

วิธีการจัดสรรงานโดยปริยายสำหรับระบบประมวลผลแบบกริด



นายณัฐกฤตย์ สงวนดีกุล

ศูนย์วิทยทรัพยากร  
จุฬาลงกรณ์มหาวิทยาลัย

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรดุษฎีบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

ปีการศึกษา 2552

ลิขสิทธิ์ของจุฬาลงกรณ์มหาวิทยาลัย

**IMPLICIT LOAD SHARING STRATEGY FOR GRID COMPUTING SYSTEM**

**Mr.Natthakrit Sanguandikul**

ศูนย์วิทยทรัพยากร  
จุฬาลงกรณ์มหาวิทยาลัย

**A Dissertation Submitted in Partial Fulfillment of the Requirements  
for the Degree of Doctor of Philosophy Program in Computer Engineering**

**Department of Computer Engineering**

**Faculty of Engineering**

**Chulalongkorn University**

**Academic Year 2009**

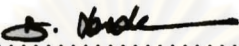
**Copyright of Chulalongkorn University**

**520054**

Thesis Title            **IMPLICIT LOAD SHARING STRATEGY FOR GRID COMPUTING SYSTEM**  
By                         **Mr.Natthakrit Sanguandikul**  
Field of Study         **Computer Engineering**  
Thesis Advisor        **Natawut Nupairoj, Ph.D.**

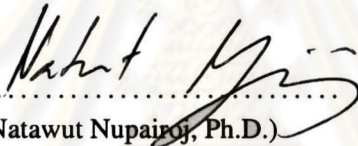
---

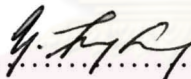
Accepted by the Faculty of Engineering, Chulalongkorn University in Partial Fulfillment of the Requirements for the Doctoral Degree

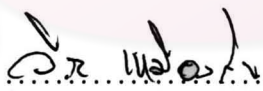
  
.....                         **Dean of the Faculty of Engineering**  
**(Associate Professor Boonsom Lerdhirunwong, Dr.Ing)**


**THESIS COMMITTEE**

  
.....                         **Chairman**  
**(Professor Prabhas Chongstitvatana, Ph.D.)**

  
.....                         **Thesis Advisor**  
**(Natawut Nupairoj, Ph.D.)**

  
.....                         **Member**  
**(Yunyong Teng-amnuay, Ph.D.)**

  
.....                         **Member**  
**(Assistant Professor Veera Muangsin, Ph.D.)**

  
.....                         **Member**  
**(Assistant Professor Putchong Uthayopas, Ph.D.)**

ณัฐกฤตย์ สงวนติกุล: วิธีการจัดสรรงานโดยปริยายสำหรับระบบประมวลผลแบบกริด. (IMPLICIT LOAD SHARING STRATEGY FOR GRID COMPUTING SYSTEM) อ.ที่  
 ปรึกษา : อ. ดร. ณัฐวุฒิ หนูไพโรจน์, 70 หน้า.

เทคโนโลยีกริดได้ถูกนำมาใช้อย่างกว้างขวางเพื่อเป็นโครงสร้างสำหรับรวบรวมทรัพยากรทางด้านประมวลผลซึ่งกระจายอยู่ตามที่ต่างๆเข้าด้วยกันโดยอาศัยเครือข่ายอินเทอร์เน็ต ระบบกริดแต่ละระบบหรือหนึ่ง “องค์กรเสมือน” นั้นอาจจะประกอบไปด้วยคลัสเตอร์ประมวลผลจำนวนมากจากหลากหลายองค์กรที่มีวัตถุประสงค์ร่วมกัน ด้วยเหตุผลดังกล่าวระบบประมวลผลแบบกริดจึงมีเอกลักษณ์ที่แตกต่างจากระบบประมวลผลอื่นๆในอดีตเช่น โหนดประมวลผลย่อยแต่ละโหนดนั้นไม่สามารถติดต่อกันได้โดยตรง หรือแม้แต่วิธีการส่งข้อมูลก็มากเนื่องจากระบบกริดนั้นทำงานบนข่ายงานบริเวณกว้าง เป็นต้น

เพื่อให้สามารถใช้ทรัพยากรประมวลผลจำนวนมากภายในระบบกริดได้อย่างมีประสิทธิภาพเราจำเป็นต้องนำเอาวิธีการจัดสรรงานเข้ามาใช้กระจายงานภายในระบบ วิธีการจัดสรรงานนั้นนับได้ว่าเป็นหนึ่งในส่วนประกอบสำคัญที่เกี่ยวข้องกับประสิทธิภาพในการประมวลผลของระบบกริด อย่างไรก็ตามวิธีการจัดสรรงานซึ่งได้ถูกนำเสนอในอดีตนั้น มักจะตัดสินใจโดยอ้างอิงจากข้อมูลประเภทขีดจำกัดที่อธิบายโดยตรงถึงลักษณะแต่ละส่วนภายในระบบประมวลผลจึงทำให้วิธีการจัดสรรงานในอดีตนั้นไม่เหมาะสมกับการนำมาใช้ในระบบประมวลผลแบบกริดเนื่องจากความยากในการเก็บรวบรวมและความไม่น่าเชื่อถือของข้อมูลประเภทขีดจำกัดนั่นเอง

ภายในงานวิจัยนี้เราได้นำเสนอข้อมูลเพื่อช่วยในการตัดสินใจประเภทใหม่ที่มีชื่อว่า “ข้อมูลซ่อนเร้น” ข้อมูลชนิดนี้เป็นข้อมูลเดี่ยวที่สามารถแสดงถึงความเร็วในการประมวลผลของระบบที่มีต่องานที่ได้รับมอบหมาย ยิ่งไปกว่านั้น ข้อมูลประเภทนี้ยังสามารถถูกเก็บรวบรวมได้ที่โหนดแฉกงานโดยตรงในขณะที่งานกำลังถูกประมวลผลอยู่ ด้วยเหตุผลดังกล่าวข้อมูลซ่อนเร้นจึงง่ายต่อการนำไปใช้โดยไม่จำเป็นต้องใช้แบบจำลองของทรัพยากรประมวลผล หรือติดตั้งบริการเก็บรวบรวมข้อมูลเพิ่มเติมแต่อย่างใด

เนื่องจากข้อมูลซ่อนเร้นนั้นไม่สามารถถูกนำไปใช้แทนข้อมูลแบบขีดจำกัดได้โดยตรง เราจึงได้นำเสนอวิธีการจัดสรรงานโดยปริยายและตัวขยายเพิ่มเติมสำหรับแต่ละองค์ประกอบเด่นๆของระบบกริดเช่น การเป็นระบบประมวลผลขนาดใหญ่ที่เกิดจากการเชื่อมต่อคลัสเตอร์ขนาดต่างๆเข้าด้วยกัน, เวลาแฝงขนาดใหญ่ภายในเครือข่ายวงกว้าง, และความแตกต่างทางด้านประสิทธิภาพในการประมวลผล เป็นต้น เราได้จำลองระบบกริดขึ้นด้วยโปรแกรมจำลองเครือข่ายเพื่อวัดประสิทธิภาพของวิธีการจัดสรรงานที่ได้นำเสนอผลลัพธ์ที่ได้จะถูกนำไปเปรียบเทียบกับผลที่ได้จากวิธีอื่นๆในอดีต ผลการทดลองแสดงให้เห็นว่าวิธีการจัดสรรงานโดยปริยายนั้นมีประสิทธิภาพเท่าเทียมหรือสูงกว่าวิธีการจัดสรรงานแบบเดิมโดยเฉพาะเมื่อมีความคลาดเคลื่อนของข้อมูลเกิดขึ้นภายในระบบ

ภาควิชา .....วิศวกรรมคอมพิวเตอร์.....  
 สาขาวิชา.....วิศวกรรมคอมพิวเตอร์.....  
 ปีการศึกษา ..... 2552 .....

ลายมือชื่อนิสิต .....  
 ลายมือชื่ออาจารย์ที่ปรึกษา .....

## 4671842421: MAJOR COMPUTER ENGINEERING  
KEYWORD: LOAD SHARING STRATEGY / GRID COMPUTING.

NATTHAKRIT SANGUANDIKUL : IMPLICIT LOAD SHARING STRATEGY FOR GRID COMPUTING SYSTEM. THESIS ADVISOR : NATAWUT NUPAIROJ, PH.D., 70 pp.

Grid technology has been extensively introduced as a computing framework for aggregating the computing resources geographically distributed over the Internet. A single grid system or a single “Virtual Organization” can be built in the form of multiple heterogeneous computing clusters from different organizations who share the same objective. Thus, grid system has unique characteristics such as no direct communication between the computing nodes in different clusters, large data transfer overhead due to WAN latency, etc.

In order to effectively use these massively computing resources within grid system, we must employ load sharing strategy to distribute workload in the system. Load sharing strategy is always one of the key components to overall performance of grid computing system. However, most strategies assign workload with respect to explicit information. This kind of information represents the characteristics of the computing resources which are difficult to be collected and unreliable to be used for making load sharing decision within grid computing system.

In this work, we propose a new metric for making load decision called “implicit information”. It is a single metric that can represent how fast a computing node can process the submitted jobs. Moreover, it can be gathered at the coordinator node which is responsible for distributing workload during the execution. Thus, this information is comprehensive and can be used for making load decision immediately without any resource models or any monitoring services.

Since implicit information cannot be used as direct substitution of explicit information, we decide to propose a new implicit strategy and its extensions for addressing unique characteristics within grid computing environment. We simulate our experiments using network simulator (NS) to evaluate the performance of our proposed strategy. We then vary the characteristics of both underlying systems and submitted applications. The obtained results of implicit strategy are compared to those from other load sharing strategy in the past. The simulation results indicate that it outperforms traditional strategies especially when information inaccuracy occurred in the system.

Department . . . . . Computer Engineering . . . . .	Student’s signature . . . . .
Field of study . . . . . Computer Engineering . . . . .	Advisor’s signature . . . . .
Academic year . . . . . 2009 . . . . .	

## Acknowledgements

To accomplish all of this work, there are many people who provide supports and reviews. First of all, I would like to thank my advisor, Dr. Natawut Nupairoj, for his grateful advices and technical knowledge.

The friendly environment inside this department is one of the key that bring me to the final phase. I would like to thank every people in the department of computer engineering and Information Systems Engineering Laboratory (ISEL) at Chulalongkorn University.

Finally, I would like to thank my parents for their kindly support and love which give me strength to reach this moment.



ศูนย์วิทยทรัพยากร  
จุฬาลงกรณ์มหาวิทยาลัย

# Contents

	Page
<b>Abstract (Thai)</b> . . . . .	iv
<b>Abstract (English)</b> . . . . .	v
<b>Acknowledgements</b> . . . . .	vi
<b>Contents</b> . . . . .	vii
<b>List of Tables</b> . . . . .	x
<b>List of Figures</b> . . . . .	xi
<b>Chapter</b>	
<b>I Introduction</b> . . . . .	<b>1</b>
1.1 Problem Statement . . . . .	1
1.2 Grid Computing . . . . .	2
1.2.1 Cluster-Based Hierarchical Structure . . . . .	3
1.2.2 WAN Communication . . . . .	3
1.2.3 Large Computing Heterogeneity . . . . .	4
1.3 Load Sharing Strategies . . . . .	4
1.3.1 Communication Structure-Based Classification . . . . .	5
1.3.2 Load Information-Based Classification . . . . .	7
1.3.3 Self-Scheduling Strategy . . . . .	7
1.4 Objectives . . . . .	8
1.5 Scope of the Work . . . . .	8
1.6 Organization . . . . .	9
<b>II Backgrounds and Assumptions</b> . . . . .	<b>10</b>
2.1 Assumptions . . . . .	10
2.1.1 System Environments . . . . .	10
2.1.2 Application Model . . . . .	11
2.1.3 Sensitivity Model of Load Information . . . . .	12
2.2 The Analysis of Stage-Based Self-Scheduling Strategies . . . . .	13
2.2.1 Prediction Model of Parallel Runtime . . . . .	14
2.2.2 The Behavior of Factoring and Descendants . . . . .	15
2.2.3 The Performance Analysis of Weighted Factoring . . . . .	16
<b>III Consuming Rate</b> . . . . .	<b>21</b>
3.1 Definition of Consuming Rate . . . . .	21
3.2 Behavior of Consuming Rate . . . . .	21
3.2.1 Consuming Rate and Number of Computing Nodes in Clusters . . . . .	22
3.3 Limitations of Consuming Rate . . . . .	22
3.3.1 Application Specific . . . . .	22
3.3.2 Require Certain Amount of Unit Tasks to Achieve Acceptable Accuracy . . . . .	23
3.3.3 Inaccurate Estimation of the Computing Power of Coordinator Node . . . . .	23
3.3.4 Non-Reusable among Other Coordinator Nodes . . . . .	23
3.4 Averaged Consuming Rate . . . . .	23

Chapter	Page
3.5 Conclusion . . . . .	23
<b>IV Implicit Load Sharing Strategy . . . . .</b>	<b>25</b>
4.1 Phases of Computation in Implicit Strategy . . . . .	25
4.1.1 Increasing Phase . . . . .	25
4.1.2 Decreasing Phase . . . . .	26
4.2 Unique Characteristics of Implicit Strategy . . . . .	27
4.2.1 Black Box Based Self-Scheduling Strategy . . . . .	27
4.2.2 Addressing Sensitivity of Load Information . . . . .	27
4.2.3 Phase-Based Adaptive Strategy . . . . .	27
4.3 Conclusion . . . . .	28
<b>V The Performance Evaluation of Implicit Strategy . . . . .</b>	<b>29</b>
5.1 The Prediction Model for Implicit Strategy . . . . .	29
5.2 The Simulated Experiments . . . . .	33
5.2.1 Load Sharing Strategies with Different Communication Structures . . . . .	33
5.2.2 Load Sharing Strategies Utilizing Explicit Information . . . . .	35
5.3 Conclusion . . . . .	36
<b>VI Extensions of Implicit Strategy . . . . .</b>	<b>37</b>
6.1 Hierarchical Structure . . . . .	37
6.1.1 CRSS Extension for Hierarchical Structure . . . . .	37
6.1.2 Performance Evaluation . . . . .	38
6.2 Large Computing Heterogeneity . . . . .	39
6.2.1 CRSS Extension for Large Computing Heterogeneity . . . . .	39
6.2.2 Performance Evaluation . . . . .	40
6.3 Inaccurate Information . . . . .	40
6.3.1 CRSS Extension for Inaccurate Information . . . . .	41
6.3.1.1 Basic Algorithm . . . . .	42
6.3.1.2 Stage Warping . . . . .	44
6.3.2 Performance Evaluation . . . . .	45
6.4 Application Classes . . . . .	47
6.4.1 Performance Evaluation . . . . .	47
6.4.1.1 Applications with Different Number of Unit Tasks . . . . .	47
6.4.1.2 Applications with Different Workload Patterns over Homogeneous System . . . . .	48
6.4.1.3 Applications with Different Workload Patterns over Heterogeneous System . . . . .	49
6.5 Conclusion . . . . .	51
<b>VII Related Works . . . . .</b>	<b>52</b>



Chapter	Page
<b>VII Conclusion</b> . . . . .	<b>54</b>
<b>References</b> . . . . .	<b>55</b>
<b>Biography</b> . . . . .	<b>58</b>



ศูนย์วิทยทรัพยากร  
จุฬาลงกรณ์มหาวิทยาลัย

## List of Tables

Table	Page
2.1 Related parameters in the system environment. . . . .	11
2.2 Related parameters of Application Model. . . . .	11
2.3 List of related variables in prediction model. . . . .	15
2.4 The parameters for simulating <i>WFSS</i> in single cluster environment. . . . .	19
5.1 The parameters for simulating <i>CRSS</i> in single cluster environment. . . . .	33
5.2 The system parameters for evaluating the effect of number of computing nodes. . . . .	34
5.3 The parameters for simulating single cluster environment. . . . .	35
6.1 The parameters for simulating multiple cluster environment. . . . .	38
6.2 The parameters for simulating highly heterogeneous environment. . . . .	40
6.3 The parameters for evaluating the performance of <i>CRSS – SW</i> . . . . .	46
6.4 The parameters for evaluating the effect of different application classes. . . . .	47


  
 ศูนย์วิทยทรัพยากร  
 จุฬาลงกรณ์มหาวิทยาลัย

## List of Figures

Figure	Page
1.1 An example of computing grid. . . . .	2
1.2 An example of computing cluster. . . . .	3
1.3 Hierarchical structure in grid computing system. . . . .	4
1.4 Classes of Load Sharing Strategies. . . . .	5
1.5 Centralized load sharing strategy. . . . .	5
1.6 Distributed load sharing strategy. . . . .	6
1.7 Hybrid/Hierarchical load sharing strategy. . . . .	6
1.8 Behavior of self-scheduling strategies. . . . .	8
1.9 Workload Allocation of Stage-Based Self-Scheduling Strategy. . . . .	8
2.1 Simulated grid computing environment. . . . .	10
2.2 Workload patterns of four different application classes. . . . .	12
2.3 Parallel runtime of WFSS with different information sensitivities. . . . .	13
2.4 Four parts of prediction model. . . . .	14
2.5 Chunk size for each request of four clusters using <i>WFSS</i> . . . . .	16
2.6 Four parts of prediction model for <i>WFSS</i> . . . . .	17
2.7 The parallel runtime of <i>WFSS</i> ( $r_{xy}=0.03125$ ). . . . .	19
2.8 The parallel runtime of <i>WFSS</i> ( $r_{xy}=0.5$ ). . . . .	19
3.1 The comparison of consuming rate and the actual aggregated computing power in a 64-node cluster. . . . .	22
4.1 An example of stage sequence in implicit strategy. . . . .	25
4.2 The chunk size per request of four clusters during runtime. . . . .	26
5.1 Four parts of prediction model for <i>CRSS</i> . . . . .	30
5.2 The parallel runtime of <i>CRSS</i> ( $r_{xy}=0.03125$ ). . . . .	33
5.3 The parallel runtime of <i>CRSS</i> ( $r_{xy}=0.5$ ). . . . .	33
5.4 Parallel runtimes as a function of computing nodes. . . . .	34
5.5 Parallel runtimes with varied estimation gap ratio. . . . .	35
5.6 Parallel runtimes with varied computing ratio. . . . .	36
6.1 Hierarchical structure in grid computing environment. . . . .	37
6.2 Parallel runtime with varied estimation gap ratio. . . . .	38
6.3 Parallel runtime with varied computing ratio. . . . .	39
6.4 Parallel runtime of implicit strategies with different relative power. . . . .	40
6.5 Chunk assignment of <i>CRSS</i> . . . . .	41
6.6 Stage number during runtime of <i>CRSS</i> . . . . .	41
6.7 Stage number during runtime of <i>AWFSS</i> . . . . .	41
6.8 Predefined amount of workload allocated in each stage during the increasing phase. . . . .	43
6.9 Predefined amount of workload allocated in each stage during the decreasing phase. . . . .	43
6.10 Chunk size assignment of <i>CRSS</i> – <i>SW</i> in ideal case. . . . .	44
6.11 Stage number during runtime of <i>CRSS</i> – <i>SW</i> . . . . .	45
6.12 Utilization graph of <i>CRSS</i> and <i>CRSS</i> – <i>SW</i> . . . . .	46
6.13 Parallel runtime of <i>AWFSS</i> , <i>CRSS</i> and <i>CRSS</i> – <i>SW</i> with varied estimation gap. . . . .	46
6.14 Parallel runtime of <i>AWFSS</i> , <i>CRSS</i> and <i>CRSS</i> – <i>SW</i> with different computing heterogeneity. . . . .	47

	Page
6.15 Parallel runtime of application with different unit tasks. . . . .	48
6.16 Parallel runtime of various applications ( $G_x = 0$ ). . . . .	48
6.17 Parallel runtime of various applications ( $G_x = 0.3$ ). . . . .	49
6.18 Parallel runtime of application with uniform pattern. . . . .	49
6.19 Parallel runtime of application with increasing pattern. . . . .	50
6.20 Parallel runtime of application with decreasing pattern. . . . .	50
6.21 Parallel runtime of application with random pattern. . . . .	51



ศูนย์วิทยทรัพยากร  
จุฬาลงกรณ์มหาวิทยาลัย

# CHAPTER I

## INTRODUCTION

### 1.1 Problem Statement

Grid technology [1][2] is introduced as a computing framework for aggregating the computing resources geographically distributed over the Internet working together in the so called “virtual organization”. A single grid system can be built in the form of multiple heterogeneous computing clusters with more than a thousand of computing nodes. Thus, grid has unique characteristics such as no direct communication between the computing nodes in different clusters, large WAN latency, etc.

