Technical Report

UCAM-CL-TR-404
ISSN 1476-2986

Number 404

UNIVERSITY OF CAMBRIDGE

Computer Laboratory

Adaptive parallelism for computing on heterogeneous clusters

Kam Hong Shum

November 1996

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

https://www.cl.cam.ac.uk/

This technical report is based on a dissertation submitted August 1996 by the author for the degree of Doctor of Philosophy to the University of Cambridge, Darwin College.

Technical reports published by the University of Cambridge Computer Laboratory are freely available via the Internet:

https://www.cl.cam.ac.uk/techreports/

ISSN 1476-2986
DOI *https://doi.org/10.48456/tr-404*

Abstract

Until recent years most parallel machines have been made up of closely-coupled microprocessor-based computers. With the advent of high-performance workstations and high-speed networking, the aggregate computational power and memory capacity of workstation clusters have become attractive and indispensable resources for parallel computing. Techniques to harness the power of workstation cluster computing, however, require the development of practical methods for controlling heterogeneous resources dynamically.

This dissertation proposes an integrated framework that comprises two related parts. The first part of the framework is a software structure that enables parallel applications to be adaptable to workload imbalances at runtime. To realize the adaptation, applications are partitioned into small components called tasks. The tasks are then grouped into grains; each grain is an object that facilitates execution of tasks on a workstation. An application can therefore optimize its performance by the reconfiguration of task-to-grain and grain-to-workstation mappings. Based on the software structure, the implementation and evaluation of workload distribution schemes for data-parallel and task-parallel applications are presented. The second part of the framework is a resource management system that allocates resources to parallel applications through competition. The applications respond to allocation decisions by dynamic reconfiguration. The objectives of the system are to maximize the speedup of the parallel applications and, at the same time, to allocate workstations fairly and efficiently to the applications. A prototype implementation which provides a testbed for studying the dynamics of competition is constructed.

In addition, a new structure for organizing replicated parallel applications is developed and an architecture for a multi-user, multi-parallel program environment based on the proposed framework is suggested. The effectiveness of the concept and the framework is demonstrated by the results of experiments conducted on the testbed. The parallel applications involved in the experiments consist of block-matrix multiplication, cycle-searching of a non-linear iterated cryptographic function, and simulators of an ATM network.

Preface

Except where otherwise stated in the text, this dissertation is the result of my own work and is not the outcome of work done in collaboration.

This dissertation is not substantially the same as any I have submitted for a degree or diploma or any other qualification at any other university.

No part of this dissertation has already been, or is being currently submitted for any such degree, diploma or other qualification.

This dissertation does not exceed sixty thousand words, including tables, footnotes and bibliography.

Publications

Some of the work presented in this dissertation has been published, and is described in references [SB94, Shu95, Shu96, SM96] and [SL96].

Trademarks

Alpha AXP and DECStation are trademarks of Digital Equipment Corporation.
Ethernet is a trademark of the Xerox Corporation.
MIPS is a trademark of MIPS Technologies Inc.
UNIX is a registered trademark of AT&T.

Contents

List of Figures

List of Tables

Glossary

AB	Application Bidder (Resource Negotiating Agent)
AR	Application Reporter
ATM	Asynchronous Transfer Mode
BB	Backward Bid
BMM	Block-Matrix Multiplication
COW	Clusters of Workstations
CPU	Central Processing Unit
CS	Cluster Scheduler (Resource Managing Agent)
CSN	Cycle searching for non-linear iterated cryptographic function
DEC	Digital Equipment Corporation
ES	Execution State
FB	Forward Bid
FNI	First N replications initiated scheduling policy
FNI-SA	FNI with Static Assignment
FNI-DA	FNI with Dynamic Assignment
FNI-DAC	FNI with Dynamic Assignment and Coalition
GWD	Global Workload Distribution
HP	Hewlett-Packard
HPF	High Performance Fortran
I/O	Input/Output
IP	Internet Protocol
LAN	Local Area Network
LWD	Local Workload Distribution
LP	Loader Process

MAN	Metropolitan Area Network
MPI	Message Passage Interface
NOW	Networks of Workstations
OS	Operating System
PB	Potential Forward Bid
PSA	Parallel Simulator of an ATM Network
PVM	Parallel Virtual Machine
RPS	Replicated Parallel Simulator
RSS	Replicated Serial Simulator
SIMD	Single Instruction Multiple Data
SPMD	Single Program Multiple Data
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

Chapter 1

Introduction

With the advent of recent technology, workstations have become a powerful and yet cost-effective computational resource. The aggregate computing and memory capacities of workstation clusters[1] have attracted a lot of parallel computing users, especially those required to solve computationally intensive applications. In the past, only dedicated, proprietary supercomputing machines were used to solve scientific applications with intense computational requirements. However, more and more of these applications can now run efficiently and economically on workstation clusters, exploiting the benefits of parallelism. Parallel computing on workstation clusters is a recent computing paradigm known as *Workstation Cluster Computing.*

There are two different configurations of workstation clusters which have distinct uses, namely *Dedicated Clusters* and *Enterprise Clusters.* The workstations of Dedicated Clusters are usually homogeneous and geographically compact. With full control of the cluster nodes, the management of these clusters can easily be implemented in a centralized fashion. Most Dedicated Clusters are designed to increase overall batch computing capabilities by running batch jobs on individual workstations in the cluster.

The other type of configuration is called an Enterprise Cluster. Workstations composing an Enterprise Cluster are typically heterogeneous and geographically dispersed. Moreover, they are individually owned by single or multiple parties. Parallel computing on these clusters relies on scavenging unused computing cy-

[1] Workstation clusters can also be termed Networks of Workstations (NOWs) or Clusters of Workstations (COWs).

cles. In this dissertation, the scope of research is primarily focussed on topics related to Enterprise Clusters.

1.1 Research Motivation

Although the popularity of using networked workstations for parallel computing has been growing steadily, the existing software support and system support for this paradigm remain *ad hoc* and insufficient. Parallel computing on workstation clusters has difficulty in guaranteeing good performance because the resources are heterogeneous and subject to dynamic workload interference from other applications that are running on the same clusters. The potential of this new computing paradigm can only be fully exploited by an integrated solution that addresses the problems of parallel software design and resource management. The goal of this integrated solution is the formation of a multi-user, multi-parallel program environment in which resources can be allocated to applications fairly and efficiently. In this environment, the applications should be adaptable to the allocations through dynamic reconfiguration.

1.2 Research Scope

This dissertation argues that a reconfigurable software structure is the fundamental basis for the adaptation of dynamic resource availability on workstation clusters. Runtime adaptation can then be realized by schemes of workload distribution based on the software structure. For resolving the problem of resource allocation between applications, a job scheduling system is essential for controlling heterogeneous resources dynamically. A prototype system called *Comedians* has been built to allocate resources to parallel applications through competition. To provide an integrated solution, the schemes of workload distribution should facilitate dynamic reconfiguration in response to the allocations made by the Comedians system. Finally, the concept of combining parallelism with replication is used to organize parallel applications in the Comedians system so that the management and performance of the applications can be improved.

