ผลกระทบของนโยบายการให้ลำดับความสำคัญภายในต่อประสิทธิภาพการจัดลำดับงานบนกริด และอัลกอริทึมการจัดลำดับงานแบบปรับตัวได้บนกริด

นางสาว ศิรประภา วิริยะประสิทธิ์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2547

ISBN 974-17-6355-7

THE IMPACT OF LOCAL PRIORITY POLICIES ON GRID SCHEDULING PERFORMANCE AND AN ADAPTIVE POLICY-BASED GRID SCHEDULING ALGORITHM

Miss Siraprapa Wiriyaprasit

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering in Computer Engineering
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2004
ISBN 974-17-6355-7

Thesis Title	The Impact of Local Priority Policies on Grid Scheduling Performance and an Adaptive Policy-based Grid Scheduling Algorithm
By	Siraprapa Wiriyaprasit
Field of study	Computer Engineering
Thesis Advisor	Veera Muangsin, Ph.D.

Accepted by the Faculty of Engineering, Chulalongkorn University in Partial Fulfillment of the Requirements for the Master's Degree

ศิรประภา วิริยะประสิทธิ์ : ผลกระทบของนโยบายการให้ลำดับความสำคัญภายในต่อประสิทธิภาพการจัดลำดับงานบนกริด และอัลกอริทึมการจัดลำดับงานแบบปรับตัวได้บนกริด. (THE IMPACT OF LOCAL PRIORITY POLICIES ON GRID SCHEDULING PERFORMANCE AND AN ADAPTIVE POLICY-BASED GRID SCHEDULING ALGORITHM) อ. ที่ปรึกษา : ดร. วีระ เหมืองสิน, 72 หน้า. ISBN 974-17-6355-7.

วิทยานิพนธ์ฉบับนี้ศึกษาปัญหาที่เกี่ยวกับการจัดลำดับงานในกริด โดยวัดผลกระทบเมื่อองค์กรบางส่วนในกริดให้ลำดับความสำคัญของงานภายในองค์กรมากกว่าข้างนอกองค์กร และเสนออัลกอริทึมแบบปรับตัวได้เพื่อลดผลกระทบดังกล่าว จากผลการทดลอง เมื่อองค์กรบางส่วนให้ลำดับความสำคัญของงานภายในองค์กรมากกว่า จะทำให้องค์กรที่เหลือได้รับผลกระทบคืองานถูกทำให้ล่าช้าออกไป อัลกอรึทึมใหม่ที่ได้เสนอนำเอานโยบายภายในองค์กรในการให้ลำดับความสำคัญของงานมาพิจารณาและใช้ในการปรับการจัดลำดับงานในกริด อัลกอริทึมใหม่สามารถลดผลกระทบต่อสมรรถนะการทำงานเนื่องจากการใช้นโยบายภายในองค์กรในการให้ลำดับความสำคัญของงานที่แตกต่างกัน และทำงานได้อย่างมีประสิทธิภาพภายใต้ความหลากหลายของปริมาณงานและสัดส่วนขององค์กรที่ใช้นโยบายต่างกัน

ภาควิชา วิศวกรรมคอมพิวเตอร์	ลายมือชื่อนิสิต...
สาขาวิชา วิศวกรรมคอมพิวเตอร์	ลายมือชื่ออาจารย์ที่ปรึกษา......................................
ปีการศึกษา 2547	ลายมือชื่ออาจารย์ที่ปรึกษาร่วม……………….…..

4670518021 : MAJOR COMPUTER ENGINEERING

KEY WORD: DISTRIBUTED SYSTEM / GRID / SCHEDULING / POLICY

SIRAPRAPA WIRIYAPRASIT : THE IMPACT OF LOCAL PRIORITY POLICIES ON GRID SCHEDULING PERFORMANCE AND AN ADAPTIVE POLICY-BASED GRID SCHEDULING ALGORITHM. THESIS ADVISOR : VEERA MUANGSIN, PH.D., 72 pp. ISBN 974-17-6355-7.

This thesis addresses a problem with job scheduling in a computational grid. It investigates the performance impact when some sites in the grid apply a priority policy in favor of local jobs and proposes an adaptive site selection algorithm for grid scheduler to reduce the severity of this impact. It is demonstrated that when some sites apply a priority policy in favor of local jobs, other sites will suffer from much longer completion times. The proposed grid scheduling algorithm takes into account local scheduling policies and adjusts the global scheduling accordingly. The results show that the new algorithm can reduce the performance impact due to different local priority policies and perform effectively under various levels of workload and fractions of sites with different policies.

Department	Computer Engineering	Student's signature..
Field of study	Computer Engineering	Advisor's signature..
Academic year	2004	Co-advisor's signature.......................................

Acknowledgements

I would like to express my gratitude to my supervisor, Dr. Veera Muangsin, for his instructive and invaluable advice. He always encourages me to find my own way. Without his guidance, this research would not have been possible. I am very fortunate to be his student.

I would like to thank Asst. Prof. Dr. Putchong Uthayopas for his expert and constructive advice.

I would like to thank all members of Scientific Parallel Computer Engineering Research Unit, Information System Engineering Laboratory and Intelligent System Laboratory who have provided help and friendship during my Master's degree study.

Finally, I thank my parents and my brother for their support, patience and encouragement that helped me to keep going.

Contents

Contents (cont.)

List of Tables

List of Figures

List of Figures (Cont.)

CHAPTER I

INTRODUCTION

1.1 Background

Grid computing [1] is a form of distributed computing and also a technology for high performance computing in which computational and data resources in a wide area network are integrated into one large computing environment. It supports formation of a Virtual Organization (VO) which consists of individual or multiple organizations. The resources owned by individual organizations in a VO are shared and coordinated. As a result, grid computing provides aggregated computing power to solve a common problem.

The grid resources are heterogeneous and owned by different organizations which have different local management systems and different local resource management. Therefore, an effective grid scheduling system or grid resource management system is needed to efficiently exploit and utilize the available grid resources.

Grid scheduling can be conducted by cooperation between autonomous local schedulers and a grid scheduler in grid middleware [2]. A grid scheduler is responsible for selecting a site for each job in order to effectively distribute workload among multiple sites in the grid.

1.2 Problem Statement

It is a general practice that a grid scheduler does not have control over local schedulers on grid sites. Therefore, local schedulers have the freedom to apply their local policies. Even though a grid site allows users from other sites to submit their jobs, there is no guarantee that the remote jobs will be treated equally to the local jobs.

In many circumstances, the site's administrator may wish to speed up local jobs by giving higher priority to local jobs than remote jobs. This is not an unlikely speculation since currently grid sites are not enforced by any global policy on resource sharing. If this is the case in many grid sites, it may cause undesirable effects to other sites in the grid. Although the grid concept permits different local scheduling policies, little work has been done on job scheduling in a grid environment where different local scheduling policies are employed.

It is still unclear whether local scheduling policies that give different priority levels to local and remote jobs will have any effect on global scheduling in a grid environment. If so, how grid scheduling can cope with such effect is yet another problem.

1.3 Objectives

The general objectives of this thesis are the understanding of the impact of local scheduling policies on the performance of grid scheduling and the improvement of a grid scheduling algorithm to cope with such effect. To be more specific, the goals of this thesis are

- To evaluate the performance impact on global scheduling in a grid environment after applying a priority policy onto local schedulers in favor of local jobs
- To propose an adaptive site selection algorithm for a grid scheduler based on priority policies of local schedulers in order to reduce the severity of such effect without interfering the autonomy of local schedulers.
- To evaluate the proposed algorithm and compare it to the conventional algorithm.

1.4 Organization of the Thesis

The contents of the thesis are divided into 6 chapters. The details of all chapters are as follows:

- Chapter 1 provides a general introduction to the research. In the beginning, the chapter points out to general practices for applying a priority policy in favor of local jobs and the problem. Then, the chapter presents the goal of this research, and outlines the contents of the remaining chapters.

- Chapter 2 reviews literature related to associated with grid scheduling.
- Chapter 3 provides the evaluation results of the impact of local priority policies. The first part describes the grid scheduling model and concepts used in the thesis. The second part explains the configuration of simulation. The performance results are shown in the last part.
- Chapter 4 describes the proposed algorithm and the experimental results compared to the results in the previous chapter.
- Chapter 5 describes the implementation of proposed algorithm in a grid portal.
- Chapter 6 gives the conclusion and suggestion for further research.

CHAPTER II

LITERATURE REVIEW

2.1 Grid

Grid [1,3] integrates computational and data resources that are geographical distributed and owned by different organizations into one large computing environment. It supports Virtual Organization (VO) which enables diverse groups of organizations and individuals to share resources easier. Consequently, members can collaborate to achieve a shared goal. Grid components are defined into layers. Higher-level services are built on lower-level services as show in Figure 2.1.

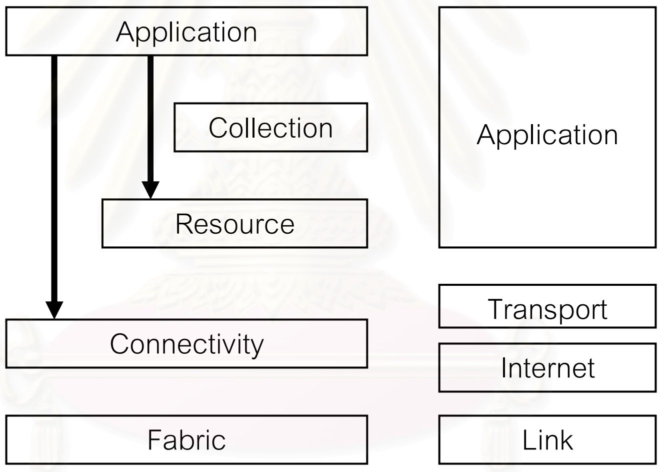


Figure 2.1: The layered Grid architecture and its relationship to the internet protocol architecture [3]

The first layer, the Grid Fabric layer, provides the unified access to resources to reduce to the complexity in accessing heterogeneous resources. The second layer, the Grid Connectivity layer, is responsible for defining the core communication and authentication protocols required for Grid-specific network transaction. The Grid Resource layer, built on the two previous protocols, defines protocols for the secure negotiation, initiation, monitoring, control, accounting, and payment of sharing operations on individual resources. The forth layer, Collective layer, contains protocols and services that are not specific to any single resources like in Resource Layer.

Instead it provides protocols and services to communicate to multiple resources. The sharing behaviors can be implemented without changing the requirements on shared resources because Collective Layer is built on top of Resource and Connectivity layer. The last layer, the Application Layer, includes the user applications and well defined protocols providing useful services such as resource management, date access and resource discovery.

There are five main necessary components to form a grid: grid resources, grid middleware, user-level grid middleware, grid application and portal [4]. There are many types of grid resources including computers, clusters, storage devices, databases, and special scientific instruments which are geographically distributed.

Grid Middleware offers the core services for remote process management, co-allocation of resources, storage access, information (registry) and discovery, security, authentication, and Quality of Service (QoS). Globus [5,6], Condor [7], and Unicore [8] are examples of grid middleware and now Globus is a de facto standard.

Next, User-Level Grid Middleware includes grid programming environment and tools offering high-level services and brokers. Brokers act as user agents for resources management and scheduling application tasks for execution on grid resources. Grid applications are developed using grid-enable languages and utilities. The examples of applications are parameter simulation and grand-challenge problem which requires massive computing power. Finally, Grid Portals offer Web-enabled application services so that users can submit and collect results for their jobs on remote resources through web-based services.

2.2 Grid Scheduling Schemes

Grid scheduling architecture composes of a collection of local schedulers and one or more grid schedulers in a grid middleware [9]. Usually, local schedulers are batch queuing systems handling submitted jobs by allocating resources from a pool of computers, e.g. a dedicated cluster or a cycle-stealing networked computer pool. Local schedulers reside in single administrative domains or single sites. Since the grid concept promotes site autonomy, local schedulers have the freedom to apply scheduling policies. End users have little knowledge on how the local scheduler interprets their requirements.

A grid scheduler is responsible for selecting a site for each job in order to effectively distribute workload among grid sites. One of the simplest algorithms to select a site is the greedy approach which iteratively assigns each job to the site that is most likely to complete the job earliest without considering the rest of pending, rescheduled or submitted jobs. The greedy approach is used by many projects [10].

Grid scheduler schemes can be classified into two categories, centralized scheme and distributed scheme [2]. In the centralized scheme, all jobs are submitted to a single grid scheduler which is responsible for making overall grid scheduling decisions and assigning each job to a specific resource. Once a job is submitted to a site, the grid scheduler does not have control over local schedulers.

On the other hand, in the distributed scheme, grid schedulers are in every site. All jobs are submitted locally to their grid schedulers. Grid schedulers periodically query to each other to collect local information. The job can be transferred to the site with lower load [11].

The centralized scheme is not very scalable because the grid scheduler must maintain a lot of information of all sites. On the other hand, the distributed scheme is the most scalable, because all jobs are submitted locally. However, this scheme has a large overhead for negotiation. Therefore, the centralized scheme is currently the most popular one.

2.3 Grid Scheduling Modes

Grid scheduling mode can be classified into two categories, namely on-line mode and batch mode [12]. In the on-line mode, a job is mapped onto a machine as soon as it arrives at the grid scheduler and considered only once for matching and scheduling. In the batch-mode, jobs are not mapped onto the resources as they arrive. Instead, they are collected in a set that is examined for mapping at prescheduled times called mapping events. Knowing the execution of a larger number of jobs enables better mapping heuristics.

The examples of simple heuristics for on-line mode scheduling are the MCT (minimum completion time) and MET (minimum execution time) heuristics. MCT assigns each job to the machine that gives the minimum completion time if the job is run on that machine. The completion time is the time since the job is submitted, waits in the queue and finishs execution. Unlike MCT, the MET heuristic considers only execution time but not the wait time. MET assigns each job to the machine that provides the minimum execution time. The MCT heuristic is commonly used as a benchmark for the on-line mode [12].

The examples of simple heuristics for batch-mode scheduling are Min-Min, Max-min and Sufferage [12]. In these three heuristics, the first step is the same. It begins with determining the machine providing the earliest completion time for every job. Then, the second step varies. In the Min-Min heuristic, the job with the earliest expected completion time is assigned to the corresponding machine. On the other hand, in Max-Min heuristic, the job having the maximum earliest completion time is assigned to the corresponding machine instead. In the Sufferage heuristic, it assigns a machine to a task that would “suffer” most in terms of expected completion time if that machine is not assigned to that job. The Sufferage value of a task is the difference between its second earliest completion time and its first earliest completion time. Finally, the process is repeated until every task is assigned to a machine.

2.4 Prediction of Execution time

Both on-line and batch mode heuristics assume that job execution times on each machine in the grid can be estimated. This assumption is commonly used when studying mapping heuristic for grid systems [12]. The examples of techniques used for prediction of execution time are instance base learning technique, short term prediction technique and test run technique.

The instance based learning technique uses a database of experiences to maintain and to make predictions [13]. A scheduling job can be an experience or a query. When a job finishes executing, it becomes an experience to the database.

In the short term prediction technique, the scheduler makes initial guess execution time of all jobs. When the tasks are complete, the scheduler used the observe execution time to adjust the prediction algorithm [14]. The technique is used in Network Weather Service (NWS) [15]

The test run technique estimates the execution time of the task by executing some sub-tasks on the set of available resources using the grid scheduling policy. The measured execution times are used as first approximation of the execution time for the remaining sub tasks [16].

The impact of accurate prediction of execution time is studied in [17]. The results show that more accurate requested execution times can improve system performance. Furthermore, users who provide more accurate requested execution times also improve the performance, even if other jobs do not provide more accurate requested execution times.

2.5 Cost Functions in Grid Scheduling

Different grid scheduling algorithms have different effects and characteristics due to different cost functions and mechanisms. The cost functions can be classified into two categories, namely performance-based cost functions and QoS-based cost functions. The goal of performance-based cost functions is to optimize the overall performance such as the average completion time, CPU utilization, throughput, average slowdown, makespan, cost, budget and deadline. In QoS-based cost functions, the goal is to guarantee some services such as maintaining the desirable network bandwidth or predictable execution time of an application.

The simplest algorithms that use performance-based cost functions include MCT, MET, Min-Min, Max-Min and Sufferage. The cost functions of MCT and MET are completion time and execution time respectively. For Min-Min and Max-Min, the cost function is completion time. For Sufferage, the cost functions are completion time and sufferage value.

CMin-Min, CMax-Min and CSufferage algorithms have been proposed to optimize both makespan and cost of running [18]. They were adapted from Min-Min, Max-Min and Sufferage which optimize only makespan, respectively. The cost function is Priority Index, which is a function of execution time, ready time, cost and accumulative cost and has a weight for each parameter.

K-Distributed model and K-Dual Queue Model have been proposed to optimize average job slowdown and turnaround time [11]. The distributed grid scheduling scheme uses this algorithm. The cost function of this model is the completion time. The grid scheduler distributes each job to the first K sites that have the earliest completion time. When a job is able to start at any sites, that site informs the grid scheduler at the site that originates the job. Then, that grid scheduler contacts the other remaining grid schedulers to cancel that job from their respective queue. However, a higher degree of “overbooking” results in an increase in the amount of work to be done at each local scheduler. Also, the amount of communication and synchronization among the grid schedulers increases.

The Computational Economy model is another model for grid scheduling [4]. It is based on a well proven approach for resource management complexity and decentralization that is present in real economies. It supports mechanisms and policies that help in regulating the supply and demand for resources. Therefore, this model can achieve performance and meet the deadline and budget constraints at the same time. The cost function for the economic model comprises of resource costs deadlines.

QoS guided Min-Min algorithm has been proposed to provide QoS of a network bandwidth. It was adapted from Min-Min by taking the QoS matching into consideration while scheduling. It maps the job with high QoS required first, and then maps the rest of the requested jobs.

The compensation-based scheduling scheme is an adaptive grid scheduling with a feedback control framework has been introduced to provide predictable completion time [16]. The difference between the monitored application performance and the desired performance is used as a cost function to perform correction by dynamically allocating additional resources. The experiment results show that compensation based scheduling was effective in reducing execution time estimation misses and total execution times of grid applications.

A policy model has been proposed to enforce both VO and resource owner scheduling policy in a data grid environment [19]. The load of each site, epoch resource allocation and burst resource allocation are used as factors in the cost function. There are policy enforcement points at both sites and VOs. Site policy enforcement points enforce the site policy by preempting the job if policy requirements are no longer met. VO policy enforcement points also operate in a similar way as the previous one.

2.6 Resource Managers

Resource management system (RMS)[1] is central to the operation of a grid [20]. It provides basic functions such as accepting requests for resources and assigning specific machine resources to a request. Resource managers can be classified into two categories: namely traditional resource managers and grid-enabled resource managers.