In order to effectively use these massively computing resources, we must employ task scheduling strategy to distribute workload in the system. Although many scheduling strategies have been proposed, the load sharing strategy is considered one of the oldest and the most important research topics in the field of distributed computing [3][4][5]. This strategy states that the performance of the distributed system can be increased by assigning an appropriate amount of workload to keep every computing resource busy until the end of computation. Hence, the underlying system will be fully utilized all the time. To address computing heterogeneity in grid system, many load sharing strategies have been proposed [6][7][8][9][10][11]. However, these strategies make load decision depending on information which directly represents various characteristics of the computing resources such as the speed of an underlying processors, available memory, and communication bandwidth. We refer to this type of information as “explicit information” as they can be explicitly determined even before submitting jobs. Although explicit information is easy to be obtained and understood, each individual information is often insufficient for making load decision. Thus, it usually needs some resource models for combining multiple explicit information together as a single metric [12][13]. As the performance of each computing node is highly related to the submitted jobs, it is very difficult to find a sophisticated model that can truly predict the performance of each resource without the complete information of computing resources and submitted works. Given the growing complexity in the computing environment and application, this methodology is no longer practical.

In this work, we propose a new load sharing strategy for grid computing. To solve the aforementioned difficulties, our strategy utilizes a new metric based on the load information called “implicit information”. Our metric is a single metric that can represent how fast a computing node can process the submitted jobs. Thus, it is comprehensive and can be used for making load decision immediately without any resource models or any monitoring service. Moreover, it can be calculated within the coordinator node which is responsible for distributing workload. This implicit information can be obtained during the execution. Thus, it is suitable for capturing the dynamic behavior of the computing system. The implicit information has been mentioned for performing implicit coscheduling between clusters in [14]. However, it focuses on understanding the communication behavior, not load sharing. Load sharing strategy for grid computing environment proposed in [15] focuses on how to determine which system is homogeneous or heterogeneous. Then, it will assign portion of workload during the first phase according to the computing het-

erogeneity and weighted value of each computing resource. While the proposed strategy claims that it is suitable for grid computing system, this strategy requires an accurate information of the computing power of each resource for making load decision. Therefore, information inaccuracy of explicit information can reduce its overall performance. In addition, we also extend an implicit strategy to address main characteristics of grid computing system such as cluster-based hierarchical system, communication latency in WAN, heterogeneity within computing resources, and inaccurate information.

## 1.2 Grid Computing

Even though grid technology has just been recently introduced to the field of distributed computing, it receives many attentions from researchers throughout the world. This technology focuses on aggregating the physically distributed computing resources over the Internet. These resources can be located in different organizations who want to share their computing resources. With this approach, the organizations that join the same virtual organization can have a large scale computing power for their computing-intensive application. Moreover, by sharing all computing resources in the organization together, the overall utilization can be increased because most of the computers usually idle while some of them overloaded. One important difference between grid technology and peer-to-peer application is that the owner of the computing resources (e.g. system administrators) can still have control over their resources. Every administrator can specify the amount of shared resources for each user from different organizations. There are many both academic and business organizations who already implemented grid technology for sharing their resources. The number of participating party can be expected larger and larger in the future.

Although there are also other different purposes of using grid technology such as data grid and access grid which is intended for aggregating a large amount of data or control the resource usage within a large collaboration group, in this work we will focus on the computing grid only. The infrastructure of computing grid usually consists of the computing clusters over *WAN*. These clusters can be located at different academic laboratories or some business departments. An example of grid computing environment is shown in Fig. 1.1.

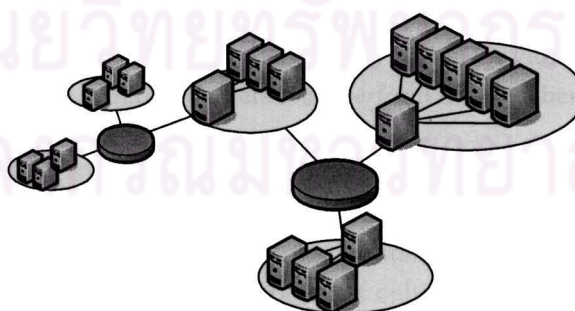


Figure 1.1: An example of computing grid.

From Fig. 1.1, we can see that even though we have tremendous amount of computing power available for the user in the virtual organization, this structure also raises some restrictions and drawbacks to the computing system. In this work, we focus on three main characteristics of grid computing consisting of cluster-based hierarchical structure, WAN communication, and large

### 1.2.1 Cluster-Based Hierarchical Structure

In order to build a high performance computing system with low budget, the clustering concept is introduced. Each computing cluster can be built by connecting a large number of computing nodes together within high speed network making them virtually seen as a single computing resource. The Beowulf infrastructure [16] has been extensively adopted for building the computing cluster. Figure 1.2 illustrates an example of the computing cluster.

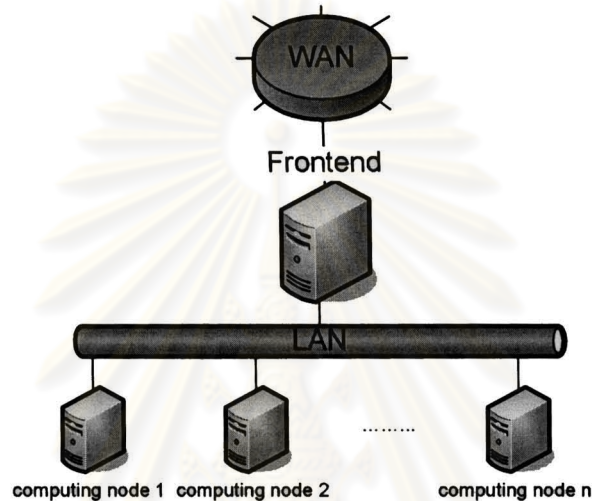


Figure 1.2: An example of computing cluster.

As presented in Fig. 1.2, there are  $n$  computing nodes connected to each other and to the front-end over *LAN*. The front-end connects to *WAN* for handling inter-cluster communications. To utilize computing cluster, users must first submit their applications at the front-end and the front-end will assign workload to each computing node according to the load distribution policy of the scheduler. With this methodology, the users will only see an entire cluster as merely a single computer with a large computing power available. However, this behavior also introduces new restriction that in order to maintain the transparency within the cluster all the communication between the worker nodes in different clusters must perform through their front-end nodes only.

Since grid computing system consists of multiple clusters, its communication topology can be in the form of hierarchical structure even in the same organization. Figure 1.3 illustrates an example organization which consists of multiple divisions, branches, and sections. Thus, the communication structure of grid system restricts not only how workload is exchanged within cluster, but also between clusters as well.

### 1.2.2 WAN Communication

Since grid technology has been proposed for aggregating computing resources from different organizations, the communication overhead can be large due to *WAN*. An amount of inter-cluster communications will be an important factor for defining load sharing strategies. In

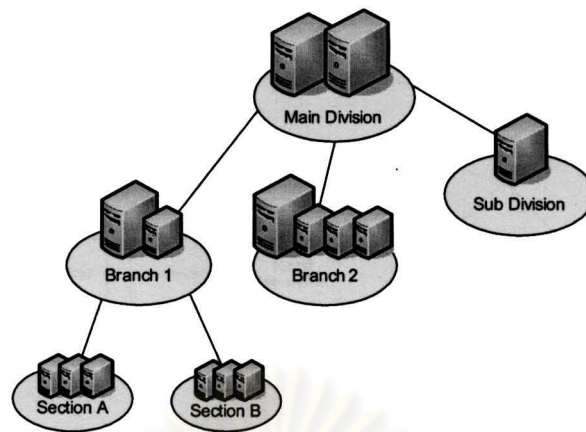


Figure 1.3: Hierarchical structure in grid computing system.

addition, information inaccuracy can also occur as a result of out-dated information [17]. Therefore, this behavior will degrade the performance of the traditional strategies that require an accurate information of the underlying system.

### 1.2.3 Large Computing Heterogeneity

The difference in computing power or computing heterogeneity within grid computing system will increase overtime due to grid's openness. This characteristic will intensify the effect of load imbalance at the end of computation. The remaining workload in some slow clusters can inflict a large amount of additional parallel runtime despite of the total computing power which has been dedicated to the submitted application.

## 1.3 Load Sharing Strategies

The main concept of the load sharing strategy focuses on assigning proper amount of workload to computing nodes for obtaining an optimal resource utilization, throughput, or response time. This strategy utilizes multiple resources simultaneously instead of using single resource to complete the works. It can be further extended to increase the reliability of an underlying system through redundant execution. The implementations of load sharing can be as a dedicated hardware or program and has been commonly used to manage workload in computer clusters, especially high-availability cluster. The performance of load sharing strategy can be varied depending on types of information being used for making load decision and how computing resources exchange their workload information. Note that, throughout this work, we use the term "load sharing" and "load balancing" interchangeably. We prefer to use "load sharing" because the intention of our proposed strategy is to optimize the parallel runtime by distributing different amounts of workload to computation nodes with respect to their computing capacities.

Load sharing strategies can be classified using two aspects, the communication structure and load information about computing resources as shown in Fig. 1.4. Communication structure classifies strategies based on how they exchange load information. This can greatly effect the communication overhead and the number of neighbor nodes during workload assignment. The



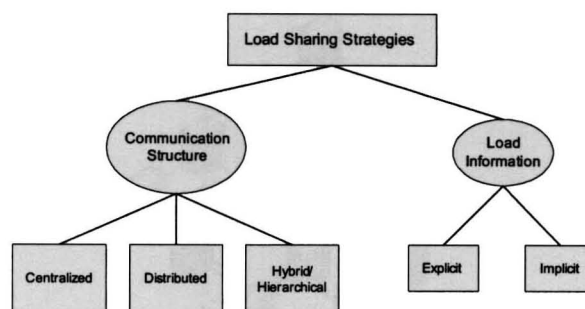


Figure 1.4: Classes of Load Sharing Strategies.

quality of load decision highly depends on how coordinator node captures the current status of an underlying system. In addition, types of load information for making load decision also effect the performance of load sharing strategies. Some metrics can be gathered easily while others are more sensitive to inaccurate information.

### 1.3.1 Communication Structure-Based Classification

By considering how each computing resource exchanges information and workload, we can divide load sharing strategies into three classes consisting of centralized, distributed, and hybrid/hierarchical strategy [18][19].

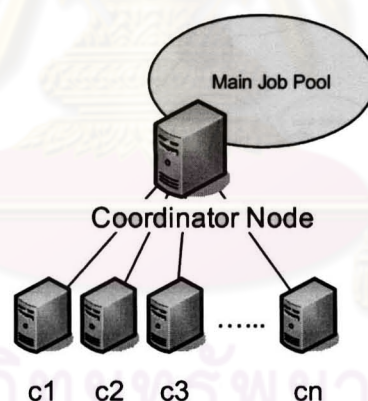


Figure 1.5: Centralized load sharing strategy.

As shown in Fig. 1.5, the centralized strategy [20] aggregates all related system information into the central coordinating node. Therefore, the central coordinating node has all information it needs to make decision. However, this class of load sharing strategy suffers from a contention problem at the central coordinating node in a large-scale computing system. Distributed strategy removes this contention problem by specifying the neighbor nodes for each computing resource. The communication model of distributed load sharing strategy is illustrated in Fig. 1.6.

Using the distributed strategy [21][22][23][24], the computing resources exchange the current information and workload with their neighbors during computation. Although this distributed strategy can remove the contention problem, it fails to capture the overall behavior of the computing system since each computing node makes load decision based on its local information only.

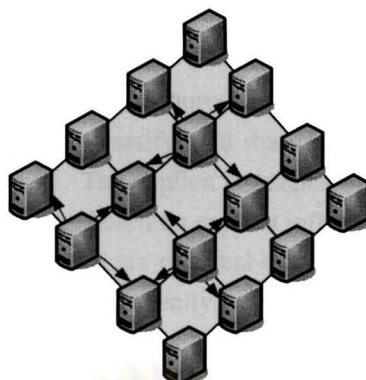


Figure 1.6: Distributed load sharing strategy.

This problem of local view can be reduced by specifying the random neighbor nodes. However, this behavior raises another restriction that each computing node must be able to directly communicating with each other over the network. This is not true in grid system since most computing resources are the computing clusters. Every computing node in each cluster must communicate through its predefined front-end. Consequently, the performance of this strategy decreases because of the contention at the front-end and from high latency in WAN between each cluster.

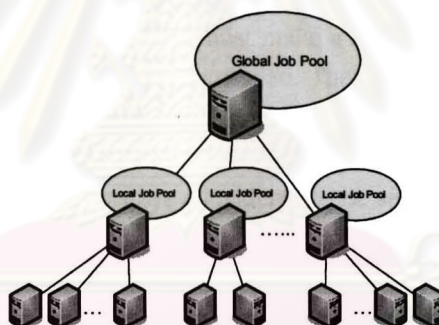


Figure 1.7: Hybrid/Hierarchical load sharing strategy.

Hybrid strategy (or hierarchical strategy) tries to combine centralized and distributed strategy together by defining a tree-like structure. Each branch in the tree represents a sub-coordinating node. This node is responsible for assigning workload to its successor in the tree structure which could be either another sub-coordinating node or the computing node. The current status of the entire system can be monitored at the root node which can be one of the computing resources or just an information server. This strategy can reduce the contention problem at the coordinating nodes while the coordinator node at the root of the tree can still capture the overall behavior of the computing system. Many related works [25][26][27] try to specify the optimal virtual tree structure over computing system. These strategies can achieve good parallel runtime since the computing nodes with the large available computing power are usually sharing the same parents in tree structure with the computing node with small computing power. However, this behavior often results in direct communication between nodes across multiple clusters which is not feasible in grid system.

### 1.3.2 Load Information-Based Classification

Load information of the computing resources can greatly effect the parallel performance of load sharing strategy. Thus, we can classify load sharing strategies into two classes: *explicit strategies* and *implicit strategies*. The explicit strategies refer to the traditional load sharing strategies which perform load decision based on explicit information of the computing resources. The explicit information represents various physical-oriented characteristics of the computing system such as processor speed, memory capacity, or current workload. To utilize explicit information, most explicit strategies define resource model for aggregating explicit information into single usable value. In contrast, the implicit strategies use more subtle approach. Instead of using physical information, the implicit strategies base their load decisions on implicit information which is basically the job processing speeds observed at the coordinator node. Implicit load sharing strategies proposed in this work is simple, yet effective. By utilizing implicit information, we can overcome the limitations founded in most self-scheduling strategies in the past. More details about the explicit and implicit information can be found in the subsequence sections.

### 1.3.3 Self-Scheduling Strategy

Self-scheduling strategy represents a large class of dynamic loop scheduling strategies. With this strategy, the coordinator node will determine when and which computing nodes to send workload to during runtime. Any idle nodes must make a request back to the coordinator again when it already finished previously assigned workload. The performance of self-scheduling strategy can be further improved by defining how the chunk sizes will be changed during an execution. Load imbalance can be occurred when some nodes are still waiting for more work to arrive. Any strategies that assign workload this way are considered to be self-scheduling strategy. This strategy can utilize different communication topologies for distributing workload although it often uses centralized communication structure. Moreover, this strategy can make load decision based on both explicit and implicit information. Therefore, this strategy can be classified into multiple subclasses according to how it is implemented. Due to its strength and simplicity, there are many load sharing strategies which based on this strategy. Figure 1.8 illustrates the behavior of self-scheduling strategy.

There is one class of self-scheduling strategy which is famous for its robustness called stage-based self-scheduling strategy. This strategy first allocates a portion of workload for each stage. These workload will be further divided according to the computing power of the requesting node. Thus, every computing node will tend to finish its previously assigned work at the same time during each stage. Load imbalance at the end of computation can be reduced with this concept. Figure 1.9 illustrates how stage-based self-scheduling strategy assigns workload during each stage. There are many strategies which can be considered as stage-based self-scheduling strategy such as *FSS*, *WFSS*, *AWFSS* including our implicit strategy (*CRSS*) as well. More details on these strategies will be provided later in the next chapter while other self-scheduling strategies will be summarized in Chapter 7.

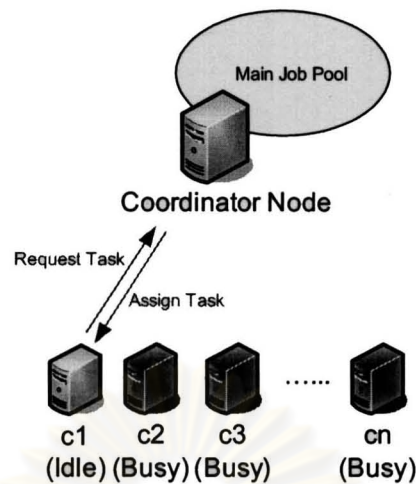


Figure 1.8: Behavior of self-scheduling strategies.

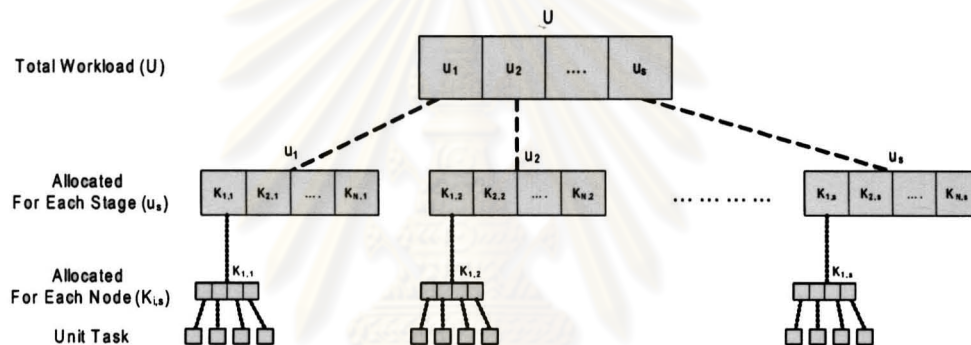


Figure 1.9: Workload Allocation of Stage-Based Self-Scheduling Strategy.

## 1.4 Objectives

The objective of this dissertation is to define a practical load sharing strategy which is simple to use and suitable for grid computing environment. Our proposed strategy must also be able to handle an information inaccuracy or different kinds of fluctuations within the computing system.

## 1.5 Scope of the Work

This work will define a new load sharing strategy which utilizes implicit information. We assume grid computing system as the collection of computing clusters connected together using wide-area network. We focus on various factors which can effect the performance of load sharing such as computing heterogeneity, communication overhead, inaccurate information, and different classes of applications. Both mathematic and simulations are defined to evaluate the performance of our proposed strategy. The simulated application will be submitted to the computing system at the coordinator node with every unit task specified. Although our proposed strategy can be extended for different type of applications with unique characteristics such as branch-and-bound applications, load sharing strategy presented in the work is for computing intensive application

with independent unit tasks only.