1.3 Dissertation Organization

Chapter 2 explains the emergence of workstation cluster computing and lists the problems that hinder the development of this computing paradigm. Related work on the development of workstation cluster computing is also reviewed. Finally, this chapter concludes that an integrated solution for solving the problems is needed.

Chapters 3 through 7 consist of the research content of this dissertation. Figure 1.1 depicts the structure of the research which mainly comprises the software support of runtime adaptation (Chapters 3 and 4) and the system support of resource management (Chapters 5, 6, and 7).

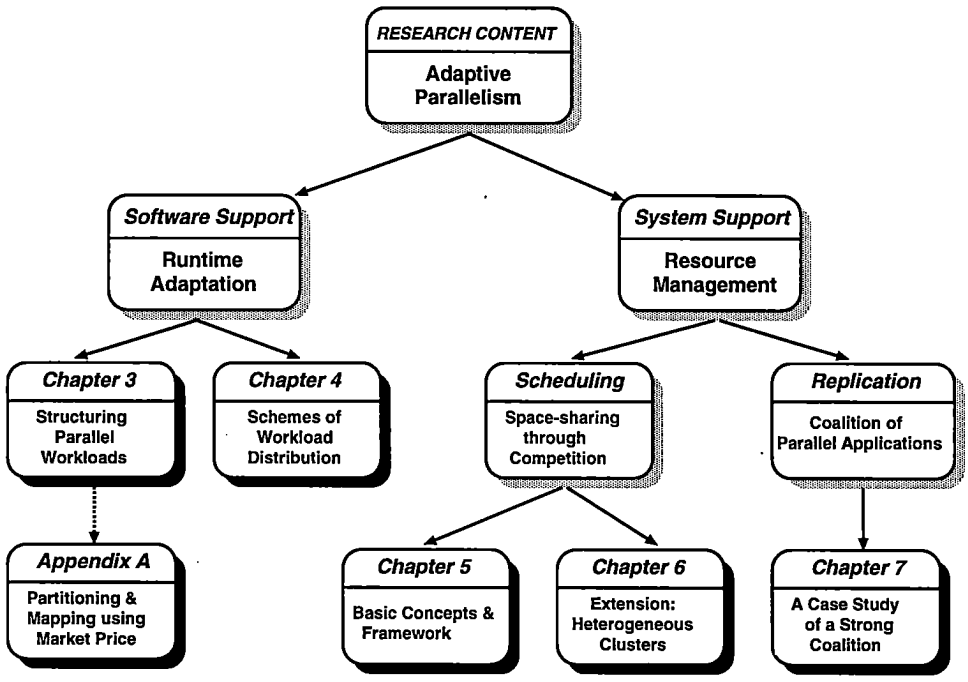


Figure 1.1: Organization of the Dissertation

Chapter 3 describes two different metrics to structure parallel workloads in a dynamic environment. The first metric is the market price derived by the supply and demand of different resources. Based on this metric, partitioning and mapping algorithms are derived, which are presented in Appendix A. The second metric is the relative processing speed of partitioned program elements running on distributed processors. A reconfigurable software structure is suggested to be supported by the second metric.

Chapter 4 presents the design and implementation of workload distribution schemes that are based on the reconfigurable software structure described in Chapter 3. The schemes allow data-parallel and task-parallel applications to be adaptable to changing environments through dynamic reconfiguration.

Chapters 5 and 6 introduce a parallel application scheduling system that applies the runtime adaptation proposed in Chapters 3 and 4. The system facilitates the space-sharing of workstations on heterogeneous clusters through competition. The architecture and experimental results of a prototype implementation are also examined.

Chapter 7 discusses the concept of combining parallelism with replication. The chapter suggests a multi-user, multi-parallel program architecture on heterogeneous clusters. The concept is then tested by experiments using a case study.

Chapter 8 concludes the dissertation with a summary of the research and some suggestions for future work.

Chapter 2

Background

2.1 Introduction

This chapter discusses the rapid emergence of workstation cluster computing. Based on the discussion, the strengths and potential research areas of this new computing paradigm are presented. Three of the most important issues that need to be addressed to further the development of workstation cluster computing are partitioning and mapping, workload distribution, and parallel application scheduling. The details of these issues are described and their related research work is reviewed. This chapter concludes that an integrated solution to address these issues is necessary to facilitate a multi-user, multi-parallel program environment on heterogeneous workstation clusters.

2.2 The Emergence of Workstation Cluster Computing

In recent years, workstation cluster computing has been emerging rapidly. Workstations are now widely available in most academic and commercial organizations. In addition, the principal user of a workstation normally utilizes only a fraction of his or her own workstation capacity [KC91, ML87]. There is an obvious need to improve the utilization of workstations in a shared network.

Since the late '80's, researchers have proposed various systems that could harness

the idle computing cycles of workstations. These systems, such as the Process Server [Hag86], the Butler [Nic87], and the Condor [LLM88], enable users to utilize a network of workstations as a shared computing environment.

Although workstations are not intended for running parallel applications, the popularity of using workstations in a shared network environment to solve computationally intensive parallel applications has been growing unceasingly since the early '90's. Instead of being driven by the research and development efforts of computer vendors, workstation cluster computing emerged naturally from the demands of parallel users and the primary support of software and hardware. What are the reasons for its popularity?

2.2.1 Cost-Performance of Processing Power

It is evident that workstation cluster computing is suitable for many workloads for scientific numerical computations [THM+94, LH95, MIC96]. Researchers [BDG+91, PWA+93, NS92] have also demonstrated the capability of solving *Grand Challenge* problems [GC91] using a workstation cluster. Workstations offer an economic platform to develop parallel computing, particularly in academic institutions, without the heavy investment of buying supercomputers. Nowadays, workstations can offer better cost-performance than supercomputers because the performance of a high-end workstation has been catching up with the performance of a low-end supercomputer, and the sale of workstations is much greater than that of supercomputers.

There is also a recent trend to run workloads other than scientific numerical computations on workstation clusters. Examples include data mining [SNM+95] and Web servers [KBM94, AYHI96]. Many more new applications such as commercially-oriented software will also benefit from the cost-effective CPU cycles of workstations. Although workstation clusters will not completely substitute supercomputers for ultimate performance, they are complementary to supercomputers in many respects.

2.2.2 High-speed Network Connections and Protocols

One of the major potential impediments to cluster computing in achieving good performance is the limitation of inter-workstation communication. However, ow-

ing to the rapid development of high-speed transmission, fast switches and protocols, the potential for performance enhancement is great.

Most of the existing workstations are interconnected by Ethernet that transmits information at around 10-100 Mbps, but with the introduction of standardized communication facilities such as Asynchronous Transmission Mode (ATM), High Performance Parallel Interface (HIPPI), and Fibre Channel Standard (FCS), the potential transmission bandwidth ranges from hundreds of megabytes per second to gigabytes per second. The development of network architectures has shifted the bottleneck of communication in the local area from the limitation of bandwidth to the overheads of protocol processing. The improvement of network interface software is currently under active research. Researchers [LHD+94, DSP+95, EBBV95, LC96] have recently shown that communication protocol overheads can be reduced by using streamlined network interface software or bypassing the general-purpose TCP/IP protocols.

