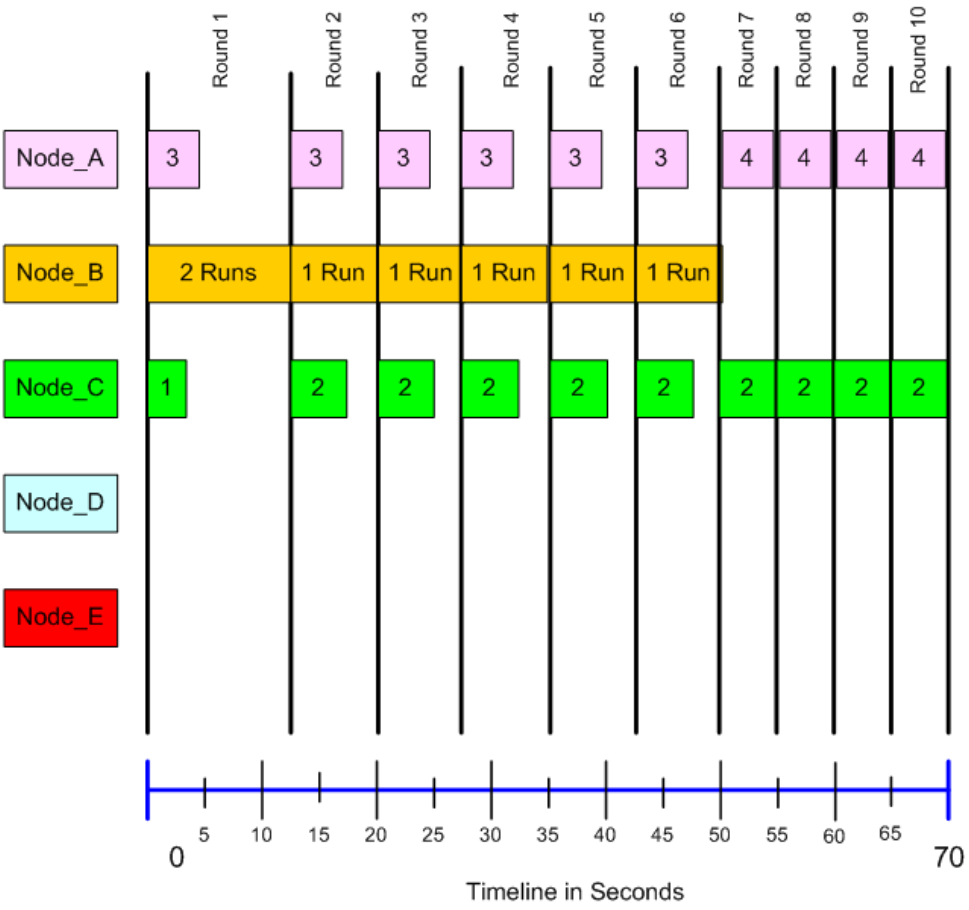


Figure 5.2: AMRS Job Distributions When Number of Rounds is 10

As the figure shows, 2 of the nodes are not assigned with any jobs and 70 seconds are spent for finishing the whole simulation whereas it can be finished in 65 seconds as in Figure 3.5.

Figure 5.3 shows simulation costs when AMRA Scheduling Algorithm is utilized. In this figure, AMRA-2 represents that the algorithm has applied 2 rounds, AMRS-3 represents that the algorithm has applied 3 rounds, and so on. As the first graphic on the figure indicates, for the cases where number of runs is 100 optimal number of rounds is 3. When number of runs increased to 400, optimal number of rounds increases to 8 (shown by second graphic on the figure) and when it is increased to 800, optimal number of rounds increases to 15 for 20 MC trials case and 20 for 50 MC trials case.

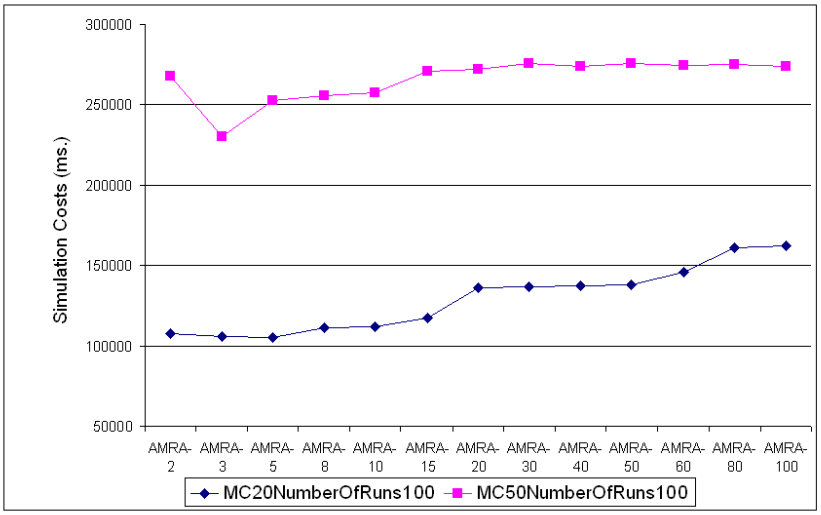
These results indicate that when task size is increased, number of rounds that is applied by AMRA should also be increased to some extent similar to the AMRS case. Another conclusion that can be reached by the results is that when simulation is extended by increasing the number of MC trials, optimal number of rounds may increase as in the case where 50 MC trials are made for 800 runs.

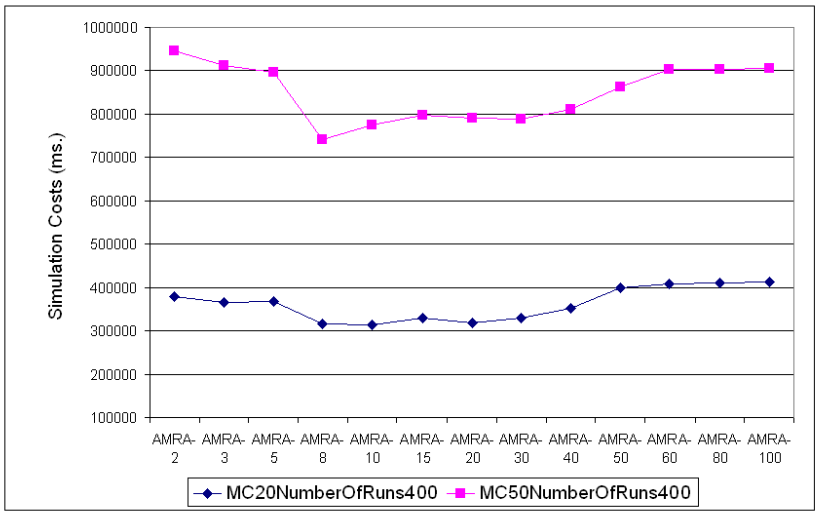
SAMRA scheduling algorithm shows a similar behavior with AMRA as shown in Figure 5.4. The first graphic on the figure indicates that optimal number of rounds is 2 when simulation contains 100 runs. This number increases to 10 and 15 for the cases where simulation consists of 400 and 800 runs respectively.

In Figure 5.5, behavior of SSSE-AMRA scheduling algorithm with different peak, kValue, and mValue values is presented. For small number of runs, the algorithm performs better when peak and kValue are kept small. When number of runs increased, peak and kValue should also be increased for achieving a better performance. This is thought to be related with the situation that when peak and kValue is kept high for small number of runs, then some nodes wait idle at the beginning or ending rounds where less number of runs are distributed.