## 1.6 Organization

This dissertation is organized as follows: In Chapter 2, we talk about backgrounds and assumptions about system environments, prediction model, and behavior of explicit strategies. Chapter 3 describes our proposed implicit information and its behavior. Implicit strategy utilizing implicit information is introduced in Chapter 4 while its performance analysis will be described in Chapter 5. Chapter 6 talks about the extensions for implicit strategy addressing the unique characteristics of grid computing system and submitted applications. The related works are summarized in Chapter 7 before we conclude our work in Chapter 8.



ศูนย์วิทยทรัพยากร  
จุฬาลงกรณ์มหาวิทยาลัย

## CHAPTER II

### BACKGROUNDS AND ASSUMPTIONS

#### 2.1 Assumptions

In our work, we assume that the system is a grid-based environment, consisting of multiple clusters working together over *WAN*. We also assume that each cluster assigns one of its node to become the front-end node responsible for distributing workload to other computing nodes in the same cluster. Moreover, one of the front-end nodes will also serve as the coordinator node which manages the submitted jobs and assigns workload to every cluster.

##### 2.1.1 System Environments

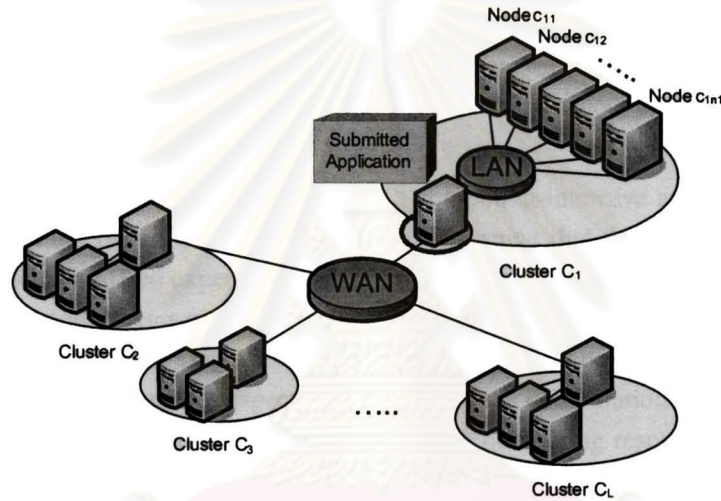


Figure 2.1: Simulated grid computing environment.

Grid system consists of  $N$  computing nodes which will be grouped together into  $L$  clusters  $\{C_1, C_2, \dots, C_L\}$ . These computing clusters communicate with each other over *WAN* with latency and bandwidth specified as  $\alpha_W$  and  $\beta_W$  respectively. The communication within each cluster will be defined using  $\alpha_L$  and  $\beta_L$  for representing latency and bandwidth within *LAN*. To simplifying our work, we assume that the overheads of the intra-cluster communication are the same in all clusters. cluster  $C_i$  consists of  $n_i$  computing nodes where  $c_{ij}$  represents node  $j$  in cluster  $i$ . We specify  $p_i$  to be the aggregated computing power in  $C_i$ . The computing power of node  $j$  in cluster  $i$  ( $\rho_{ij}$ ) will be specified in comparison with the total computing power  $P$  according to  $r_{ij}$ . Thus:

$$\begin{aligned}
 P &= \sum_{i=1}^L p_i \\
 &= \sum_{i=1}^L \sum_{j=1}^{n_i} \rho_{ij}
 \end{aligned} \tag{2.1}$$

All parameters related to simulated computing environment are shown in Table 2.1.

Table 2.1: Related parameters in the system environment.

Variables	Definitions
$L$	Total number of computing clusters
$C_i$	Computing cluster $i$
$c_{ij}$	Computing node $j$ in cluster $i$
$N$	Total computing nodes
$n_i$	Number of computing nodes in cluster $i$
$P$	Total computing power in grid system
$p_i$	Computing power of cluster $i$
$\rho_{ij}$	Computing power of node $j$ in cluster $i$
$R_i$	Computing ratio of cluster $i$
$r_{ij}$	Computing ratio of node $j$ in cluster $i$
$\alpha_W$	Inter-cluster latency
$\beta_W$	Inter-cluster bandwidth
$\alpha_L$	Intra-cluster latency
$\beta_L$	Intra-cluster bandwidth

### 2.1.2 Application Model

Throughout this work, we focus mainly on a computing-intensive application which consists of a large number of independent unit tasks. We assume that there is only one submitted application for each simulated experiment. Grid user submits his/her application to the coordinator node. Then, the coordinator node will distribute these unit tasks to other clusters and its local computing nodes. There are  $U$  unit tasks per application. The total computation and communication size is represent as  $W$  and  $V$  respectively. Note that the computation size of each unit task can be seen as a weighted value effecting how long each computing resource will take to finish each task. The computing resource with computing power specified as one task per second means it can finish the unit task with computation size specified as one within one second. The larger computation size will result as an additional time for the same computing resource to finish it. The related parameters being used throughout this work are shown in Table 2.2.

Table 2.2: Related parameters of Application Model.

Variables	Definitions
$U$	Total number of tasks in submitted application
$W$	Computation size of submitted application
$w_q$	Computation size of each unit task
$V$	Communication size of submitted application
$v_q$	Communication size of each unit task

To model various application types, we define four distinct classes of applications [28]. Each application class consists of multiple independent tasks with different computation size of each task ( $w_q$ ) based on predefined workload pattern. The predefined workload pattern can be either uniform, increasing, decreasing, or random distribution, which can represent popular applications such as Matrix Multiplication, SOR, Reverse Adjoint Convolution, LU Decomposition, and Gauss Jordan Elimination, respectively. Figure 2.2 illustrates the workload patterns of all four application classes.

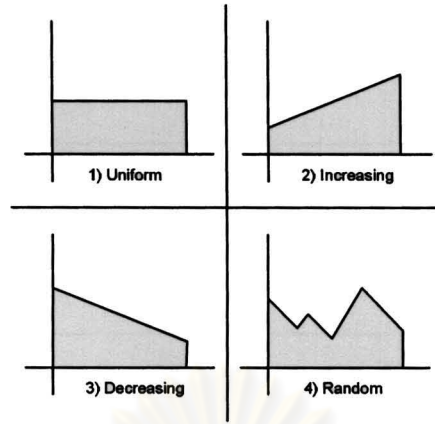


Figure 2.2: Workload patterns of four different application classes.

For application with uniform pattern, all unit tasks have the same constant task size, which is  $\{w_q = W/U\}$ . This is not the case for the application with increasing workload pattern, whose unit task sizes are bigger near the end of computation. On the contrary, the application with decreasing workload pattern will have smaller computation size near the end of computation. The exponential random distribution will be used when simulating the application class with random distribution pattern. In order to ensure the fairness in our simulated experiment, we specify the total computation size of every application class to be the same. Moreover, we also evaluate the performance of load sharing strategies when the number of unit tasks is limited. Finally, we change the computing heterogeneity of an underlying system to study the behavior of load sharing strategies with different applications.

### 2.1.3 Sensitivity Model of Load Information

Typically, every load sharing strategy makes a load sharing decision based on the estimated computing power of the computing resources. We define  $\hat{\rho}_{ij}$  to represent the estimated computing power of node  $c_{ij}$ . On one hand, some strategies use static  $\hat{\rho}_{ij}$  estimated from pre-determined CPU and memory capacities. On the other hand, many strategies measure  $\hat{\rho}_{ij}$  during run-time. Therefore,  $\hat{\rho}_{ij}$  can be different from the actual value,  $\rho_{ij}$ . We refer to this difference as the sensitivity of load information. This sensitivity value plays an important role in load sharing strategies as it can greatly effect their performance. To model information sensitivity, we define estimation gap ratio ( $g_{ij}$ ) to represent the ratio of incorrect estimation of the computing power at node  $c_{ij}$ . An estimated computing power of node  $c_{ij}$  ( $\hat{\rho}_{ij}$ ) can be calculated as:

$$\hat{\rho}_{ij} = (1 + g_{ij}) * \rho_{ij} \quad (2.2)$$

The estimated total computing power ( $\hat{P}$ ) can then be calculated as:

$$\hat{P} = \sum_{i=1}^L \sum_{j=1}^{n_i} \hat{\rho}_{ij} \quad (2.3)$$



To illustrate the effect of information sensitivity, Fig. 2.3 shows the parallel runtime of explicit strategy (*WFSS*) when the information sensitivity about one computing node ( $g_{xy}$ ) is varied.

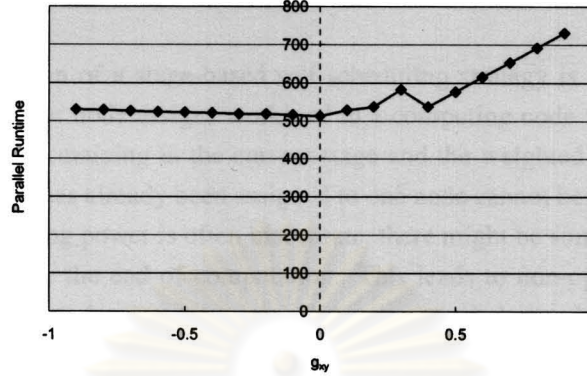


Figure 2.3: Parallel runtime of *WFSS* with different information sensitivities.

From Fig. 2.3, we can see that *WFSS* will achieve its best result when there is no information sensitivity occurring in the computing system ( $g_{xy}$  is specified as zero). The obtained parallel runtime will become worse when the information sensitivity about node  $y$  in cluster  $x$  ( $g_{xy}$ ) either increases or decreases. The positive/negative value of information sensitivity will result as an over/under-estimation about the computing power of a specified node. Given a homogeneous system, an over-estimation about the computing power of one computing node tends to effect overall performance more than when we under-estimates it because other computing nodes must wait until an over-estimated node to finish all of its workload at the end of computation. Thus, an amount of idle computing power will be large in these cases.

## 2.2 The Analysis of Stage-Based Self-Scheduling Strategies

In this work, we propose a mathematical model to evaluate the parallel performance of a load sharing strategy. To simplify our model, we assume that the system has only one cluster called cluster  $C_x$  available. In this case,  $L = 1$  and  $P = p_x$ . The estimated computing powers of all other computing nodes except computing node  $c_{xy}$  will always be correct. Therefore, there will be only one computing node which is incorrectly estimated for its computing power. We will use  $c_{xy}$  to represent this node throughout an entire work. As for an information sensitivity ( $g_{xy}$ ), we will use only non-negative values when we evaluate the performance of *WFSS* because load information about an underlying system is usually in the form of an upper-bound value resulting as an over-estimation about the available computing power and explicit strategies also exhibits the same behavior with both positive and negative values of  $g_{xy}$ . Thus:

$$\hat{\rho}_{xj} = \begin{cases} \rho_{xj} & \text{if } j \neq y \\ (1 + g_{xy}) * \rho_{xj} & \text{otherwise} \end{cases} \quad (2.4)$$

This assumption will effect the parallel runtime depending on which load sharing strategy is being used. Moreover, We also assume that the file size of submitted job is small with respect

to the communication bandwidth. Therefore, the communication overhead is very short and can be omitted.

### 2.2.1 Prediction Model of Parallel Runtime

The entire execution of a stage-based self-scheduling strategy is divided into stages. In each stage, the coordinator node assigns workload to a computing node with respect to the total chunk size of workload remaining in the current stage and the weighted value of that particular node. The workload that has already been assigned to one node cannot be moved to another node. As the estimated computing power is often inaccurate, there might be some nodes stay idle while other nodes are busy near the end of computation. This leads to non-optimal parallel runtime. Let  $c_{xy}$  be the node whose  $\hat{\rho}_{xy}$  is overestimated  $\hat{\rho}_{xy} > \rho_{xy}$  and all other nodes have accurate estimation. In this case,  $c_{xy}$  will finish its works later than the rest. Thus, the execution stages of self-scheduling algorithm can be grouped into 4 parts as shown in Figure 2.4. Note that the box in the figure represents a number of unit tasks assigned during each stage. The boxes with dark color indicate the works belong to  $c_{xy}$ , which may be executed at the other nodes near the end of computation.

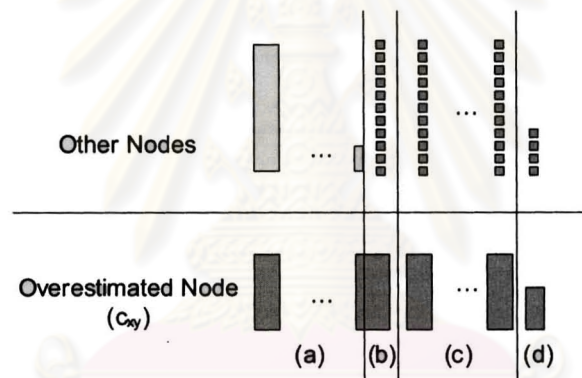


Figure 2.4: Four parts of prediction model.

- a. Parallel runtime until all nodes, except  $c_{xy}$ , finish their portions of workload ( $t_A$ )
- b. Parallel runtime for  $c_{xy}$  to finish its current stage while other normal nodes execute work from  $c_{xy}$ 's portion ( $t_B$ )
- c. Parallel runtime for  $c_{xy}$  to finish its subsequence stage while other normal node continue to execute more works from  $c_{xy}$ 's portion ( $t_C$ )
- d. Parallel runtime for  $c_{xy}$  to finish its last stage ( $t_D$ )

The total parallel runtime ( $T$ ) can be calculated as:

$$T = t_A + t_B + t_C + t_D \quad (2.5)$$

Note that the parallel runtime in part  $C$  will occur only when there are enough workload in  $c_{xy}$ 's portion that can keep other nodes busy until  $c_{xy}$  execute the last stage or else this part will be omitted. The related variables used in this model is shown in Table 2.3.

Table 2.3: List of related variables in prediction model.

Variable Names	Meaning
$T$	Total parallel runtime
$t_{(A/B/C/D)}$	Parallel runtime in part A,B,C,D
$\omega'_{xi,(A/B/C/D)}$	Remaining tasks in node $i$ of cluster $x$ during part A,B,C,D
$\omega'_{(A/B/C/D)}$	Remaining tasks at the beginning of part A,B,C,D
$\tilde{\omega}_{xi,(A/B/C/D)}$	Expected tasks for node $i$ of cluster $x$ during part A,B,C,D
$\tilde{\omega}_{(A/B/C/D)}$	Expected tasks for every node during part A,B,C,D

## 2.2.2 The Behavior of Factoring and Descendants

In this work, we choose one of the load sharing strategies which is famous for the robustness called ‘‘Factoring’’ ( $FSS$ ) [29] and its descendants to represent explicit strategy. This strategy proposes new notation called *stage*. Every computing node will receive equal chunk size during each stage. The total chunk size distributed during stage  $s$  ( $u_s$ ) can be obtained as:

$$u_s = \left\lceil \frac{\omega'_s}{\delta} \right\rceil \quad (2.6)$$

Given  $\omega'_s$  as the remaining unit tasks at the beginning of stage  $s$ , we can define a decreasing stage size between each stage. The parameter  $\delta$  is computed by a probability distribution or is suboptimally chosen as  $\delta = 2$ . Since  $FSS$  was proposed for homogeneous system where every computing node has the same computing power, the chunk size assigned for node  $j$  of cluster  $i$  during stage  $s$  ( $K_{ij,s}$ ) can be calculated as:

$$K_{ij,s} = \left\lceil \frac{u_s}{N} \right\rceil \quad (2.7)$$

From Eq. (2.7), we can see that  $FSS$  distributes the largest chunk in the first stage and will decrease the chunk size in the subsequent stages with an equal proportion. During each stage, every processor will receive an equal chunk size of workload.  $FSS$  can reduce communication overhead by sending large chunks at the beginning while it achieves sub-optimal runtime by sending small chunks near the end of computation.

To address heterogeneity within the computing system, ‘‘Weighted Factoring’’ ( $WFSS$ ) [30] is proposed as an extension of  $FSS$ . In this strategy, the amount of total unit tasks allocated during each stage is the same as defined in  $FSS$  (using the same  $u_s$ ). However, unlike  $FSS$ ,  $WFSS$  can be considered as explicit strategy since it uses explicit information of the computing resources to further assign workload allocated within each stage.

Using  $WFSS$ , the chunk size of the first request is the largest and decreases toward the

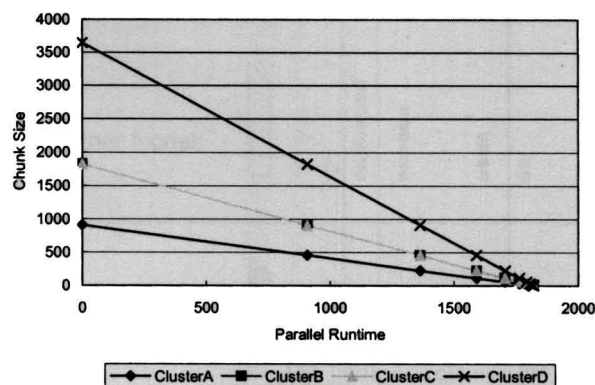


Figure 2.5: Chunk size for each request of four clusters using *WFSS*.

end of computation. From Fig. 2.5, we can see that each cluster receives a decreasing chunk size according to its available computing power. For example, cluster *D* has the computing power 4 times larger than cluster *A*.

As an extension of *WFSS* for addressing inaccurate estimator, “Adaptive Weighted Factoring” (*AWFSS*) [31] has been introduced. This strategy further extends *WFSS* by introducing new weighted value called “Weighted Average Performance”. This weighted value ( $WAP_{ij,k}$ ) of node  $j$  of cluster  $i$  during stage  $k$  can be calculated as follows:

$$WAP_{ij,k} = \frac{\sum_{s=1}^k t_{ij,s} * s}{\sum_{s=1}^k K_{ij,s} * s} \quad (2.8)$$

Where  $t_{ij,s}$  is an execution time for node  $j$  from cluster  $i$  to finish all  $K_{ij,s}$  iterations in stage  $s$ . This weighted value will be re-calculated every stage using the newly obtained computing rates of each resource. Therefore, the explicit information will be used as a weighed value during the first stage only. With this average value, *AWFSS* can address the dynamic behavior of the heterogeneous computing system. However, since *AWFSS* assigns half of the available workload during the first stage, the problem of an inaccurate explicit information can still effect the performance of this strategy.

For simplicity, we will analyze the behavior of *WFSS* as an example of explicit strategy. Both *WFSS* and *AWFSS* will be simulated and compared with our implicit strategy later in the following chapters.

### 2.2.3 The Performance Analysis of Weighted Factoring

The behavior of *WFSS* highly depends on the accuracy of the computing power being used for making load sharing decision. We will use *WFSS* as a representative of explicit strategies. By using our prediction model, we can predict the parallel runtime of *WFSS* and identify the factors that affect the parallel runtime of explicit strategy. The behavior of *WFSS* within our prediction model is illustrated in Figure 2.6

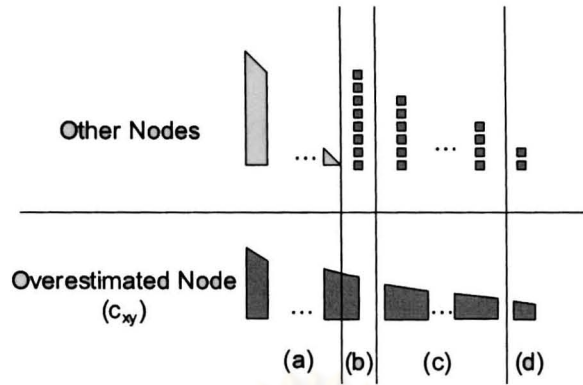


Figure 2.6: Four parts of prediction model for *WFSS*.