2.2.3 Aggregate Computational and Memory Capacities

The aggregate computational and memory capacities of workstation clusters can satisfy the requirements of many demanding applications that cannot be solved economically on a single machine. In particular, the aggregate DRAM of workstations can be used as a giant cache for disk [ACP94]. Furthermore, the multiple workstation disks can function cooperatively as redundant arrays of inexpensive (workstation) disks (RAID) using the LAN as the I/O backplane [ACP94].

In addition, the aggregate heterogeneous computing power of different clusters can be harnessed by parallel applications as a unified resource. The concept of *clusters* can be generalized by heterogeneous computing. A heterogeneous computing environment may consist of a combination of vector supercomputers, massively parallel computers, specialist high-performance computers, and workstation clusters. The computing clusters of the environment can be connected by fibre-optic links over a wide-area network [NRS92]. Figure 2.1 shows that workstation clusters can be the building blocks of an infrastructure for a heterogeneous computing environment. The goal of using heterogeneous computing [FS93] is to obtain superspeed processing for computationally demanding applications with diverse computing needs.

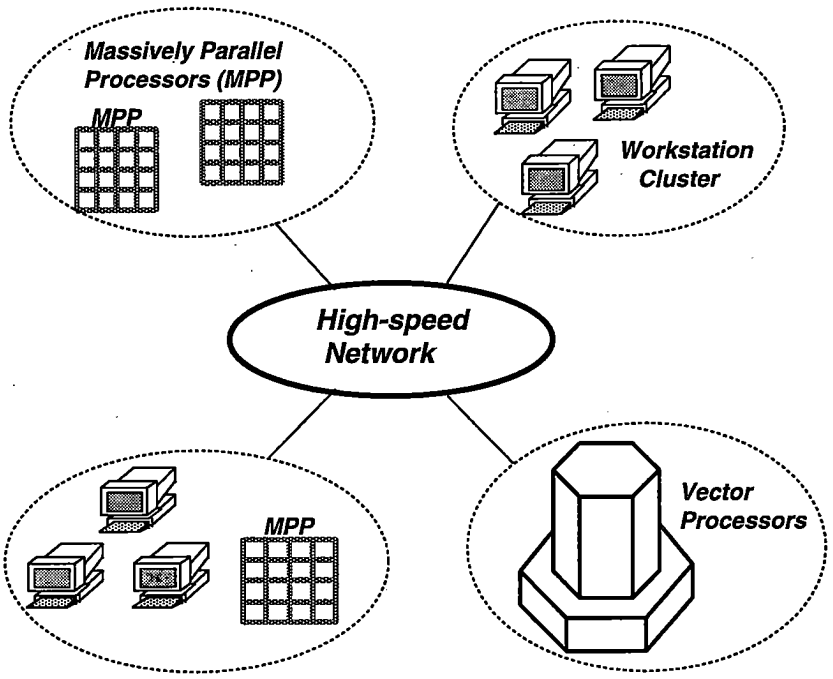


Figure 2.1: Generalization of Cluster Computing

2.2.4 Parallel Programming Paradigms

One of the key factors for the early success of workstation cluster computing is the readiness of software support. Turcotte [Tur93] has presented a comprehensive overview of no less than 60 software packages that provide parallel programming support for a network of workstations. Nevertheless, the methods for developing software systems for workstation cluster computing is still in an early stage. The following sections describe the recent research directions for parallel computing on workstation clusters.

Message Passing Environments

The availability of message passing programming libraries, such as PVM [GBD+93], P4 [BL92], Express [FK94], CHIMP [CHI], and MPI [GLS94] facilitates implementation platforms that are accessible to many people through the Internet. These libraries allow unused computing cycles on workstations to be harnessed for the development of parallel computing.

The message passing paradigm relies on the explicit transmission of messages between systems. The message passing libraries can be regarded as *coordination li-*

braries [CG92] because serial computations are glued together by communication and synchronization operations provided by the libraries. Among the libraries, PVM and MPI are the most widely used.

Parallel Virtual Machine (PVM) was developed by Oak Ridge National Laboratory, University of Tennessee, and Emory University, and is considered as the *de facto* standard of message passing. It provides a mechanism for initializing processes on distributed machines and communicating among these processes. It promotes the utilization of a heterogeneous network of computers as a single computational resource.

Message Passing Interface (MPI) has emerged as the future standard of message passing in parallel computing environments. An MPI Forum was set up in 1993, and over 40 academic, governmental and industrial organizations contributed to the Forum. By making use of the experience of previous message passing systems, MPI aims to provide greater functionality, unified design and an emphasis on user access to high performance protocols and hardware. In 1994, the final MPI-1 standard was released; it included the specification of a much richer set of features than previous message passing libraries. Extensions such as real-time communications, parallel I/O, and active messages will be added in the forthcoming MPI-2 standard [DOSW96].

Distributed Shared Memory

The development of other high-level programming models that aim at more efficient coding and higher software development productivity is also proceeding at a rapid pace. Distributed shared memory (DSM) is one such programming model that can be applied efficiently for cluster computing. DSM systems allow data access through a globally shared memory even though data on each distributed node does not reside in a physically shared memory. The actual movement of data between nodes is hidden from programmers. The primary advantage of this model over the message passing model is easy parallel programming, because programmers do not need to specify interprocessor communication explicitly, and data structures can be accessed in a single address space. However, the implementation of the DSM system has to consider the problems of memory structure and data consistency. As the number of distributed workstations increases, the overheads in maintaining DSM systems also increase. Linda and TreadMarks are two examples of DSM systems for workstation clusters.

Linda [CG89] is a coordination language from Yale University based on a shared associative memory. It is created by extending a sequential language (C or Fortran) with five additional statements. The contents of the memory are organized as tuples. Tuples are defined as objects that contain data and information on how to process the data. A collection of tuples is called tuple space. There are process tuples and data tuples; parallelism is achieved when the process tuples exchange data by generating, reading, and consuming data tuples simultaneously.

TreadMarks [ACD+96] is a DSM system built at Rice University. It is a user-level DSM system that runs on UNIX workstations. The shared memory in TreadMarks is organized as a linear array of bytes via a relaxed memory model called release consistency. It uses a multi-writer protocol to alleviate problems caused by data inconsistency and application granularity.

High Performance Fortran

High Performance Fortran (HPF) [For93] is a programming language that supports data-parallel programming based on the style of implicit parallelism. The objective of HPF is to develop a portable and efficient language for parallel computer architectures. Discussions on a HPF standard were started in 1991 and the first specification appeared in May 1993. The design of HPF was evolved from the Fortran 90 [For91] programming language. Fortran 90 improved Fortran 77 by adding new features such as array operations, pointers, dynamic storage allocation, and user-defined data types. HPF extended Fortran 90 to support parallelism in a natural way. The enhancements include the support of array alignment and distribution, data parallel constructs and compiler directives for data partitioning.