2.6.1 Traditional Resource Managers

[1] In this thesis, the terms 'resource management system', 'resource manager' and 'scheduler' are used interchangeably.

A traditional resource manager is used within a set of computational resources in a particular domain. The examples of traditional resource managers are as follows.

PBS (Portable Batch System) [21], the Portable Batch System, is a batch scheduler. PBS provides most of known job scheduling policies such as FIFO (first-in-first-out), SJF (shortest-job-first), Fairshare, etc. PBS allows each site to create its own scheduler and allows a job to be launched from a cluster and executed on another cluster. In addition, PBS can create logical queues. A job can be automatically routed to a specific logical queue based on the predefined condition of each logical queue. Also, a job can specify the destination queue itself.

The Maui scheduler [22] is an external job scheduler used on clusters and capable of enforcing complex scheduling policies. Maui uses weights and various algorithms to efficiently schedule jobs instead of FIFO order. The policy can be tuned efficiently. In addition, Maui can be used as an external scheduler for other resource management systems such as PBS and SGE. Maui makes and enforces its decisions by querying and controlling a resource management system. For example, PBS manages the job queues and the computational resources. Maui queries PBS to obtain the job and the node information. Then, Maui directs PBS to manage jobs in accordance with specified Maui's policies, priorities, and reservations. The performance gain when Maui is used as the external scheduler of PBS is studied in [23].

The Sun Grid Engine (SGE) [24] is another batch queuing system. It was formerly named Codine before Sun purchased it. The current version of SGE supports only the default option which implements first-in-first-out (FIFO). The concept of queues in the SGE package is defined per host basis and SGE does not support logical queues.

Nimrod [25, 26] uses a simple declarative parametric modeling language to automate the execution of parameter sweep applications and uses the concept of computational economy.

Condor [7] is a resource management system designed to support high throughput computing (HTC). Condor discovers the idle resources and allocates those resources to the application tasks. When the owner of resource returns to use it, that resource will be deallocated. Therefore, Condor always respects the autonomy of the system. In addition, Condor provides an extensible resource description language called Classified Ads to specify the resource requirements in a more detailed and controlled way.

SQMS (Simple Queue Management System) [27] is a simple resource scheduler based on thread and networking technology. SQMS has been developed in at Parallel Research Group at Kasetsart University. All logical components are designed to be pluggable. Therefore, SQMS are very flexible to support the new types of jobs and load balancing policies.

2.6.2 Grid-Enabled Resource Managers.

Currently there have been many implementations of Grid-enabled resource managers. The examples of grid-enable resource managers are as follow.

Nimrod/G [28] is an extension of Nimrod to support grid and uses Globus middleware services for remote access, resource discovery and scheduling job over grid. Nimrod/G still uses the economic model and supports parameter sweep applications. Especially, it supports user-defined deadline and budget constraints for scheduling. The grid-enabled Nimrod/G broker is implemented as part of a new framework called GRACE (Grid Architecture for Computational Economy).

Condor-G [29] is of the grid-enabled version of Condor. Condor-G address es issues of failure, credential expiry, and interjob dependencies that are not supported by Nimrod/G

SCEGrid [30] is the extension of SQMS and functions on top of Globus. It simplifies the usage of grid systems by automating the process of resources selection and allocation. Currently, SCEGrid supports only sequential jobs without job dependencies. The other features including resource commerce model, smarter load balance policy, job work-flow and parallel job support are planed to be supported in the future.

CHAPTER III

THE IMPACT OF LOCAL PRIORITY POLICIES

This chapter presents an evaluation of the performance impact of applying a priority policy onto local schedulers in favor of local jobs.

3.1 Grid Scheduling Model

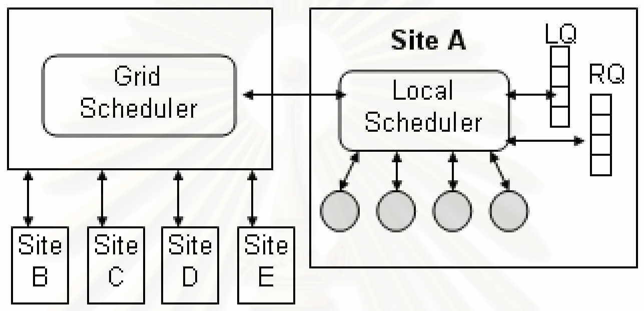


Figure 3.1: The grid scheduling model

The grid scheduling model under investigation is the centralized on-line model, which is a widely accepted model at present. The evaluated scheduling model is depicted in Figure 3.1. A grid scheduling system is divided into two levels, namely a grid scheduler and local schedulers. Every job is first submitted to the grid scheduler. Then, the grid scheduler contacts a group of local schedulers, called candidates, to query the predicted completion times if the job was run on those sites. The minimum completion time (MCT) approach is used by the grid scheduler. Therefore, the grid scheduler actually submits the job to the local scheduler that returns the shortest predicted completion time. If the grid scheduler assigns a job to the local scheduler on the same site where the job is originated, the job is called a *local job*. If it is submitted to other sites, it becomes a *remote job*.

For simplicity, the grid in our study consists of a number of clusters, each of which employs a local batch scheduler using a First-Come-First-Serve (FCFS) scheduling algorithm. All jobs are sequential and computational intensive programs, of which the time for data I/O is negligible, compared to computation time. Each site offers dedicated and space-shared resources and runs jobs in an exclusive fashion.

The grid sites employ two different priority-based scheduling policies. The first policy is to treat local jobs and remote jobs equally. This can be implemented by putting local and remote jobs in the same queue (single-queue model). The other policy is the *local-job-first* policy which gives a higher priority to local jobs. This can be implemented by using two separate queues (dual-queue model), one for local jobs and the other for remote jobs. The local-job queue (LQ) is given a higher priority than the remote-job queue (RQ). Therefore, a remote job can be dispatched only when all jobs in the local queue have been dispatched.

3.2 Method of Completion Time Estimation

The estimated completion time (or estimated response time) of a job on a site can be calculated by summing up the estimated execution time and the estimated wait time of the job on that site.

The estimated execution time of a job on a machine can be calculated by dividing the workload with the machine speed. To estimate the wait time, the method of start time prediction describe in [13] is used. The start time is the time when the job was selected to execute. Therefore, the relative start time value is the wait time. The start time is calculated by performing a simulation of the scheduling algorithm and policy, which results in estimated start times for each of the jobs waiting in the queue(s). This simulation is performed by using predictions of the estimated execution times of jobs.

3.3 Analysis of Real Workload

In order to provide the simulation with a realistic workload, a workload trace on a real machine is studied. The Feitelson's parallel workloads archive [31] is a great source of real workload traces, especially on parallel machines. One of the traces that are most frequently used in literature is the CTC SP2 log. This log contains records between June 1996 to May 1997 on the 512-node IBM SP2 located at the Cornell Theory

Center (CTC). Therefore, the characteristics and distribution of this workload trace are studied and used as that basis for the workload model for the empirical study in this thesis.

The cumulative distribution function (CFD) is the probability that the variable takes a value less than or equal to a particular value [32]. CDF is a good way of examining a probability distribution of the workload because it does not depend on the quantization interval.

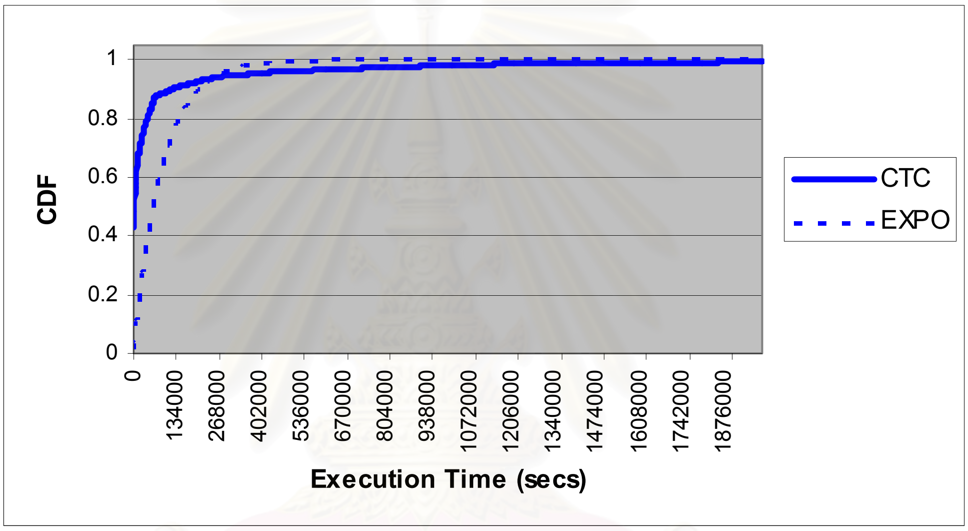


Figure 3.2 Comparison the CDF of execution times between CTC workload and exponential distribution

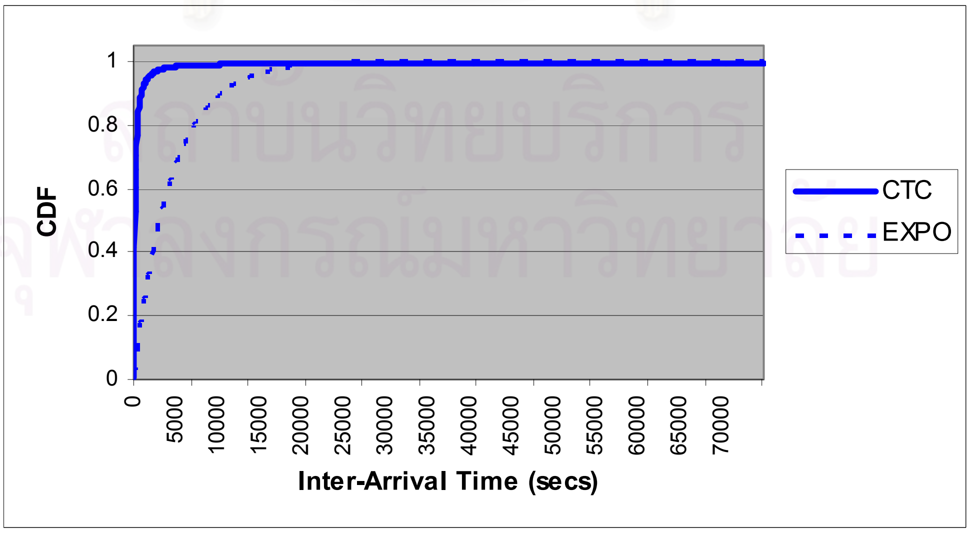


Figure 3.3 Comparison the CDF of inter-arrival times between CTC workload and exponential distribution

The distribution of the CTC workload fits closely the Hyper Erlang distribution of Common Order, which is a generalization of the exponential, the hyper exponential, and the Erlang distribution [33]. The exponential distribution is a similar distribution but much simpler and more widely used. Therefore, it is used instead of the Hyper-Erlang distribution for synthetic workload generation in the following simulation study.

For simplicity, all parallel jobs in the trace are converted to sequential jobs by multiplying the number of allocated processors by the average CPU time used. From the archive, the average execution time and average inter-arrival time are 90,083 seconds and 379 seconds respectively. These two values are used in modeling the exponential distribution. The comparison of CDF of the inter-arrival times and execution times between CTC workload and the corresponding exponential distribution are shown in Figure 3.2 and Figure 3.3, respectively. The exponential distribution is generated by using the average execution time and average inter-arrival time of CTC workload.

In addition, the utilization of CTC workload is only 55% [31]. This is inadequate to put enough pressure onto the system in order to observe any significant effects. Therefore, the distribution of the synthetic workload will be tuned up to increase the utilization.

3.4 Simulation Configuration

The simulation model was implemented on a java-based discrete-event grid simulation toolkit called *GridSim* [34]. The simulated grid environment consists of 5 sites (A, B, C, D, and E). Each site has one cluster with four computing nodes. The size and complexity of the evaluated system is limit because each simulation experiment takes quite a long time.

The grid scheduler uses the MCT approach described in section 3.1. Simulation parameters and queue configurations are summarized in Table 3.1 and Table 3.2 respectively.

Table 3.1: Simulation Parameters

Parameter	Case	Site A	Site B	Site C	Site D	Site E
Machine Speed (MIPS)	1	500	500	500	500	500
	2	700	600	500	400	300
	3	300	400	500	600	700
Number of jobs	1	880	880	880	880	880
	2	1232	1056	880	704	528
	3	528	704	880	1056	1232
Inter-arrival time of jobs: Heavy load (**H**) (Exponential Dist. , Second)	1	450	450	450	450	450
	2	321	375	450	563	750
	3	750	563	450	375	321
Inter-arrival time of jobs: Medium load (**M**) (Exponential Dist. , Second)	1	525	525	525	525	525
	2	375	438	525	656	875
	3	875	656	525	438	375
Inter-arrival time of jobs: Light load (**L**) (Exponential Dist. , Second)	1	600	600	600	600	600
	2	429	500	600	750	1000
	3	1000	750	600	500	429
Job Sizes (Exponential Dist., Million Instructions)	ALL	10^6				

Table 3.2: Queue Configurations

Identifier	Site(s) using dual queues	Site(s) using single queue
Q0	-	A, B, C, D, E
Q1	A	B, C, D, E
Q2	A, B	C, D, E
Q3	A, B, C	D, E
Q4	A, B, C, D	E
Q5	A, B, C, D, E	-

The speed of machines in each site are categorized into three cases with the mean of 500 million instructions per second (MIPS). In case 1, the machine speeds of all sites are 500 MIPS. However, in case 2 and case 3, the machine speeds of all sites are different to represent heterogeneity. In case 2, the machine speeds of site A, B, C, D and E are sorted by descending that are 700, 600, 500, 400 and 300 respectively in order to analyze the performance impact when the sites with high speed machines start to use dual-queue model. On the other hand, in case 3, the machine speeds are sorted by ascending in order to analyze the performance impact when the sites with low speed machines start to use dual-queue model.

From the total of 4,400 jobs, only 4,000 jobs in the middle are used for performance evaluation. The number of jobs which are originated at each site is designed to correspond to machine speed. As a result, the site with high speed machines will generate more jobs. Job sizes are also determined randomly with an exponential distribution with the mean of 10^6 million instructions.

Consider the speed of the machines, the number of nodes and the job sizes, the system will have 100% utilization if the inter-arrival time of jobs entering the grid scheduler is equal to or less than 100 seconds. As a result, three levels of inter-arrival time : 90, 105 and 120 are chosen to represent heavy load, medium load and light load, respectively. The utilization levels for these three inter-arrival times are 85%, 100% and 100% respectively. As a result, the inter-arrival times of jobs at each site are determined randomly with an exponential distribution with the mean of 450, 525 and 600, respectively. For case 2 and case 3 where the number of jobs at each site are not equal, the inter-arrival of each site will be adjusted according to the number of jobs of that site.

The number of sites using dual queues varies from 0 to 5 sites, as shown in Table 3.2. For example, in configuration Q0, all sites use the single-queue model. On the other hand, in configuration Q5, all sites use the dual-queue model. Each experiment was performed 10 times with different seeds for the random number generator and the average values are reported.

3.5 Performance Evaluation of MCT Algorithm

The performance metrics to be considered are the average completion time (Figure 3.4-3.6), the maximum of average completion times of site A to E (Figure 3.7-3.9), the standard deviation of average completion times of site A to E (Figure 3.10-3.12) and the maximum gain and loss of average completion times of site A to E (Figure 3.13-3.15). The performance plots of individual sites are labeled as A, B, C, D and E and the performance of all sites is labeled as ALL.

3.5.1 The Average Completion Times of MCT Algorithm

The relationship between average completion times and queue configurations is depicted in Figure 3.4-3.6. The results show that the average completion time of all sites changed very little compared to the completion times of individual sites, which are significantly affected. Four characteristics can be concluded as follows.

Firstly, the sites that employ the dual-queue model have better average completion times than single-queue sites. For example, in configuration Q2, site A and B use the dual queues while site C, D and E use the single queue. Under Q2 in heavy load of case 1 (Figure 3.4a), the average completion time of site A and B are 14,308 and 15,027 seconds respectively. On the other hand, the average completion time of site C, D and E are 27,572, 27,989 and 27,187 seconds respectively.

Secondly, switching from the single queue to dual queues reduces the average completion time of the site but increases the average completion times of all other sites in the grid, especially the remaining single-queue sites. For example, in Figure 3.4a, when site C switches to use dual queues (from Q2 to Q3), its average completion time reduces from 27,572 to 14,718 seconds; in other words, 47% reduction. However, the average completion times of all other sites increase.

Thirdly, as the fraction of dual-queue sites increases, the difference between average completion time of single-queue and dual-queue sites gets larger. Therefore, the last site that remains using the single queue mode will suffer most. The effect is more severe when the workload is heavier. For example, in Figure 3.4a, the average completion time of site E in Q4 is 52899.34 seconds, which is 133% greater than the total average completion time.

Finally, in a particular queue configuration, the completion times of sites in single-queue mode are nearly equal in all cases and workloads. On the other hand, sites in dual-queue mode, the average completion times likely correspond to their machine speeds. This is obviously seen in medium load and light load. Therefore, the average completion times of dual-queue sites in Case 1 are nearly equal as shown in Figure 3.4b and Figure 3.4c. The average completion times of dual-queue sites in Case 2 are sort by ascending as shown in Figure 3.5b and Figure 3.5c. Lastly, in Case 3, the average completion time are sorted by descending as shown in Figure 3.6b and Figure 3.5c

3.5.2 The Maximum Average Completion Times of MCT Algorithm

The relationship between queue configuration and maximum average completion time of all sites is depicted in Figure 3.7-3.9. Label H, M and L represent heavy load, medium load and light load respectively. The maximum points usually come from the single-queue sites except in configuration Q5 where all sites use dual queues. The figures obviously show that the maximum average completion time increases as the fraction of dual-queue sites increases from Q0 to Q4. Therefore, the users in single-queue sites will suffer most from this policy.

In addition, when all sites use dual-queue model (Q5), the maximum average completion time is not always reduced from those of Q4. For example, in heavy load of all cases (Figure 3.7a, 3.8a and 3.9a), the maximum average completion time of Q5 is less than that one of Q4. However, in light load of case3 (Figure 3.9c), the maximum average completion times of Q5 is increased.

3.5.3 The Standard Deviation of Average Completion Times of MCT Algorithm

The standard deviations of average completion times in each queue configuration are depicted in Figure 3.10-3.12. The result can be interpreted in the same manner as the figures of the maximum average completion time. First, the standard deviation increases as the fraction of dual-queue site increases. Also, when all sites use dual-queue model, the standard deviation of the average completion time is not always less than that of Q4.