- a. Parallel runtime until all nodes, except  $c_{xy}$ , finish their portions of workload ( $t_A$ )

Let  $t_A$  be the time that this part ends. Since we assume that only the computing power of  $c_{xy}$  has been overestimated, other nodes, beside  $c_{xy}$ , will finish their works at the same time. Thus, we can calculate  $t_A$  by considering the completion time of any nodes, except  $c_{xy}$ .

$$t_A = \frac{U}{\hat{P}} \quad (2.9)$$

- b. Parallel runtime for  $c_{xy}$  to finish its current stage while other normal nodes execute work from  $c_{xy}$ 's portion ( $t_B$ )

To determine an amount of workload remaining in  $c_{xy}$  during part *B*, we must first find the stage that  $c_{xy}$  is in when part *B* starts. Given the decreasing stage size of *WFSS* with constant ratio as 2, we have the following equation:

$$\frac{\rho_{xy}}{\hat{\rho}_{xy} * (1 - (1/2)^m)} = 1 \quad (2.10)$$

Based on equation(2.10), the stage of  $c_{xy}$  when part *B* starts (and finishes) is  $\lceil m \rceil$ . Thus:

$$t_B = \frac{\omega'_{xy,B}}{\rho_{xy}} \quad (2.11)$$

where

$$\omega'_{xy,B} = \left(\frac{U}{\hat{P}}\right) * ((1 - (1/2)^{\lceil m \rceil}) * \hat{\rho}_{xy} - \rho_{xy}) \quad (2.12)$$

The amount of unit tasks necessary for every node to stay busy during part *B* ( $\tilde{\omega}_B$ ) becomes:

$$\tilde{\omega}_B = \omega'_{xy,B} + (t_B * (P - \rho_{xy})) \quad (2.13)$$

If the remaining tasks in part *B* is less than  $\tilde{\omega}_B$ , the execution stops with parallel runtime equals to the parallel runtime in *A* and *B* only.

- c. Parallel runtime for  $c_{xy}$  to finish its subsequence stage while other normal node continue to execute more works from  $c_{xy}$ 's portion ( $t_C$ )

We will calculate how many stages that node  $c_{xy}$  can go further while every node is busy. To begin with, the remaining tasks in part  $C$  ( $\omega'_C$ ) can be calculated as:

$$\omega'_C = \left(\frac{U}{\hat{P}}\right)(\hat{\rho}_{xy} - \rho_{xy}) - \tilde{\omega}_B \quad (2.14)$$

Since  $WFSS$  always reduces stage size by half every stage, we can calculate the amount of work which will be allocated to  $c_{xy}$  given the number of stages that has been executed. Note that part  $C$  start from stage  $[m] + 1$  which can be calculated from (2.10). An amount of workload assigned to  $c_{xy}$  during stage  $j$  in part  $C$  can be calculated as:

$$\tilde{\omega}_{xy,j} = (1/2)^j * N * \frac{\hat{\rho}_{xy}}{\hat{P}} \quad (2.15)$$

Therefore, we can find the last stage of node  $c_{xy}$  that every node is still busy before phase  $D$  starts at  $[n]$  by comparing with  $\omega'_C$  as:

$$\frac{\omega'_C}{\sum_{j=m+1}^n \left[ \tilde{\omega}_{xy,j} + \left(\frac{\tilde{\omega}_{xy,j}}{\rho_{xy}}\right) * (P - \rho_{xy}) \right]} = 1 \quad (2.16)$$

If there is not enough tasks for every node to stay busy, all of the remaining tasks will be executed in phase  $D$ . An amount of work necessary for every node to stay busy and total parallel runtime in phase  $C$  can be obtained as follows:

$$\tilde{\omega}_C = \sum_{j=m+1}^{\lfloor n \rfloor} \left[ \tilde{\omega}_{xy,j} + \left(\frac{\tilde{\omega}_{xy,j}}{\rho_{xy}}\right) * (P - \rho_{xy}) \right] \quad (2.17)$$

where

$$t_C = U * \left(\frac{\hat{\rho}_{xy}}{\hat{P}}\right) * ((1/2)^m - (1/2)^n) * \frac{1}{\rho_{xy}} \quad (2.18)$$

d. Parallel runtime for  $c_{xy}$  to finish its last stage ( $t_D$ )

In this part, we will compare how much works of  $c_{xy}$  remain that can be distributed to other nodes. We begin with calculating an amount of work for the overestimated node  $c_{xy}$  to be executed in this part.

$$\omega'_D = \omega'_C - \tilde{\omega}_C \quad (2.19)$$

$$\tilde{\omega}_D = \min(\omega'_D, \left(\frac{U}{2}\right) \left(\frac{\hat{\rho}_{xy}}{\hat{P}}\right) (1/2)^{\lfloor n \rfloor - I + 1}) \quad (2.20)$$

$$t_D = \frac{\tilde{\omega}_{xy,D}}{\rho_{xy}} \quad (2.21)$$

By combining all of the parallel runtime from every part together, we can have the overall predicted parallel runtime. From (2.12), we can see that when  $\hat{\rho}_{xy}$  is increased as a result of larger gap between the estimated and real computing power of node  $c_{xy}$  ( $g_{xy}$ ),  $\omega'_{xy,B}$  will increase. This means both  $c_{xy}$  and other normal computing nodes need more works in order to stay busy. If there is not enough remaining work, load imbalance between computing

nodes will occur degrading the overall parallel performance. This behavior will also appear in part *C* where other nodes need more work in order to stay busy given an increasing  $\hat{\rho}_{xy}$  as illustrated in (2.15). Hence, our prediction model for *WFSS* indicates that an overestimated computing power can lead to a degraded parallel performance because of an additional load imbalance between computing nodes.

Using our prediction model, we can analyze the performance of *WFSS* by comparing the model to the simulation results obtained from *Network Simulator(NS)* [32] with the parameters given in Table 2.4. From the given simulation parameters, the parallel runtime of the prediction model and simulated experiments are illustrated in Figure 2.7 and 2.8, respectively. Note that  $r_{xy}$  represents the computing ratio of an overestimated node.

Table 2.4: The parameters for simulating *WFSS* in single cluster environment.

Variable Names	Values
Number of tasks ( $U$ )	16,384
Total number of compute nodes ( $N$ )	32
Number of clusters ( $L$ )	1
Intra-cluster communication (Latency, Bandwidth) ( $\alpha_L, \beta_L$ )	1ms , 100Mb/s

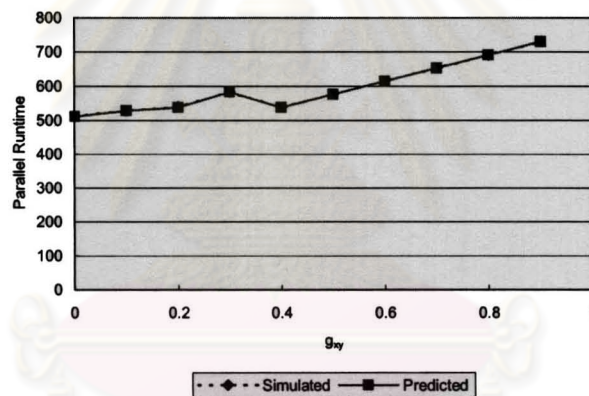


Figure 2.7: The parallel runtime of *WFSS* ( $r_{xy}=0.03125$ ).

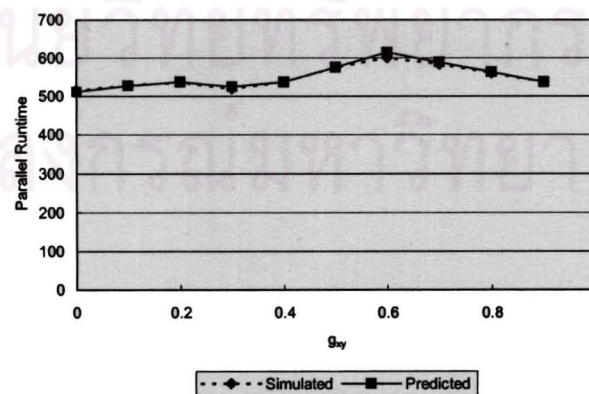


Figure 2.8: The parallel runtime of *WFSS* ( $r_{xy}=0.5$ ).

From simulated experiments, we can see that the performance of *WFSS* is best when there is no estimation gap ( $g_{xy} = 0$ ). The parallel runtime of *WFSS* tends to increase in according to

an estimation gap ratio ( $g_{xy}$ ). Both prediction model and simulated experiments clearly indicate that the overestimation of computing power at node  $c_{xy}$  can degrade the parallel runtime of explicit strategy. This behavior is the result of load imbalance between computing nodes at the end of computation especially when the relative computing power of node  $c_{xy}$  ( $r_{xy}$ ) is small in comparison with other nodes.



ศูนย์วิทยทรัพยากร  
จุฬาลงกรณ์มหาวิทยาลัย



## CHAPTER III

### CONSUMING RATE

#### 3.1 Definition of Consuming Rate

A good load sharing strategy requires a good load sharing metric that can truly represent the computing power of the working node while taking network condition into consideration and yet simple enough to measure accurately. While the idea of defining general-purpose load metric has been proposed in [33], it does not introduce any new metrics or load sharing strategies at all. In this work, we propose an implicit information called “consuming rate”, as load sharing metric [34]. This simple metric can satisfy as a good load sharing metric when being used properly.

In general, stage-based self scheduling strategy requires the computing nodes to request workload of the next stages from the coordinator/front-end nodes when they complete the execution of their current stages. As the coordinator/front-end nodes know the chunk size that has been assign to the computing nodes and can measure the time between requests, they can estimate the computing nodes' capacities using the consuming rate. The consuming rate ( $cr_{ij,s}$ ) is simply a rate of how fast a requesting node  $c_{ij}$  can process unit tasks at stage  $s$ . The definition of  $cr_{ij,s}$  is as followed:

$$cr_{ij,s} = \left( \frac{K_{ij,s-1}}{Intv_{ij,s}} \right) \quad (3.1)$$

Where  $cr_{ij,s}$  represents the consuming rate of node  $j$  of cluster  $i$  at stage  $s$ , which can be calculated from the chunk size assigned in previous stage ( $K_{ij,s-1}$ ) and the interval time between requests of node  $c_{ij}$  from the previous request to this one ( $Intv_{ij,s}$ ). It is obvious that these two variables can be collected easily at the coordinator or front-end nodes which are responsible for assigning workload.

Since  $cr_{ij,s}$  is measured at the coordinator node, this metric takes an account for both actual computing power and communication bandwidth, even if the owner of each computing cluster may not dedicate the entire cluster to process the submitted application. Thus, we can make load decision immediately without the need to define any complex resource models with multiple metrics or measuring any indicators at the computing nodes on the networks. The coordinator node just needs to only keep track of the rate of requests from each working node.

#### 3.2 Behavior of Consuming Rate

While this consuming rate is fairly simple, measuring it accurately requires the understanding of its natures. The value of consuming rate usually depends on the amount of assigned workload and the underlying computing infrastructure. Obviously, the consuming rate can be measured accurately when there are enough workload assigned to the working node. In this section, we will study the behavior and effect of consuming rate upon different factors such as the number of computing nodes in the clusters or the chunk size per each request.

### 3.2.1 Consuming Rate and Number of Computing Nodes in Clusters

In Grid computing environment, a cluster appears as a single computing resource. Thus, the coordinator node will treat the entire cluster as a single worker node. However, a cluster actually consists of a large number of computing nodes inside. Hence, collecting the consuming rate of the entire cluster by assigning a few unit tasks to a computing resource in grid may not accurately represent the actual computing power of that resource.

Figure 3.1 presents the behavior of our consuming rate as a function of chunk size given a sample cluster with 64 computing nodes. For simplicity, we assume that each computing node can process one task per unit time and always has the same computing power throughout an entire execution in this example.

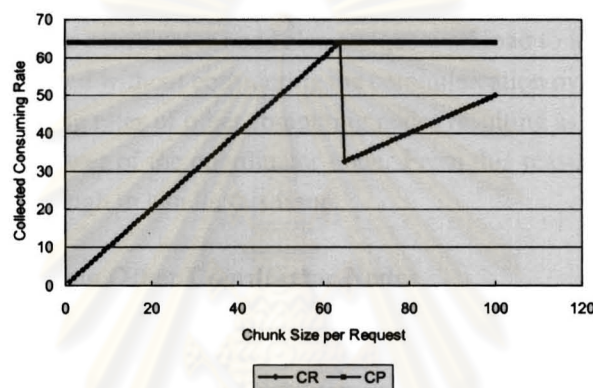


Figure 3.1: The comparison of consuming rate and the actual aggregated computing power in a 64-node cluster.

In Figure 3.1, the collected consuming rate depends on the chunk size assigned in each request. As chunk size increases, the number of working nodes in the sample cluster increases. The consuming rate can truly represent the actual computing power when we assign workload in multiples of the total number of computing nodes in that cluster. In addition, when we assign larger chunk size, the consuming rate becomes more accurate than when assigning smaller one. Therefore, our consuming rate is a bad estimator of the computing power of an entire cluster when the assigned chunk size per request is small.

### 3.3 Limitations of Consuming Rate

Although consuming rate is simple and powerful enough to be used for making load decision, it also has some limitations that we need to consider when we define implicit strategy.

#### 3.3.1 Application Specific

Different from other explicit information such as the speed of CPU, the consuming rate is application specific. The consuming rate can change considerably even with the same computing system given different applications. Some applications require large communication bandwidth while others might focus on the computing power of an underlying system instead. An obtained consuming rate can also vary according to the different set of parameters of the same application.

Therefore, we must be very careful when we decide to use the consuming rate collected in the past.

### 3.3.2 Require Certain Amount of Unit Tasks to Achieve Acceptable Accuracy

Although the consuming rate should be collected on-the-fly during an execution, there must be a certain amount of unit tasks in the submitted application. Since an accuracy of the obtained consuming rate can change with respect to the chunk size, the lack of unit tasks in the submitted application can worsen the performance of our implicit strategy. The performance comparison with different number of unit tasks will be shown in the following section.

### 3.3.3 Inaccurate Estimation of the Computing Power of Coordinator Node

In some cases that the coordinator node also assigns workload to itself, the consuming rate of that node will be calculated without considering the communication overhead. Hence, it will be different from the consuming rates of other computing nodes resulting as an overestimation about the available computing power of the coordinator node. From this reason, the proposed implicit strategy must be robust enough to handle this issue.

### 3.3.4 Non-Reusable among Other Coordinator Nodes

The consuming rate is also depend on the communication structure between the coordinator node and other computing nodes. Therefore, the consuming rate should not be used immediately when we change the coordinator node. However, we might use old consuming rate as a reference when we start calculating the consuming rate again at new coordinator node.

## 3.4 Averaged Consuming Rate

As for grid computing system which consists of multiple clusters, this consuming rate will truly represent an available computing power of the requesting cluster only when chunk size is equal or more than the number of the computing nodes in that cluster. Otherwise, the computing power of idle nodes will not be taken into consideration. Since an accuracy of the consuming rate depends on how much workload have been assigned during each request, we define an average value of the consuming rate so that the consuming rate obtained from assigning larger chunk will effect this value more than the others. The averaged consuming rate ( $\bar{c}r_{i,k}$ ) of  $C_i$  at stage  $k$  can be calculated as

$$\bar{c}r_{ij,k} = \frac{cr_{ij,k}(K_{ij,k-1}) + \bar{c}r_{ij,k-1} * \sum_{s=1}^{k-2} K_{ij,s}}{\sum_{s=1}^{k-1} K_{ij,s}} \quad (3.2)$$

## 3.5 Conclusion

Consuming rate represents the capability of the computing resource to process the submitted application. It is a single metric which can be used immediately for making load decision. This value can be obtained at the coordinator node without the need to implement any monitoring

services.

Although the consuming rate can be collected and used easily, it also has the limitations and unique characteristics which are different from those of explicit information. From the mathematical analysis, we find that the consuming rate can be a good estimator of the computing power by allocating a large chunk size per request. However, the variance in the runtime will also increase proportionally with the chunk size. These issues must be addressed when we define implicit load sharing strategy as we will show in the following chapter.



ศูนย์วิทยทรัพยากร  
จุฬาลงกรณ์มหาวิทยาลัย

## CHAPTER IV

### IMPLICIT LOAD SHARING STRATEGY

To overcome the shortcomings of traditional load sharing strategy, we propose a new self-scheduling strategy called consuming rate self-scheduling (*CRSS*). Based on *WFSS*, our algorithm assigns workload into stages. The consuming rate (*cr*) will be used for making decision of workload assignment to the working nodes in each stage.

#### 4.1 Phases of Computation in Implicit Strategy

In our implicit load sharing strategy, we utilize *cr* for making load decision without knowing *cr*'s initial values. We can obtain *cr* only during the execution by assigning chunks to the requesting nodes. Although sending large chunks allows us to obtain accurate consuming rates, it may also introduce load imbalance at the end of computation. Thus, we define two phases of execution: increasing and decreasing phases. During the increasing phase, we gradually increase the stage size to obtain accurate consuming rates. After the increasing phase completes, the decreasing phase starts. Utilizing the *cr* measured in the increasing phase, the decreasing phase can reduce the variance of runtime during the end of computation. Note that it does not matter how we assign chunks as long as they are large enough to keep every node busy and not too large to create load imbalance near the end of computation. Hence, there are only two phases in our implicit strategy. Figure 4.1 illustrates how implicit strategy allocates workload during each stage where  $U$  is 16,384 and the number of stages in the increasing phase is specified as 5.

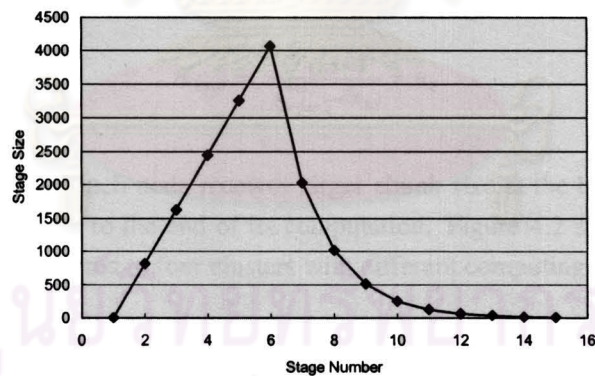


Figure 4.1: An example of stage sequence in implicit strategy.

#### 4.1.1 Increasing Phase

Increasing phase is important for obtaining accurate consuming rates. During this phase, the stage sizes are defined in arithmetic sequence. This yields better parallel runtime than using other sequence such as geometrical sequence since other sequences may assign too small chunks at the beginning resulting to an inaccurate consuming rate. In addition, we allocate an equally half of a total workload for both increasing and decreasing phases.

Let  $u_s$  be the total chunk size in stage  $s$ . Suppose that there are  $I$  stages in the increasing

phase. We gradually increases the assigned chunk size with a constant value  $\Delta$ . As we split total available tasks in half and  $u_1$  is specified as 1, the constant value ( $\Delta$ ) can be computed as follows:

$$\Delta = \left\lceil \frac{\frac{U}{I} - 2}{I - 1} \right\rceil \quad (4.1)$$

After the increasing phase completes, the coordinator node will have the consuming rate of each working node. Our algorithm then performs the decreasing phase.