2.3 Addressed Issues & Related Research

The paradigm of parallel computing on workstation clusters inherits many issues that hinder the full exploitation of its potential power. Three of the most important issues are partitioning and mapping, workload distribution and parallel application scheduling. These issues are related to one another and they demand solutions from both software and system support.

2.3.1 Partitioning and Mapping

Problem Definition

The problems of partitioning and mapping in parallel machines involve the assignment of parallel processes to processors. One problem arises when mapping between processes and processors cannot be represented as a uniform one-to-one relation. Partitioning is the development of methods to decompose applications into small partitions that can efficiently be executed in parallel. Mapping is the distribution of the partitions over suitable processors. If these problems are not addressed properly, the performance of parallel computing will be affected adversely by workload imbalances or excessive communication overhead owing to poor process allocation.

The common solution to these problems is to minimize execution time by minimizing communication overheads and balancing workloads. However, there is always a dilemma between using less processors to minimize interprocessor communication and using more processors (but before the saturation effect occurs) to maximize parallel execution. For cluster computing, the problems are further complicated by the dynamic availability of communication bandwidth and computational power.

Related Work

The conventional strategies of partitioning and mapping can be classified as being either static or dynamic. In the case of static mapping, the information regarding processors in the system, as well as processes of an application is assumed to be known *a priori*. Since this problem is NP-hard [Cof74, LK78, GJ79], optimal solutions can only be formulated for some special cases with strict assumptions [Hu61, CG72, Sto77, Bok81]. Previous research primarily focussed on finding heuristic solutions under different situations [NT93]. Nevertheless, these heuristic solutions apply only to particular communication or hardware characteristics [Sah84, MA86] such as the interconnection of processors. In the case of dynamic mapping, a mapping decision is made for a process only when it is executed [SCMS82, SS84] because little knowledge is known or available *a priori*. Only the dynamic approach of partitioning and mapping can be applied to distributed multi-user systems like workstation clusters because resource availability changes when the number of running applications varies.

2.3.2 Workload Distribution

Problem Definition

To support adaptive execution, applications require the capacity for distributing processors' workloads adaptively according to runtime conditions. As depicted in figure 2.2, there are two components of adaptive workload distribution, namely dynamic load balancing and adaptation of external workload interference. The runtime cost of supporting adaptive execution may be significant, yet the potential gains can often outweigh the cost incurred.

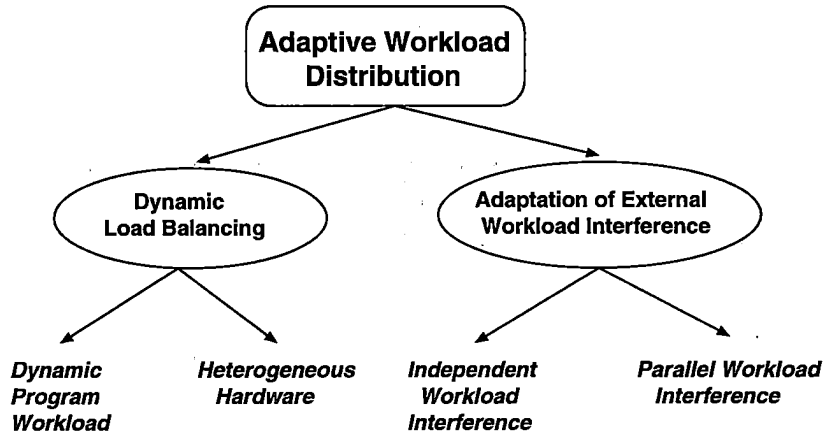


Figure 2.2: Problems Tackled by Adaptive Workload Distribution

Dynamic Load Balancing

Dynamic load balancing is one of the most important and widely studied issues [Qui94] in parallel processing because good performance can only be achieved in a well-balanced situation. Dynamic program workload and heterogeneity in workstations and network connections are two causes of workload imbalance that need to be tackled by dynamic load balancing.

The program workload imbalance arises when demand in processing power varies during the lifetime of an application. For example, in the *land-battle simulation* described by Nicol and Saltz [NS88], different phases of the simulation demand different levels of processing power. Higher processing power is needed when the simulation process changes phase as combating units come close to fighting. A reassignment of workload is therefore required to maintain optimal performance

of the simulation. Another example is the *protein structure prediction* presented in an article by Martino *et al.* [MJS+94]. The processing time of millions of protein structures can be reduced by parallel computing. In the sequence of structures, the atomic coordinates of the atoms change owing to the movement of atoms in the protein structure. The number of neighbouring atoms for each atom changes from structure to structure, and the workload varies accordingly. Static assignment of the atoms across processors is therefore inadequate.

Heterogeneity is common in distributed systems. Although most workstations are connected by a local area network, the scale of workstation cluster computing has been extended to wide-area networks [Str95, DPCS95]. In many network settings the number of workstations ranges from tens to hundreds. Workstations and their interconnections are often heterogeneous in architecture and configuration. This leads to uneven distribution of computation and communication power.

Adaptation to External Workload Interference

In a shared-network environment, applications are subjected to workload interference from other users. The problem of workload interference, which is quite unique to the applications running on workstation clusters because resources are shared among them, can hamper reasonable performance. There are two types of workload interference, one of which comes from independent jobs generated by the owners or users of individual workstations. Since the owners have a high priority in using the workstations, parallel applications should be unobtrusive to them. The other type of workload interference comes from parallel applications. Resource availability can be altered by the arrival of independent workload as well as parallel workload. To be adaptable to different types of workload interference, applications should have the abilities to change the number and distribution of active processes at runtime. The abilities include migrating workload from one machine to another and changing the demand on resources according to availability.

Related Work

Despite the importance of adaptive execution, most of the related work only provides support for either dynamic load balancing or adaptation of the workload interference. In the work related to workload distribution, studies have been carried out at different levels. At the language development level, Dataparallel C

[NQ93] is a parallel programming language that supports dynamic program load balancing on a network of heterogeneous workstations. However, Dataparallel C can only be used for data-parallel applications, and its programming model is restricted to SIMD. The distributed object migration environment (Dome) [ABL+95] from Carnegie Mellon University provides a library of distributed objects for parallel programming that performs dynamic load balancing. Piranha [CFGK95] provides an adaptive version of master-worker parallelism which is implemented on top of Linda. Depending on availability, the number of processors working on a computation may vary in Piranha.

At the application development level, different approaches to implement adaptive runtime support for data-parallel programs are suggested by Prouty *et al.* [POW94], Edjlali *et al.* [EASS95], and Kaddoura *et al.* [KR95]. Dynamic process migration is also used as an adaptive mechanism in response to external workloads. Two different schemes are introduced by Konuru *et al.* [KCO+94] and Casas *et al.* [CCK+95] to provide a migratable system using PVM.