As previously mentioned in Section 3.3.3, ISSSE-AMRA brings a small improvement over SSSE-AMRA, so its behavior is very similar with different values of peak, kValue, and mValue as Figure 5.6 depicts.

After all test cases are completed, a comparison is made among the scheduling algorithms. For this comparison, execution times of the algorithms when they applied 15 rounds is used. Figure 5.7 presents the comparison graphic.





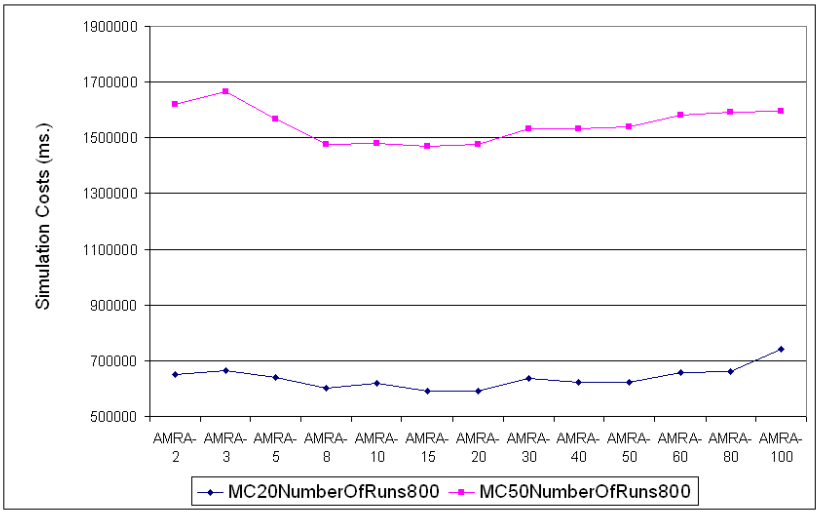


Figure 5.3: AMRA Scheduling Algorithm Execution Times

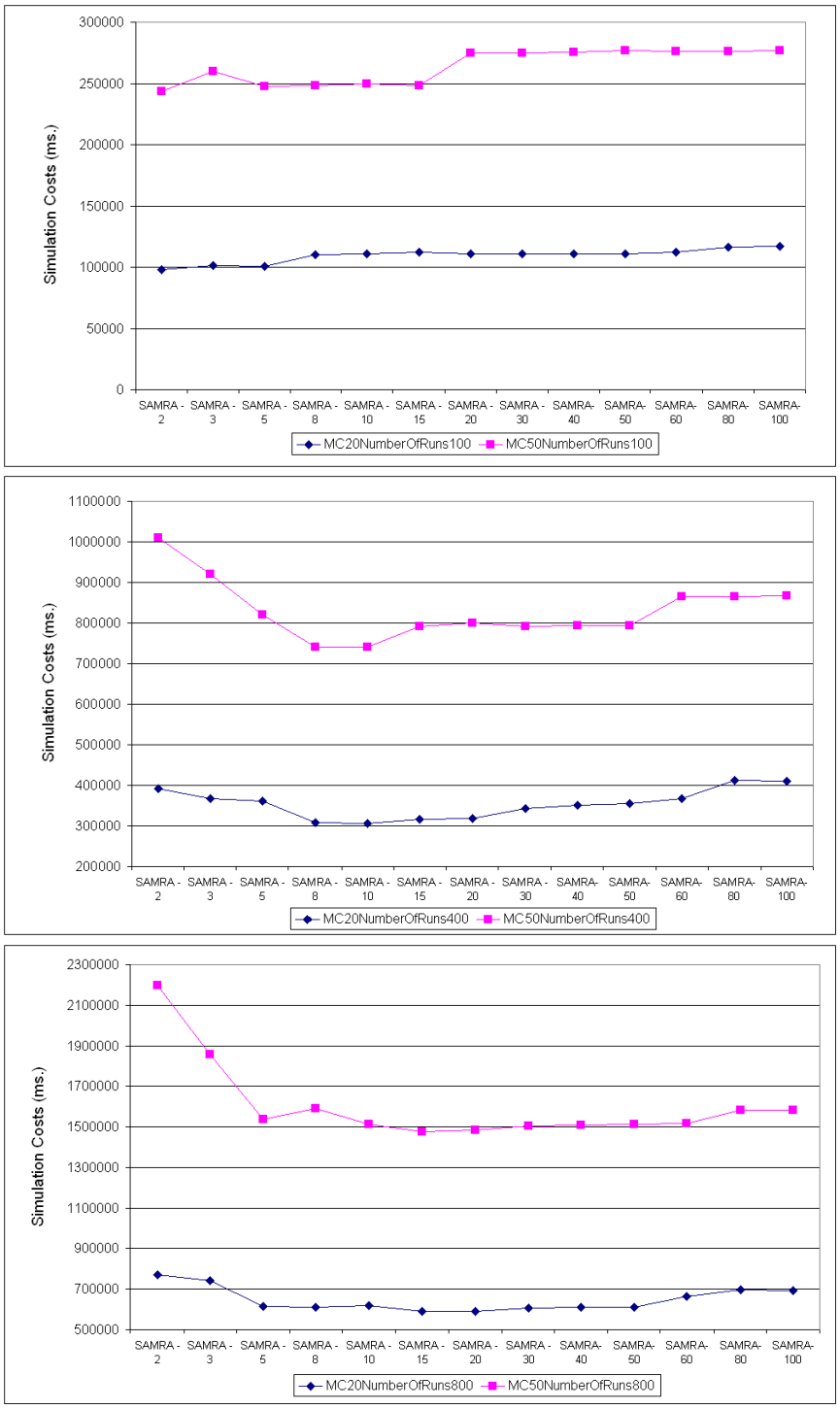


Figure 5.4: SAMRA Scheduling Algorithm Execution Times

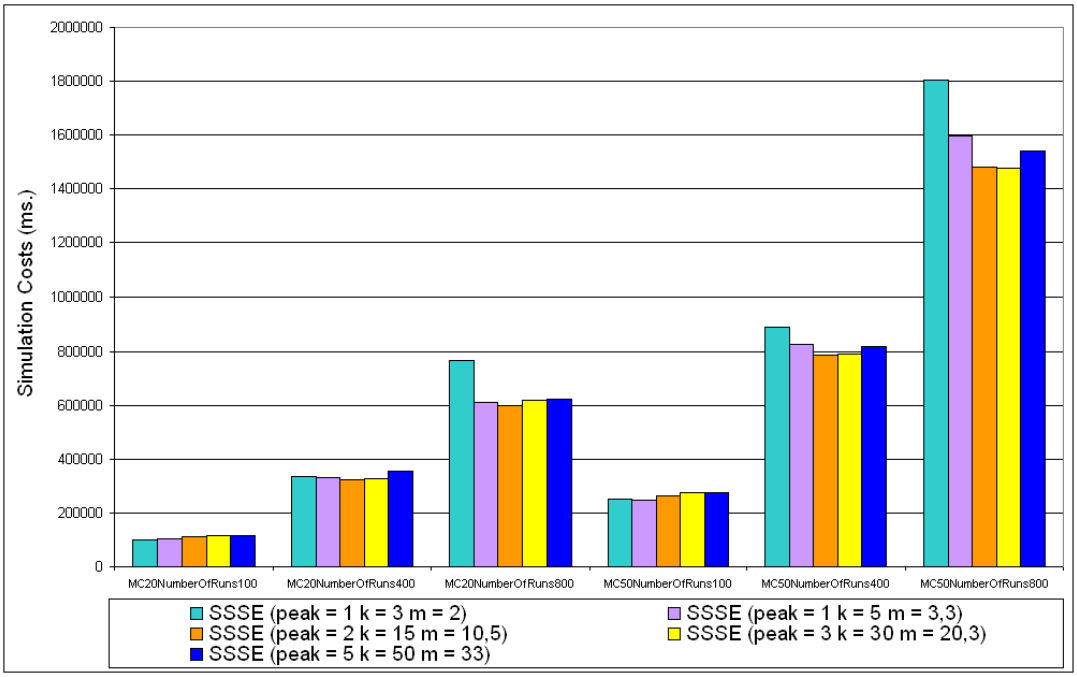


Figure 5.5: SSSE-AMRA Scheduling Algorithm Execution Times

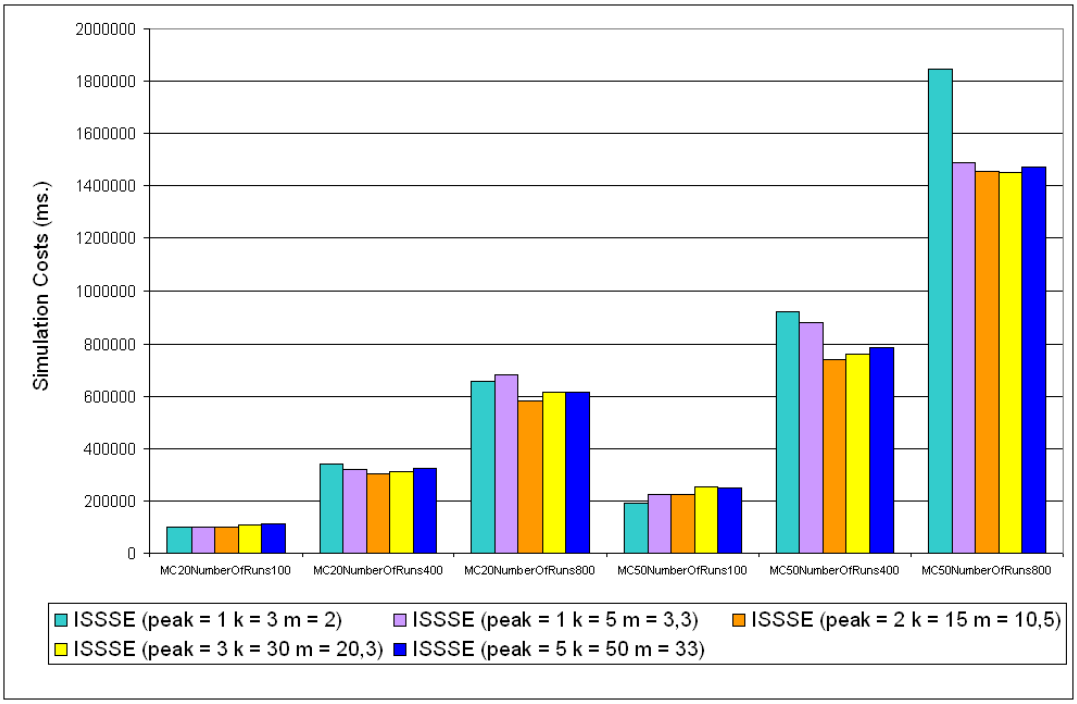


Figure 5.6: ISSSE-AMRA Scheduling Algorithm Execution Times

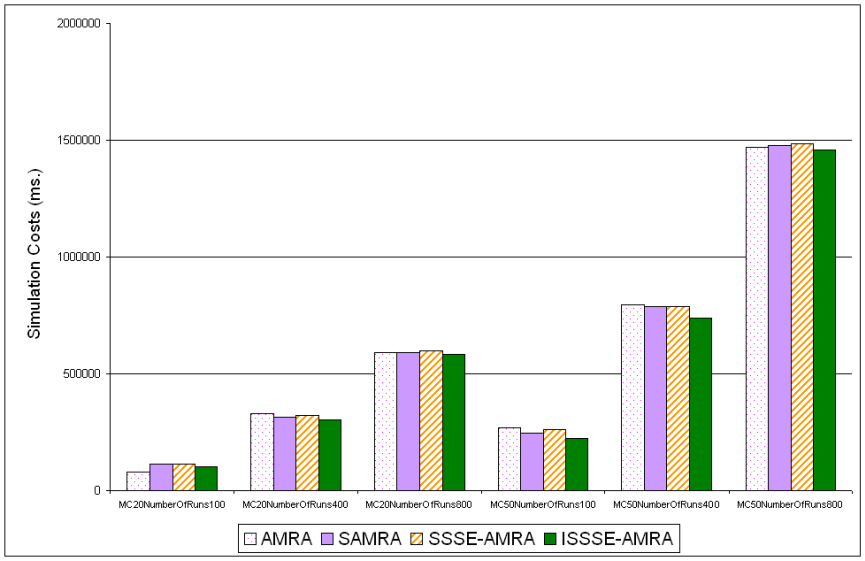


Figure 5.7: Comparison of Sim-PETEK Scheduling Algorithms

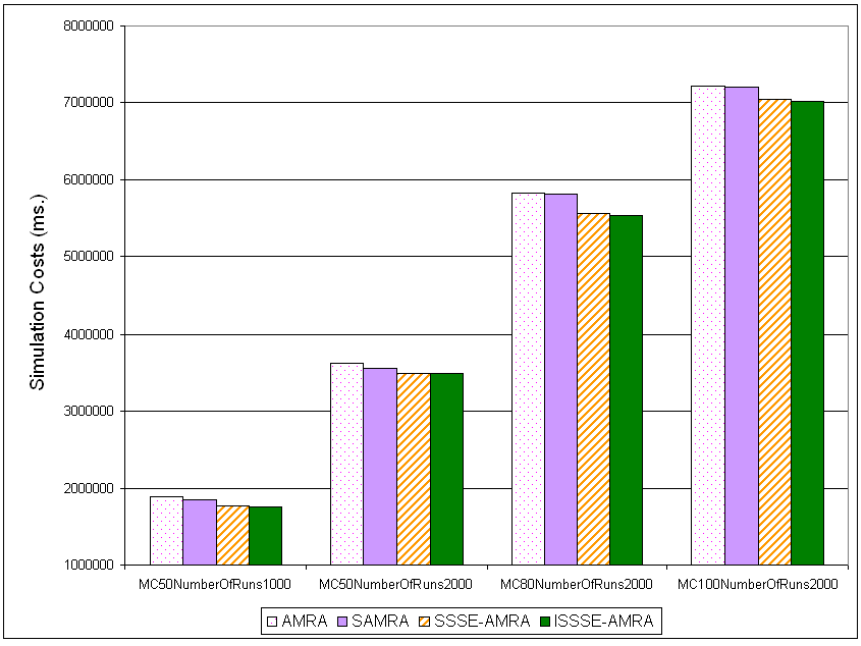


Figure 5.8: Comparison of Sim-PETEK Scheduling Algorithms with Increased Number of Runs and MC Trials

From this comparison graphic, it can determined that ISSSE-AMRA shows the best performance in most of the cases. However, a conclusion can not be made about the behavior of AMRA, SAMRA, and SSSE-AMRA. It is easily detected that they do not behave in convenience with the theoretical results in Chapter 3 which has shown that SSSE-AMRA would perform better than AMRA and SAMRA, and SAMRA would perform better than AMRA. We think that these results may have occured becuase of the fact that communication costs are not used in our scheduling model and our tests are running for simulations with small number of runs for which communication costs are not ignorable. At this point, further tests are applied with increased number of runs and MC trials where communication costs can be ignored. These tests consists of 50 MC trials for 1000 runs, 50 MC trials for 2000 runs, 80 MC trials for 2000 runs, and 100 MC trials for 2000 runs. Figure 5.8 presents the results which are convenient with the theoretical findings of Chapter 3. ISSSE-AMRA shows the best performance, AMRA shows the worst performance, and SSSE-AMRA performs worse than ISSSE-AMRA and better than SAMRA as expected.