#### 4.1.2 Decreasing Phase

The purpose of decreasing phase is to make our implicit strategy more robust against an inaccuracy of our consuming rate. During this phase, the chunk size is decreased by a constant proportion ( $\delta$ ) to achieve near-optimal parallel runtime similar to *WFSS*. This constant proportion can be calculated using a probability distribution or suboptimally specified as 2. Thus, the chunk size of stage  $s$  ( $u_s$ ) becomes:

$$u_s = \left\lceil \frac{U}{2 * \delta^{(s-I)}} \right\rceil, s \geq I + 1 \quad (4.2)$$

Given  $u_s$  and the average consuming rate during the previous stages  $\bar{c}r_{ij,s-1}$ , the chunk size for node  $c_{ij}$  during stage  $s$  ( $K_{ij,s}$ ) can now be determined as:

$$K_{ij,s} = \frac{\bar{c}r_{ij,s-1}}{\sum_{j=1}^{n_i} \bar{c}r_{ij}} * u_s \quad (4.3)$$

Using equation(4.3), each node receives larger chunk size at the beginning. Later, it will receive a smaller chunk size to the end of its computation. Figure 4.2 shows the sample chunk size per each request distributed to four clusters with different computing power during runtime.

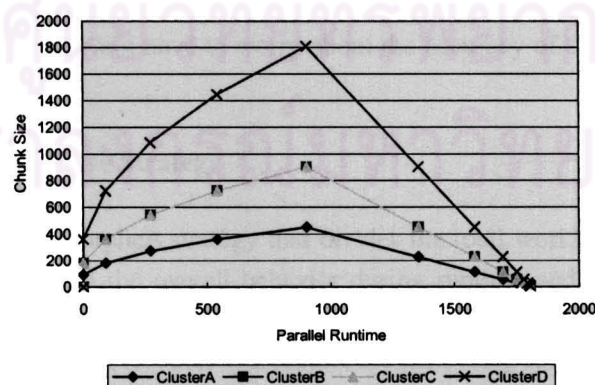


Figure 4.2: The chunk size per request of four clusters during runtime.

In Figure 4.2, we assume that cluster *D* has computing power two times larger than cluster

$B$  and  $C$ . In addition, cluster  $B$  and  $C$  are also assumed to have computing power two times larger than cluster  $A$ . Using our algorithm, cluster  $D$  will receive chunk size per request two times larger than any clusters while cluster  $B$  and  $C$  receive an equal chunk size.

## 4.2 Unique Characteristics of Implicit Strategy

Using consuming rate, our proposed implicit strategy has some unique characteristics that address many shortcomings of other explicit strategies in the past. These characteristics do not only dictate the performance of our strategy, but also allow our strategy to focus on global load sharing for grid computing system.

### 4.2.1 Black Box Based Self-Scheduling Strategy

Since our implicit strategy makes load decision based on the consuming rate, it treats all clusters as black boxes. Thus, it can assign workload properly even without knowing the details of an underlying computing system. Therefore, our strategy can be considered as a black box based self-scheduling strategy. Unlike explicit strategies, this characteristic really simplifies our work since we do not need to understand the relationship between every piece of system information. Implicit strategy can address the changes in both the computing system and submitted application without the need to redefine the resource model. In addition, it is also suitable to be used as a global load sharing strategy for distributing workload to each cluster since it can achieve sub-optimal performance without forcing how workload is distributed by local strategy.

### 4.2.2 Addressing Sensitivity of Load Information

As explicit strategies make decision based on complex resource model, the accuracy of load information can vary according to the quality of resource model, computing system, and submitted application. Moreover, some explicit information might need to be updated more frequently than the others. Therefore, we can say that the sensitivity of load information used in explicit strategies is considerably high. Unlike explicit strategies, the performance of our implicit strategy is not related to the sensitivities of any explicit information at all because it makes load decision based on the consuming rate that reflects the overall performance of each resource during an execution. With implicit strategy, we do not have to worry about the accuracy of the monitoring service or the resource model any more.

### 4.2.3 Phase-Based Adaptive Strategy

The behavior of our implicit strategy that divides the total workload into multiple stages gives it the power to control the overall behavior during runtime and still be able to address the changes occurring in an underlying system. By defining two phases of computation, we can make sure that every computing resource will receive the increasing chunk size at the beginning of computation while receives the decreasing chunk size near the end of computation. The increasing phase is for obtaining accurate consuming rates while the decrease phase is for obtaining sub-optimal parallel runtime. According to how consuming rates are re-calculated at the end of every stage throughout an entire execution, our implicit strategy can adjust to the dynamic behavior of an underlying system because it uses these consuming rates to further assign workload allocated

in each stage to the requesting node.

### 4.3 Conclusion

Our proposed implicit strategy (*CRSS*) consists of two phases of computation to address the unique characteristics of consuming rate. The increasing phase is defined for obtaining an accurate consuming rate since we do not have any information about the computing resource before an execution. In the other hand, the decreasing phase is defined for obtaining sub-optimal parallel runtime near the end of computation. According to how our implicit strategy can be extended as a hierarchical strategy, its performance is better than other centralized and distributed strategy in the past. Moreover, since our implicit strategy does not rely on explicit information at all, the estimation gap in explicit information does not degrade the performance of our strategy.

Despite some limitations according to implicit information, our proposed implicit strategy has many attractive features which are suitable for grid computing system. It can be used as an effective load sharing strategy without adding additional complexities into the underlying system. Therefore, our implicit strategy might be the right solution for today's large scale computing system.



## CHAPTER V

### THE PERFORMANCE EVALUATION OF IMPLICIT STRATEGY

#### 5.1 The Prediction Model for Implicit Strategy

Since our implicit strategy does not use any explicit information regarding to the computing power of each node, an estimated gap ratio of overestimated node ( $g_{xy}$ ) does not affect its parallel performance at all. However, there are some differences in collecting consuming rate of the front-end node and other computing nodes. Suppose there is only one cluster in the system named  $C_x$ . Let  $c_{xf}$  denote a front-end node which is also a coordinator node. The consuming rate of  $c_{xf}$  does not include any communication overhead during each request because it fetches workload directly from itself while this communication overhead will be included in consuming rate of other computing nodes. Since it is ineffective to assign workload large enough to eliminate the effect of network latency, the absence of communication overhead at the front-end node will lead to assigning too large chunk size to the front-end node. Since we predict the parallel runtime of *CRSS* in a single cluster environment, given  $c_{xf} \in \{c_{x1}, c_{x2}, \dots, c_{xN}\}$ , the estimated computing power ( $\hat{\rho}_{xf}$ ) or the consuming rate ( $cr_{xf}$ ) can be calculated as:

$$\hat{\rho}_{xf} = cr_{xf} = \rho_{xf} \quad (5.1)$$

Let  $c_{xk}$  be a node besides  $c_{xf}$ . The estimated computing power of node  $c_{xk}$  ( $\hat{\rho}_{xk}$ ) or the consuming rate of node  $c_{xk}$  ( $cr_{xk}$ ) is:

$$\hat{\rho}_{xk} = cr_{xk} = \frac{\bar{K}}{\frac{\bar{K}}{\rho_{xk}} + 2 * \alpha_L} \quad (5.2)$$

Where  $\alpha_L$  is the communication latency within computing cluster. We then estimate an average chunk size ( $\bar{K}$ ) assigned to each computing node by averaging over every stage in implicit strategy.

$$\bar{K} = \frac{1}{2} * \left( \left( \frac{U}{2} * \frac{\rho_{xk}}{P} * \frac{1}{I} \right) + \left( \frac{U}{2} * \frac{\rho_{xk}}{P} * \frac{1}{\log_2 U + 1} \right) \right) \quad (5.3)$$

From equation(5.1) and equation(5.2), we can see that the consuming rate of front-end node is obtained differently from other computing nodes. Since we assume that the size of unit tasks is small compared to the available communication bandwidth, only the startup communication latency is considered. Therefore, the total estimated computing power ( $\hat{P}$ ) can now be determined as:

$$\hat{P} = \hat{\rho}_{xf} + \sum_{xk \neq xf} \hat{\rho}_{xk} \quad (5.4)$$

From the consuming rate of both front-end and computing nodes together with the stage size predefined in our implicit strategy, we can predict the parallel runtime of our implicit strategy using the prediction model as shown in Figure 5.1

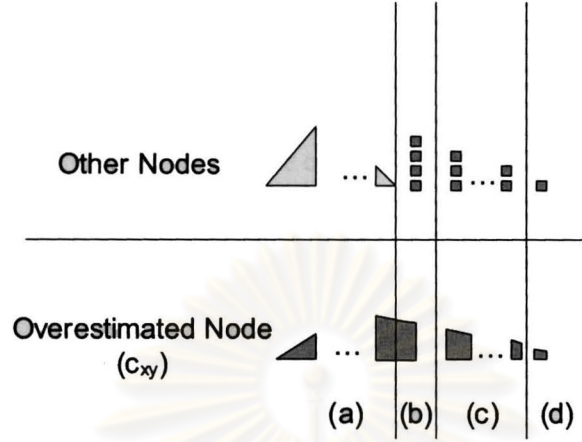


Figure 5.1: Four parts of prediction model for  $CRSS$ .

- a. Parallel runtime until all nodes, except  $c_{xf}$ , finish their portions of workload ( $t_A$ )

Let  $c_{xk}$  be a computing node. Given  $\rho_{xk}$  and  $cr_{xk}$  of node  $c_{xk}$  in implicit strategy, we have

$$t_A = \left( \frac{U}{\hat{P}} \right) \left( \frac{\hat{\rho}_{xk}}{\rho_{xk}} \right) \quad (5.5)$$

- b. Parallel runtime for  $c_{xf}$  to finish its current stage while other normal nodes execute work from  $c_{xf}$ 's portion ( $t_B$ )

To determine an amount of workload remaining in  $c_{xf}$ , we must find its current stage after part A. First, we must determine whether  $c_{xf}$  is already in its decreasing or not by calculating an amount of work executed by  $c_{xf}$  compared with a total work. If an executed work is more than half of total work, we can tell that  $c_{xf}$  is already in the decreasing phase. Given both the increasing and decreasing stage size of  $CRSS$ , we can calculate the current stage of  $c_{xf}$  as followed:

If  $c_{xf}$  still in the increasing phase, its current stage will be equal to  $\lceil m \rceil$  from (5.6).

$$\frac{U \left( \frac{\hat{\rho}_{xk}}{\rho_{xk}} \right) \left( \frac{\rho_{xf}}{\hat{\rho}_{xf}} \right)}{\left( m \frac{(2+(m-1)\Delta)}{2} \right)} = 1 \quad (5.6)$$

However, if  $c_{xf}$  already enter a decreasing phase, the current stage of  $c_{xf}$  in part B will be equal to  $\lceil m \rceil$  from (5.7) instead.

$$\frac{\left( \frac{\hat{\rho}_{xk}}{\rho_{xk}} \right) \left( \frac{\rho_{xf}}{\hat{\rho}_{xf}} \right)}{\left( \frac{1}{2} \right) + \left( \frac{1}{2} \right) m^{-I+2}} = 1 \quad (5.7)$$

With the current stage of  $c_{xf}$ , we can calculate a parallel runtime in this part from the remaining works in  $c_{xf}$  ( $\omega'_{xf,B}$ ) when  $c_{xf}$  is still in the increasing phase as

$$\omega'_{xf,B} = (\lceil m \rceil \frac{(2 + (\lceil m \rceil - 1)\Delta)}{2}) \left( \frac{\hat{\rho}_{xf}}{\hat{P}} \right) - \left( \frac{U}{\hat{P}} \right) \left( \frac{\hat{\rho}_{xk}}{\rho_{xk}} \right) * \rho_{xf} \quad (5.8)$$

If  $c_{xf}$  is already in the decreasing phase, we have

$$\omega'_{xf,B} = (1 + (1/2)^{\lceil m \rceil - I + 1}) \left( \frac{U}{2} \right) \left( \frac{\hat{\rho}_{xf}}{\hat{P}} \right) - \left( \frac{U}{\hat{P}} \right) \left( \frac{\hat{\rho}_{xk}}{\rho_{xk}} \right) * \rho_{xf} \quad (5.9)$$

From equation(5.8) and equation(5.9), we can calculate the parallel runtime in this part as:

$$t_B = \frac{\omega'_{xf,B}}{\rho_{xf}} \quad (5.10)$$

The amount of unit tasks necessary for every node to stay busy during this part is

$$\tilde{\omega}_B = \omega'_{xf,B} + (t_B * (P - \rho_{xf})) \quad (5.11)$$

If the remaining tasks in part  $B$  is less than  $\tilde{\omega}_B$ , an execution stop with parallel runtime equals to the parallel runtime in part  $A$  and  $B$  only.

- c. Parallel runtime for  $c_{xf}$  to finish its subsequence stage while other normal node continue to execute more works from  $c_{xf}$ 's portion ( $t_C$ )

We will calculate how many stages that  $c_{xf}$  can go further while every node is busy given  $\omega'_C$  as:

$$\omega'_C = \left( \frac{U}{\hat{P}} \right) (\hat{\rho}_{xf} - \left( \frac{\hat{\rho}_{xk}}{\rho_{xk}} \right) * \rho_{xf}) - \tilde{\omega}_B \quad (5.12)$$

If  $c_{xf}$  is still in the increasing phase, first we must determine whether there is enough job left for it to enter the decreasing phase or not by comparing a remaining jobs with a necessary amount of work for keeping every node busy until  $c_{xf}$  enters the decreasing phase or  $\tilde{\omega}_{xf,C}^{(I)}$ .

$$\tilde{\omega}_{xf,C}^{(I)} = \left( \frac{\hat{\rho}_{xf}}{\hat{P}} \right) \left( \frac{U}{2} - \lceil m \rceil \frac{2 + (\lceil m \rceil - 1)d}{2} \right) \quad (5.13)$$

If there is enough work for  $c_{xf}$  to be busy until entering the decreasing phase, a number of tasks for every node to be busy until it enters the decreasing phase ( $\tilde{\omega}_C^{(I)}$ ) can be calculated as

$$\tilde{\omega}_C^{(I)} = \tilde{\omega}_{xf,C}^{(I)} + \left( \frac{\tilde{\omega}_{xf,C}^{(I)}}{\rho_{xf}} \right) (P - \rho_{xf}) \quad (5.14)$$

We can calculate the last stage of an overestimated node during the decreasing phase as  $\lceil n \rceil$  from a following equation:

$$\frac{\omega'_C - \tilde{\omega}_C^{(I)}}{\left( \frac{\hat{\rho}_{xf}}{\hat{P}} \right) \left( \frac{U}{2} \right) \left( \frac{1}{2} \right)^{\lceil n \rceil - I + 1} \left( 1 + \left( \frac{P - \rho_{xf}}{\rho_{xf}} \right) \right)} = 1 \quad (5.15)$$

An amount of work necessary for every node to stay busy and total parallel runtime in phase  $C$  can be obtained as follows:

$$\tilde{\omega}_C = \tilde{\omega}_C^{(I)} + \left( \frac{\hat{\rho}_{xf}}{\hat{P}} \right) \left( \frac{U}{2} \right) \left( \frac{1}{2} \right)^{(\lfloor n \rfloor - I + 1)} \left( 1 + \left( \frac{P - \rho_{xf}}{\rho_{xf}} \right) \right) \quad (5.16)$$

where

$$t_C = \left( \frac{1}{\rho_{xf}} \right) \left[ \tilde{\omega}_{xf,C}^{(I)} + \left( \frac{\hat{\rho}_{xf}}{\hat{P}} \right) \left( \frac{U}{2} \right) \left( (1/2)^{(\lfloor n \rfloor - I + 1)} \right) \right] \quad (5.17)$$

d. Parallel runtime for  $c_{xf}$  to finish its last stage ( $t_D$ )

In this part, we will find how much work is left over for other computing nodes except  $c_{xf}$  to execute. If  $c_{xf}$  is already in the decreasing phase, we have

$$\omega'_D = \omega'_C - \tilde{\omega}_C \quad (5.18)$$

yields

$$\tilde{\omega}_{xf,D} = \min(\omega'_D, \left( \frac{U}{2} \right) \left( \frac{\hat{\rho}_{xf}}{\hat{P}} \right) (1/2)^{\lfloor n \rfloor - I + 1}) \quad (5.19)$$

results in

$$t_D = \frac{\tilde{\omega}_{xf,D}}{\rho_{xf}} \quad (5.20)$$

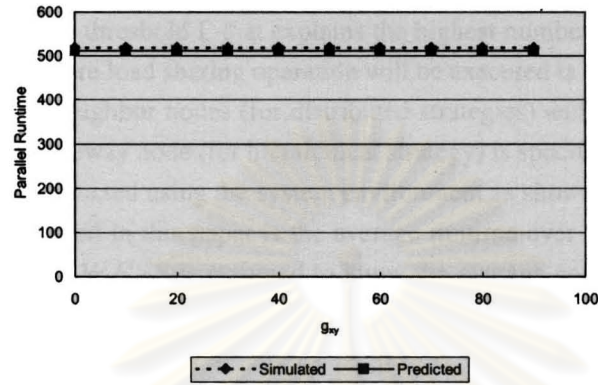
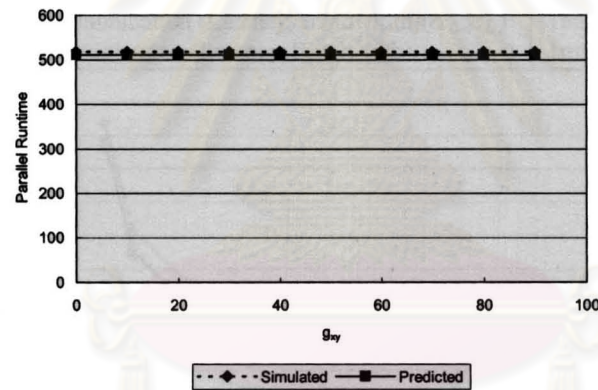
By combining all of the parallel runtime from every part together, we can have an overall predicted parallel runtime. From (5.8) and (5.9), we can see that the remain tasks at  $c_{xf}$  in phase  $B$  will increase as the relative power of  $\hat{\rho}_{xf}$  and total estimated computing power ( $\hat{P}$ ) increases. This behavior is a result of how the consuming rate of the front-end node ( $\hat{\rho}_{xf}$ ) is calculated differently from other computing node. Therefore, it will cause other computing nodes to ask for more work from the remaining tasks in the system. If there is not enough work to be assigned, a load imbalance will occur. This behavior will appear again during part  $C$  where other node will need more work given an increasing overestimation as in (5.14) and (5.16). Therefore, we can say that an overestimation within consuming rate can also cause an additional parallel runtime due to a load imbalance as occurs in implicit information. However, the main difference between these two strategies is in the origin of an overestimation. An overestimation in explicit information may come from an incomplete information about the computing system or the application while an overestimation in our implicit strategy is come from a different behavior between front-end and computing nodes when they make a request for more work.

We compare the parallel runtime of our implicit strategy by comparing between the prediction model and the simulated experiments using the system environment as shown in Table 5.1.

Figure 5.2 and 5.3 show that our prediction model can accurately estimate the parallel performance of  $CRSS$ . Moreover, both prediction model and simulated experiments indicate that

Table 5.1: The parameters for simulating *CRSS* in single cluster environment.

Variable Names	Values
Number of tasks ( $U$ )	16,384
Total number of compute nodes ( $N$ )	32
Number of clusters ( $L$ )	1
Intra-cluster communication (Latency, Bandwidth) ( $\alpha_L, \beta_L$ )	1ms , 100Mb/s

Figure 5.2: The parallel runtime of *CRSS* ( $r_{xy}=0.03125$ ).Figure 5.3: The parallel runtime of *CRSS* ( $r_{xy}=0.5$ ).

an information inaccuracy within explicit information is not related to the parallel performance of implicit strategy at all. We will further compare the parallel performance of our proposed strategy with other explicit strategies regarding to other parameters such as with different application patterns in the next section.