2.3.3 Parallel Application Scheduling

Problem Definition

Since parallel applications will demand a considerable amount of resources, resource scheduling is important both for maximizing the performance of individual applications and optimizing the utilization of resources. The traditional local scheduler in the UNIX kernel of distributed workstations is not an adequate answer to this problem, because parallel workloads need co-ordination between workstations. A distributed or global scheduling system is essential to resolve the contention of resources among the parallel applications running on the same workstation clusters.

For the owners of workstations, a simple priority setting at each workstation is not enough to safeguard the ownership of the workstations because the owners may not be willing to share CPU cycles with computationally intensive applications that normally take hours to finish. For the parallel applications, the effectiveness of execution depends on the quality of resource scheduling.

Related Work

Work related to scheduling parallel applications on workstation clusters includes DQS [GS91] which supports batch scheduling of parallel jobs using a centralized queueing mechanism. DQS can also support both PVM and MPI applications. Although DQS can control the execution of applications running on a network, it can only start and stop them but it cannot relocate them. Condor [LLM88] initially offered process migration of sequential jobs, but there is a recent integration of PVM and Condor called CARMI [PL95] which supports parallel job scheduling by job suspension and resumption. The Prospero system (PRM) [NR94] provides processor selection and allocation, task mapping, program loading, and execution for a scalable infrastructure. The PRM is also scheduled to apply the functions provided by Condor to support task migration. In [SOW94] and [CCO+95], gang scheduling is proposed to allocate time slices and node domains among the tasks of different applications.

The Spawn system [WHH+92] applies a microeconomic paradigm to support coarse-grain parallel programs and the remote execution of independent jobs. It uses auctions to allocate resources among clients that bid monetary funds and bidders increase their bids linearly over time. Funds encapsulate resource rights and serve as a form of priority, and price equates to the supply and demand of processing resources. The auction dynamics can be volatile, however, and the overhead of bidding for resources at time slices is great.

2.4 The Need for an Integrated Solution

As shown in Sections 2.3.1 to 2.3.3, the past research efforts in tackling the problems of partitioning and mapping, workload distribution, and parallel application scheduling do not co-ordinate with one another. The underlying motivations for these research efforts, however, do have two points in common – the maximization of application performance and the optimization of resource utilization. The current trend of development is to tackle some of the problems by combining the capacities of different systems, such as PVM and Condor. This is because the existing approaches cannot provide a complete solution to both the programming support and resource scheduling for parallel applications.

For a multi-user environment, there is an obvious need to investigate an integrated solution to address all the above problems at the same time. If this integrated

solution is realized, different applications should be able to use the resources of workstation clusters as a unified resource or a virtual machine. Users should no longer need to worry about which *set of machines* their applications are running on in order to get the best performance.

Chapter 3

Structuring Workloads for a Dynamic Environment

3.1 Introduction

This chapter describes two metrics to structure parallel workloads for a dynamic workstation cluster computing environment. Criteria for structuring applications are examined. The first metric is the market price acquired from the supply and demand of different resources. The second metric is the relative processing speed of partitioned program elements running on distributed processors. A comparison and an evaluation of these two metrics are presented in this chapter. The second metric is chosen as the basis for building a reconfigurable software structure to support adaptive parallelism on workstation clusters.

3.2 Criteria for Structuring Parallel Workloads

3.2.1 Granularity

Although the workloads of applications can be quantitatively defined by different descriptions [CS93], two of the most important workload characteristics are computation and communication requirements. The relation between these two characteristics can be represented by a parameter called *granularity*. The granularity of an application is determined by the computation-to-communication ratio

of its processes. This ratio is measured by the number of calculations performed by a process to the total size of messages sent by the process. The processes of a *fine-grain* application perform few operations between sending messages. In contrast the processes of a *coarse-grain* application spend a relatively high amount of time in computation between communication intervals. An extreme case of the coarse-grain applications is called *embarrassingly-parallel* [FJL+88], in which no communication is needed between processes.

Although communication technology has been advancing quickly, the overhead of communication is still a limiting factor in achieving a good performance in cluster computing. However, a lot of target applications that range from medium-grain to coarse-grain are suitable for cluster computing. The minimization of communication overhead and the maximization of parallelism are two conflicting criteria in controlling granularity. Careful control of the granularity of an application by partitioning and grouping workloads is important to enhance performance.

3.2.2 System Dynamics and Heterogeneity

As described in Chapter 2, the availability of resources is highly dynamic in a cluster computing environment because the computation power and communication bandwidth of workstations are shared by multiple users. The applications should be adaptable to the changing capacity of computation and communication in workstation clusters. In addition, the architectures and configurations of workstations and communication interconnects are usually heterogeneous. The methods for structuring parallel workloads are exacerbated by the problem of heterogeneity. Static schemes of workload structuring obviously cannot provide the adaptability required for cluster computing.

From the above brief review, the need for a suitable structuring method which can tackle the issues of granularity, system dynamics, and heterogeneity is evident. On the one hand, the structuring method needs a metric to measure the relative importance of computation and communication workloads. On the other hand, the method should allow repartitioning at runtime in order to sustain good performance in a dynamic environment.

3.3 Market Price as a Structuring Metric

In this approach to structuring, an application is represented by a connected task-graph. Figure 3.1 exemplifies a task-graph in which a node indicates a task and an edge is referred to as the communication required between the nodes it connects. From the graph, the task connectivity and the computation and communication requirements of an application can be outlined. *Tasks* are the basic elements of an application. Usually the number of tasks is greater than the number of workstations, therefore tasks have to be grouped into clusters of tasks called *grains* for mapping as shown in Figure 3.1. Each grain is allocated to a *single* workstation for execution.

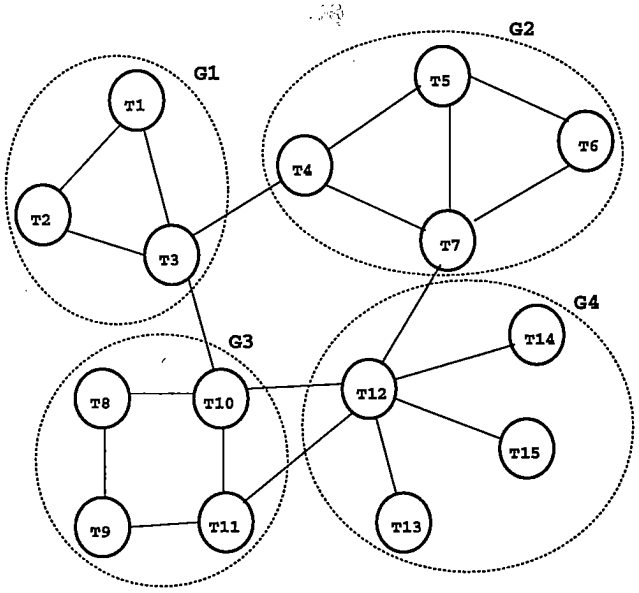


Figure 3.1: An Example of a Task-Graph

Market price is used as a metric to structure parallel workloads because it can evaluate the relative importance between computation and communication workloads. The market price is governed by the demand and supply of the traded commodities and the competition created between the buyers and the sellers, respectively. In market terms, grains are the buyers of the commodities: computation and communication capacities. The workstations are the sellers of these commodities. The buyers should eventually bid to get the best possible deal in terms of their power of payment and urgency of completion time. The tasks are related to each other by their communication requirements in terms of unit data (bytes or messages), while the workstations are related to each other by the market price of communication per unit data. In some studies [AJ88, BKM89,

BNG92], however, the communication intensity between the tasks and the system topologies have been used to partition parallel applications by conventional methods. The sellers should eventually get the best possible price paid for the commodities that they auction. In doing so, both the grains and the workstations may compete with other grains and workstations.