Moreover, the scheduling algorithms are examined in order to analyze their DLT convenience. Figure 5.9 presents simulation execution times with AMRA scheduling algorithm on a node basis. From this figure, it can be detected that all nodes do not finish their task parts exactly at the same time but there are not any huge differences.

Figure 5.10 shows simulation execution times with SAMRA. Similar with AMRA, all nodes do not finish their tasks exactly at the same time. Differences in finish times are not high and they are less than the differences of AMRA meaning that SAMRA is more convenient with DLT optimality principle than AMRA.

Making same analysis with SSSE-AMRA and ISSSE-AMRA, the graphics shown in Figures 5.11 and 5.12 are obtained. As it can be seen from the figures, task finishing times of the nodes are very near to each other meaning that DLT optimality principle is assured better than AMRA and SAMRA. Figure 5.12 further shows duplicate and redundant times for the nodes. Duplicate times denote the time spans where a node, Node_i, which is faster than another node, Node_j, is assigned with Node_j's job in order to finish the simulation earlier. In such a case, the time spent by Node_j is denoted as redundant time on the graphic.

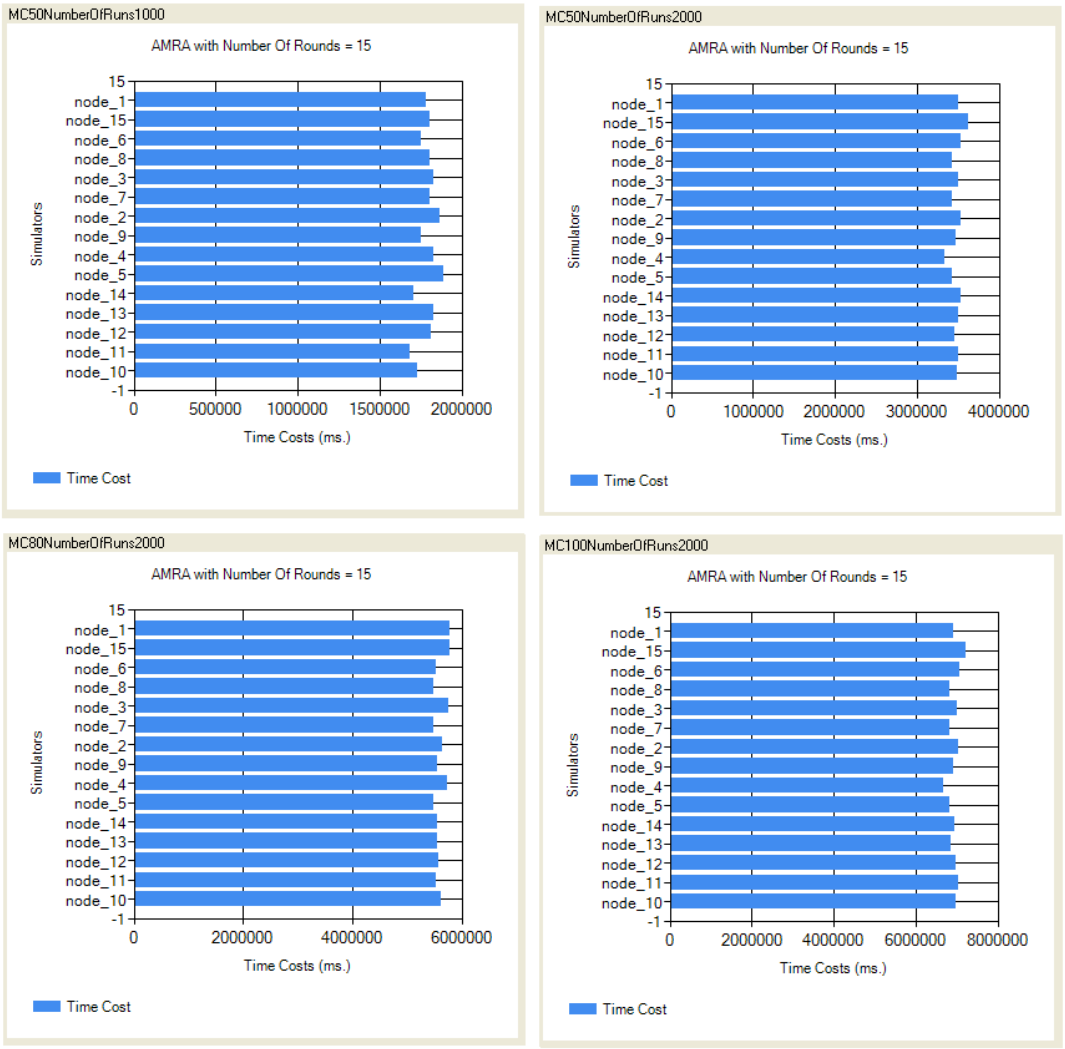


Figure 5.9: Node Execution Times with AMRA

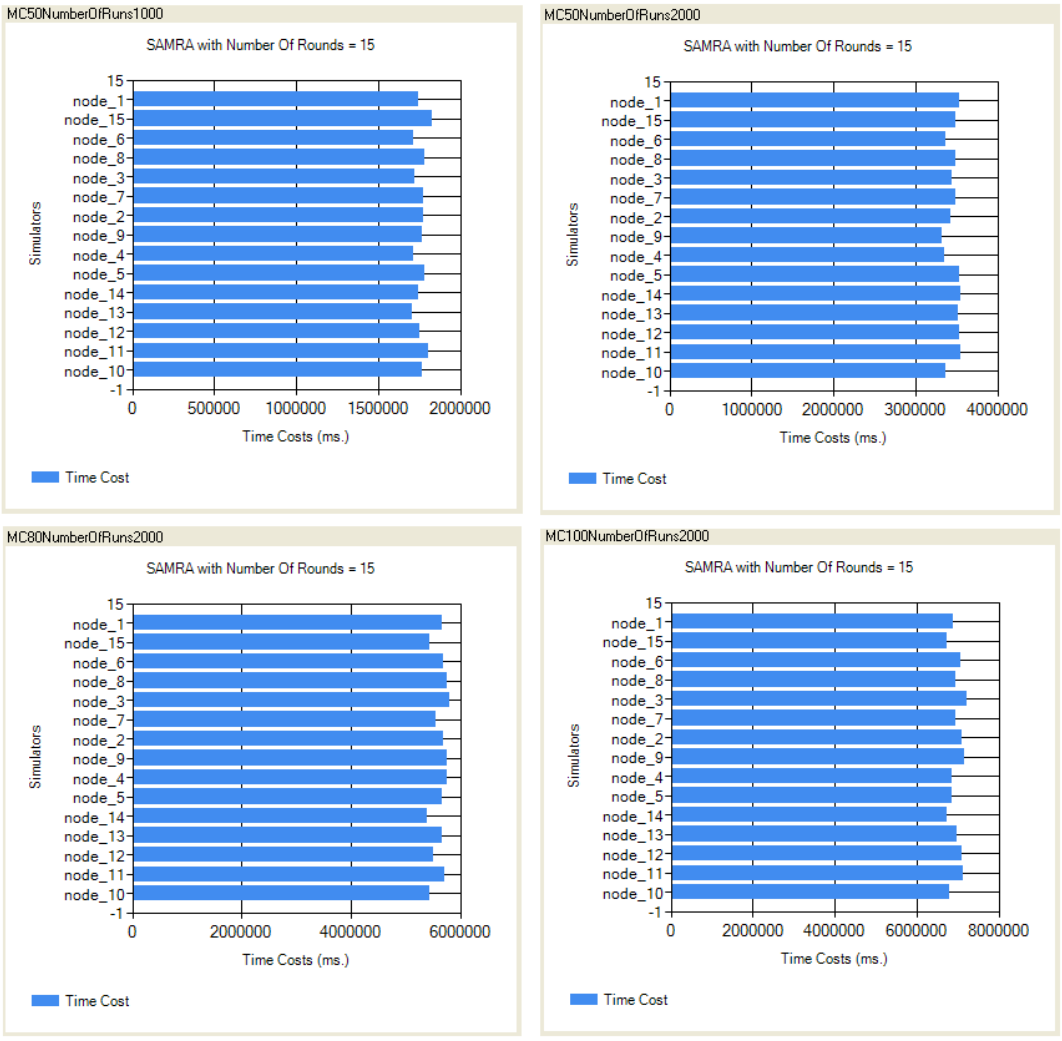


Figure 5.10: Node Execution Times with SAMRA

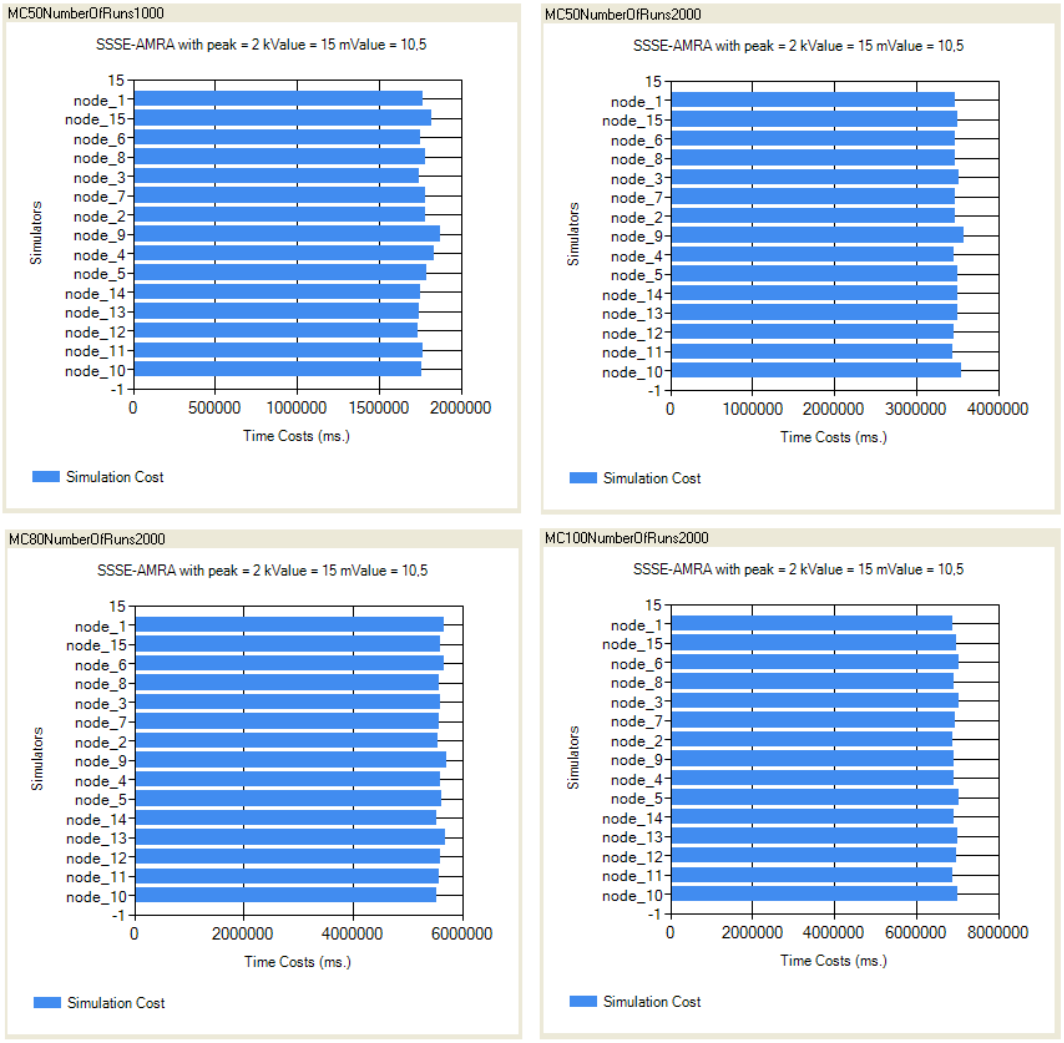


Figure 5.11: Node Execution Times with SSSE-AMRA

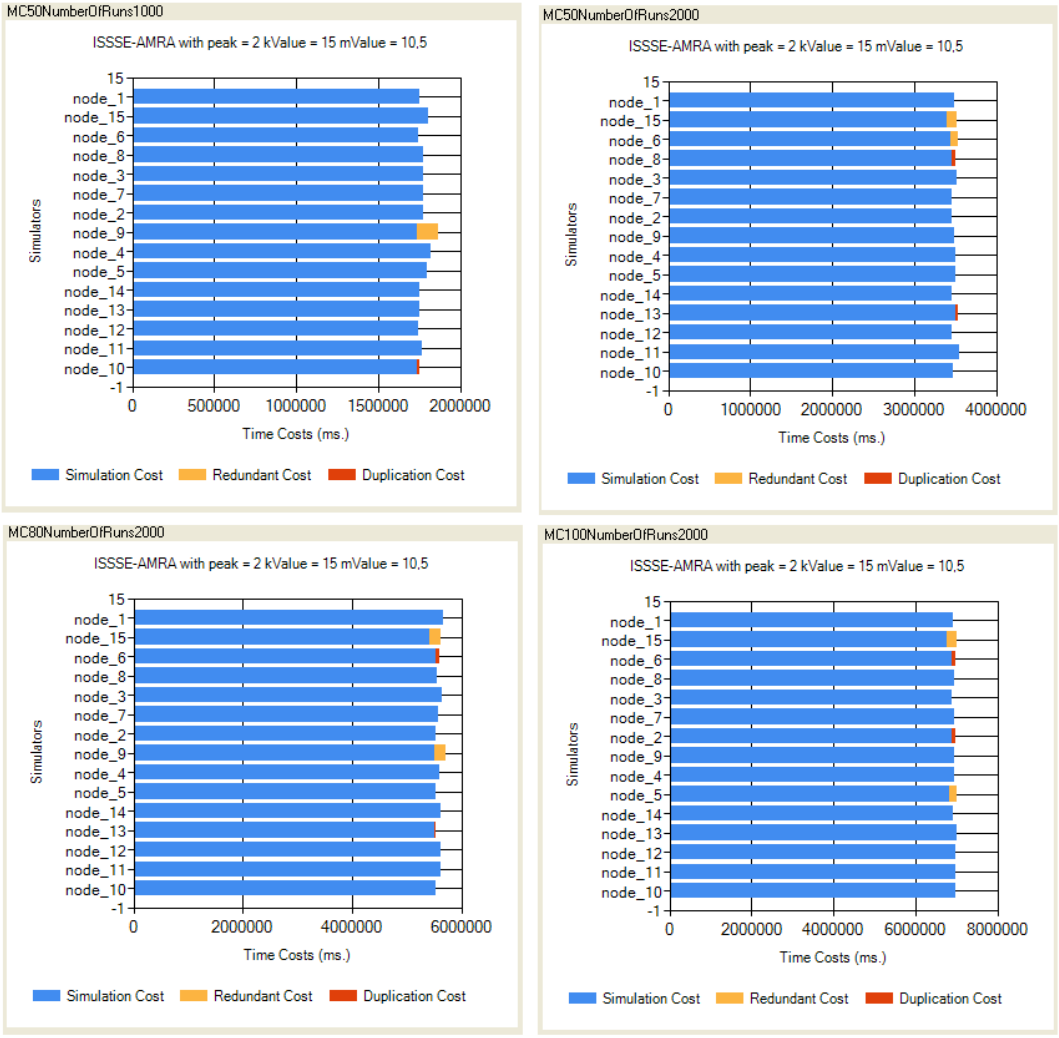


Figure 5.12: Node Execution Times with ISSSE-AMRA

5.2.2.2 Results of Second Group of Tests

As mentioned previously, tests of this group are utilized for comparing the performance of the most effective scheduling algorithm among Sim-PETEK scheduling algorithms and the scheduling approach of [23] using statistical calibration techniques which is called as ”Calibrated Scheduler” in this study.