## 5.2 The Simulated Experiments

### 5.2.1 Load Sharing Strategies with Different Communication Structures

In this section, we evaluate the performance of load sharing strategies with different communication structures by varying the total number of computing nodes. In the simulated environment, there is a half of computing nodes which has computing power two times larger than the rest. The communication network in both between and within subgroups is considered as LAN.

Both  $C - CSS$  and  $CSS$  represent chunk self-scheduling strategy which assign workload constantly as 8 tasks per request.  $C - CSS$  is an example of centralized load sharing strategy which has only one coordinator node in the system. Every computing node must make a request directly to this node. As for  $CSS$  which is hierarchical strategy, the gateway nodes in a lower layer which have  $n$  descendants will receive  $n * 8$  tasks per request.  $WFSS$  and  $CRSS$  mentioned in this section are also a hierarchical strategy which use the same strategy in both upper and lower level. For distributed strategies( $LM$ ), the load sharing operation is assumed to be performed once every 4 tasks are executed and the threshold  $\Gamma$  that explains the highest number of workload difference between two neighbors before load sharing operation will be executed is also specified as 4 tasks. Moreover, the number of neighbor nodes (for distributed strategies) and the number of member nodes that join the same gateway node (for hierarchical strategy) is specified as 8 nodes. The simulated experiments are conducted using the system environment as shown in Table 5.2. Note that the parallel runtime presented in this paper is the average runtime over 20 simulation runs with different random seeds and  $WFSS$  is assumed to know the average computing power of every computing node before an execution. Therefore,  $WFSS$  has an advantage over other strategies.

Table 5.2: The system parameters for evaluating the effect of number of computing nodes.

Variable Names	Values
Number of tasks ( $U$ )	16,384
Intra-cluster communication (Latency, Bandwidth)( $\alpha_L, \beta_L$ )	1ms , 100Mb/s
Inter-cluster communication (Latency, Bandwidth)( $\alpha_W, \beta_W$ )	1ms , 100Mb/s

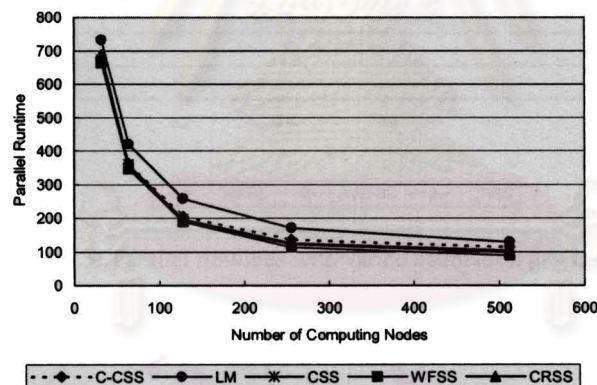


Figure 5.4: Parallel runtimes as a function of computing nodes.

Figure 5.4 illustrates the behavior of load sharing strategies given the computing system with different number of computing nodes. There are two major observations from this figure. First, since we use local network communication as a communication structure in this experiment, the centralized strategy ( $C - CSS$ ) can perform much better than the distributed strategy ( $LM$ ). However, the parallel runtime of the centralized strategy is worse than the hierarchical load sharing strategies due to the congestion at the coordinator node. Second, the hierarchical extension of our implicit load sharing strategy ( $CRSS$ ) can perform as good as the hierarchical extension of weighted factoring ( $WFSS$ ). Keep in mind that  $WFSS$  benefits the most from this setting as it is assumed to have the perfect knowledge of the computing power of each computing node.

### 5.2.2 Load Sharing Strategies Utilizing Explicit Information

In this section, we compare the parallel runtime of *WFSS*, *AWFSS*, and *CRSS* over various computing environment using different application patterns. The parameters used in our simulation are shown in Table 5.3 (unless stated otherwise).

Table 5.3: The parameters for simulating single cluster environment.

Parameter	Values
Number of tasks ( $U$ )	16,384
Total number of compute nodes ( $N$ )	32
LAN (Latency, Bandwidth)	1ms , 100Mb/s
Computing ratio of node $c_{xy}(r_{xy})$	0.03125
Estimation gap ratio of node $c_{xy}(g_{xy})$	0.3

First, we evaluation the effect of information inaccuracy by varying the estimation gap ratio ( $g_{xy}$ ) of computing node  $c_{xy}$ . Note that the computing power of every node will remain the same in all experiments.

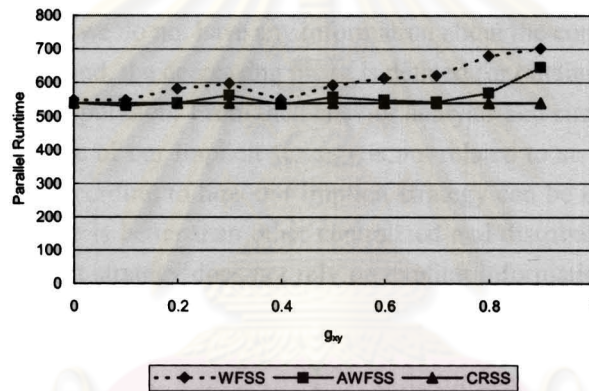


Figure 5.5: Parallel runtimes with varied estimation gap ratio.

From Figure 5.5, we can see that the parallel runtime of both *WFSS* and *AWFSS* will increase according to an information inaccuracy in the computing system. However, since *CRSS* does not utilize explicit information for making a load decision, its parallel performance is not effected by the varied different ratios. Moreover, the parallel runtime of *CRSS* is still comparable to *WFSS* and *AWFSS* when an information accuracy is low while its parallel performance is much better than *WFSS* and *AWFSS* given a large information inaccuracy.

To evaluate the parallel performance of different strategies in heterogeneous computing system, we simulate several experiments with varied computing ratio of  $c_{xy}(r_{xy})$ .

Figure 5.6 shows that *WFSS* achieves better parallel runtime when the computing power of an overestimated node dominate the entire cluster. This behavior comes from how *WFSS* and *CRSS* estimate the computing power. While an overestimation of *WFSS* is originated from an information inaccuracy, an overestimation in *CRSS* is a product of both computing power and communication latency. Therefore, an overestimation in *CRSS* will increase with  $\rho_{xy}$  resulting in a worse parallel runtime.

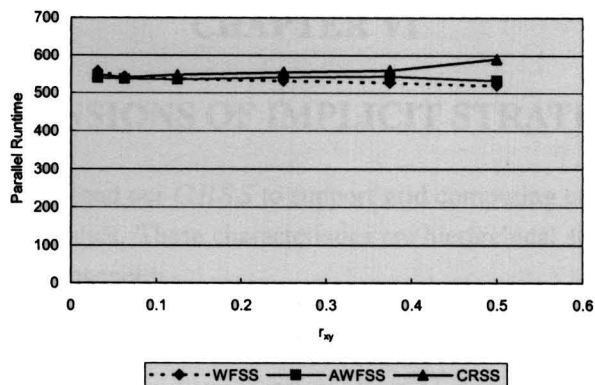


Figure 5.6: Parallel runtimes with varied computing ratio.

### 5.3 Conclusion

Our proposed implicit strategy (*CRSS*) consists of two phases of computation to address the unique characteristics of consuming rate. The increasing phase is defined for obtaining an accurate consuming rate since we do not have any information about the computing resource before an execution. In the other hand, the decreasing phase is defined for obtaining sub-optimal parallel runtime near the end of computation. From the behavior analysis and simulated experiments, we can see that the performance of our implicit strategy is not related to an information inaccuracy of explicit information. According to how our implicit strategy can be extended as a hierarchical strategy, its performance is better than other centralized and distributed strategy in the past. Moreover, since our implicit strategy does not rely on explicit information at all, the estimation gap in explicit information does not degrade the performance of our strategy. In a single cluster environment, *CRSS* can obtain a comparable parallel runtime as other explicit strategies. It can even achieve a better result when there is a large estimation gap occurred in the computing system. However, basic *CRSS* will have the worst parallel runtime over highly heterogeneous system. This issue will be addressed later by introducing an extension of implicit strategy for a computing system with large computing heterogeneity.

Despite some limitations, our proposed implicit strategy has many attractive features which are suitable for grid computing system. It can be used as an effective load sharing strategy without adding additional complexities into the underlying system. Therefore, our implicit strategy might be the right solution for today's large scale computing system.



## CHAPTER VI

### EXTENSIONS OF IMPLICIT STRATEGY

In this chapter, we extend our *CRSS* to support grid computing environment by considering grid's unique characteristics. These characteristics are hierarchical structure, large latency in WAN, and computing heterogeneity.

#### 6.1 Hierarchical Structure

As grid usually consists of multiple clusters, the topology of grid is hierarchical by nature. This hierarchical structure can degrade the performance of traditional load sharing strategies significantly. For example, load sharing strategies in [26][27][25] suffer from large communication overhead and information inaccuracy problems. This is because every load sharing operation must perform across WAN since the virtual binary tree created by coupling the fast and slow computing resources together. Moreover, hierarchical structure also causes load imbalance problem to distributed load sharing strategy since each computing node within the computing cluster can only communicate over LAN.

##### 6.1.1 CRSS Extension for Hierarchical Structure

To support the hierarchical structure of multi-cluster environment, *CRSS* organizes the coordinator node and all front-end nodes in hierarchical fashion. Figure 6.1 illustrates an example of hierarchical *CRSS* in grid computing environment. In this structure, the coordinator distributes workload to other front-ends and the front-ends distribute workload to their working nodes. *CRSS* can be applied at both levels without any modifications. Under this configuration, the coordinator node treats each cluster as a single working node represented by the front-end of that cluster. Once the front-end receives new workload from the coordinator, it will define a new sequence of stage size. This stage size will be used to determine a chunk size for its local computing nodes afterward.

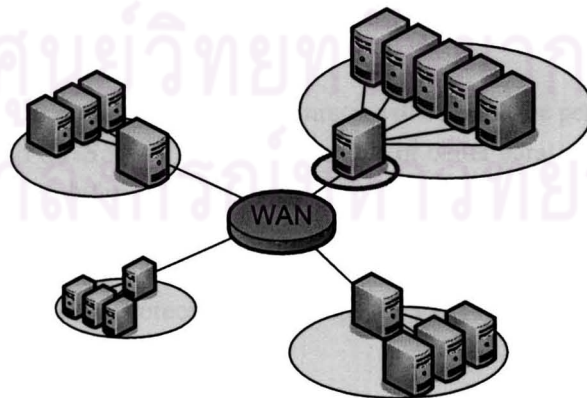


Figure 6.1: Hierarchical structure in grid computing environment.

### 6.1.2 Performance Evaluation

To demonstrate the effectiveness of the extensions, we conduct simulations using parameters specified in Table 6.1 (unless stated otherwise). In this section, we compare the performance of our *CRSS-SW* with *CRSS* and *AWFSS*, which are representatives of implicit and explicit strategy. The experiment results are simulated using Network Simulator (*NS*) [32].

Table 6.1: The parameters for simulating multiple cluster environment.

Parameter	Values
Number of tasks ( $U$ )	16,384
Total number of compute nodes ( $N$ )	64
Number of clusters ( $L$ )	4
Intra-cluster communication (Latency, Bandwidth)( $\alpha_L, \beta_L$ )	1ms , 100Mb/s
Inter-cluster communication (Latency, Bandwidth)( $\alpha_W, \beta_W$ )	30ms , 2Mb/s
Computing ratio of overestimated cluster ( $G_x$ )	0.0625
Estimation gap ratio of overestimated cluster ( $R_x$ )	0.3

In this section, we conduct the simulated experiments over a multiple cluster environment. We compare the performance of *CRSS* with the *hetero-AWFSS*, which is the extensions of *AWFSS* to support the hierarchical structure. First, we compare two strategies over a computing system with information inaccuracy by varying gap ratio ( $G_x$ ).

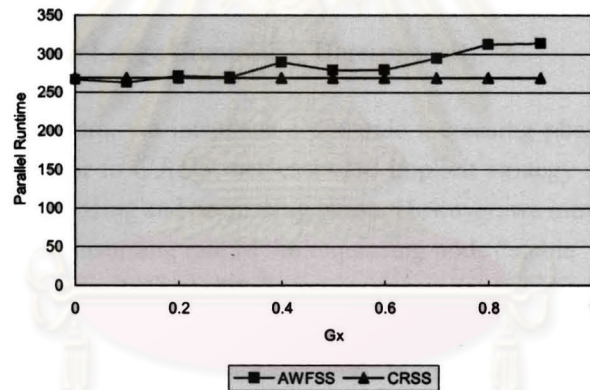


Figure 6.2: Parallel runtime with varied estimation gap ratio.

Figure 6.2 shows that an information inaccuracy will degrade the performance of *AWFSS*. Therefore, our proposed *CRSS* will have a comparable or better parallel runtime than *AWFSS* especially when there is an information inaccuracy occurred in the system.

To simulate computing heterogeneity, we vary the computing power of  $p_x$  by specifying different computing ratio ( $R_x$ ). Moreover, we also specify gap ratio as 0.3 to simulate information inaccuracy in the computing system.

Figure 6.3 shows that the parallel runtime of *AWFSS* is better than *CRSS* when an overestimated cluster has a very small computing ratio. By varying the computing power of an overestimated cluster, we can see that the performance of *CRSS* is bad only when there is a large computing heterogeneity between each cluster. This issue related to computing heterogeneity will be addressed in the following section.

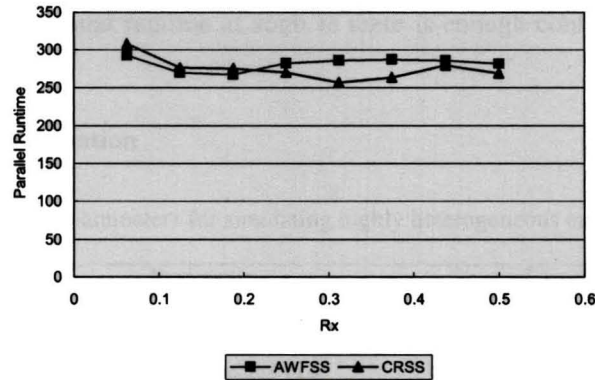


Figure 6.3: Parallel runtime with varied computing ratio.

## 6.2 Large Computing Heterogeneity

With grid's openness, the differences in computing powers between grid resources will grow over time. This gap of differences in the computing powers really increases the risk of facing the load imbalance problem. Unintentionally sending one more job to the computing node with one hundred times slower than other computing nodes will result in a very bad parallel runtime despite of the total computing power which has been dedicated to the submitted application.

### 6.2.1 CRSS Extension for Large Computing Heterogeneity

To address this problem, we introduce a dynamic increasing phase to specify stage size during an execution. Similar to *CRSS*, our extended implicit strategy still uses two phases of execution consisting of increasing and decreasing phase. However, we introduce a new term called "stable rate". We call the consuming rate of the requesting node "stable" only when an obtained consuming rate is within a predefined differences percentage ( $\epsilon$ ) from the consuming rate in a previous stage given a two times larger chunk size assigned in the previous stage. According to our preliminary experiments, we will specify  $\epsilon$  as 25%. With this way, we can assure that we have already assigned large enough chunk size for the requesting node when we obtain the stable rate. The behavior of our dynamic increasing phase is defined as follows:

- a. At the beginning, each cluster will be given a single task per request.
- b. The chunk size for the requesting nodes will be increased with a constant ratio of 2 regardless of their consuming rates until an obtained consuming rate become stable.
- c. The requesting node whose consuming rate is already stable will receive the same chunk size as in the previous stage.
- d. Repeat b and c until the consuming rate of every node becomes stable or the remaining tasks are below half of the total tasks. Then, the phase of execution will change to the decreasing phase.

With this modified strategy we do not risk assigning too much tasks related to how we specified stage size before an execution begin. A decreasing chunks will be sent to each node

in order to obtain near optimal runtime as soon as there is enough confident in every collected consuming rate.

### 6.2.2 Performance Evaluation

Table 6.2: The parameters for simulating highly heterogeneous environment.

Parameter	Values
Number of tasks ( $U$ )	16,384
Total number of compute nodes ( $N$ )	64
Number of clusters ( $L$ )	4
Intra-cluster communication (Latency, Bandwidth)( $\alpha_L, \beta_L$ )	1ms , 100Mb/s
Inter-cluster communication (Latency, Bandwidth)( $\alpha_W, \beta_W$ )	30ms , 2Mb/s
Computing ratio of overestimated cluster ( $G_x$ )	0.0625

We evaluate the parallel runtime of this extension of implicit strategy by varying the relative computing power ratio in the system. Its parallel runtime will be compared with our base implicit strategy.

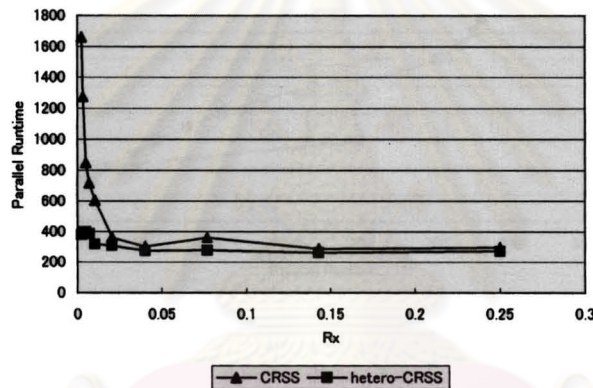


Figure 6.4: Parallel runtime of implicit strategies with different relative power.

As shown in Fig. 6.4, *CRSS* obtains bad parallel runtime when the computing power of the specified cluster is very small in comparison with other clusters. Computing heterogeneity increases the effect of load imbalance at the end of computation especially when there are some remaining works left in the specified cluster. *Hetero-CRSS* addresses this problem by not using the obtained consuming rates until they are stable. Thus, load imbalance from an over-estimated information is reduced.

### 6.3 Inaccurate Information

Inaccurate estimators of the computing resources can cause both explicit and implicit strategies to allocate too much workload for each stage. This leads to performance degradation since over-estimated clusters can not catch up with other clusters in the later stages, which results to load imbalance near the end of computation. Moreover, this problem becomes worse in grid system as the underlying resources are heterogeneous. Figure 6.5 illustrates the situation when the execution ends while cluster *A* is still in its increasing phase. Thus, *CRSS* will obtain a very bad parallel runtime. From Fig. 6.6, we can see that the stage number of cluster *A* is far behind

those of other clusters. This behavior is a result of inaccurate information which can be occurred in *AWFSS* as shown in Fig. 6.7.

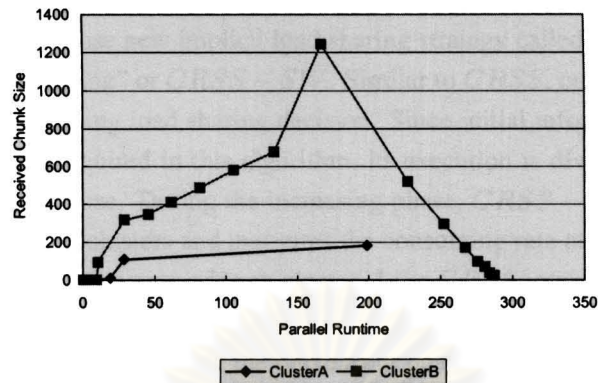


Figure 6.5: Chunk assignment of *CRSS*.