3.5.4 The Maximum Gain and Loss of Average Completion Times

The maximum gain and loss of average completion times in a queue configuration are depicted in Figure 3.13-3.15. The maximum gain in a queue configuration is the maximum percentage increment in average completion time from the average completion time of each site in Q0. On the other hand, the maximum loss in a queue configuration is the maximum percentage decrement in average completion time from the average completion time of each site in Q0. Therefore, the plot of both maximum gain and maximum loss shows the performance boundary of all sites in a queue configuration.

An important characteristic obtained from the graphs is that the peak performance loss is always greater than the peak performance gain. For example, in Figure 3.13b, the maximum gain of all queue configurations is 16.5% which is in Q1, and the maximum loss of all queue configurations is 43.5% which is in Q4.

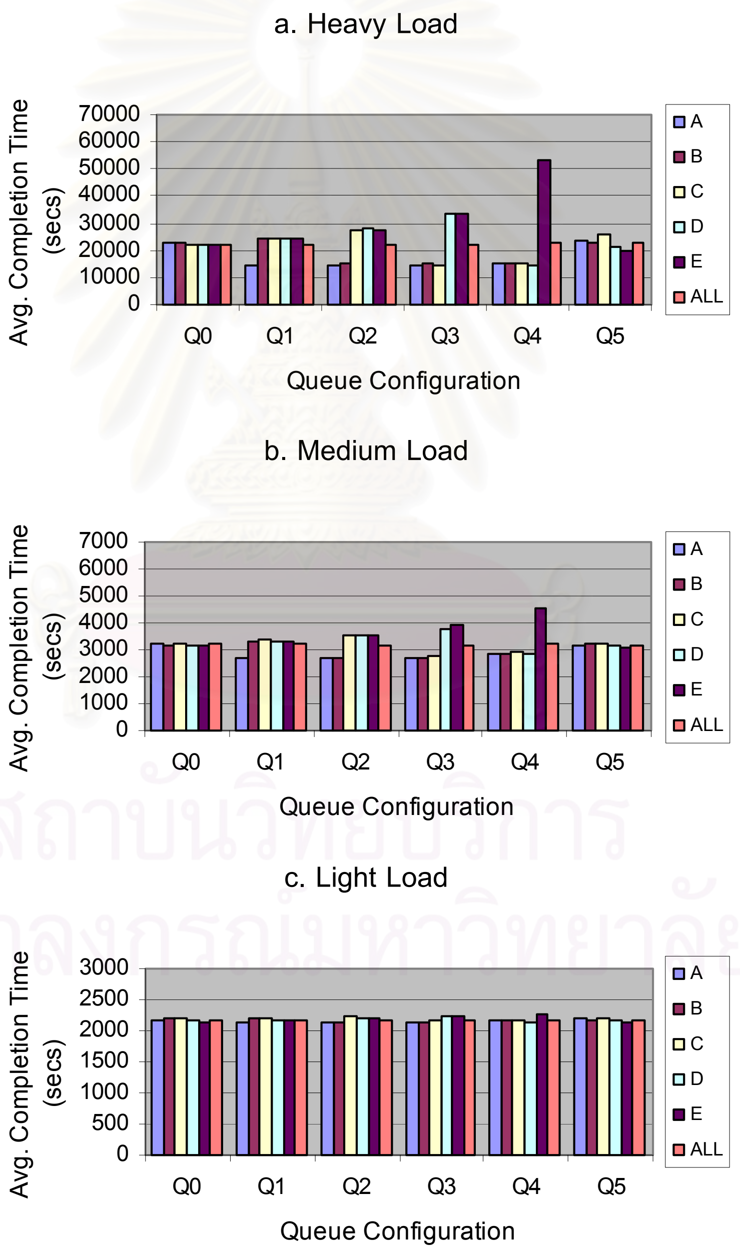


Figure 3.4: Average completion time of MCT algorithm under Case 1

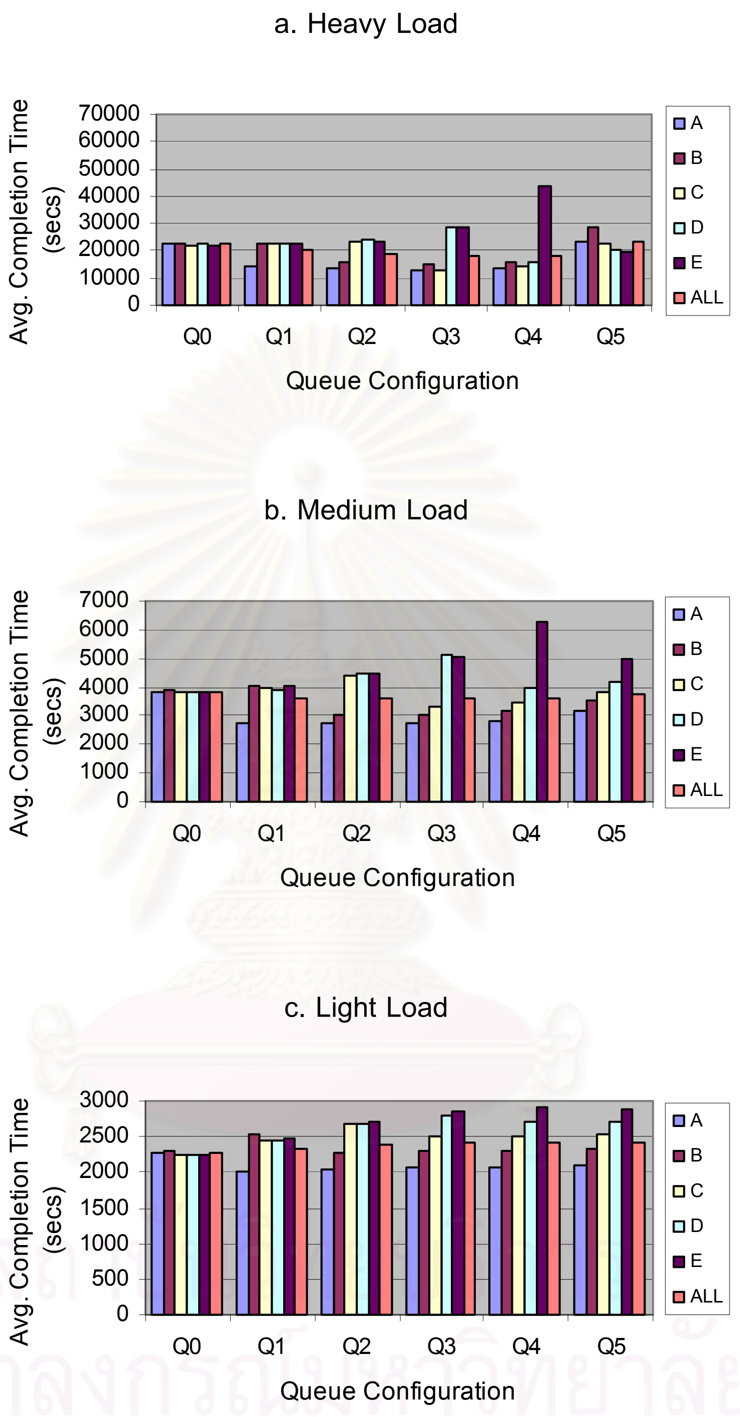


Figure 3.5: Average completion time of MCT algorithm under Case 2

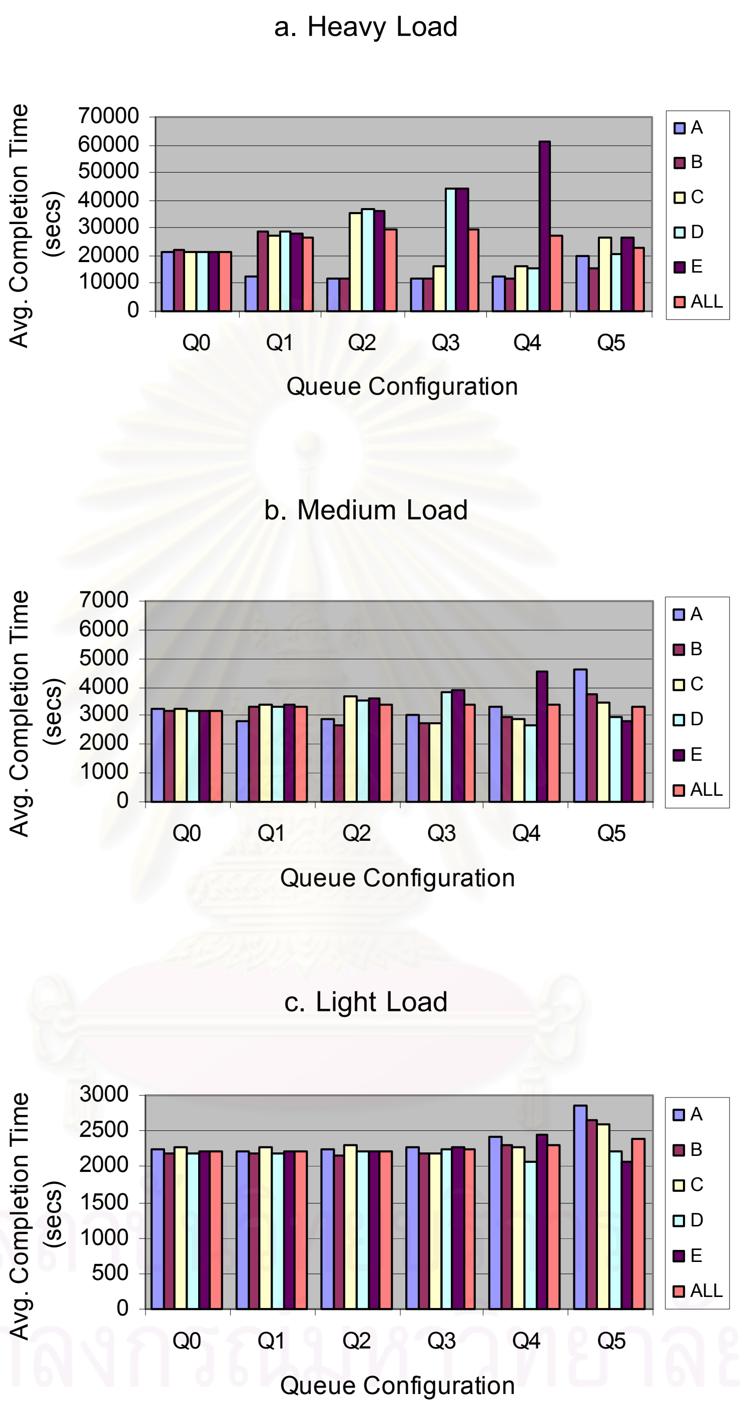
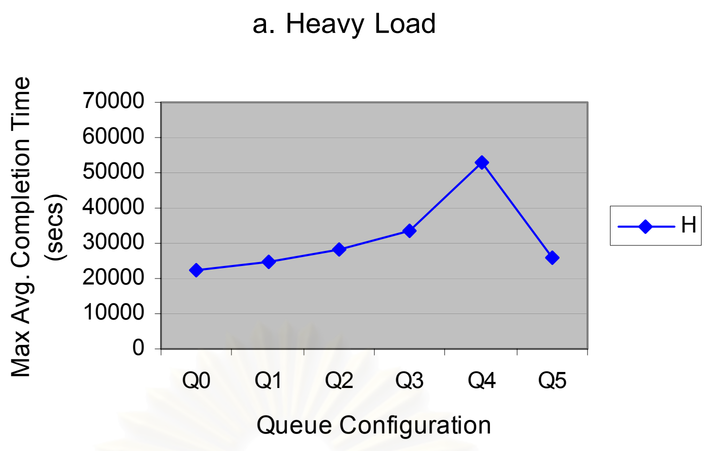
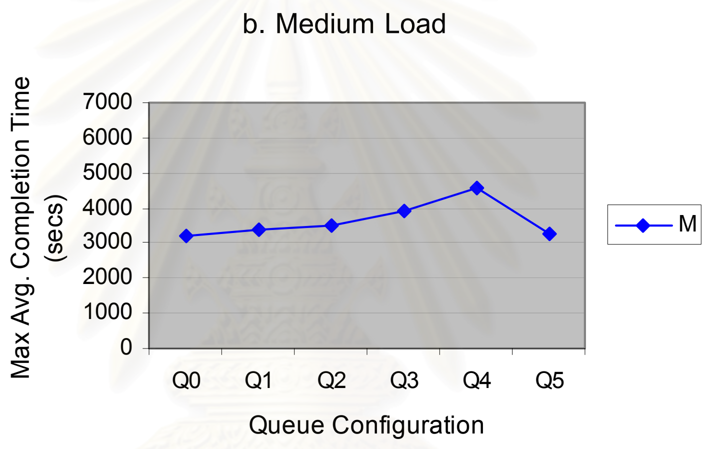


Figure 3.6: Average completion time of MCT algorithm under Case 3





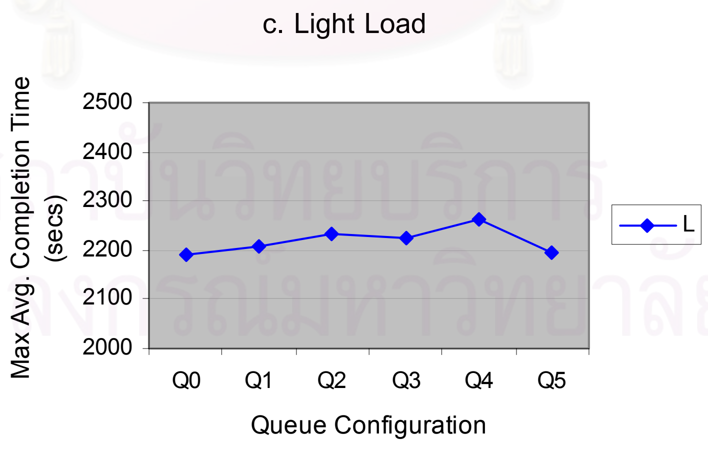
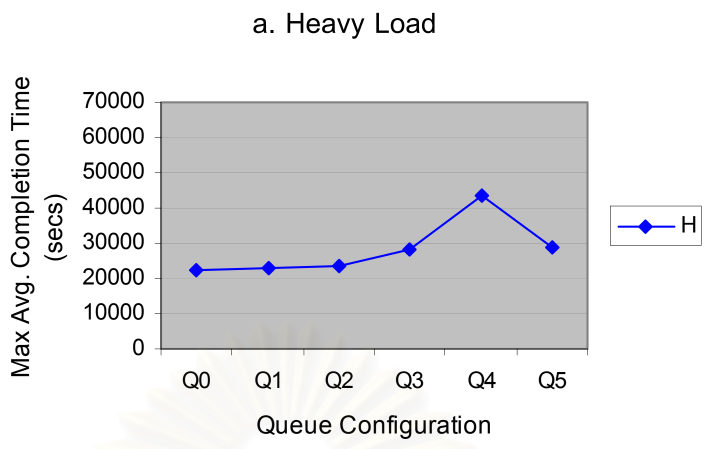
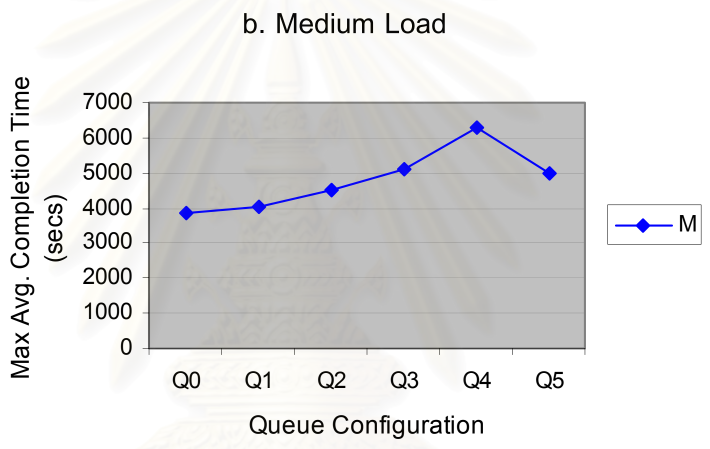


Figure 3.7: Maximum average completion time of MCT algorithm under Case 1





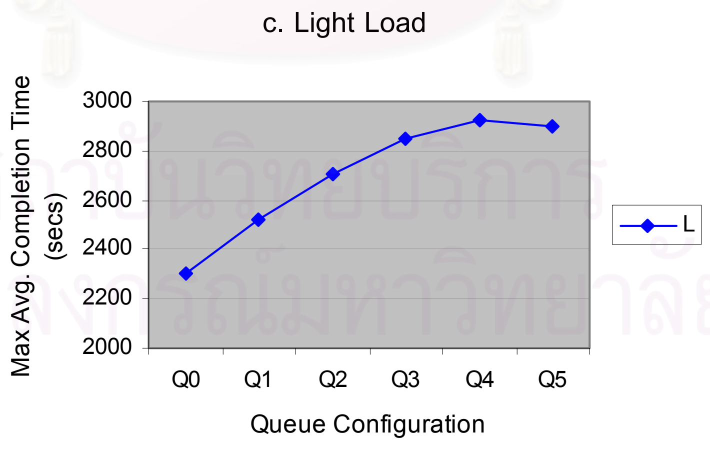
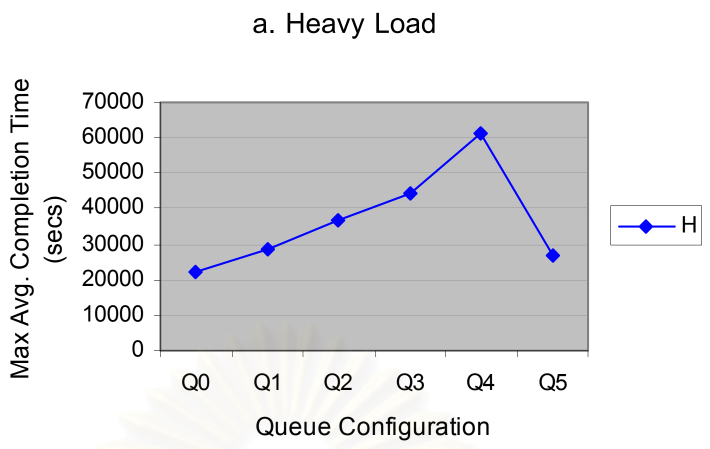
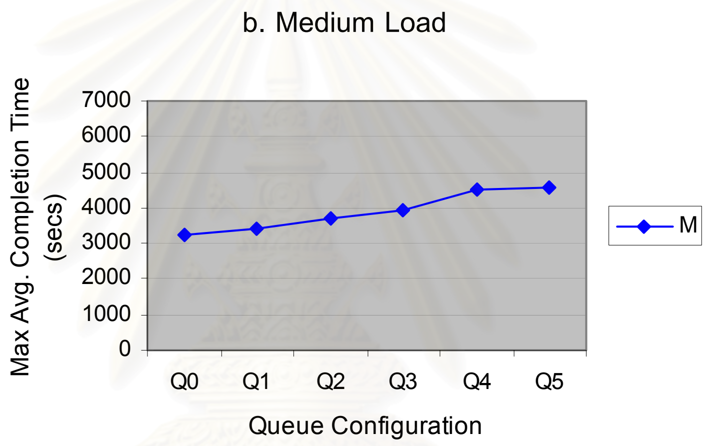