Examining the results of first group of tests in Section 5.2.2.1, ISSSE-AMRA scheduling algorithm is found to show the best performance with Sim-PETEK architecture. For this reason, comparisons are made between ISSSE-AMRA and Calibrated Scheduler. Figure 5.13 presents the comparison graphic. As the graphic depicts ISSSE-AMRA runs for 3 different values of peak, kValue, and mValue. Results of second group of tests are compatible with the finding with first group of tests which says that ”better performance is achieved with ISSSE-AMRA by increasing peak and kValue to some extent when number of runs is increased”.

In the comparison graphic of Figure 5.13, it is clear that ISSSE-AMRA performs better than Calibrated Scheduler in all of the test cases.

When algorithms are inspected in terms of DLT optimality principle, graphics shown in Figure 5.14 are obtained. These graphics only present the bahaviour when 100 MC trials are applied for 1000 runs. Behaviours of the algorithms are very similar in other tests.

One important situation about the graphics of Figure 5.14 which attracts attention is that, there are idle times for some of the nodes when Calibrated Scheduler is used as the scheduling approach. These idle times are thought to be caused by the probing phase of the algorithm.

ISSSE-AMRA’s better performance is thought to be related with follows:

- There are not any idle times in ISSSE-AMRA.
- ISSSE-AMRA can collect more accurate information about the nodes by sending less number of runs in several rounds at the beginning and prevents huge number of runs to be sent to a poor node. Calibrated Scheduler applies the probing in one only round which may not be sufficient in a system which contains frequent changes.
- Task redistribution approach of ISSSE-AMRA prevents wait conditions for bottlenecked nodes. Calibrated Scheduler does not provide any mechanism for such kinds of cases.