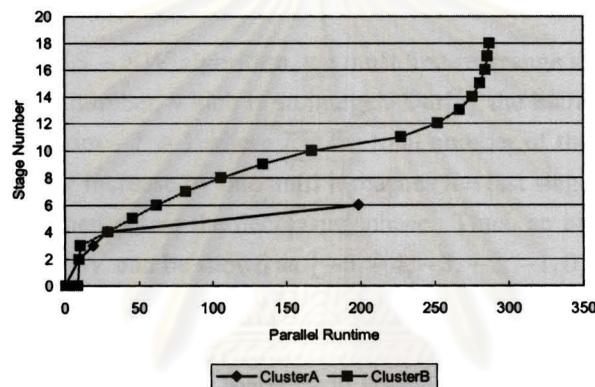


Figure 6.6: Stage number during runtime of *CRSS*.

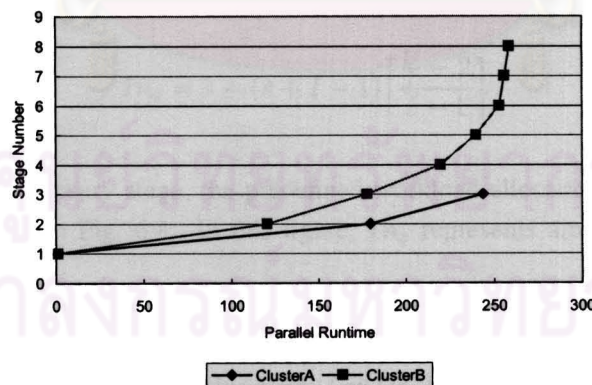


Figure 6.7: Stage number during runtime of *AWFSS*.

### 6.3.1 CRSS Extension for Inaccurate Information

To address this practical issue, we introduce stage-warping approach for implicit load sharing strategy. This new concept adjusts an incorrect assignment by allowing clusters to skip their predefined stages to catch up with other clusters during the execution. The performance of pro-

posed implicit strategy is compared to both explicit and implicit strategies with different classes of applications over simulated grid computing environment.

In this paper, we propose new implicit load sharing strategy called ‘‘Consuming Rate Self-Scheduling with Stage Warping’’ or *CRSS – SW*. Similar to *CRSS*, our *CRSS – SW* utilizes implicit information for making load sharing decision. Since initial information regarding to the system capabilities is not required in this algorithm, its execution is divided into the increasing phase and the decreasing phase. During the increasing phase, *CRSS – SW* assigns half of the total number of unit tasks to clusters and measures the consuming rate at the coordinator node to estimate the actual computing power of each cluster. Like *CRSS*, our *CRSS – SW* allocates chunk size during the increasing phase in a linear fashion to ensure that it obtains accurate consuming rates. *CRSS – SW* then enters the decreasing phase to balance the workload between each cluster for the remaining half of the total number of unit tasks.

### 6.3.1.1 Basic Algorithm

To describe the *CRSS – SW* algorithm, we must first rearrange stage numbers during the execution. Let  $s$  be a stage number, which is an integer. During the increasing phase,  $s$  is a non-positive number, starting from  $-I + 1$  where  $I$  is the total number of the increasing stages. The stage number will gradually increase by one until it reaches the last stage of the increasing stage  $\{s = 0\}$ . Our algorithm then enters the decreasing phase. Thus, an example of stage number sequence of our *CRSS – SW* can be shown as  $\{-5, -4, -3, -2, -1, 0, 1, 2, 3, 4\}$ .

In addition to rearrange the stage numbers, we also specify the predefined stage size which is the number of unit tasks that will be assigned to clusters during each stage. Let  $In_s$  be the predefined amount of workload in stage  $s$  of the increasing phase. The value of  $In_s$  can be calculated as shown in Eq. (6.1).

$$In_s = 1 + (s + I - 1) \left\lceil \frac{U - 2}{I - 1} \right\rceil \quad (6.1)$$

As  $In_s$  is doubled every stage, the predefined workload allocation during the increasing phase can be illustrated in Fig. 6.8. In this figure,  $In_s$  represents amount of workload to be completed at the stage  $s$ .

As for the decreasing phase, we define the decreasing chunk sizes with a constant ratio  $\delta$  which will be specified as 2 throughout this work. Let  $De_s$  be the predefined amount of workload for stage  $s$  of the decreasing phase. Equation (6.2) illustrates how we calculate  $De_s$  with respect to the total unit tasks ( $U$ ). The predefined workload allocation in all stages during the decreasing phase is illustrated in Fig. 6.9.

$$De_s = \left\lceil \frac{U}{2 * \delta^s} \right\rceil \quad (6.2)$$

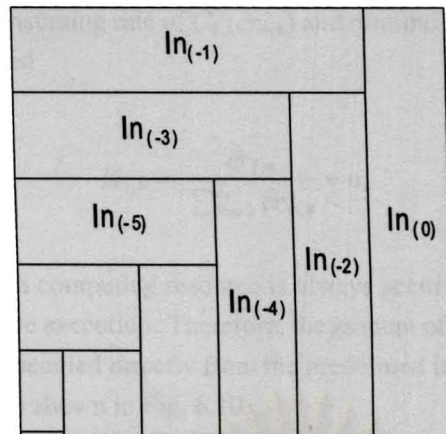


Figure 6.8: Predefined amount of workload allocated in each stage during the increasing phase.

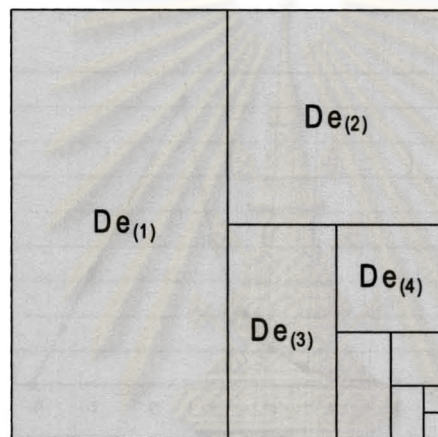


Figure 6.9: Predefined amount of workload allocated in each stage during the decreasing phase.

With these definitions, we can now explain our *CRSS – SW* algorithm. As mentioned earlier, our *CRSS – SW* consists of two phases, the increasing phase and the decreasing phase. the goal of the increasing phase is to obtain accurate consuming rate of each cluster. However, estimating the computing powers by measuring consuming rates with small number of tasks can be very inaccurate and misleading since the computing nodes in the clusters may not be fully utilized. To improve the accuracy of the estimation, we divide the increasing phase into two sub-phases. The first sub-phase is to find a stable consuming rate of every cluster before we adjust it in the second sub-phase. We first start assigning small chunk sizes to clusters and double the chunk sizes exponentially. During this sub-phase, we keep measuring the consuming rate of each cluster at the coordinator node until it becomes stable without using the obtained consuming rate at all. A consuming rate is considered stable only if it does not increase upon an increasing chunk size. The cluster that is considered stable will receive an equal chunk size while waiting for other clusters to become stable. After the consuming rate of every cluster is stable, *CRSS – SW* enters the second sub-phase of the increasing phase to further adjust the obtained consuming rate.

Through out the second sub-phase of the increasing phase and the entire decreasing phase, the size of the workload that the coordinator node will assign to the cluster  $C_i$  ( $K_{i,s}$ ) will be cal-

culated from the averaged consuming rate of  $C_i$  ( $\bar{c}r_{i,s}$ ) and runtime amount of workload allocated during stage  $s$  ( $u_s$ ) as followed

$$K_{i,s} = \frac{\bar{c}r_{i,s}}{\sum_{k=1}^L \bar{c}r_{k,s}} * u_s \quad (6.3)$$

If the estimator of each computing resource is always accurate, every cluster will arrive at the same stage during an entire execution. Therefore, the amount of workload allocated for stage  $s$  during runtime ( $u_s$ ) can be specified directly from the predefined increasing and decreasing stage sizes according to Eq.(6.4) as shown in Fig. 6.10.

$$u_s = \begin{cases} In_s & \text{if } s \leq 0 \\ De_s & \text{otherwise} \end{cases} \quad (6.4)$$

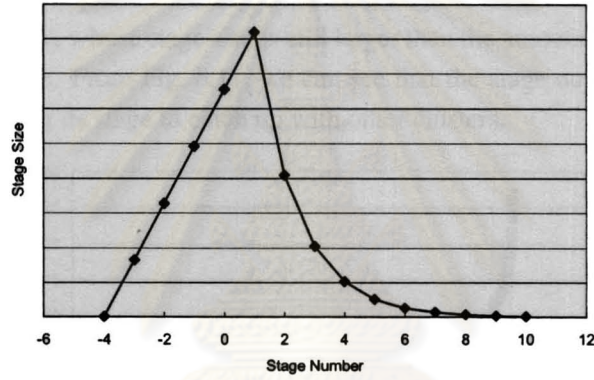


Figure 6.10: Chunk size assignment of  $CRSS - SW$  in ideal case.

### 6.3.1.2 Stage Warping

In the ideal case, all clusters will progress to each stage at the same pace. In other words, all clusters will enter the same stage  $s$ , complete all tasks allocated in that stage ( $In_s$  and  $De_s$ ), and move to the next stage ( $s + 1$ ) at the same time. In reality, this will never happen. Some clusters will enter the later stages before other clusters. To solve this problem, our  $CRSS - SW$  uses a technique called stage-warping. This technique allows the clusters whose stages are behind others to skip (or warp) from their current stages to the foremost stage which other clusters have already reached.

The objective of stage-warping is to define runtime stage sizes based on predefined stage sizes of both the increasing and the decreasing phase. These runtime stage sizes include the remaining workload in the previous stages. Every cluster will request for more workload from the foremost stage. With this behavior, the leftover workload in the previous stages will be re-assigned again to every cluster and the effect of inaccurate estimators which causes some clusters to stay behind can be reduced. Each time one of the computing clusters enters the new foremost stage, the runtime stage size will be defined using the predefined stage sizes ( $In_s$  or  $De_s$ ) together with the number of total tasks ( $U$ ) and the remaining tasks at the beginning of stage  $s$  ( $\omega'_s$ ). The



runtime stage size ( $u_s$ ) in the increasing phase will be specified as shown in Eq. (6.5).

$$u_s = In_s + \sum_{j=-I+1}^{s-1} In_j - U + \omega'_s \quad (6.5)$$

As for the runtime decreasing stage sizes, they will be calculated considering the predefined decreasing stage size ( $De_s$ ) and  $\omega'_s$  as shown in Eq. (6.6).

$$u_s = \begin{cases} De_s + (\frac{U}{2} + \omega'_s - U) & \text{if } s = 1 \\ De_s + (\frac{U}{2} + \sum_{k=1}^{s-1} De_k + \omega'_s - U) & \text{if } s > 1 \end{cases} \quad (6.6)$$

Note that there is the first stage warping which will occur immediately after the first sub-phase in the increasing phase. This first stage warping will make every cluster to enter their second sub-phase from the same stage number. The beginning stage of the second sub-phase is specified as the smallest stage in  $In_j$  whose stage size is still larger than the amount of workload distributed during the first sub-phase. From Fig. 6.11, we can see that the stage number of cluster A skips from time to time keeping its stage to catch up with other clusters.

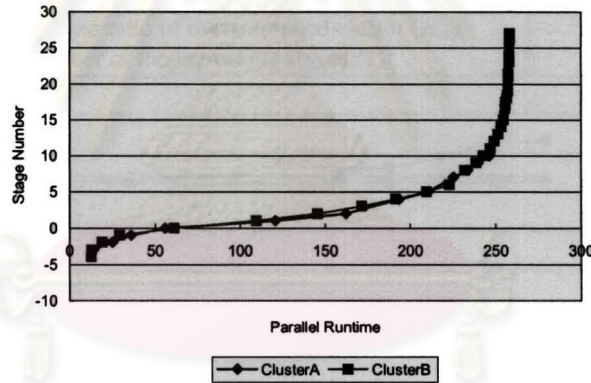


Figure 6.11: Stage number during runtime of  $CRSS - SW$ .

The resource utilization of  $CRSS - SW$  compared with  $CRSS$  over highly heterogeneous computing system is shown in Fig. 6.12. We can see that  $CRSS - SW$  can increase the utilization during the increasing phase while it also reduces load imbalance near the end of computation.

### 6.3.2 Performance Evaluation

We will compare the performance of  $CRSS - SW$  with  $AWFSS$  and  $CRSS$  with different estimation gap and computing heterogeneity. We begin with varying the estimation gap ( $G_x$ ) of computing cluster  $C_x$  according to the system parameter in Table 6.3.

From Fig. 6.12, we can see that the parallel runtime of  $AWFSS$  will become worsen with an increasing estimation gap while the parallel runtime of our implicit strategies are unvaried as we expected. Note that the performance of  $CRSS - SW$  is slightly better than  $CRSS$  even in a

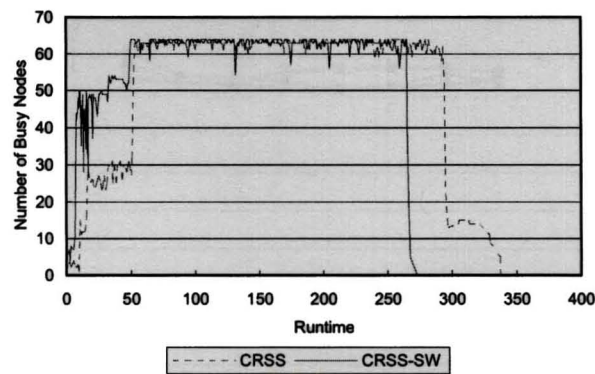


Figure 6.12: Utilization graph of *CRSS* and *CRSS – SW*.

Table 6.3: The parameters for evaluating the performance of *CRSS – SW*.

Parameter	Values
Number of tasks ( $U$ )	16,384
Total number of compute nodes ( $N$ )	64
Number of clusters ( $L$ )	4
Intra-cluster communication (Latency, Bandwidth)( $\alpha_L, \beta_L$ )	1ms , 100Mb/s
Inter-cluster communication (Latency, Bandwidth)( $\alpha_W, \beta_W$ )	30ms , 2Mb/s
Computing ratio of overestimated cluster ( $G_x$ )	0.0625
Estimation gap ratio of overestimated cluster ( $R_x$ )	0.3
Number of the increasing stages ( $I$ )	20

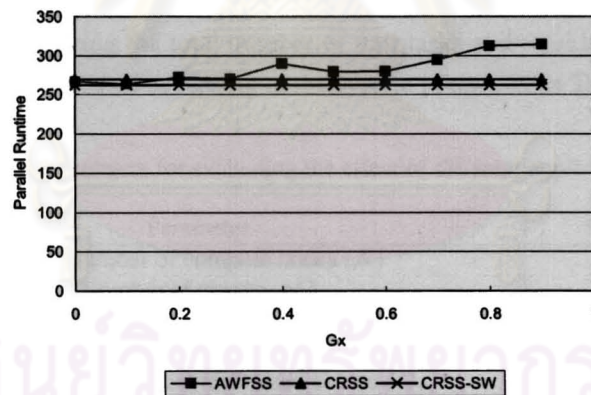


Figure 6.13: Parallel runtime of *AWFSS*, *CRSS* and *CRSS – SW* with varied estimation gap.

homogeneous environment.

Figure 6.14 illustrates the performance of load sharing strategies with different computing heterogeneities. The simulated results indicate that both *AWFSS* and *CRSS* can become highly fluctuate with different computing ratio ( $R_x$ ) of computing cluster  $C_x$ . In the other hand, *CRSS – SW* can achieve the best results in most cases and its parallel runtime barely change at all. This behavior can be implied that *CRSS – SW* can tolerant to computing heterogeneity and inaccurate information than *CRSS* throughout an entire execution.

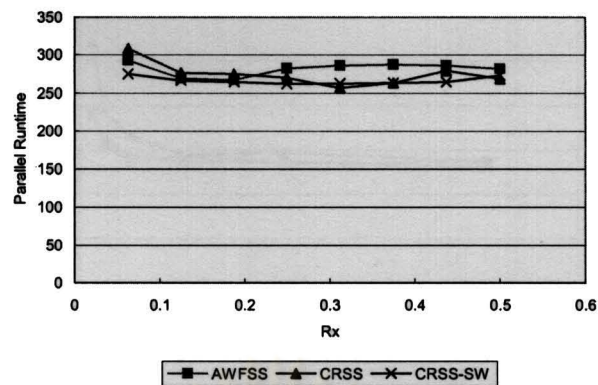


Figure 6.14: Parallel runtime of *AWFSS*, *CRSS* and *CRSS-SW* with different computing heterogeneity.

## 6.4 Application Classes

Since grid technology is introduced as a computing framework for computing-intensive applications, load sharing strategy must be able to handle workload from different classes of applications effectively.

### 6.4.1 Performance Evaluation

In this section, we will study the performance of load sharing strategies with different application classes by changing the total number of unit tasks and workload pattern. Simulated computing environment is created according to the system parameter in Table 6.4.

Table 6.4: The parameters for evaluating the effect of different application classes.

Parameter	Values
Total number of compute nodes ( $N$ )	64
Number of clusters ( $L$ )	4
Intra-cluster communication (Latency, Bandwidth)( $\alpha_L, \beta_L$ )	1ms , 100Mb/s
Inter-cluster communication (Latency, Bandwidth)( $\alpha_W, \beta_W$ )	30ms , 2Mb/s
Computing ratio of overestimated cluster ( $G_x$ )	0.0625
Estimation gap ratio of overestimated cluster ( $R_x$ )	0.3

#### 6.4.1.1 Applications with Different Number of Unit Tasks

The number of unit tasks in the submitted application is actually one of the most important factors that can effect the performance of our implicit strategy. Accurate consuming rates can not be obtained when there is only a limited number of unit tasks. In this section, we will compare the parallel runtime of both *CRSS* and *CRSS-SW* with *AWFSS* over different number of unit tasks.

From Fig. 6.15, we can see that while *CRSS* obtains a very bad parallel runtime when number of unit tasks is limited. However, *CRSS-SW* can still achieve a comparable performance to *AWFSS*. These simulated results indicate that with proper defined extensions, our implicit strategy can handle even the application with small number of unit tasks.

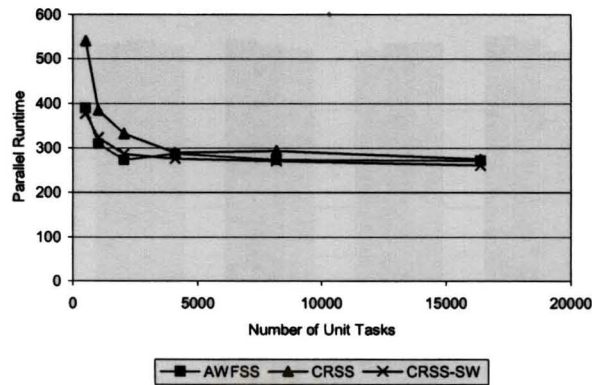


Figure 6.15: Parallel runtime of application with different unit tasks.

#### 6.4.1.2 Applications with Different Workload Patterns over Homogeneous System

We evaluate the parallel performance of both explicit and implicit strategies including *AWFSS*, *CRSS* and *CRSS – SW* over four application classes. We first perform the simulated experiments assuming ideal environment such that there is no estimation error ( $G_x = 0$ ). The obtained results are illustrated in Fig. 6.16.

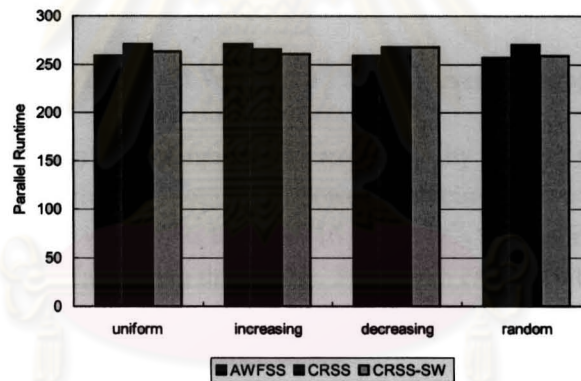


Figure 6.16: Parallel runtime of various applications ( $G_x = 0$ ).