The details of the partitioning and mapping algorithms [SB94] that use the market price as a structuring metric are described in Appendix A. The algorithms are based on a micro-economic approach which deals with parallel applications, but assumes the existence of independent jobs executing on individual workstations as well. The algorithms consider two main phases: the partitioning phase and the mapping phase. The application and the system data in each phase are translated into prices. These phases can be reiterated to restructure the workload of an application whenever the demand for computation or communication requirements of the application changes, or the availability of computation or communication capacities of the workstations varies.

The algorithms underlie the development of simple, fast, and flexible solutions to the partitioning and mapping problems over clusters of workstations, inspired by market rules. They also aim to provide a balanced load distribution by partitioning the workloads of applications dynamically. There is no special requirements on the interconnection scheme of the workstations or inter-task communication requirements, in contrast to most of the earlier studies.

3.4 Structuring by Relative Processing Speed

In the above approach, the market price is used as the metric for measuring workloads. Applications repartition according to changes in the market prices of different resources. However, information about computation and communication workloads is usually not known before program execution. In addition, this approach requires a mechanism to maintain the up-to-date values of the market prices in the whole system.

An alternative metric to structuring the workloads is the relative processing speed of partitioned program elements. Unlike the metric described above, this metric does not assume knowledge of workloads and the state of market prices. Applications are initially structured as partitioned program elements by its users. Then the structure can be reconfigured adaptively to the dynamic environment

according to the values of the metric. The values of the metric are obtained when different partitioned program elements in the same or different applications race or compete one another for completion. Based on the reconfigurable software structure, applications are constructed as adaptive computation agents. The details of the reconfigurable structure and competition are described in the following sections.

3.4.1 Reconfigurable Software Structure

Akin to the basic structure described in Section 3.3, applications are decomposed into small elements called *tasks* that cannot be further decomposed. The computational structure of a problem can be decomposed according to its data structure (domain decomposition) for data-parallel applications or its functional structure (functional or spatial decomposition) for task-parallel applications. After decomposition, the tasks can be grouped into agglomerations called *grains*; each grain is an object that facilitates execution of a variable number of tasks on a *single* workstation. The size of a grain is determined by the number of encapsulated tasks in a grain.

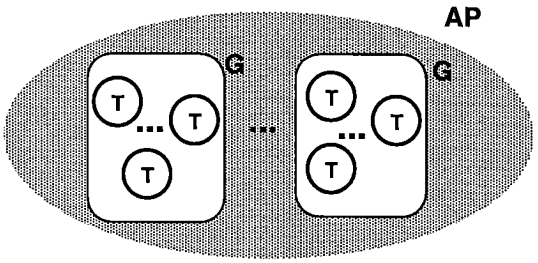


Figure 3.2: The Basic Reconfigurable Structure of an Application

Figure 3.2 depicts the basic reconfigurable structure of an application. The numbers of tasks and grains can vary adaptively according to the environment by changing *task-to-grain* and *grain-to-workstation mappings*. The task-to-grain mapping relates logical task numbers to logical grain numbers; it defines the number of tasks in each grain and the granularity of an application. The grain-to-workstation mapping relates logical grain numbers to physical identifiers for execution. This mapping governs the assignment of grains into workstations and determines the choice of workstations.

Since the task-to-grain and grain-to-workstation mappings can be updated dynamically, the task distribution and execution location of each grain can vary in different situations. These may include differences in processing speed between workstations, and dynamic workload interference on individual workstations. An application can optimize its performance by adopting the best task-to-grain and grain-to-workstation mappings for the current environment.

3.4.2 Partitioning Levels

Runtime adaptation will definitely introduce overheads into execution time. The time for searching for suitable task-to-grain and grain-to-workstation mappings in a changing environment is a major overhead. To minimize the searching time, the possible task-to-grain mappings are confined by predefined groupings of tasks called *partitioning levels*. The configurations of the partitioning levels are defined by the programmer and stored as static information in an application. Programmers can select a set of possible partitioning levels that may be suitable for different situations of resource availability. Although the computation structure and communication topology of an application will affect the partitioning levels of an application, detailed workload information is not required.

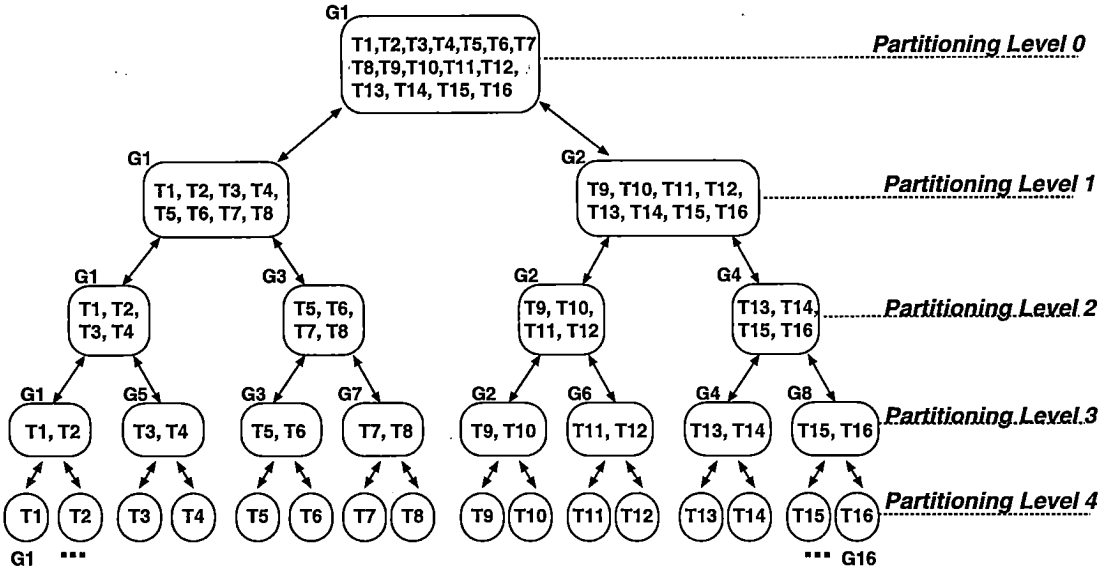


Figure 3.3: Splitting and merging of the grains (by a factor of two) of an application between different partitioning levels

Figure 3.3 shows an application which consists of 16 tasks. The grains of this example can split or merge by a factor of two between two partitioning levels. These partitioning levels provide very coarse-grain control for adaptability. To obtain better adaptability, the grain-size at each partitioning level may be fine-tuned. Therefore, the task-to-grain mapping of an application at each partitioning level is variable.