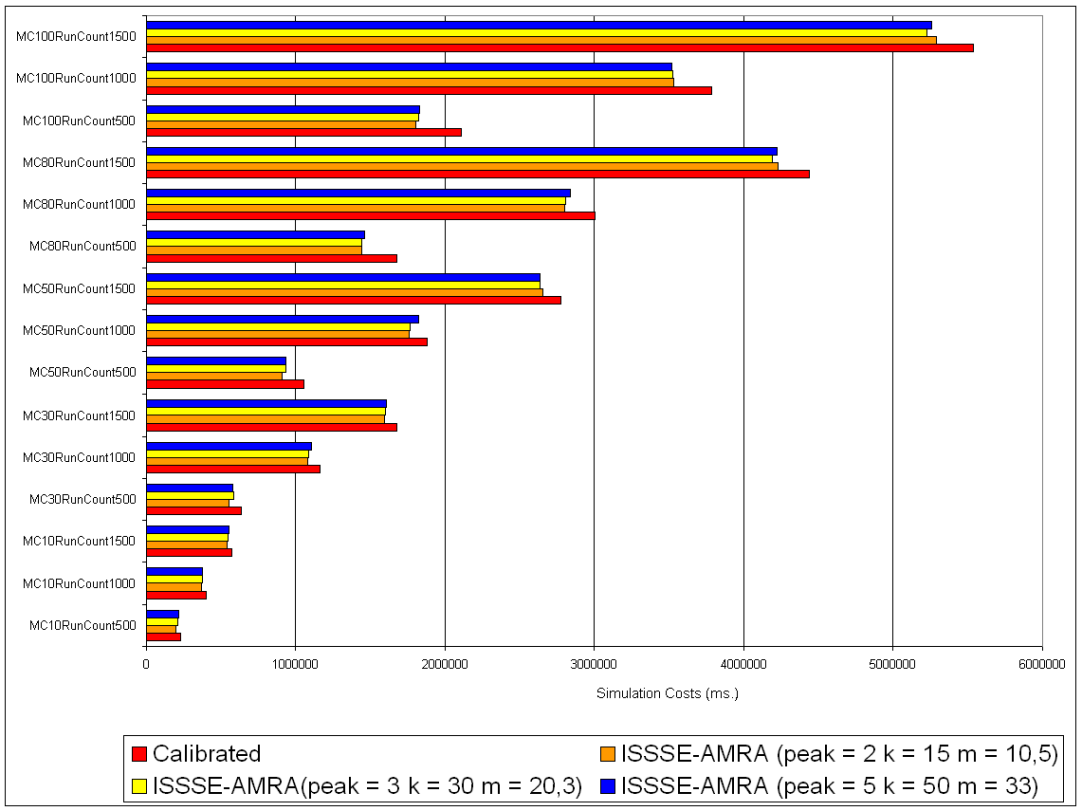


Figure 5.13: Comparison of ISSSE-AMRA and Calibrated Scheduler

Figure 5.14: Node Execution Times with ISSSE-AMRA and Calibrated Scheduler

5.2.2.3 Results of Third Group of Tests

As it is mentioned in the previous sections, tests in this group are organized for observing the effect of adaptivity in scheduling.

The comparison graphic of Figure 5.15 depicts that adaptive version of ISSSE-AMRA performs better than the nonadaptive version in all of the test cases which is the expected behavior.

5.3 Discussion

There are several projects in literature like AppLeS in [14] and Nimrod/G in [12] targeting the PSA deployment in distributed systems.

Scheduling approach of AppLeS uses static and dynamic information about resources as well as application-level information like number of tasks and size of data files for making scheduling decisions. Since the framework targets long-running applications, it refines its scheduling decisions periodically during the application execution. The scheduling algorithm especially focuses on the scenarios where large input data files are shared among several task fractions. It tries to maximize the re-use of such files by replicating the files and dispatching the tasks close to their relevant files. This is an NP-complete scheduling problem and heuristics named as Min-min, Max-min, Sufferage, and XSufferage are used by AppLeS framework for the solution [16]:

- **Min-min** is the heuristic which gives priority to the task that can be completed earliest.
- **Max-min** is the heuristic which gives priority to the task that can be completed latest.
- **Sufferage** has the main idea that a resource should be assigned to the task that would suffer the most if not assigned to that host.
- **XSufferage** applies Sufferage heuristic in cluster level.