Figure 3.8: Maximum average completion time of MCT algorithm under Case 2





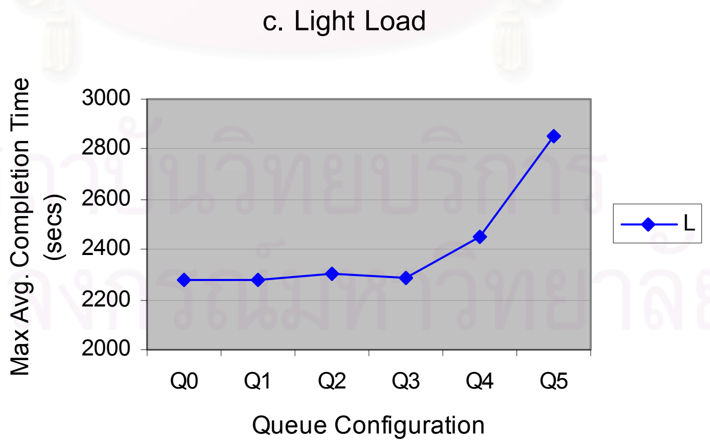
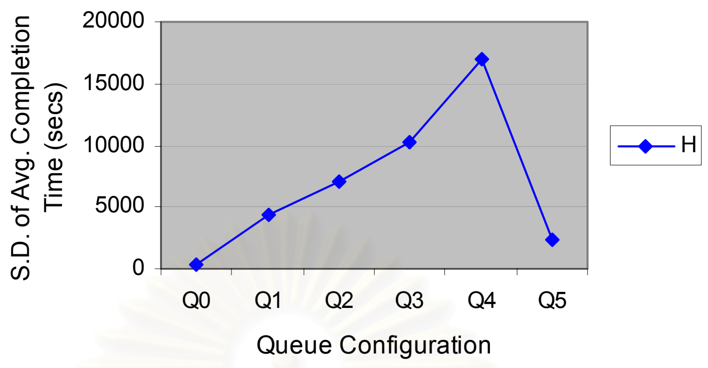
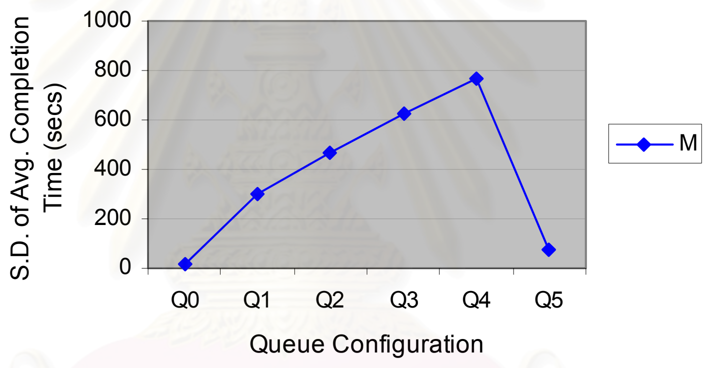


Figure 3.9: Maximum average completion time of MCT algorithm under Case 3

a. Heavy Load



b. Medium Load



c. Light Load

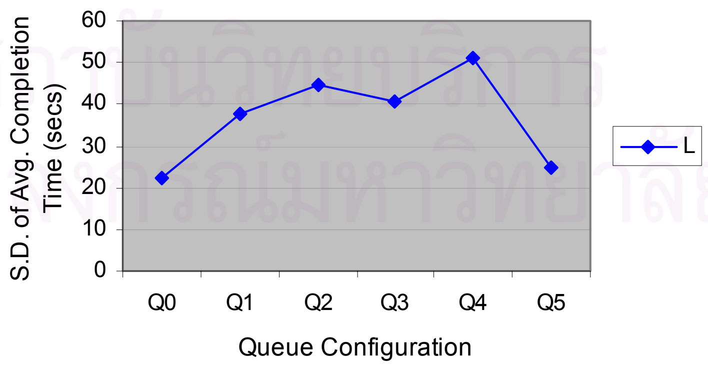
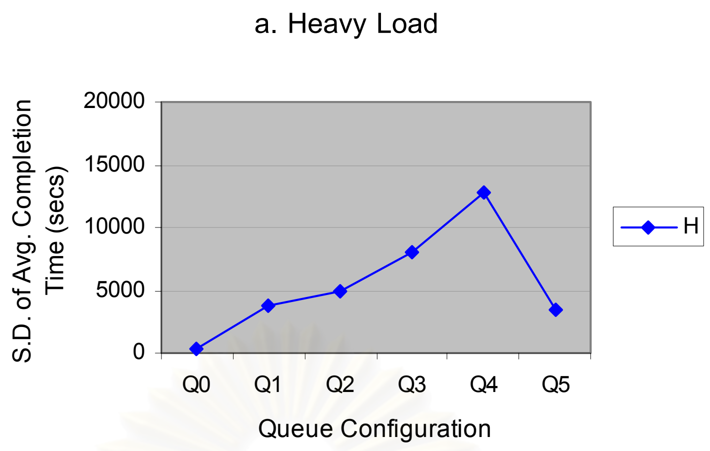
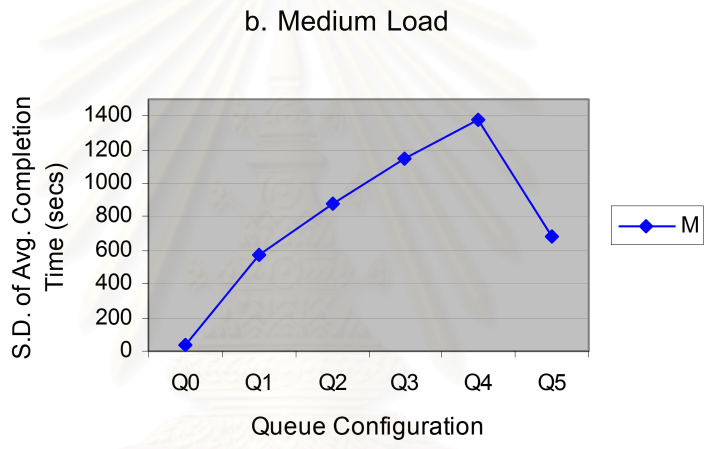


Figure 3.10: S.D. of average completion time of MCT algorithm under Case 1





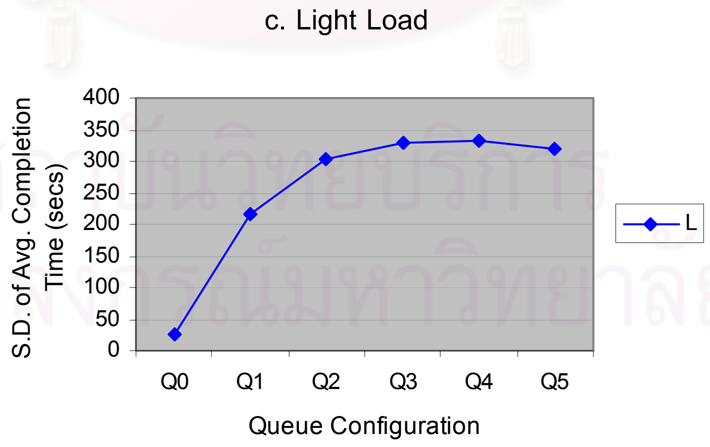
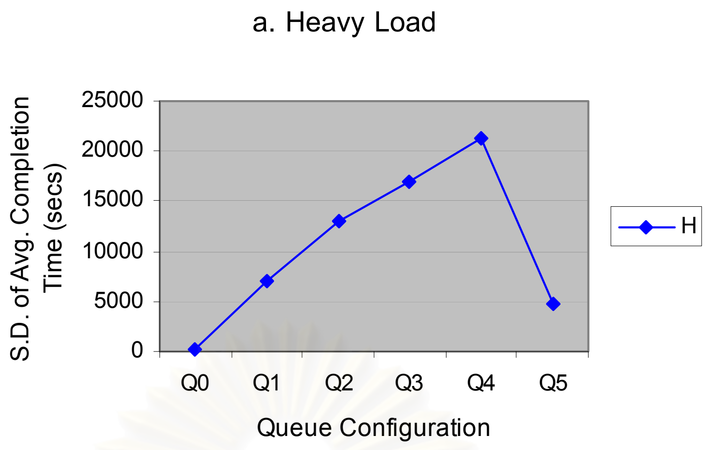
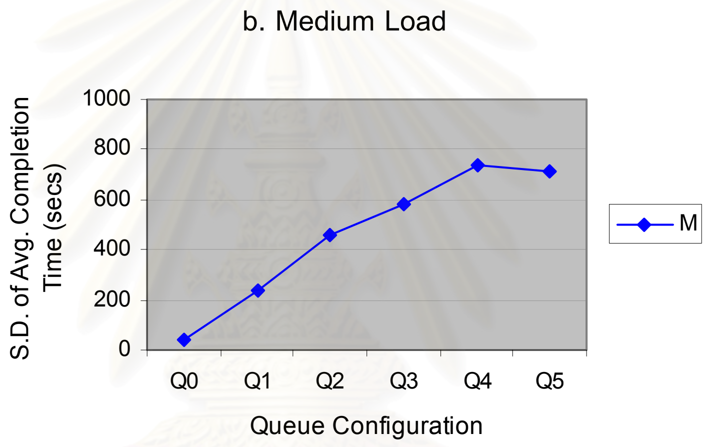


Figure 3.11: S.D. of average completion time of MCT algorithm under Case 2





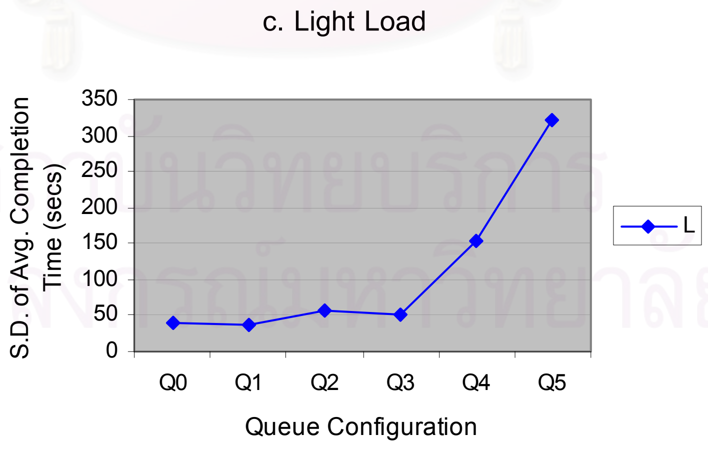


Figure 3.12: S.D. of average completion time of MCT algorithm under Case 3

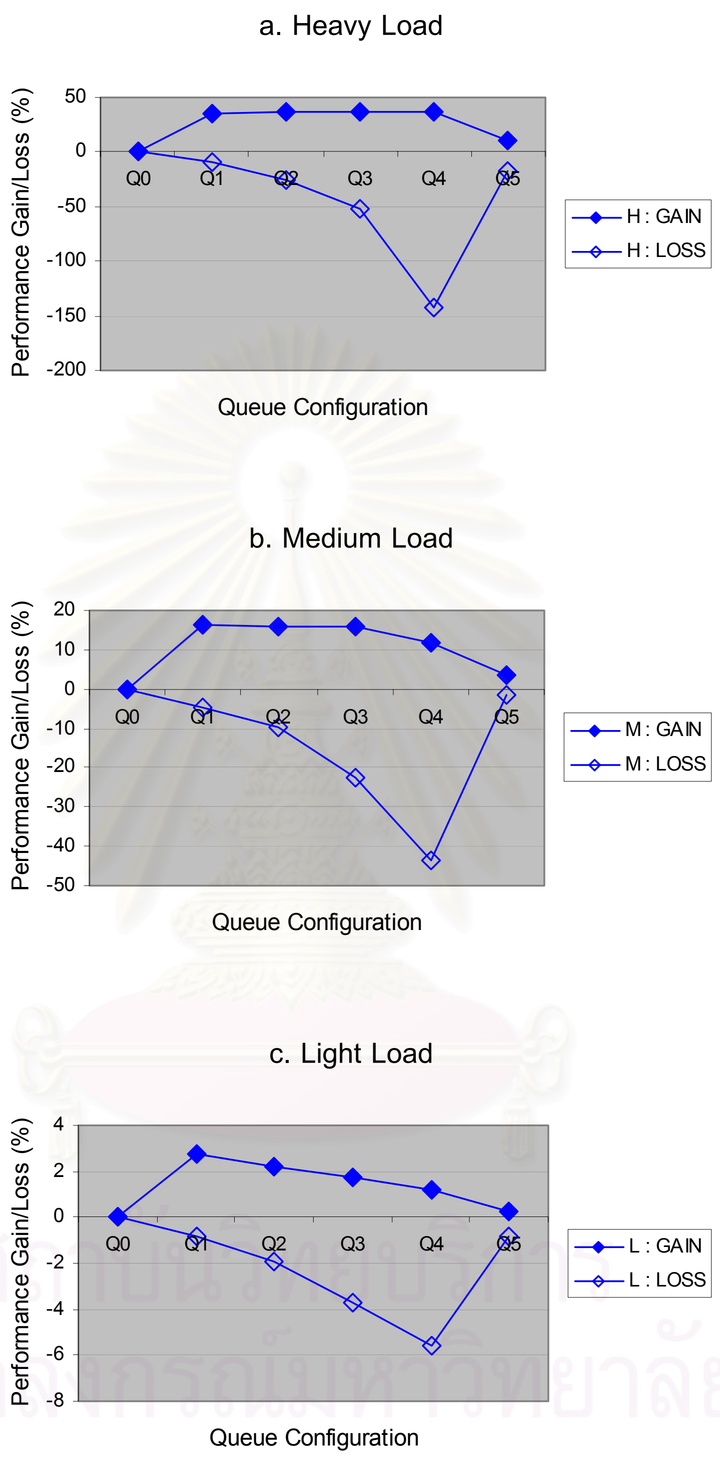


Figure3.13: Maximum gain and loss of completion time of MCT algorithm under Case 1

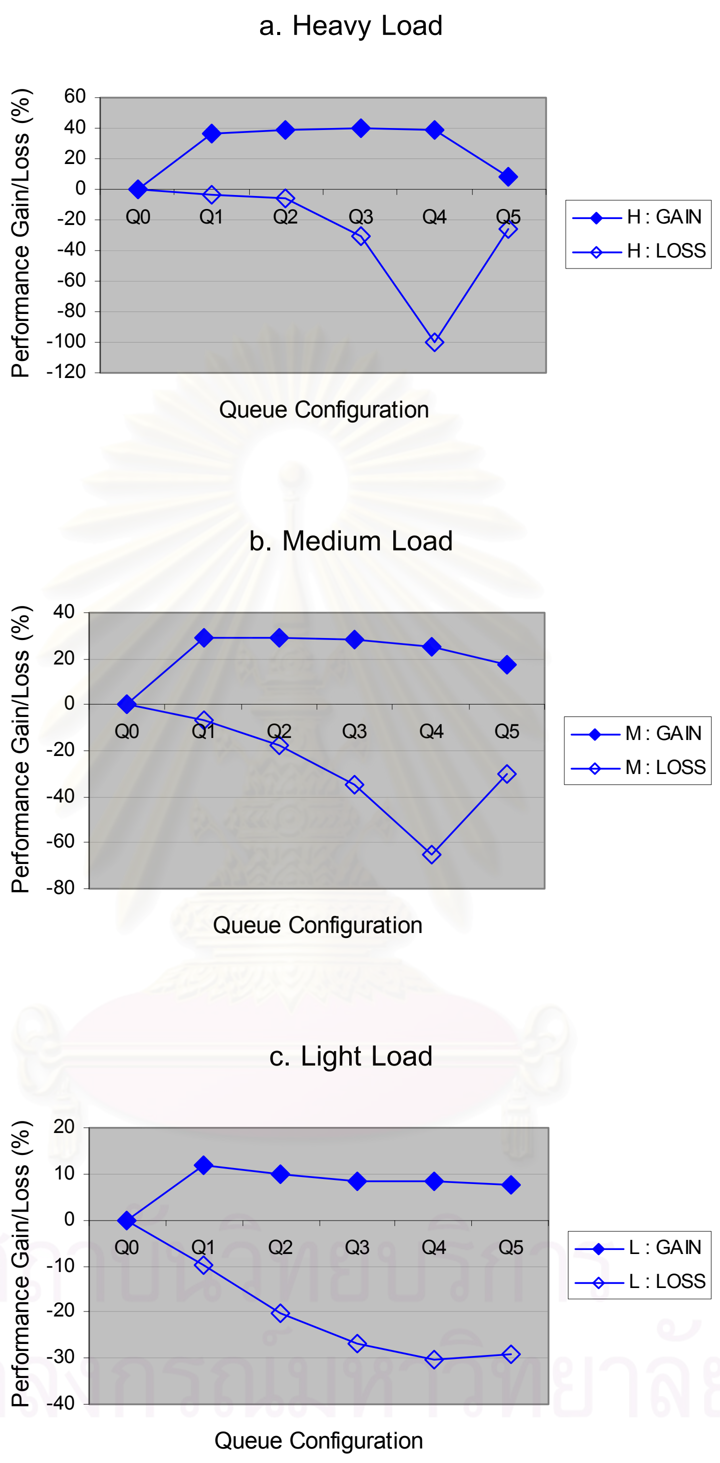


Figure3.14: Maximum gain and loss of completion time of MCT algorithm under Case 2