3.4.3 Relation between Software Structure & Adaptation

In a large and dynamic environment, a global control for all applications is not possible because up-to-date global system states cannot be maintained efficiently. For parallel applications running on the environment, they should have self-regulating capabilities that are based on local information. The parallel applications can act as adaptive computation agents that are adaptable to external changes by adjusting their execution profiles and resource consumption. The goal of designing a reconfigurable software structure is to form a basis for supporting adaptive applications. An application is said to be adaptive if it can take actions in response to changes in its surroundings.

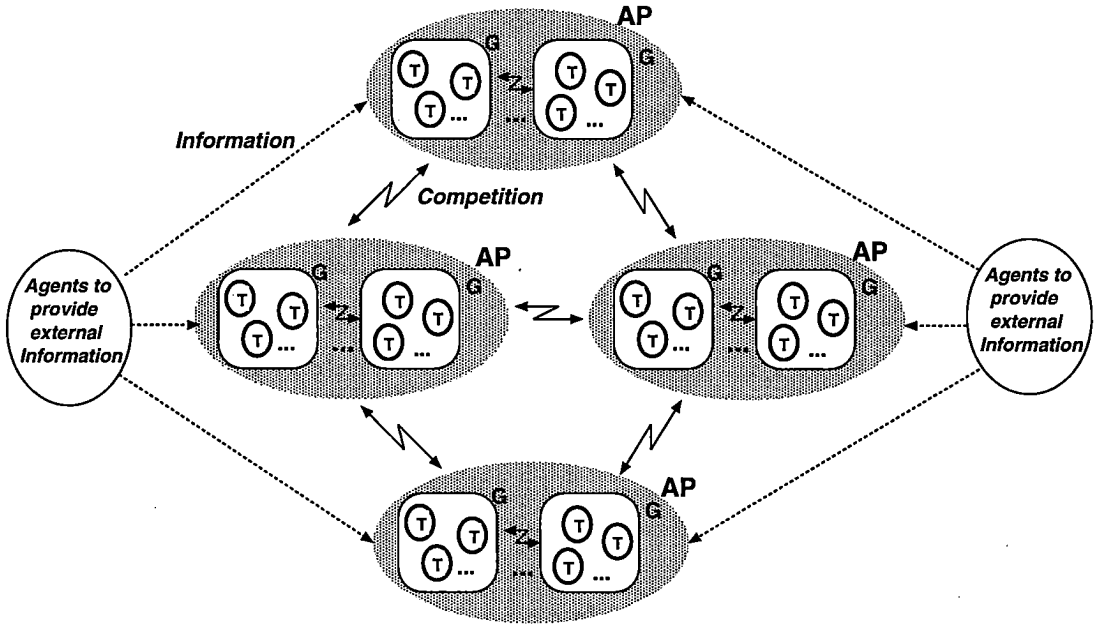


Figure 3.4: Abstract Model of a Dynamic Environment for Adaptive Applications

Figure 3.4 shows an abstract model of a dynamic environment for adaptive applications. Competition is used as a general approach to facilitate adaptation in the environment. There are two types of competition that can lead to structure reconfiguration: intra-application competition and inter-application competition. Intra-application and inter-application competition are driving forces for the actions of adaptation. Table 3.1 compares the major differences between these two forms of competition.

	Intra-application Competition	*Inter-application Competition*
Scope	Between grains	Between applications
Information Required	Relative speed of processing tasks	Resource availability & Application status
Sources of Information	Neighbouring grains	External monitoring agents
Effects of Competition	Resolve local workload imbalances	Resolve allocation of resources among applications

Table 3.1: Comparison between Intra-application and Inter-application Competition

3.4.4 Intra-Application Competition

One of the most important criteria to achieve optimal performance from parallel computing is load balancing, that is, workload distribution among distributed processors is balanced in proportion to their computation and communication power. The traditional methods of load balancing involve matching between application workloads (computation load and communication load) and hardware capacities (processor speed and network bandwidth). These methods require detailed knowledge of application workloads and hardware characteristics. A new metric that does not require information about workloads is derived through the intra-application competition. The metric is the relative processing speed of tasks between grains; it is used to compare the resultant performance owing to the computation and communication of tasks on distributed machines. This metric is independent of application types and execution platforms.

For intra-application competition, competition appears in the form of racing between neighbouring grains. Workload imbalances can be detected by comparing workloads between grains using the proposed metric. Workloads can then be distributed locally between the grains of the same application. The conditional actions of adaptation through intra-application competition are discussed thoroughly in Chapter 4.

3.4.5 Inter-Application Competition

For inter-application competition, external information is needed to improve resource utilization by avoiding unnecessary resource contention among applications. The external information also provides guidance for an application to expand or migrate to newly available workstations or withdraw its computations from workstations that cause workload imbalances. As shown in Figure 3.4, the information should be passed to applications from external monitoring agents. The information includes parameters for evaluating resource availability, such as the utilization rate of CPU, network bandwidth, and memory.

Besides the information about resource availability, status information about the other competing applications is also required to make resource allocations. Inter-application competition will not be fair and efficient if the applications do not have access to information about resource availability and competing applications. In Chapters 5 and 6 a parallel application scheduling system is proposed to provide all the supporting functions required for inter-application competition.

3.5 Summary

This chapter presented two different metrics to structure parallel workloads in a dynamic environment. The first metric measures workloads using market prices adjusted by the supply and demand of different resources. The second metric uses the relative processing speed of tasks between grains to structure workloads. A reconfigurable software structure which is supported by the second metric was suggested. The software structure is the basis of the work to support adaptive parallelism on workstation clusters through intra-application and inter-application competition.

Chapter 4

Adaptive Execution of Parallel Applications

4.1 Introduction

The purpose of this chapter is to present two complementary schemes of adaptive workload distribution based on the reconfigurable software structure described in Chapter 3. The schemes allow parallel applications to reconfigure themselves at runtime in order to be adaptable to workload imbalances. The workload imbalances can be the results of dynamic program workload, heterogeneity, and different forms of workload interference. In other words, the performance of a parallel application may suffer if the amount of heterogeneous resources alters during its lifetime in the environment. The tailor-made schemes for workload distribution are implemented for different data-parallel and task-parallel applications. The evaluation of experimental results is described near the end of this chapter.

4.2 Adaptive Workload Distribution

Figure 4.1 shows the problems that need to be addressed by adaptive workload distribution and the solutions provided. The white oval blocks are the classification of these problems which have been described in Section 2.3.2, and the grey oval blocks represent the solutions provided by adaptive workload distribution.