Furthermore, a greedy algorithm using assigning work to hosts as soon as they become available is implemented for scheduling purposes and different scheduling approaches are compared. XSufferage heuristic has seen to be performing best when large input files are shared

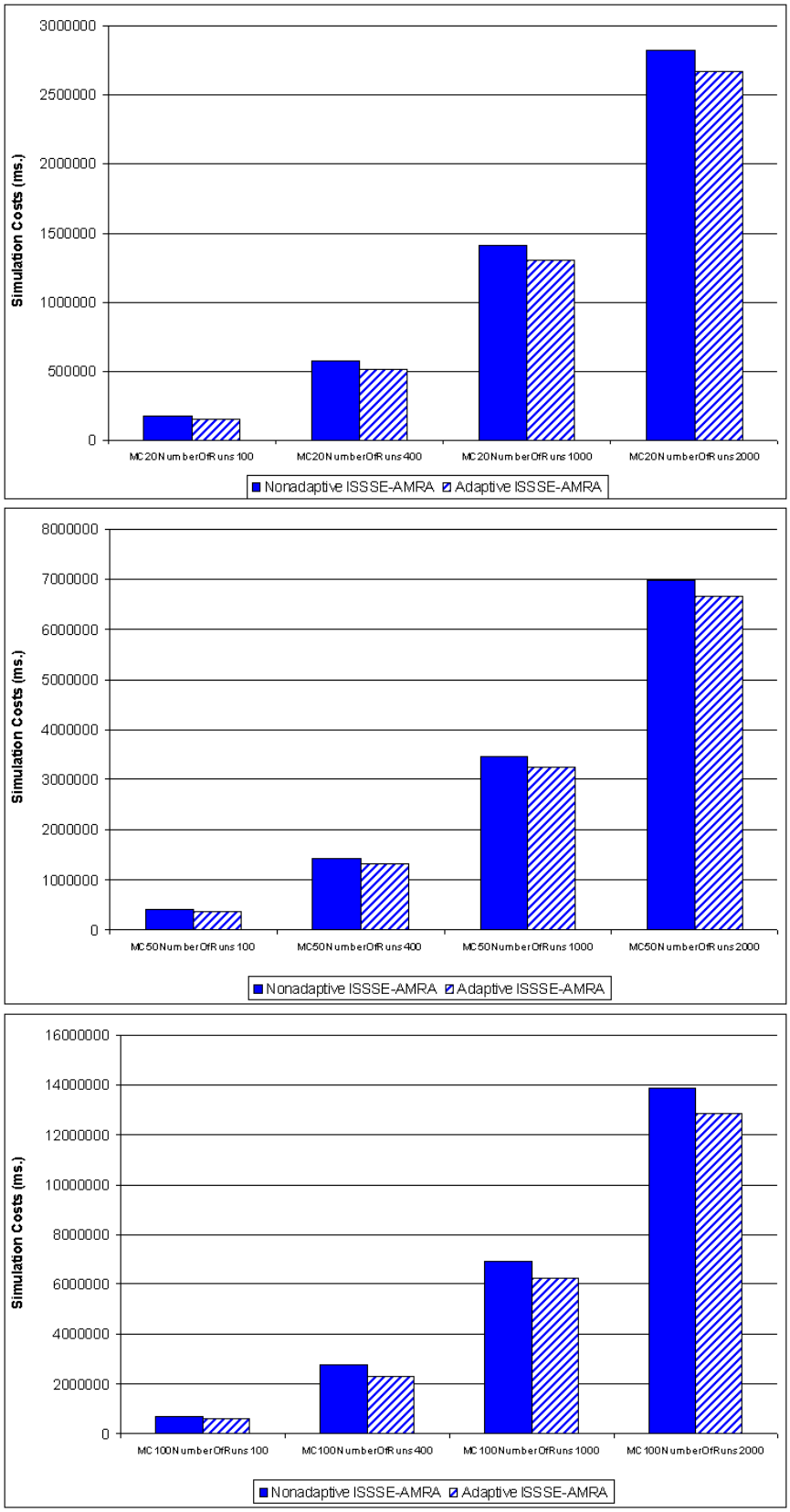


Figure 5.15: Comparison of Adaptive and Nonadaptive Versions of ISSSE-AMRA

by several tasks and performance predictions have errors within reasonable amounts. However, in a system where there is a significant variance in resource availability, greedy approach has seen to be more appropriate [13].

Scheduling approach of Nimrod/G tries to integrate the computational economy as a part of the scheduling system, i.e. what the scheduling system tries to do is to find sufficient resources for meeting user's deadline and cost. Various kinds of parameters such as resource configuration, resource state, resource capability, access speed, and task priority are used for arriving optimal schedules. Resources offering the best price and meeting the deadline can be selected and used in task execution [12].

In Sim-PETEK architecture, there do not exist any cases where huge file access is needed because simulators in the system only read the simulation definitions from short XML files and produce the different values for input parameters automatically. For this reason, the scheduling approach of AppLeS using heuristics is not appropriate for Sim-PETEK. However, AppLeS's approach to make periodic scheduling decisions according to the variances of the system is similar to the adaptive approach of Sim-PETEK scheduling algorithms where periods are defined by rounds.

Sim-PETEK, not like Nimrod/G, is not designed in a way to meet the deadline and cost requirements of users. However, if it is desired, the architecture can be extended and computational economy can be integrated into the existing scheduling algorithms.

CHAPTER 6

CONCLUSION

The study in thesis has focused on the development and analysis of scheduling algorithms for Sim-PETEK. For this purpose, 5 different scheduling algorithms are designed and implemented. Our common design approach of such algorithms is that they are developed as adaptive and multi-round. Adaptive approach is followed because Sim-PETEK is designed to run in heterogeneous computational environments and multi-round approach is followed since the simulation applications using Sim-PETEK are parameter sweep applications which are arbitrarily divisible.

AMRS is the first scheduling algorithm that has been developed for Sim-PETEK. This algorithm starts with assigning an initial expected execution power value to each computational node according to its number of CPU cores. After initialization, job dispatching rounds start and continues until all runs of the simulation are finished. Between the rounds, expected execution power values of the nodes are updated according to nodes' performance in the previous rounds and runs are distributed in accordance with this updated value.

AMRA improves AMRS by preventing wait conditions in the rounds. This is held by immediately making a new job assignment to the nodes which finish their runs.

AMRA is further improved and named as SAMRA. What SAMRA does for this improvement is that, it adds a probing phase before the first round and provides more compliancy with Sim-PETEK design. Probing is handled by assigning small number of runs to the nodes for estimating their computational powers. Sim-PETEK compliancy is provided by arranging number of runs in a job according to the resource properties.

SSSE-AMRA tries to solve idle wait conditions at the last round by a slow start-slow end scheduling approach. The algorithm starts with dispatching small number of runs in a round,

increases that number to some extent, and then decreases. Assigning small number of runs at the beginning provides the algorithm to collect performance metrics about the nodes as a precaution for preventing bottleneck of slow nodes. Similarly, by assigning small number of runs at the last rounds, SSSE-AMRA prevents wait conditions for slow nodes.

SSSE-AMRA is further improved by ISSSE-AMRA by redistributing the jobs of slow nodes to the faster ones in the last iteration. This small improvement aims to prevent redundant waits for the bottlenecked nodes.

Various tests are applied for analyzing the developed scheduling algorithms. This analysis has shown that number of rounds should be increased or decreased in accordance with the number of runs that the simulation contains. Another observation after the analysis is that ISSSE-AMRA generally shows better performance than the others.