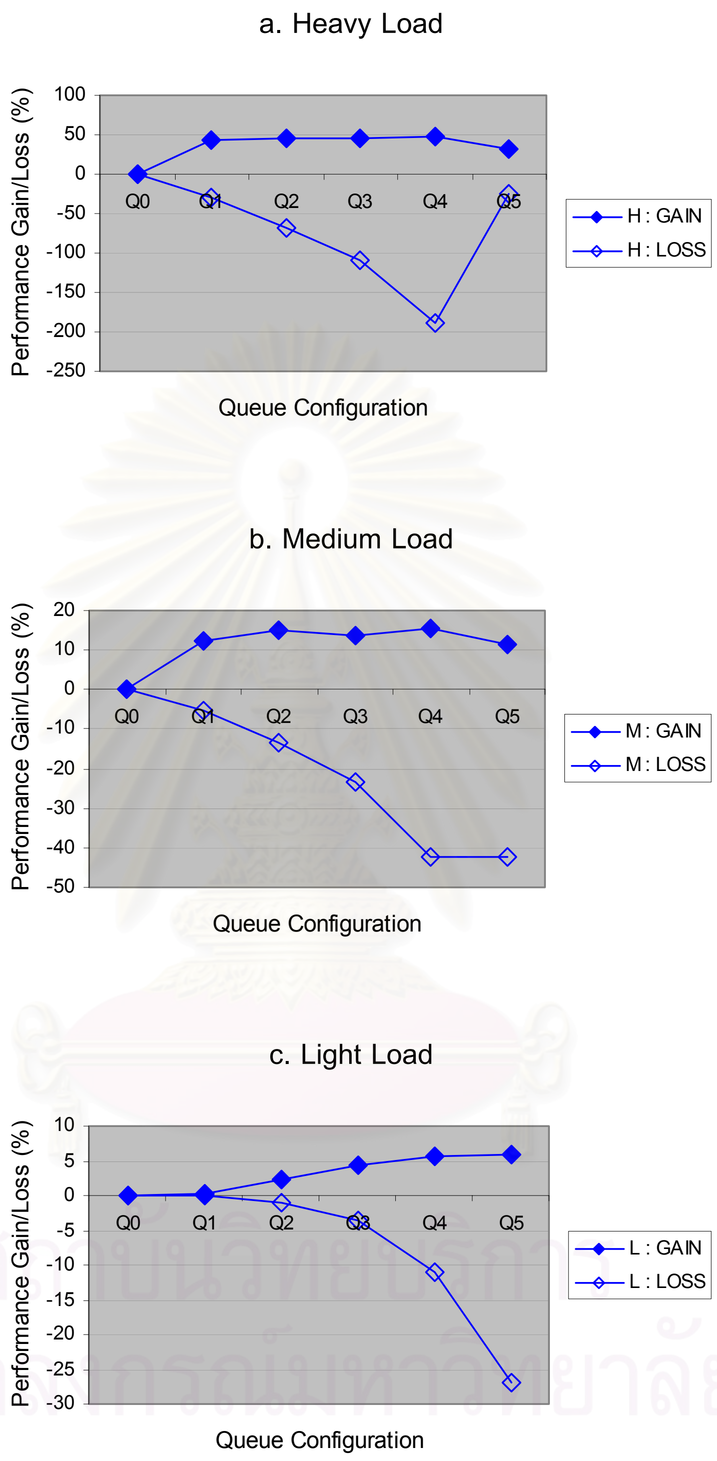


Figure3.15: Maximum gain and loss of completion time of MCT algorithm under Case 3

CHAPTER IV

THE PROPOSED ALGORITHM AND PERFORMANCE EVALUATION

In this chapter, a new algorithm for grid scheduling is proposed to overcome the performance impact that was identified in the previous chapter.

4.1 The Proposed Adaptive Site Selection Algorithm

At present, global policy enforcement mechanism in a grid environment has not been put into practice. Under the constraint that a grid scheduler cannot control local schedulers but only acts as a broker, it must be able to react to the behavior and performance of individual local schedulers.

As seen in the previous chapter, the local-job-first policy causes the completion times of remote jobs to be longer than predicted. A grid scheduler should be able to take into account the local policy before submitting jobs to local schedulers.

Therefore, this work proposes a new adaptive site selection algorithm. In this algorithm, the grid scheduler will select the site that gives the minimum of completion time multiplied by a weight value (Eq.1)

$$selected\_site = i \;\; if \; Tc_i \times W_i \; = \min_{1 \le i \le k} \left\{ Tc_i \times W_i \right\} \quad \text{(Eq.1)}$$

where *k*, Tc_i and W_i denote the number of sites, completion time of site *i* and weight associated with site *i*, respectively.

The weight is assigned by the grid scheduler and can be different for each site and each job. The purpose of the weight value is to add or reduce the possibility that the grid scheduler will select a target site according to some criteria. Several approaches for weight assignment are possible.

Our approach focuses on reducing the undesirable effects on single-queue sites. Therefore, the grid scheduler uses local priority policy as a criteria for site selection by making it more difficult to submit a job to another site which uses different priority policy. Another criteria is the fractions of sites that use the same policy. The criteria are converted into weight values.

The weight assignment can be divided into two cases by using computing power, which is the summation of the machine speeds.

In the first case, the computing power of single-queue sites is greater than or equal to the computing power of dual-queue sites. The weight of all sites is “1”. As a result, in this case, this algorithm has the same behavior as the MCT algorithm. This allows some sites to switch to use the local-job-first policy in order to reduce their completion times without causing too much trouble. The rational is that single-queue sites dominate the system and double-queue sites will have little effect.

In the second case, the computing power of single-queue sites is less than that of dual-queue sites. The weight of a site is “1” if the site uses the same policy as the job-owner site. Otherwise, its weight is set to a value greater than 1. As a result, the jobs are more likely to go to the site using the same policy. Jobs are sent to a remote site using a different policy only when the predicted completion time at that remote site is really attractive.

```
If  CP ( single-queue sites ) > CP (dual-queue sites) {
        $W_i$ = 1 for all $i$
} else {
        $W_i$ = 1 if  $P_i = P_o$
        $W_i$ = a ; a > 1  if $P_i \neq P_o$
}
```

Figure 4.1 Weight Assignment Algorithm

The logic of weight assignment can be described by the algorithm in Figure 4.1 which is based on the following definition. Function *CP (Group)* will evaluate the computing power of the specified group. W_i is response for the weight of site *i*. P_i is the policy of site *i*. P_o is the policy of the job-owner site.

4.2 Weight Value Selection

The value of weight (W_i = a) can vary from one site to another, and can even change over time. For simplicity and the scope of empirical study, the value of weight when it is greater than 1 will be set to a constant that is generally good. Standard deviation of average completion time can be used to identify the constant.

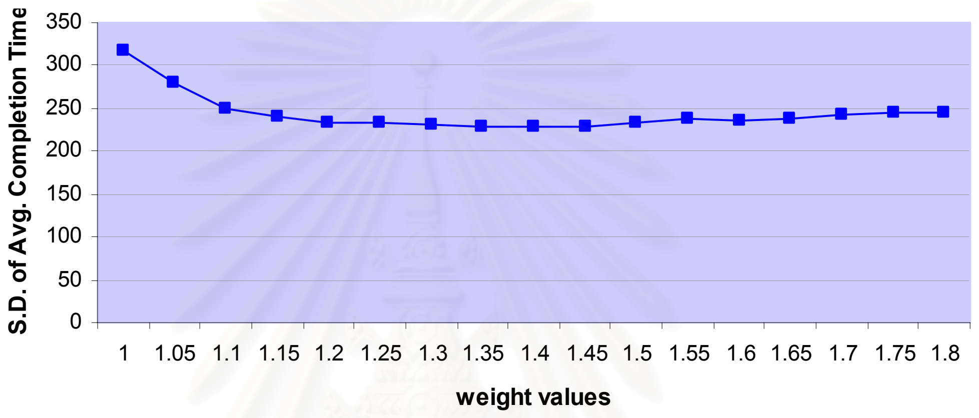


Figure 4.2: The relationship between standard deviation of Average Completion Time and constant weight values

The relationship between standard deviation of average completion times and the constant weight value of the medium load under case 1 are shown in Figure 4.2. The value "1.3" is chosen as a constant for the algorithm for every case and workload. It is one of the good values that make the standard deviation of average completion time reduce in this case. However, this weight value can be set to other constant values rather than 1.3. If this value is high, the probability that grid scheduler will send jobs across policy groups is reduced. If this value is low, the probability is increased.

4.3 Performance Evaluation of the Proposed Algorithm

Because the fraction of policy groups and the computing power of each site are used by the proposed algorithm to adjust the weight as described in section 4.1, the algorithm takes effect differently in each case. In Case 1, the algorithm takes effect on

Q3 and Q4. In Case 2, the algorithm will take effect sooner that is Q2, Q3 and Q4. Finally, In Case 3, the algorithm will take effect only in Q4.

The performance metrics to be considered are the same as metrics presented in the section 3.5. First, the average completion time when the proposed algorithm is applied are depicted in Figure 4.3-4.5. Second, the maximum of average completion time are depicted in Figure 4.6-4.8. Third, the standard deviation of average completion time of before and after applying the proposed algorithm are depicted in Figure 4.9-4.11. Finally, the maximum gain and loss of average completion times are depicted in Figure 4.12-4.14. Label H (NEW), M (NEW) and L(NEW) represent heavy load, medium load and light load respectively for the proposed algorithm. Label H (MCT), M (MCT) and L (MCT) represent heavy load, medium load and light load respectively for the MCT algorithm.

4.3.1 The Average Completion Times of the Proposed Algorithm

The relationship between average completion times and queue configurations is depicted in Figure 4.3-4.5.

The results show that the average completion time of individual sites are less significantly affected. The average completion times of single-queue sites in the queue configurations affected by the proposed algorithm are reduced. For example, in Q3 of Figure3.4a (MCT algorithm), the average completion time of site A, B and C which are dual-queue sites 14,283, 15,196 and 14,718 seconds respectively, and the average completion time of site D and E which single-queue site are 33,775 and 33,371 seconds respectively. When the proposed algorithm applied, the average completion time of site A, B, C are increased to 21,657, 21,652 and 23,038 respectively and the average completion time of sites D and E are reduced to 26,203 and 26,119 respectively

Only one characteristic before applying the proposed algorithm still remain. Sites in dual-queue mode, the average completion times likely correspond to their machine speed in medium load and light load of all cases.

4.3.2 The Maximum Average Completion Times of the Proposed Algorithm

The relationship between queue configuration and maximum average completion time of all sites is depicted in Figure 4.6-4.8.

For Case 1 and Case 2 (Figure 4.6, 4.7), the proposed algorithm always performs better than the MCT algorithm. For example, in Q4 of Figure 4.6a and Figure 4.7a, the maximum of average completion times is reduced by 44% and 35% respectively. However, in Q4 of Case 3 (Figure 4.8), the maximum average completion times is 45% and 14% better for heavy load and medium load but only 5% worse for light load. Therefore, the maximum completion times are generally reduced by the proposed algorithm.

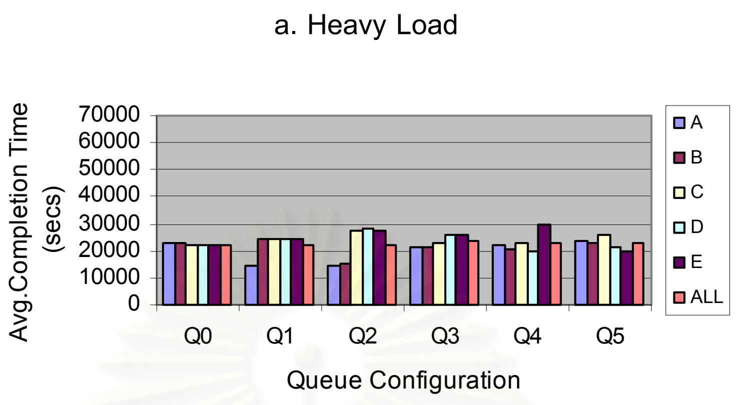
4.3.3 The Standard Deviation of Average Completion Times of the Proposed Algorithm

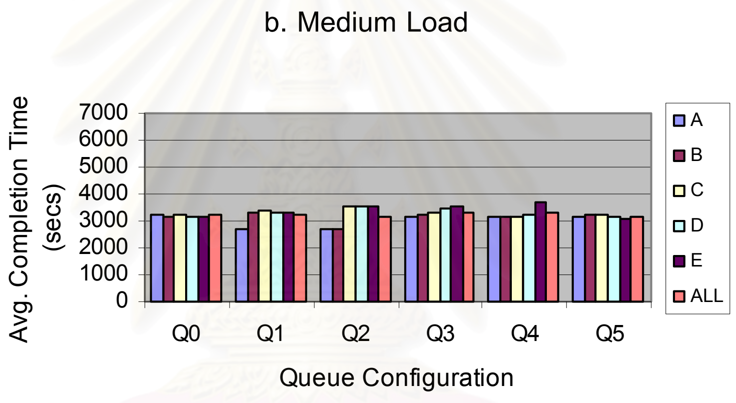
The standard deviations of average completion times in each queue configuration are depicted in Figure 4.9-4.11. All of them are also improved by proposed algorithm in the same manner as the maximum completion time except the Q4 of Figure 4.11 which increases 20 seconds.

4.3.4 The Maximum Gain and Loss of Average Completion Times of the Proposed Algorithm

The maximum gain and loss of average completion times in a queue configuration are depicted in Figure 4.12-4.14. Compared to the results obtained from the MCT algorithm, the gap between the maximum gain and the loss is reduced in most cases and workloads. For example, in Q4 of Figure 4.12a, the maximum gain is reduced from 37% to 12%, and the maximum loss is reduced from 142% to 36%. However, in Q4 of Figure 4.14a, only the maximum loss is increased from 11% to 14%.

In summary, the new algorithm allows some sites to exploit the local-job-first policy and get some performance gain while other sites experience some performance loss. When the fraction of the policy groups reaches a threshold, the adaptive algorithm tries to reduce the performance loss, with the reduction of the performance gain as a tradeoff.





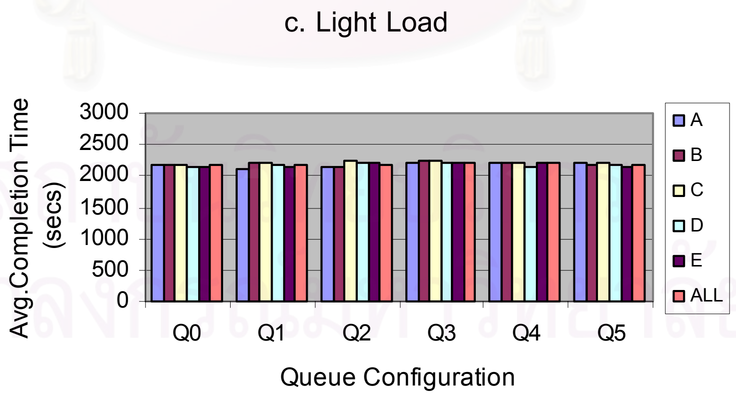
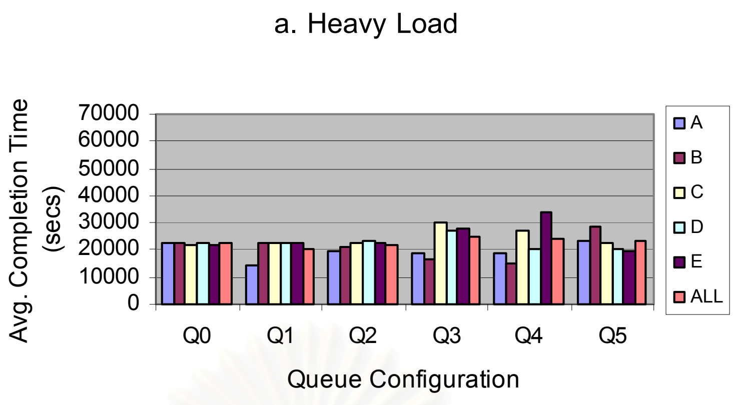
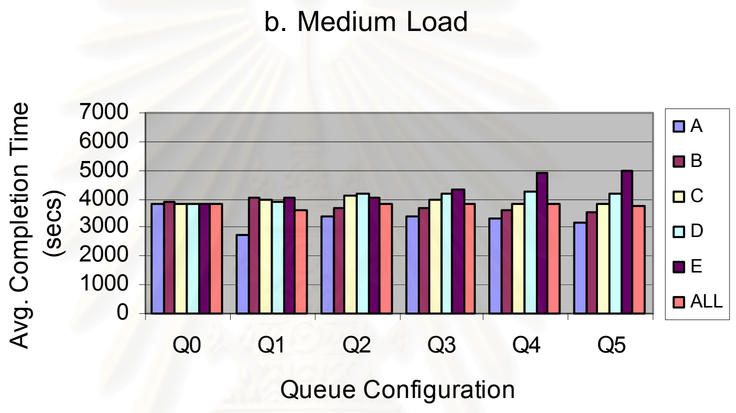


Figure 4.3: Average completion time of before and after applying the proposed algorithm under Case 1





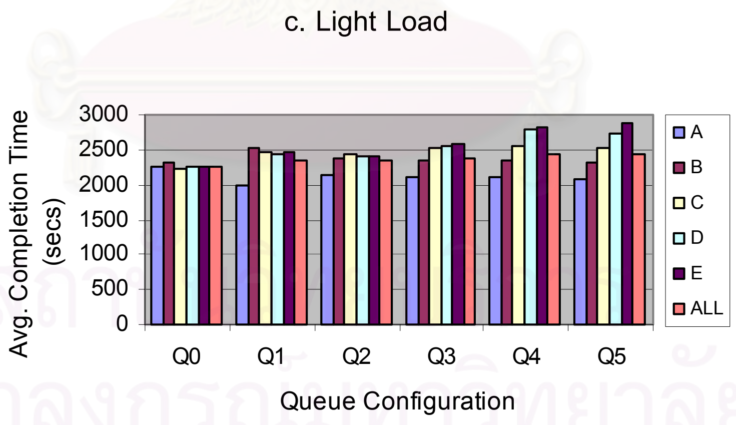
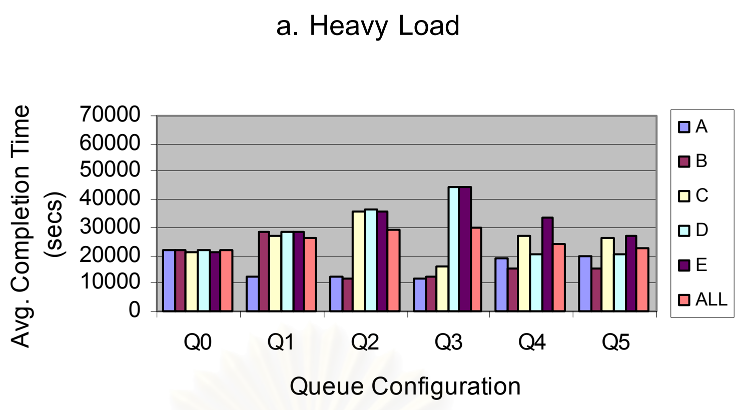
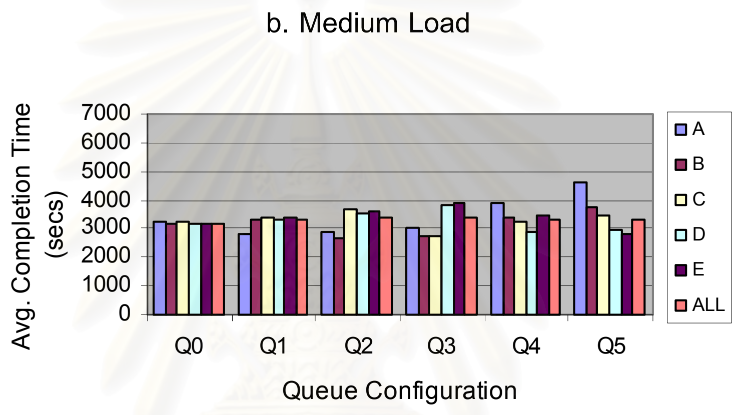