As shown in Fig. 6.16, our proposed algorithm, *CRSS – SW* can achieve a comparable parallel runtime compared to *AWFSS*'s under ideal environment. *CRSS – SW* performs even better especially when the application class is the increasing workload pattern. Obviously, the applications with increasing pattern have larger computation sizes near the end of computation. This tends to increase load imbalance. As *AWFSS* and *CRSS* do not employ the stage-warping technique, they may assign too large chunk sizes near the end of computation. For *CRSS – SW*, the stage-warping technique allows the under-estimated clusters to steal workload of the over-estimated ones while preventing the coordinator node to assign too large chunk sizes to these clusters. On the contrary, *AWFSS* performs best when the workload pattern of the submitted application is decreasing. As expected, application with decreasing computation size tends to reduce load imbalance issues near the end of computation.

Figure 6.17 shows the experimental results of non-ideal environment where there is an es-

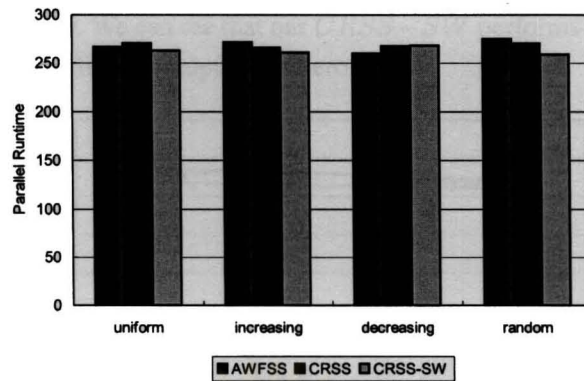


Figure 6.17: Parallel runtime of various applications ( $G_x = 0.3$ ).

timization gap in the system. It is very clear that our *CRSS – SW* performs better than *AWFSS* in most cases especially with increasing workload pattern. Without accurate explicit information, *AWFSS* performs poorly as it distributes workload based on incorrect assumptions. For *CRSS* and *CRSS – SW*, the increasing phase allows both algorithms to obtain the accurate estimations of the underlying system. However, *CRSS – SW* performs slightly better as it distributes workload faster and finds the stable estimation during the increasing phase, as well as, utilizes the stage-warping for the entire execution.

#### 6.4.1.3 Applications with Different Workload Patterns over Heterogeneous System

We study the effects of the computing heterogeneity by varying the computing power of cluster  $C_h$ , which is  $p_h$ . As the total computing power of all clusters is 64, the underlying system is considered heterogeneous when  $p_h$  is less than 16 (cluster  $C_h$  is slower than the others) or  $p_h$  is more than 16 (cluster  $C_h$  is faster than the others). Since both cases exhibit similar behaviors, we will present only the results of the experiments with  $p_h$  is less than 16. Note that we also assume the estimation gap ( $G_x$ ) to be 0.3 in all experiments.

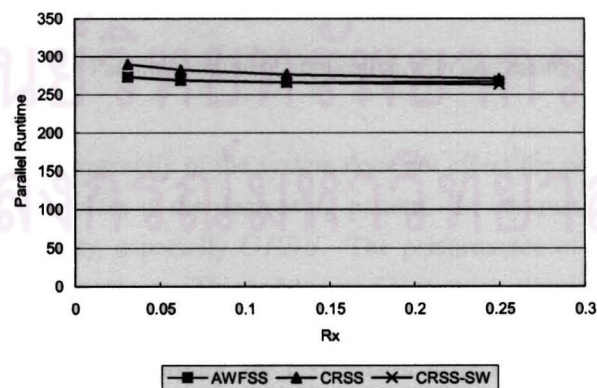


Figure 6.18: Parallel runtime of application with uniform pattern.

Figure 6.18 shows the performance of three load sharing strategies with constant-workload application. Although every strategy does not fluctuate much with different computing heterogeneity given a constant-workload, *CRSS* tends to have worse parallel runtime in according to

the computing heterogeneity. We can see that our *CRSS – SW* performs slightly better than both *AWFSS* and *CRSS* on different computing heterogeneity.

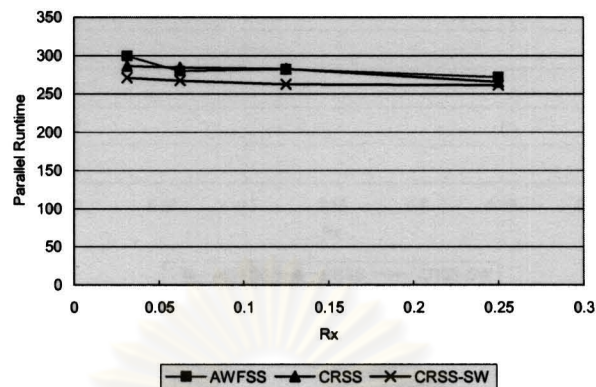


Figure 6.19: Parallel runtime of application with increasing pattern.

When the application class is the increasing-workload pattern as presented in Fig. 6.19, however, *CRSS – SW* can achieve a noticeable improvement over *AWFSS* and *CRSS*. This is very similar to the results of the homogeneous system presented in the previous section. The inaccuracy of explicit information can severely decrease the performance of *AWFSS*.

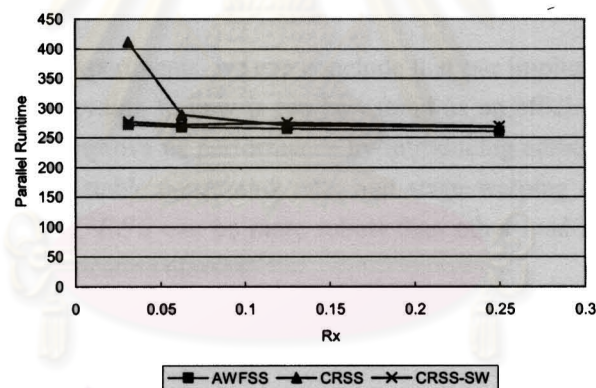


Figure 6.20: Parallel runtime of application with decreasing pattern.

In Fig. 6.20, the heterogeneity of the system does not effect the performance of *CRSS – SW* when the application class is the decreasing pattern. In contrast, *AWFSS* and *CRSS* suffer from the heterogeneity, especially *CRSS*. The performance of *CRSS* becomes much worse as the heterogeneity increases. This is because  $C_h$  is much slower than the other clusters. Thus, it is in the middle of the increasing phase while others have already reached the end of the computation. Without the stage-warping, the coordinator node keeps assigning large chunks to the slower cluster  $C_h$  and allows other faster clusters to steal very little portions of  $C_h$  workload.

For the application with random-workload pattern, Fig. 6.21 shows that the performance of *AWFSS* decreases when the computing heterogeneity in the system increases. Note that the performance of *CRSS* and *CRSS – SW* converses when the heterogeneity is high.

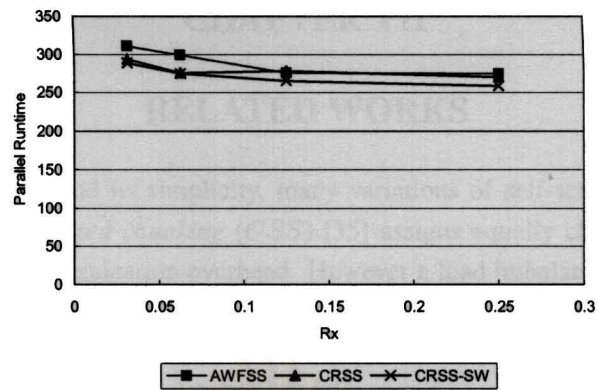


Figure 6.21: Parallel runtime of application with random pattern.

## 6.5 Conclusion

In this chapter, we introduce the extensions of implicit strategy for addressing unique characteristics of grid computing. These characteristics consist of cluster-based computing system, large communication overhead, and computing heterogeneity. In addition, we also focus on inaccurate information and different application classes which can effect the performance of load sharing strategies.

From the simulated experiments, we can conclude that our implicit strategy is suitable for grid computing system according to how it can be extend as an efficient hierarchical strategy. Moreover, we can further improve its performance by introducing additional extensions such as movable coordinator node, stable consuming rate, and stage warping concept. Together with proper defined extensions, *CRSS* can be more robust than other load sharing strategies in the past even with different application classes.

## CHAPTER VII

### RELATED WORKS

Due to its strength and its simplicity, many variations of self-scheduling strategies have been proposed. *Uniform-sized chunking (CSS)* [35] assigns equally chunk size ( $K$ ) tasks per request to reduce the communication overhead. However a load imbalance at the end of computation can significantly worsen the parallel runtime if we specify chunk size too large. *Guided self-scheduling (GSS)* [36] addresses this problem by allocating large chunks at the beginning of a computation while sending smaller chunks near the end of computation to achieve a better parallel runtime. The chunk size scheduled for the next idle node is the total remaining tasks divided by the number of available processors. For constant-length iterations and uneven starting times, this strategy is proofed that all computing nodes will finish within single iteration of each other. However, this strategy sometimes allocates too large chunk size in early stages. In *Trapezoid self-scheduling (TSS)* [37], the available tasks will be allocated to the requesting node with linearly decreasing chunk size. This technique reduces the risk of assigning too large first chunk for each computing node. *Fixed Increase Self-Scheduling (FISS)* [38] tries to overlap the communication and computation by sending an increasing chunk size instead.

The prefetching technique such as proposed in [39] has been proposed for addressing large latency in the communication network. This technique assumes that the access pattern of a submitted application can be obtained or predicted. Therefore, the coordinator node will be able to send workload to other clusters before those clusters actually finish all the work. With this way, every cluster can continue executing more work without requesting and waiting for a next batch of workload to arrive. However, this technique is not compatible with our implicit information which can only be obtained during an execution. Although we can add this prefetching technique into our implicit strategy by allowing clusters to request for more workload before they actually finish executing every unit task assigned during the previous stage, this behavior can increase load imbalance at the end of computation as a result of an over-estimated cluster still asking for more work even though there is a remaining work left in it. The prefetching technique can be efficiently utilized only when an information accuracy about the underlying system and the submitted application is high enough.

*RUMR* [40] is an extension of load sharing strategy named *UMR* or “Uniform Multi-Round” algorithm. *UMR* intends to hide the communication overhead with the computation time by sending an increasing chunk size. It begins with assigning a small chunk to every cluster which inflicts only a small communication overhead. Then, it sends out larger chunks during each round because all clusters also need more time before they can finish all the work assigned previously. Assuming that the computing power and communication bandwidth are constant and known before the execution begins, this strategy can calculate the chunk size to be distributed each round by specifying that every chunk in the same round is equal. While *UMR* is an increasing strategy, *RUMR* consists of both increasing and decreasing phase. *RUMR* adds the decreasing phase near the end of computation to make it more robust when there is an information inaccuracy occurring in the system. Note that although our proposed methodology of handling implicit information resembles to this strategy, the main purpose behind the idea is different. The proposed



methodology in [40] has an intention to overlapped communication and computation while our work uses an increasing phase to calculate the consuming rate of each computing resource.



ศูนย์วิทยทรัพยากร  
จุฬาลงกรณ์มหาวิทยาลัย

## CHAPTER VIII

### CONCLUSION

Grid computing has been widely used for aggregating computing resources across multiple organizations. In order to effectively utilize the available computing power in grid, we need to implement load sharing strategy which increases an overall parallel performance by ensuring that each resource will receive workload proportionally to its computing power. In this work, we propose a new load sharing strategy utilizing implicit information called “implicit load sharing strategy”. Unlike explicit information which consists of many different parameters of the computing system or the application, implicit information is a single metric that can represent the computing power of each resource. While an explicit information will soon become impractical considering the complexity within the system which will continue to grow everyday, our implicit information will become an important metric for making a load decision. However, since the accuracy of the implicit information heavily depends on the chunk size allocated to the requesting node, our implicit strategy addresses this issue by defining two phases of computation, increasing and decreasing phase. At the beginning, the computing nodes will be assigned increasing chunk sizes for obtaining their consuming rates while they will receive decreasing chunk sizes near the end of computation to achieve sub-optimal runtime. Moreover, we also extend our implicit strategy to address the unique characteristics in grid computing environments. These characteristics consist of a cluster-based infrastructure, a large latency in WAN, and a computing heterogeneity between each computing resource. There are several concepts proposed in this work which can further improve the performance of implicit strategy such as movable coordinator node, stable rate, and stage warping. Based on the simulated experiments, our proposed strategy performs comparable or better than other popular strategies without using any explicit information.

Our future works will focus on two issues: node selection and fault tolerance. In some cases, it is more beneficial to choose only a subset of computing nodes from available candidates with respect to the submitted job and differences in computing powers. However, since our implicit information can only be obtained during an execution, we must carefully define how to choose the computing nodes. Moreover, we also want to add fault tolerance capacity to address the dynamic behavior in grid computing system because some computing nodes or clusters can be unreachable at anytime. Our implicit strategy may address this problem by sending a redundant workload to other available candidate nodes to obtain their consuming rate in advance.

## References

- [1] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid. *J. HPC Apps.* 15, 3(2001): 200–222.
- [2] I. Foster and C. Kesselman. The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco, CA, 1999.
- [3] Y. T. Wang and J. T. Morris. Load sharing in distributed systems. *IEEE Trans. Comp.* C-34, 3(1985): 204–217.
- [4] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *JPDC* 7(1989): 279–301.
- [5] M. Willebeek-LeMair and A. P. Reeves. Strategies for dynamic load balancing on highly parallel computers. *IEEE Trans. Parallel Distrib. Syst.* PDS-4, 9(1993): 979–993.
- [6] J. Balasubramanian, D. C. Schmidt, L. W. Dowdy, and O. Othman. Evaluating the performance of middleware load balancing strategies. In *EDOC*, 2004.
- [7] R. Biswas, M. A. Frumkin, W. Smith, and R. F. V. der Wijngaart. Tools and techniques for measuring and improving grid performance. In *IWDC*, 2002.
- [8] A. T. Chronopoulos, S. Penmatsa, and N. Yu. Scalable loop self-scheduling schemes for heterogeneous clusters. In *CLUSTER*, 2002.
- [9] E. Putrycz. Design and implementation of a portable and adaptable load balancing framework. In *CASCON*, 2003.
- [10] R. Wolski. Experiences with predicting resource performance on-line in computational grid settings. *SIGMETRICS* 30, 4(2003): 41–49.
- [11] G. Sabin, R. Kettimuthu, A. Rajan, and P. Sadayappan. Scheduling of parallel jobs in a heterogeneous multi-site environment. In *JSSPP*, 2003.
- [12] J. D. Teresco, J. Faik, and J. E. Flaherty. Resource-aware scientific computation on a heterogeneous cluster. *CiSE* 7, 2(2005): 40–50.
- [13] J. Cao, D. P. Spooner, S. A. Jarvis, S. Saini, and G. R. Nudd. Agent-based grid load balancing using performance-driven task scheduling. In *IPDPS*, 2003.
- [14] A. C. Arpaci-Dusseau, D. E. Culler, and A. M. Mainwaring. Scheduling with implicit information in distributed systems. In *SIGMETRICS*, 1998.
- [15] C.-T. Yang, K.-W. Cheng, and K.-C. Li. An efficient parallel loop self-scheduling on grid environments. In *NPC*, 2004.
- [16] D. J. Becker, T. Sterling, D. Savarese, J. E. Dorband, U. A. Ranawake, and C. V. Parker. Beowulf: A parallel workstation for scientific computation. In *ICPP*, 1995.
- [17] K. Shen, T. Yang, and L. Chu. Cluster load balancing for fine-grain network services. In *IPDPS*, 2002.
- [18] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. Evaluation of job-scheduling strategies for grid computing. In *GRID*, 2000.

- [19] S. P. Dandamudi and K. C. M. Lo. A hierarchical load sharing policy for distributed systems. Mascots 0(1997): 3.
- [20] T.-H. Kim and J. M. Purtilo. Load balancing for parallel loops in workstation clusters. In ICPP, 1996.
- [21] A. Hac and X. Jin. Dynamic load balancing in a distributed system using a decentralized algorithm. In ICDCS, 1987.
- [22] M. D. Feng and C. K. Yuen. Dynamic load balancing on a distributed system. In SPDP, 1994.
- [23] R. Luling, B. Monien, and F. Ramme. Load balancing in large networks: A comparative study. In IPDPS, 1991.
- [24] M. Arora, S. K. Das, and R. Biswas. A de-centralized scheduling and load balancing algorithm for heterogeneous grid environments. In ICPP, 2002.
- [25] I. Mitrani and J. Palmer. Optimal tree structures for large service networks. Technical report, University of Newcastle upon Tyne, January 2004.
- [26] S. N. Crivelli and T. Head-Gordon. A new load-balancing strategy for the solution of dynamical large-tree-search problems using a hierarchical approach. IBM Journal of Research and Development 48, 2(2004): 153–160.
- [27] A. Katartzis, M. N. Garofalakis, I. Mourtos, and P. G. Spirakis. A hierarchical adaptive distributed algorithm for load balancing. JPDC 64(2004).
- [28] Y. W. Fann, C. T. Yang, C. J. Tsai, and S. S. Tseng. IPLS: An intelligent parallel loop scheduling for multiprocessor systems. In ICPADS, 1998.
- [29] S. F. Hummel, E. Schonberg, and L. E. Flynn. Factoring: A method for scheduling parallel loops. Comm. of the ACM 35, 8August 1992: 90.
- [30] S. F. Hummel, J. P. Schmidt, R. N. Uma, and J. Wein. Load-sharing in heterogeneous systems via weighted factoring. In SPAA, 1996.
- [31] I. Banicescu and V. Velusamy. Performance of scheduling scientific applications with adaptive weighted factoring. In IPDPS, 2001.
- [32] S. McCanne and S. Floyd. VINT Network Simulator. <http://www-mash.CS.Berkeley.EDU/ns/> (1999).
- [33] T. Kunz. The influence of different workload descriptions on a heuristic load balancing scheme. IEEE Trans. on Softw. Eng. 17, 7(1991): 725.
- [34] N. Sanguandikul and N. Nupairoj. Implicit information approach for self-scheduling load sharing policy. In PDCS, 2005.
- [35] C. P. Kruskal and A. Weiss. Allocating independent subtasks on parallel processors. IEEE Trans. Software Eng. 11, 10October 1985: 1001–1016.
- [36] C. D. Polychronopoulos and D. J. Kuck. Guided self-scheduling: A practical scheduling scheme for parallel supercomputers. IEEE Trans. Comp. 36, 12(1987): 1425–1439.
- [37] T. H. Tzen and L. M. Ni. Trapezoid self-scheduling: A practical scheduling scheme for parallel compilers. IEEE Trans. Parallel Distrib. Syst. 4, 1(1993): 87–98.

- [38] T. Philip and C. R. Das. Evaluation of loop scheduling algorithms on distributed memory systems. In PDCS, 1997.
- [39] P. J. Rhodes and S. Ramakrishnan. Iteration aware prefetching for remote data access. In eScience, 2005.
- [40] Y. Yang and H. Casanova. RUMR: Robust scheduling for divisible workloads. In HPDC, 2003.



ศูนย์วิทยทรัพยากร  
จุฬาลงกรณ์มหาวิทยาลัย

## Biography

Natthakrit Sanguandikul is a Ph.D. candidate in Computer Engineering at Chulalongkorn University. He also received his B.Eng and M.Eng from Chulalongkorn University in 2002 and 2003 respectively. His research interests include distributed computing, high performance grid computing, load balancing, and network security.



ศูนย์วิทยทรัพยากร  
จุฬาลงกรณ์มหาวิทยาลัย