Moreover, several tests are organized for comparing the performance of ISSSE-AMRA and the scheduling approach of [23]. Results of these tests have revealed that ISSSE-AMRA performs better than Calibrated Scheduler from 4% up to 15%. For this reason, ISSSE-AMRA is determined to be the Sim-PETEK scheduling algorithm.

Comparing the scheduling approach of Sim-PETEK with other PSA running systems such as AppLeS, it is seen that periodic scheduling decision updates of AppLeS corresponds to rounds of Sim-PETEK Scheduler and provides adaptivity. Another PSA running system, Nimrod/G, integrates computational economony in its scheduling system which has not been considered by Sim-PETEK, however; it can be integrated into the scheduling algorithms of Sim-PETEK if it is desired.

As a future extension, some other tests which will be running in a wider heterogeneous computational environment such as grid can be organized. These tests will be giving us the opportunity to make further analysis on the behavior of scheduling algorithms.

In this study, optimal values for some parameters such as number of rounds, peak and kValue are determined by experimentation. For getting rid of these experiments and saving time, another extension can work on formulating such kind of values in terms of task size (i.e. number of runs in our case) and other kinds of system properties like number of computational resources.

REFERENCES

[1] OGSA, OGSI, and GT3.
http://gdp.globus.org/gt3-tutorial/multiplehtml/ch01s01.html, last visited on 22 August 2010.

[2] Open Grid Services Architecture.
http://www.globus.org/ogsa, last visited on 8 August 2010.

[3] Parameter Sweep.
http://sourceforge.net/apps/mediawiki/qucs/index.php?title=Parameter_Sweep, last visited on 9 August 2010.

[4] scientific application - Computer Dictionary Definition.
http://www.yourdictionary.com/computer/scientific-application, last visited on 22 August 2010.

[5] The Web Services Resource Framework.
http://www.globus.org/wsrf, last visited on 8 August 2010.

[6] Windows Communication Foundation.
http://en.wikipedia.org/wiki/Windows_Communication_Foundation, last modified on 22 July 2010, last visited on 27 July 2010.

[7] I. Banicescu and V. Velusamy. Load Balancing Highly Irregular Computations with the Adaptive Factoring. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS.02)*, 2002.

[8] O. Beaumont, A. Legrand, and Y. Robert. Scheduling divisible workloads on heterogeneous platforms. *Parallel Computing*, 29:1121–1152, 2003.

[9] V. Bharadwaj, D. Ghose, V. Mani, and T. G. Robertazzi. *Scheduling Divisible Loads in Parallel and Distributed Systems*. IEEE Computer Society Press, 1996.

[10] 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, 2003.

[11] D. Bozağaç, G. Karaduman, A. Kara, and M.N. Alpdemir. Sim-PETEK : A Parallel Simulation Execution Framework for Grid Environments. In *Summer Computer Simulation Conference (SCSC'09)*, pages 275–282, 2009.

[12] R. Buyya, D. Abramson, and J. Giddy. Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid. In *High Performance Computing in the Asia-Pacific Region*, pages 283–289, 2000.

[13] H. Casanova and F. Berman. *Parameter Sweeps on the Grid with APST, in Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons, Ltd, Chichester, UK, 2003.

[14] H. Casanova, F. Berman, G. Obertelli, and R. Wolski. The apples parameter sweep template: User-level middleware for the grid. In *Supercomputing, ACM/IEEE 2000 Conference*, pages 60–78, 11 2000.

[15] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. Heuristics for Scheduling Parameter Sweep applications in Grid Environments. In *9th Heterogeneous Computing Workshop(HCW)*, pages 349–363, 2000.

[16] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. Heuristics for scheduling parameter sweep applications in grid environments. In *Heterogeneous Computing Workshop*, pages 349–363, 2000.

[17] F. Deniz. Variable Structure and Dynamism Extensions To a DEVS Based Modeling and Simulation Framework. Master's thesis, METU, 2010.

[18] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. The Physiology of the Grid. *International Journal of Supercomputer Applications*, 15(3):200–222, 2001.

[19] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid. *International Journal of Supercomputer Applications*, 15(3):200–222, 2001.

[20] N. Fujimoto and K. Hagihara. Near-Optimal Dynamic Task Scheduling of Independent Coarse-Grained Tasks onto a Computational Grid. In *Parallel Processing, 2003*, pages 391–398, 2003.

[21] Y. Gao, Rong H., and J. Z. Huang. Adaptive Grid Job Scheduling with Genetic Algorithms. *Future Generation Computer Systems*, 21.

[22] D. Ghose, H. J. Kim, and T. H. Kim. Adaptive Divisible Load Scheduling Strategies for Workstation Clusters with Unknown Network Resources. *IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS*, 16(10):897–907, 2005.

[23] H. González-Vélez and M. Cole. Adaptive statistical scheduling of divisible workloads in heterogeneous systems. *Journal of Scheduling*, 13(4):427–441, 2009.

[24] M. T. Heath. *Scientific Computing: An Introductory Survey*. McGraw-Hill Higher Education, 2002.

[25] M. Humphrey, G. Wasson, K. Jackson, J. Boverhof, M. Rodriguez, J. Gawor, J. Bester, S. Lang, I. Foster, S. Meder, et al. State and events for web services: a comparison of five WS-resource framework and WS-notification implementations. In *14th IEEE International Symposium on High Performance Distributed Computing, 2005. HPDC-14. Proceedings*, pages 3–13, 2005.

[26] A. Kara, D. Bozagac, and M.N. Alpdemir. Sima: A devs based hierarchical and modular modelling and simulation framework. 2. National Defensive Applications Modelling and Simulation Conference, 4 2007.

[27] N. T. Loc and S. Elnaffar. A Dynamic Scheduling Algorithm for Divisible Loads in Grid Environments. *Journal Of Communications*, 2(4):57–64, 2007.

[28] N. T. Loc, S. Elnaffar, T. Katayama, and Bao H. T. MRRS: A More Efficient Algorithm for Scheduling Divisible Loads of Grid Applications. In *IEEE/ACM International Conference on Signal-Image Technology and Internet-based Systems (SITIS'06)*, 2006.

[29] T. Ma and R. Buyya. Critical-Path and Priority based Algorithms for Scheduling Workflows with Parameter Sweep Tasks on Global Grids. In *Proceedings of the 17th International Symposium on Computer Architecture on High Performance Computing*, pages 251–258, 2005.

[30] C. Peiris, D. Mulder, and S. Cicoria. *Pro WCF: Practical Microsoft SOA Implementation*. Apress, 2007.

[31] B. Sotomayor. *The Globus Toolkit 4 Programmer's Tutorial*. 2005.

[32] A. S. Tanenbaum. *Modern Operating Systems, 2nd Edition*. Prentice Hall PTR, 2001.

[33] G. Theodoropoulos, Y. Zhang, D. Chen, R. Minson, S.J. Turner, W. Cai, Y. Xie, and B. Logan. Large scale distributed simulation on the grid. *EPSRC e-Science Sister Project GR/ S*, 82862, 2003.

[34] Y. Yang and H. Casanova. RUMR: Robust Scheduling for Divisible Workloads. In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)*, 2003.

[35] Y. Yang and H. Casanova. UMR: A Multi-Round Algorithm for Scheduling Divisible Workloads. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)*, 2003.

[36] Y. Yang, K. Raadt, and H. Casanova. Multiround Algorithms for Scheduling Divisible Loads. *IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS*, 16(11):1092–1102, 2005.