Figure 4.4: Average completion time of before and after applying the proposed algorithm under Case 2





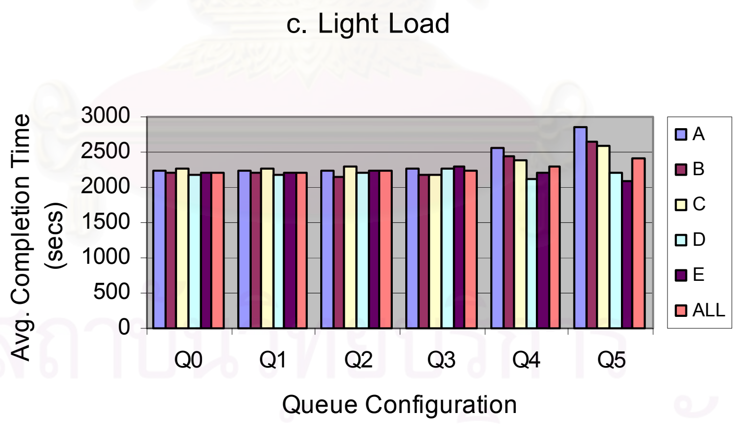


Figure 4.5: Average completion time of before and after applying the proposed algorithm under Case 3

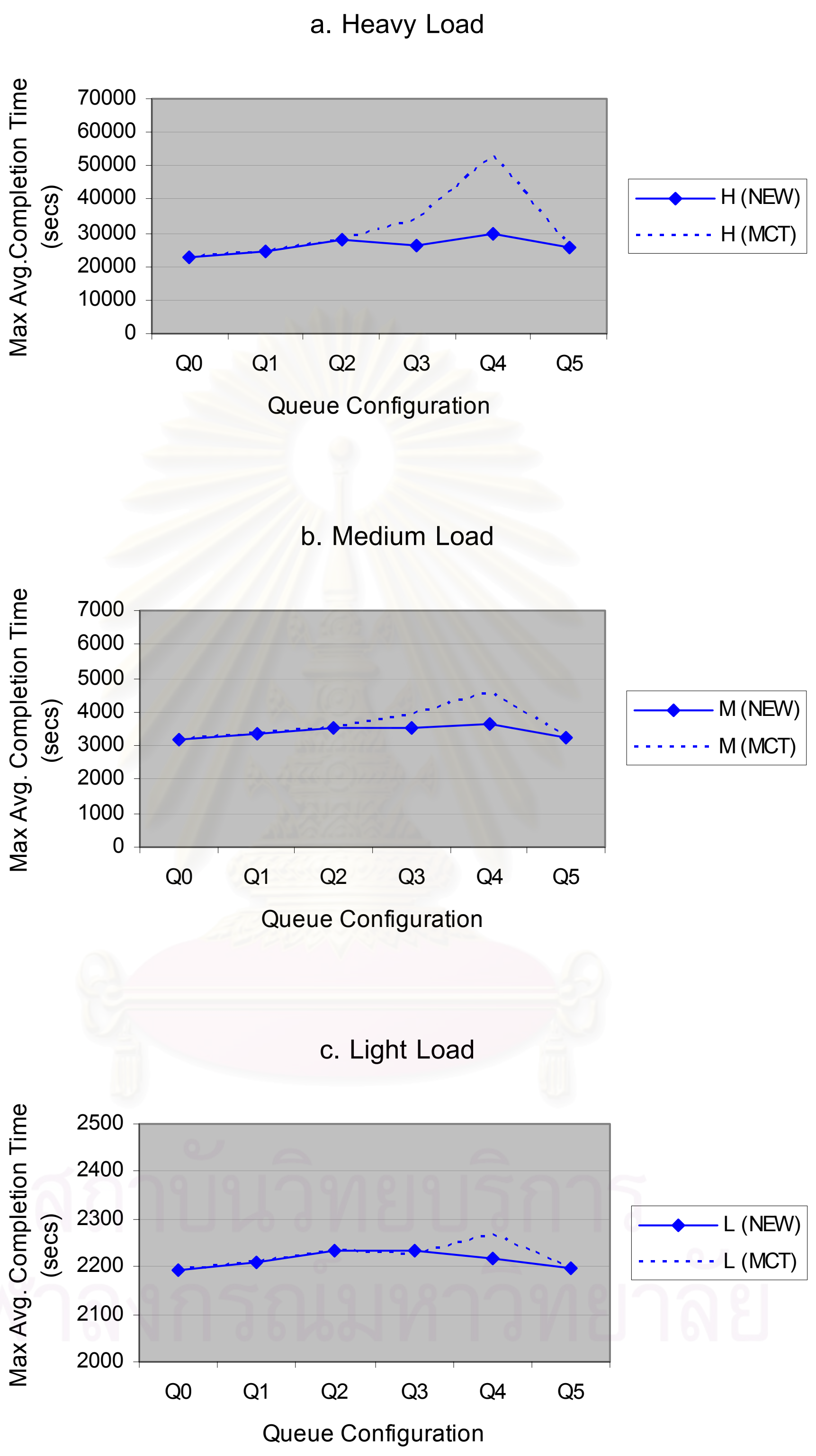
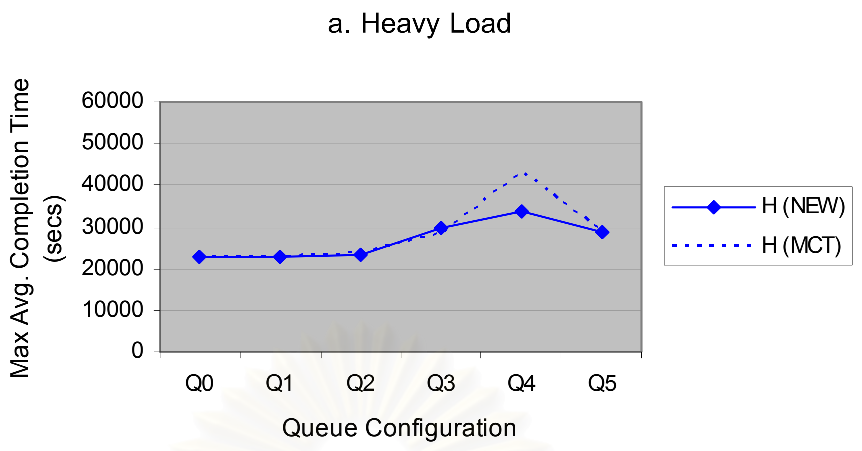
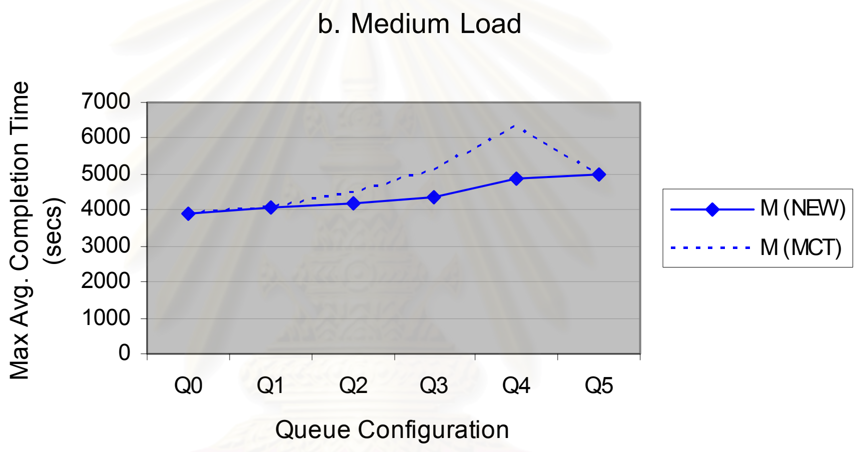


Figure 4.6: Maximum average completion time of before and after applying the proposed algorithm under Case 1





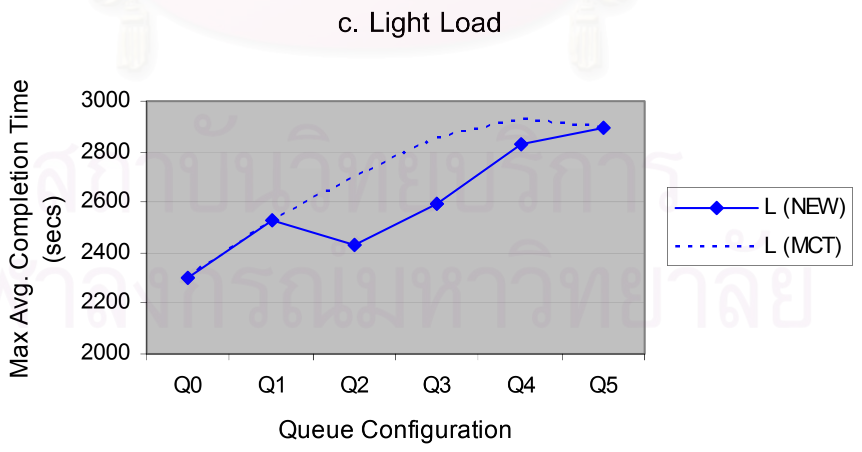


Figure 4.7: Maximum average completion time of before and after applying the proposed algorithm under Case 2

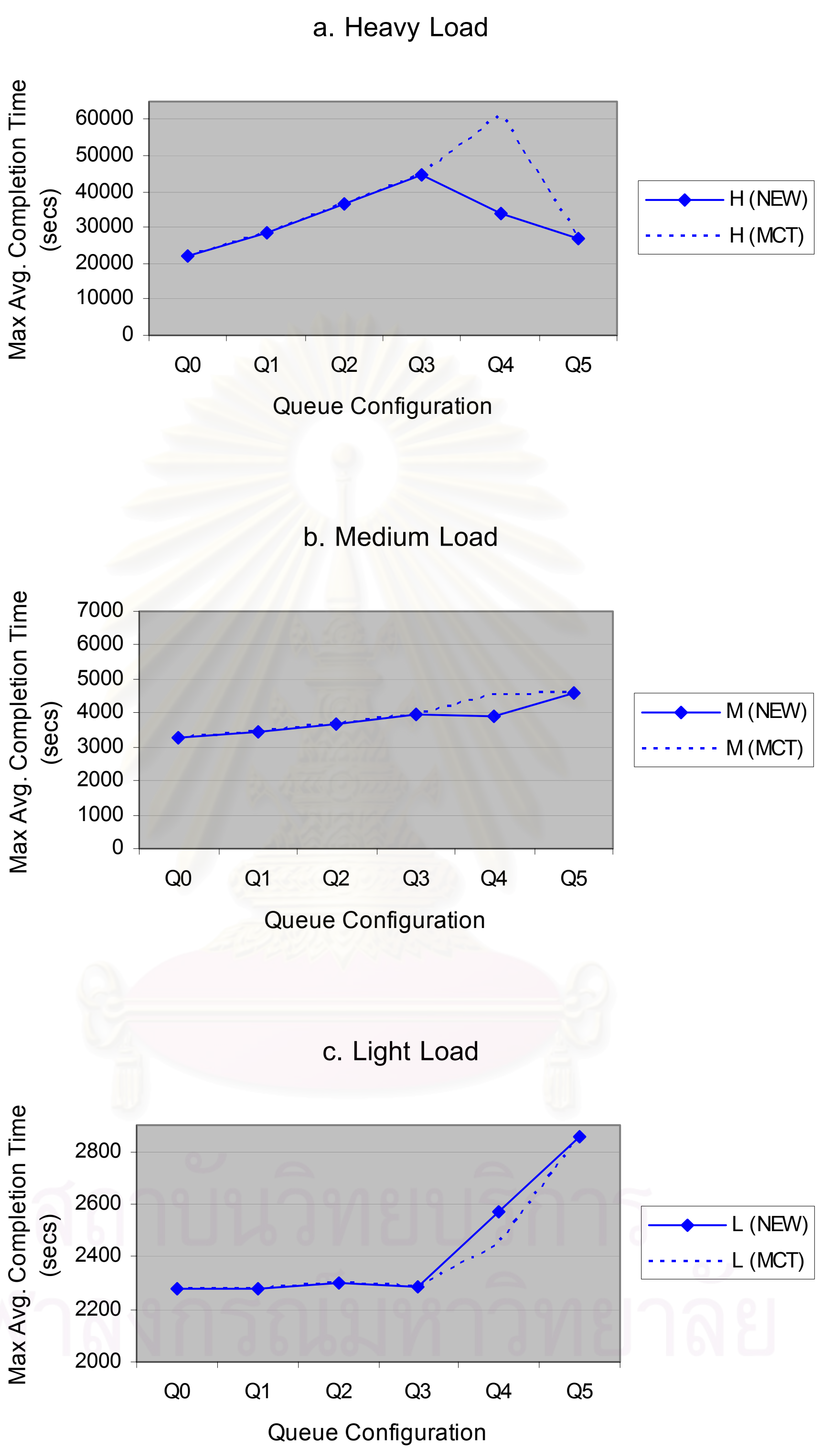


Figure 4.8: Maximum average completion time of before and after applying the proposed algorithm under Case 3

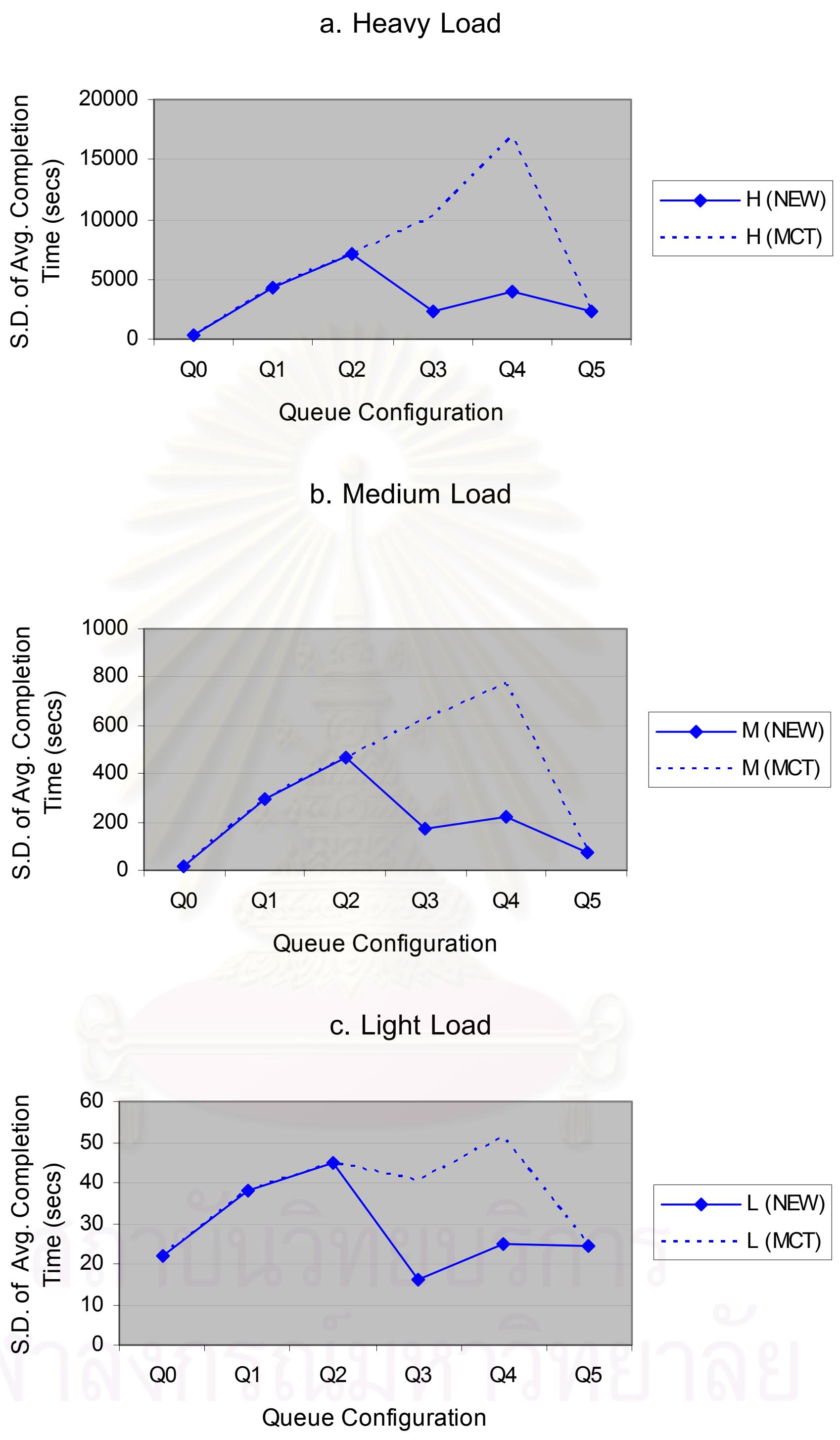


Figure 4.9: S.D. of average completion time of before and after applying the proposed algorithm under Case 1

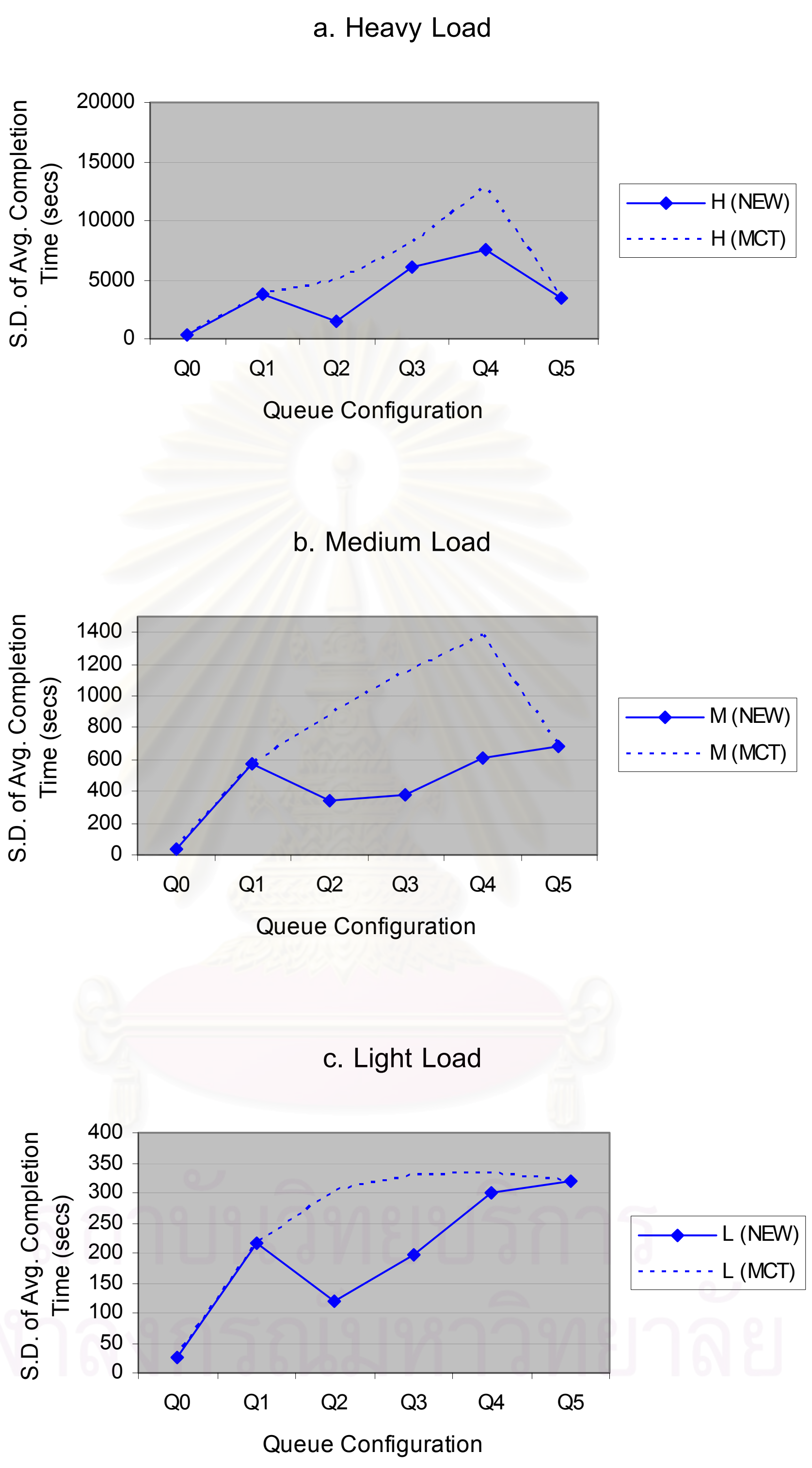


Figure 4.10: S.D. of average completion time of before and after applying the proposed algorithm under Case 2

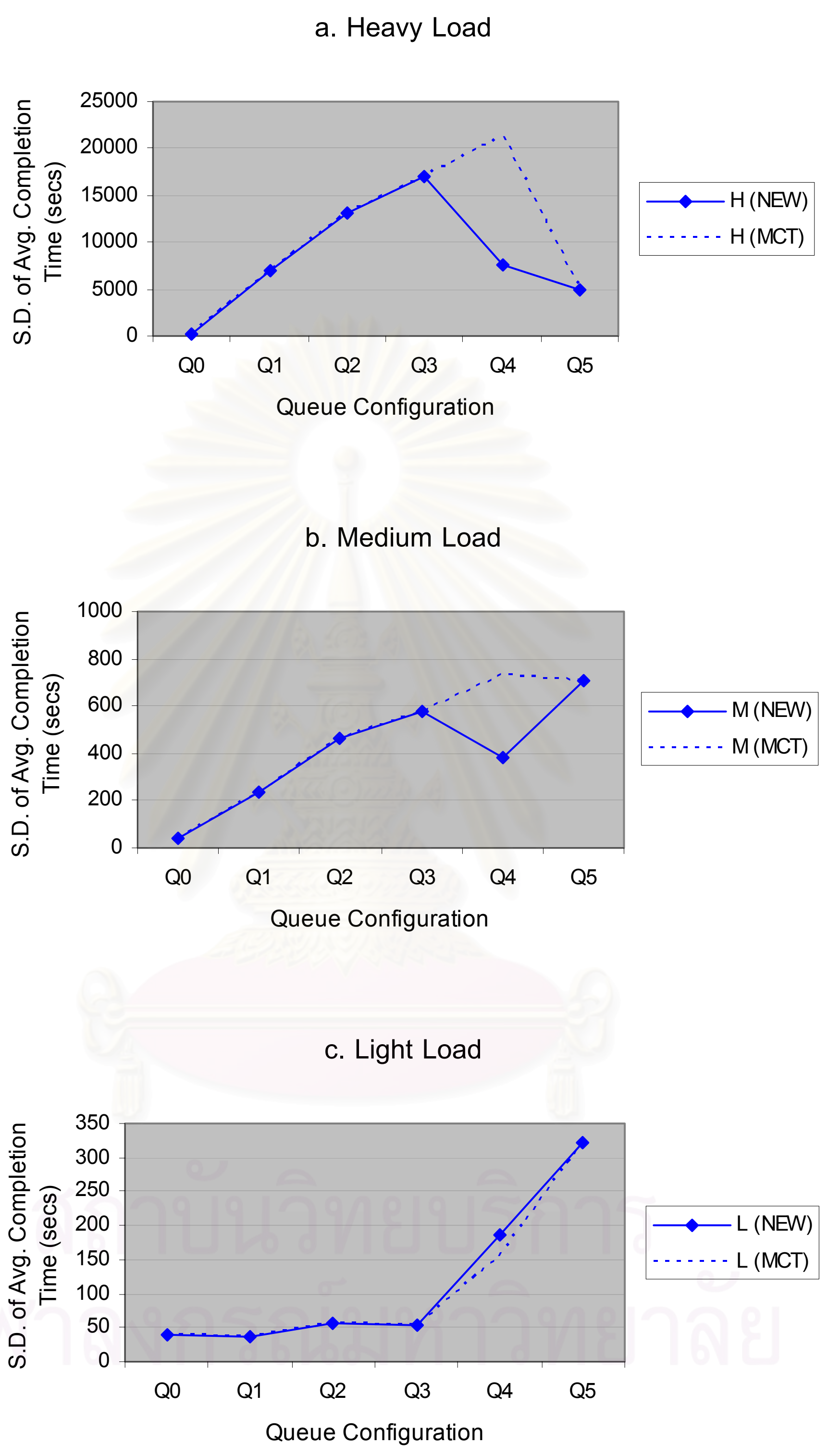
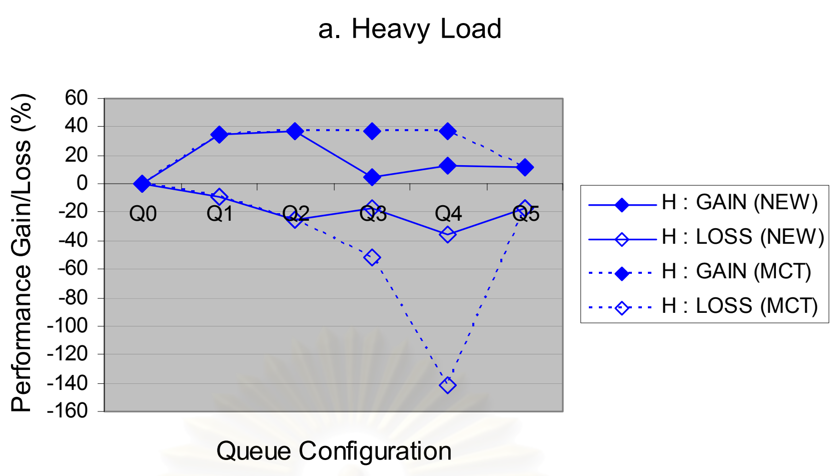
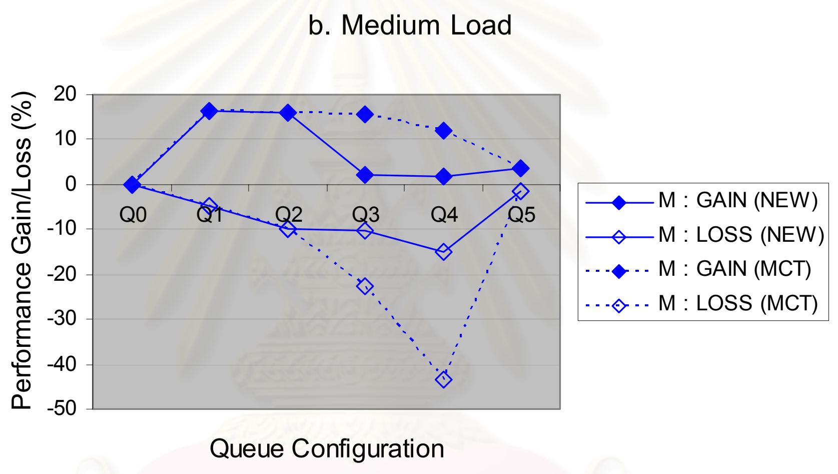


Figure 4.11: S.D. of average completion time of before and after applying the proposed algorithm under Case 3





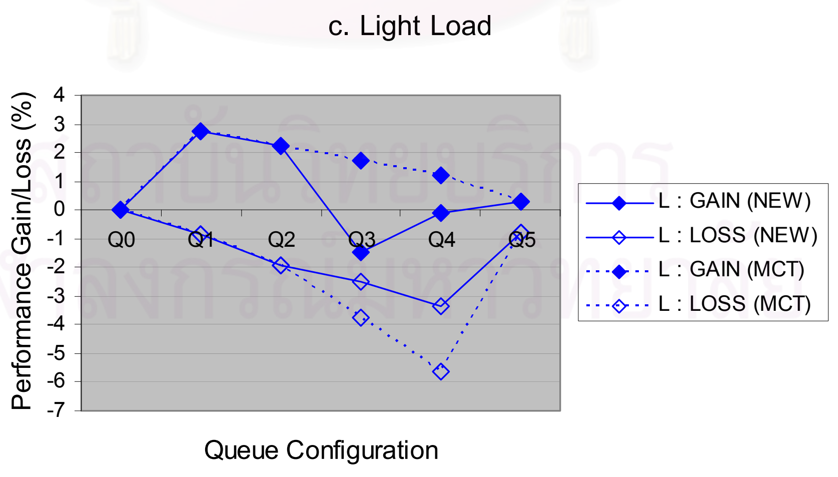


Figure 4.12: Maximum gain and loss of completion time of before and after applying the proposed algorithm under Case 1

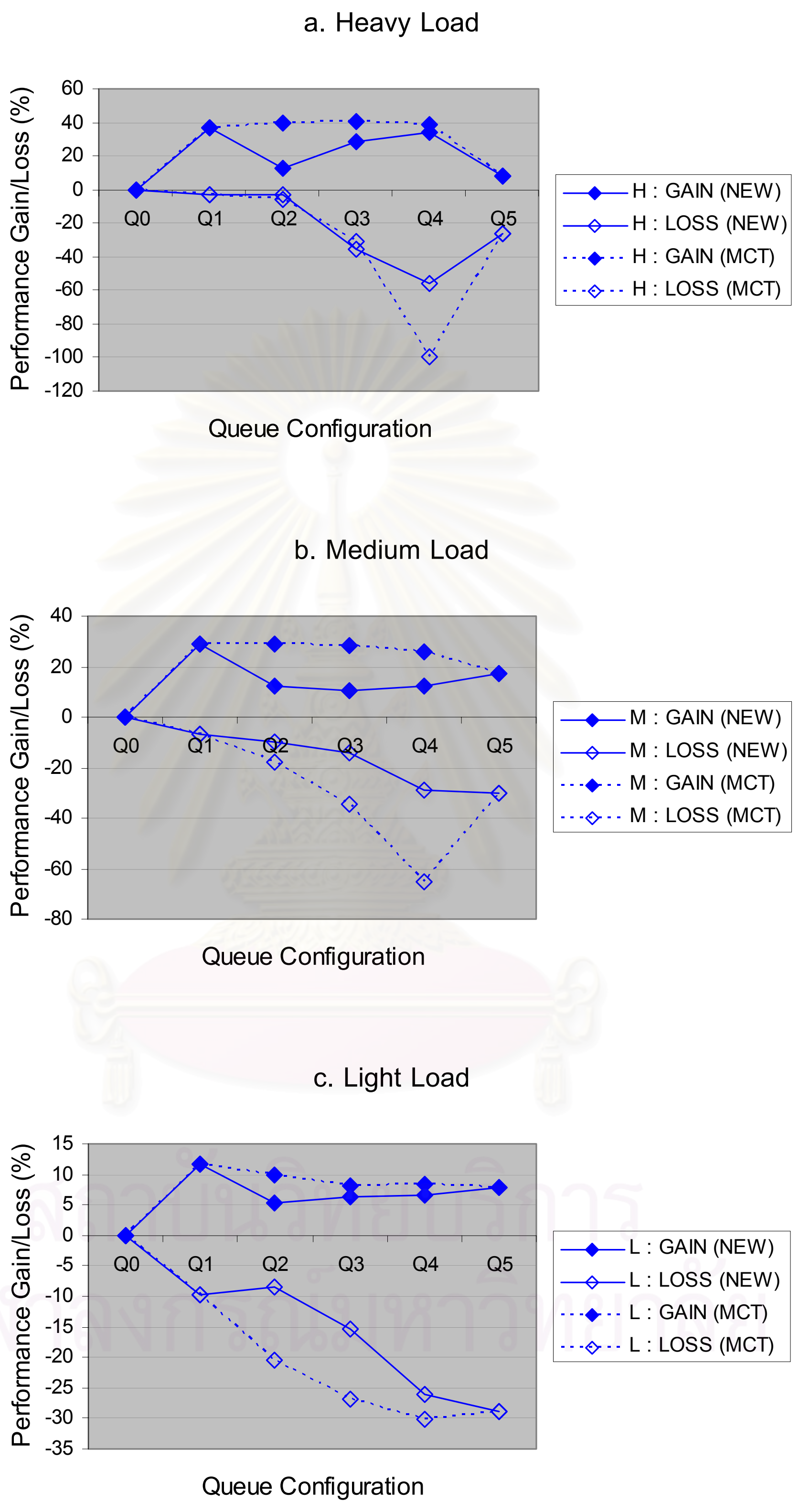
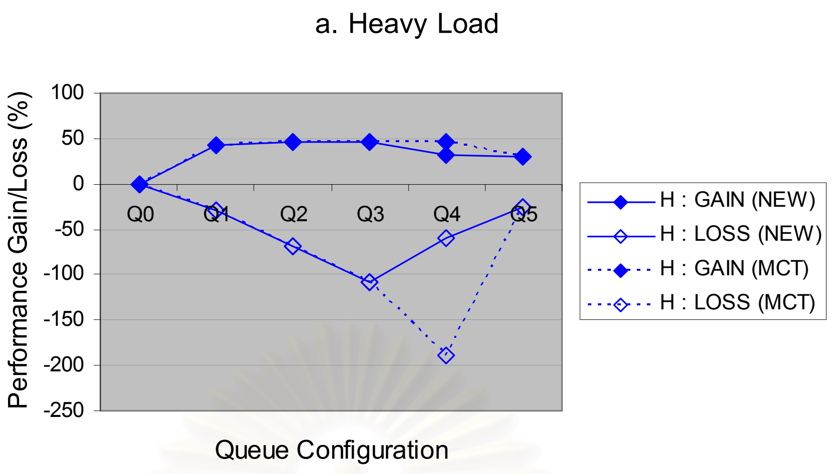
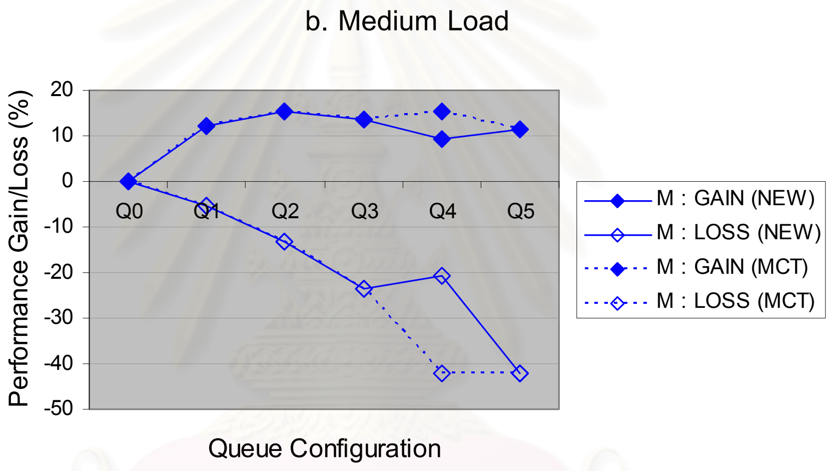


Figure 4.13: Maximum gain and loss of completion time of before and after applying the proposed algorithm under Case 2





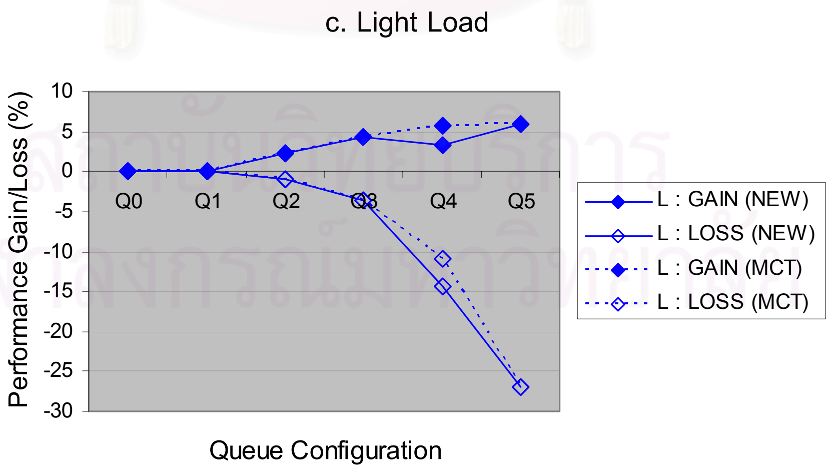


Figure 4.14: Maximum gain and loss of completion time of before and after applying the proposed algorithm under Case 3

CHAPTER V

IMPLEMENTATON

This chapter presents the requirements and methods for implementing the proposed grid scheduling algorithm. A simple implementation on a grid portal demonstrates the feasibility of a real implementation.

5.1 Implementation Requirements

The proposed algorithm can be implemented into existing grid schedulers. However, there are some requirements to be met including:

- When queried by the grid scheduler, a local scheduler must provide information for the grid scheduler to determine whether the local scheduler uses the local-job-first policy.
- When queried by the grid scheduler, a local scheduler must provide information for the grid scheduler to determine how long it will take to complete a job if the job is submitted to that site.
- The grid scheduler must be able to handle heterogeneity regarding to machine speeds and execution time estimations.

These requirements can be met in several ways. If adding extra capabilities to local schedulers is possible, then it is preferable to make the local scheduler be able to answer the queries from the grid scheduler. However, sometimes it is hard to do so and it is left to the grid scheduler to figure out needed information from the data that is usually provided by unmodified local schedulers. For example, most available schedulers such as PBS and the like have a command to show the list of jobs in the queue(s) and their status including expected execution times. Grid Information Services (GIS) or Monitoring and Directory Service (MDS) of Globus can also be used for grid sites to report their local scheduling policies. The estimated execution time must be adjusted accordingly to the performance of different sites.

5.2 Implementation on a Grid Portal

SPACE Grid Portal [35] is a web interface to the grid to provide an easy-to-use interface so that user can work on a grid system more easily and to maintain basic functions Grid have already serviced. It was developed to support Computational Chemistry research works at Department of Chemistry, Faculty of Science, Chulalongkorn University. Globus is installed on each cluster in the grid. All local schedulers are PBS/Maui. Other software includes GridPort[36] and MyProxy[37]. The main applications to be run on the grid is Gaussian98 [38] and AutoDock [39].

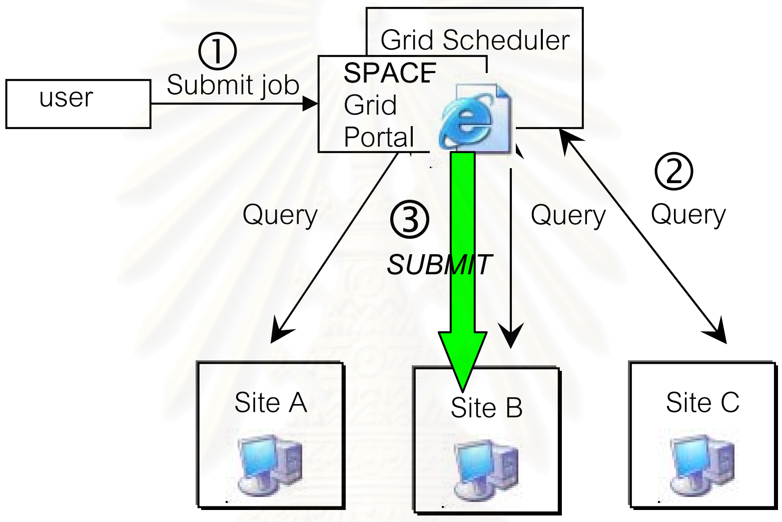


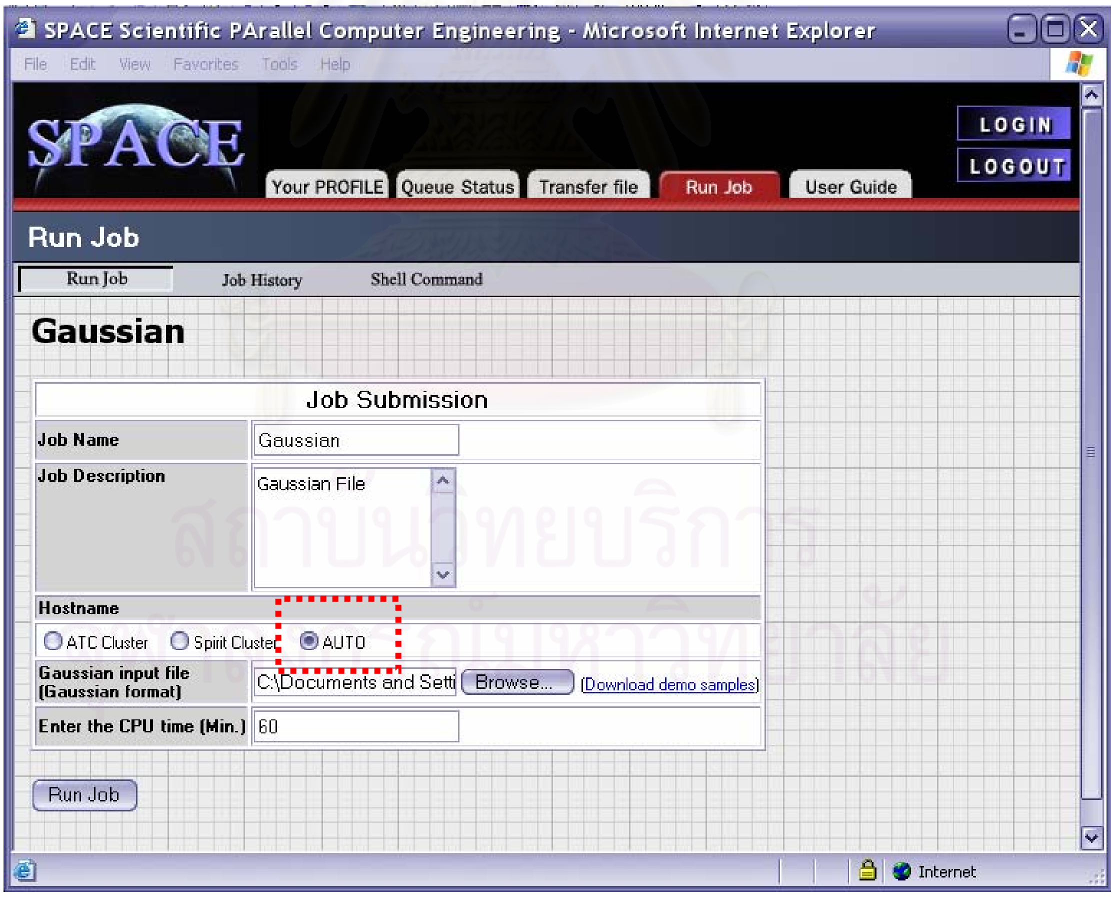
Figure 5.1: Flow chart of job submission through SPACE Grid Portal

The grid scheduling algorithm has been implemented in SPACE Grid Portal as Perl script files. Therefore, the grid scheduler accepts only jobs submitted at SPACE grid portal. The mechanism of submitting a job to SPACE Grid Portal is shown in Figure 5.1. First, a user submits a job to SPACE grid portal through the web interface. The execution time of the job must be estimated and specified by the user. Second, the job description is passed to the grid scheduler. The grid scheduler queries the queue status of each site and calculates the completion time of the job on each site. Finally, the grid scheduler will select the suitable site by using the proposed algorithm.

The example of submission page for Gaussian application is shown in Figure 5.2. When the user chooses the hostname “AUTO”, the proposed algorithm will be used to select a suitable site automatically.

5.3 Limitation

- In the current implementation, the estimation of execution time specified by a user is not adjusted with respect to machine speeds since the execution time is estimated by the user. Therefore, the performance of the algorithm depends on the accuracy of the estimated completion time.
- All cluster information must be specified to configuration files. The cluster information includes the number of nodes on a cluster and type of policy.



Job Submission	
Job Name	Gaussian
Job Description	Gaussian File
Hostname	
ATC Cluster Spirit Cluster [✓ AUTO]	
Gaussian input file (Gaussian format)	C:\Documents and Setti
Enter the CPU time (Min.)	60

Figure 5.2: SPACE Grid portal: Run Job page

CHAPTER VI

CONCLUSION

6.1 Summary

In a computational grid, the local schedulers which reside in grid sites cooperate with but are not controlled by a grid scheduler. As a result, the local schedulers have the freedom to apply their own scheduling policies. Some sites may apply a policy in favor of local jobs by giving a higher priority to local jobs than remote jobs.

This thesis has evaluated the performance impact on global grid scheduling when some sites apply this kind of policy and to propose a new adaptive site selection algorithm to cope with such effects.

Two different priority-based algorithms are employed at grid sites. The first policy is to treat remote jobs and local jobs equally. This can be implemented by putting local jobs and remote jobs in the same queue (single-queue model). The other policy, local-job-first policy, gives higher priority on local jobs than remote jobs in order to speed up the local jobs. This can be implemented by using two separate queues (dual-queue model).

Minimum completion time (MCT) approach is used as a benchmark for grid scheduling algorithms. The simulation results demonstrate three important characteristics. Firstly, dual-queue sites always have better average completion time than single-queue sites. Therefore, switching from the single-queue model to dual-queue model reduces the average completion time of the site but increases the average completion times of all other sites, especially the remaining single-queue sites. Secondly, as the fraction of dual-queue site increases, the difference between average completion time of single-queue sites and dual-queue sites increase. This difference is greater when the workload is heavier. Therefore, the last site that remains using the single-queue will suffer most. Finally, the average completion times among dual-queue sites correspond to their machine speeds. The sites that have higher speed will have better average completion times. However, the average completion times among single-queue sites are not affected by the machine speed. Therefore, the completion times of all single-queue sites are nearly equal.

In order to reduce the performance impact on single-queues site, a new adaptive site selection algorithm for grid scheduling has been proposed. In its general form, the algorithm applies a weight factor onto the cost function associated to each site in order to adjust the probability of assigning a job to the site. In this algorithm, the grid scheduler examines the local priority policies, fractions of sites that use the same policies and computing powers of all sites before submitting the jobs. The proposed algorithm allows some sites to exploit the local-job-first policy and get some performance gain while other sites experience tolerable performance loss. When the fraction of the dual-queue sites reaches a threshold, the proposed algorithm reduces the probability of submitting a job to the site that uses a different policy. Therefore, jobs are more likely go to the sites using the same policy.

The results of the proposed algorithm show that when the fraction of the dual-queue sites exceeds a threshold, the difference between the average completion times of single-queue sites and dual-queue sites is reduced. The average completion times of single-queue sites are significantly reduced, whereas the average completion times of dual-queue sites are slightly increased. Therefore, the proposed algorithm reduces the performance loss on single-queue sites with little reduction in performance gain as a tradeoff. This promotes more fairness in grid scheduling. In addition, the proposed algorithm can effectively work on various fractions of dual-queue sites and load conditions, especially when the load is heavier.

The work also suggests that a preferable situation is when a site may temporarily switch to the single-queue model when needs arise and not too many sites in the grid work in that mode. In the other words, the fraction of dual-queue sites should be limited within a threshold of the proposed algorithm.

The proposed grid algorithm has been implemented into the SPACE grid portal to demonstrate a real implementation. This grid scheduler accepts Gaussian and Autodock jobs submitted at the portal.

6.2 Future Work

There are many possible extensions to the study in this thesis, as follow.

- Introducing a new weight assignment method

In weight assignment in the proposed algorithm, the weight is either 1 or a constant value greater than 1. The new weight assignment may introduce more dynamic weight assignment by increasing or decreasing the weight value assigned each individual site according to the performance of the site at that time.

- Evaluate on more realistic grid environments

Evaluation and analysis of policy-based grid scheduling algorithms in larger and more heterogeneous grid environments.

6.3 Final Remark

The work in this thesis has been published in [40].

References

1. Foster, I. and Kesselman, C. The Grid: Blueprint for a Future Computing Infrastructure. Morgan Kaufmann, 1999.
2. Shan, H. and Oliker, L. Job Superscheduler Architecture and Performance in Computational Grid Environments. Proceedings of Supercomputing, 2003.
3. Foster, I., Kesselman, C., and Tuecke, S. The Anatomy of the Grid - Enabling Scalable Virtual Organizations. Lecture Notes in Computer Science Publisher 2150 (2001).
4. Baker, M., Buyya, R. and Laforenza, D. The Grid: International Efforts in Global Computing. Proceedings of Advances in Infrastructure for Electronic Business, Science,and Education on the Internet (SSGRR 2000), 2000.
5. Foster, I. and Kesselman, C. Globus: A Metacomputing Infrastructure Toolkit. The International Journal of Supercomputer Applications and High Performance Computing 11 (1997): 115-128.
6. Foster, I. and Kesselman, C. The Globus Project: A Status Report. Proceedings of the Seventh Heterogeneous Computing Workshop, 1998.
7. Raman, R., and Solomon, M. Matchmaking: Distributed Resource Management for High Throughput Computing. Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, 1998.
8. Romberg, M. The UNICORE Grid Infrastructure. Scientific Programming : Special Issue on Grid Computing 10 (2002): 149-158.
9. Czajkowski, K., Foster, I., Karonix , N., Kesselman, C., and Martin , S. A Resource Management Architecture for Metacomputing Systems. In Proceeding of IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, 1998.
10. Huedo, E., Montero, R. S. and Llorente, I. M. An Experimental Framework for Executing Applications in Dynamic Grid environments. ICASE Technical Report, 2002.

11. Subramani, V., Kettimuthu, R., Srinivasan, S. and Sadayappan, P. Distributed Job Scheduling on Computational Grids using Multiple Simultaneous Requests. Proceedings of the 11th IEEE Symposium on High Performance Distributed Computing (HPDC 2002), 2002.
12. Maheswaran, M., Ali, S., Siegel, H. J., Hensgen, D. A. and Freund, R. F. Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems. Proceedings of Heterogeneous Computing Workshop, 1999.
13. Smith, W. and Wong, P. Resource Selection Using Execution and Queue Wait Time Predictions. NAS Technical Report Number: NAS-02-003, 2003.
14. He, X., Sun, X. H. and Laszewski, G. A QoS Guided Scheduling Algorithm for Grid Computing. Proceedings of the International Workshop on Grid and Cooperative Computing (GCC02), 2003.
15. Wolski R., Spring, N. and Hayes, J. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. Future generation Computing Systems 15 (1999): 757-768.
16. Teo, Y. M., Wang, X. and Gozali, J. P. A Compensation-based Scheduling Scheme for Grid Computing, Proceedings of the 7th International Conference on High Performance Computing and Grid in Asia Pacific Region (HPC Asia 2004), IEEE Computer Society Press, 2004.
17. Chiang, H., Dusseau-Arpaci, A., and Vernon, M. K. The Impact of More Accurate Requested Runtimes on Production Job Scheduling Performance, Proceedings of the 8th Workshop on Job Scheduling Strategies for Parallel Processing, 2002.
18. Phatanapherom, S. Model and implementation of efficient grid resource scheduler. Master's Thesis, Department of Computer Engineering, Kasetsart University, 2003.
19. Dumitrescu, C., Foster, I. and Wilde, M. Policy-based Resource Allocation for Virtual Organizations. iVDGL/GriPhyN Technical Report, 2003.

20. Krauter, K., Buyya, R. and Maheswaran, A. M. A Taxonomy and Survey of Grid Resource Management Systems. Software Practice and Experience 32 (2002): 135-164.

21. Veridian Systems. Portable Batch System [Online]. (n.d.). Available from: http://www.openpbs.com [2004, September]

22. Cluster Resources. Maui scheduler [Online]. (n.d.). Available from: http://www.supercluster.org/maui/ [2004, September]

23. Brett Bode, D. M. H., Ricky, K., Lei, Z. and Jackson, D. The Portable Batch Scheduler and the Maui Scheduler on Linux Clusters, Proceedings of Usenix Conference, 2000.

24. Sun Microsystem. Sun Grid Engine [Online]. (n.d.). Available from: http://gridengine.sunsource.net [2004, September]

25. Buyya, R., Murshed, M. and Abramson, D. A Deadline and Budget Constrained Cost-Time Optimization Algorithm for Scheduling Task Farming Applications on Global Grids, Proceedings of the 2002 International Conference on Parallel and Distributed Processing Techniques and Applications, 2002.

26. Abramson, D., Giddy, J. and Kotler, L. High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?, Proceedings of International Parallel and Distributed Processing Symposium (IPDPS), 2000.

27. Sriprayoonskul, S. and Uthayopas, P. SQMS: A Batch Scheduling System for Large PC Cluster, Proceedings of the 5th Annual National Symposium on Computational Science and Engineering, 2001.

28. Buyya, R., Abramson, D. and Giddy, J. Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid, Proceedings of the 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000), 2000.

29. Frey, J., Tannenbaum, T., Livny, M. and Foster, I. Condor-G: A Computation Management Agent for Multi-Institutional Grids, Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01), 2001.

30. Phatanapherom, S. and Uthayopas, P. On The Building of a Job Scheduler System for Globus Grid Environment, Proceedings of APAN Conference, 2002.

31. Feitelson, D. The parallel workload achieve [Online]. (n.d.). Available from: http://www.cs.huji.ac.il/labs/parallel/workload/ [2004, September]

32. NIST/SEMATECH e-Handbook of Statistical Methods [Online]. (n.d.). Available from: http://www.itl.nist.gov/div898/handbook/ [2004, September]

33. Jann, J., Pattnaik, P., Franke, H., Wang, F., Skovira, J. and Riodan, J. Modeling of workload in MPPs. Job Scheduling Strategies for Parallel Processing (1997): 95-116.

34. Buyya, R. and Murshed, M. Using the GridSim Toolkits for Enabling Grid Computing Education. Journal of Concurrency and Computation 14 (2002).

35. SPACE Research Unit. SPACE Grid Portal [Online]. (n.d.). Available from: http://space.cp.eng.chula.ac.th/portal/index.thml [2004, September]

36. Thomas, M., Mock, S., Boisseau, J., Dahan, M., Mueller, K. and Sutton, D. The GridPort Toolkit Architecture for Building Grid Portals. Proceedings of the 10th IEEE International. Symposium on High Perferformance Distributed Computing, 2001

37. Novotny, J., Tuecke, S. and Welch, V. An Online Credential Repository for the Grid: MyProxy. Proceeding of the 10th IEEE Symposium on High Performance Distributed Computing, 2001.

38. Gaussian Official Site [Online]. (n.d.). Available from: http://www.gaussian.com/ [2004, September]

39. AutoDock [Online]. (n.d.). Available from: http://www.scripps.edu/mb/olson/doc/autodock/ [2004, September]

40. Wiriyaprasit, S. and Muangsin, V. The Impact of Local Priority Policies on Grid Scheduling Performance and an Adaptive Policy-based Grid Scheduling Algorithm, Proceedings of the 7th International Conference on High Performance Computing and Grid in Asia Pacific Region (HPC Asia 2004), IEEE Computer Society Press, 2004

Biography

Siraprapa Wiriyaprasit was born in Bangkok, Thailand on February 10, 1981. She received the B.Eng. degree from Chulalongkorn University in 2002. Then, she worked at the Stock Exchange of Thailand for one year. After that, she continued her study in the master program at the department of computer engineering, Chulalongkorn University.

Her research interests include distributed and parallel computing, especially Grid computing.