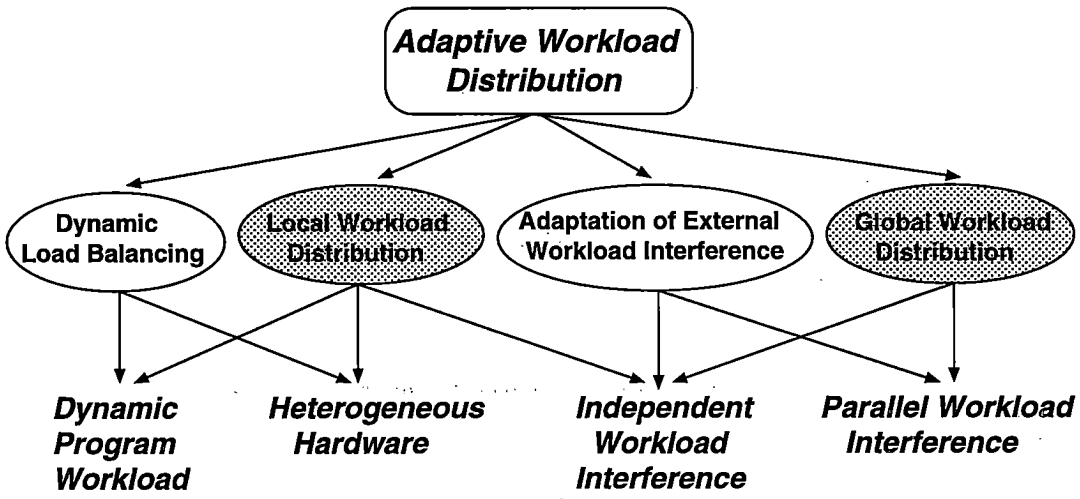


Figure 4.1: Problems Tackled and Solutions Provided by Adaptive Workload Distribution

Two schemes of workload distribution are proposed for automatic adaptation at runtime, namely *Local Workload Distribution* (LWD) and *Global Workload Distribution* (GWD). These schemes take different approaches to distributing workload, but they can complement each other to enhance overall adaptability.

For Local Workload Distribution, tasks are grouped locally between neighbouring grains. Grains and their neighbours race one another until all their tasks are finished. If the relative processing speed of tasks between grains is different, the tasks of the slower grains will be shared with the faster grains. Since the slowest grain limits the performance of an application, the overall speedup can be increased by sharing the workload of the slowest grain with its neighbouring grains. It is possible that all the tasks of a grain are taken over by its neighbouring grains. Basically, the LWD is designed to tackle the problem of dynamic load balancing, but it can also resolve certain workload imbalances due to independent workload interference.

For Global Workload Distribution, an application can switch from one partitioning level to another partitioning level through splitting, merging, or migrating the grains of the application. Although these partitioning levels restrict the number of possible task groupings, they reduce the time to search for suitable task-to-grain and grain-to-workstation mappings at runtime. The GWD aims to provide solutions for the adaptation to external workload interference.

4.3 Local Workload Distribution

The Local Workload Distribution (LWD) distributes tasks among neighbouring grains by comparing the relative processing speed of tasks between the grains. The grains coordinate with their neighbouring grains so that workloads among them can be shared dynamically and locally by racing one another until all their tasks are finished. Through grain racing, the relative speed of processing tasks can be compared between grains. If the processing speeds between the grains are different, workloads will be balanced by sharing the tasks of slow grains with faster grains. The LWD is enforced by conditional actions, therefore it will not be activated if the workloads of the grains are balanced. The details of the conditional actions for different applications are described in Sections 4.4.2 and 4.5.3. In contrast to the master-workers parallelism, the control of the LWD is distributed, so the LWD provides a scalable load-balancing approach that can be applied to large scale parallel applications.

In the following sections, tailor-made implementations of the LWD for three parallel applications which have different computation and communication characteristics and requirements are discussed: (1) a block-matrix multiplication program (BMM); (2) a cycle-searching of a non-linear iterated function program for cryptography (CSN); (3) a parallel simulator of an ATM Network (PSA). The first two applications are classified as data-parallel applications and the third application is referred to as a task-parallel application. For data-parallel applications, the same algorithm is applied to a large data set. Each task receives a fraction of the whole job related to its part of the data. For task-parallel applications, tasks are defined by spatial or functional decomposition. These examples are used to illustrate the fact that the LWD is amenable to different paradigms of parallelism.

4.4 Data Parallelism

4.4.1 Data-Parallel Applications

Many computationally intensive scientific computations are made up of operations on a large data set by multiple loops. To parallelize these computations, each processor applies the same operations to a subset of the data set. This regular parallelism on data is commonly realized as the data-parallel paradigm.

Data parallelism is a natural paradigm for a large number of computational problems [Fox89]. High Performance Fortran (HPF) is the most recent international effort to derive a language that aims to enhance the performance of data-parallel applications on different hardware platforms. The extensions of HPF from Fortran 90 include advanced constructs and compiler directives for data distribution. However, the current standard for HPF does not include adaptive workload distribution.

Two examples of data-parallel applications to be discussed in this chapter are the block-matrix multiplication program (BMM) and the cycle-searching of an non-linear iterated function program for cryptography (CSN). For the BMM, input and product matrices are partitioned into NxN matrix blocks. Each task is to formulate a product block that is calculated by the equation: $C_{x,y} = \sum_{i=1}^{N} A_{x,i} \cdot B_{i,y}$. There is no communication between tasks, so every grain can work independently.

For the CSN, the input data set can also be arranged as a two-dimensional data array. Different initial values lead to either new cycles or repeating cycles in a non-linear iterated function. The program takes each initial value and searches for which cycle it ends up in. For the sequential version of the program, cycle information is stored by a big hash table; it is used to determine whether two values lead to the same cycle. For the parallel version, each task takes a fraction of the initial values and searches for cycles with reference to its local hash table. No inter-task communication is required except infrequent broadcasts of cycle information. It is carried out to unify cycle information in the local hash tables. Initially, the application may take a long time to search for and create the information on new cycles, but the time for searching decreases rapidly as more cycles have been recorded. Unlike the BMM, the CSN consumes a lot of memory and its execution time may not be directly proportional to the size of its input data set.

4.4.2 Distributing Workloads between Grains

For both of the example applications, a two-dimensional data set can be partitioned and numbered as shown in Figure 4.3. Initially the subsets of the data set can be decomposed uniformly into small data domains; these domains are then assigned as grains. The grains are further decomposed into smaller uniform data domains as tasks. The grains are processed in different individual workstations concurrently, whereas the tasks in each grain are processed sequentially in order

Conditional Action 1
If all its tasks are processed,
Informs its neighbouring grains about its completion,
Asks its next neighbouring grain for more unprocessed tasks
until all its neighbouring grains finish all their remaining tasks.
Conditional Action 2
If requests from other neighbouring grains are received while processing its own tasks,
Dispatches n unprocessed tasks to the requesting grain.

Figure 4.2: Conditional Actions of the LWD in each grain for Data-Parallel Applications

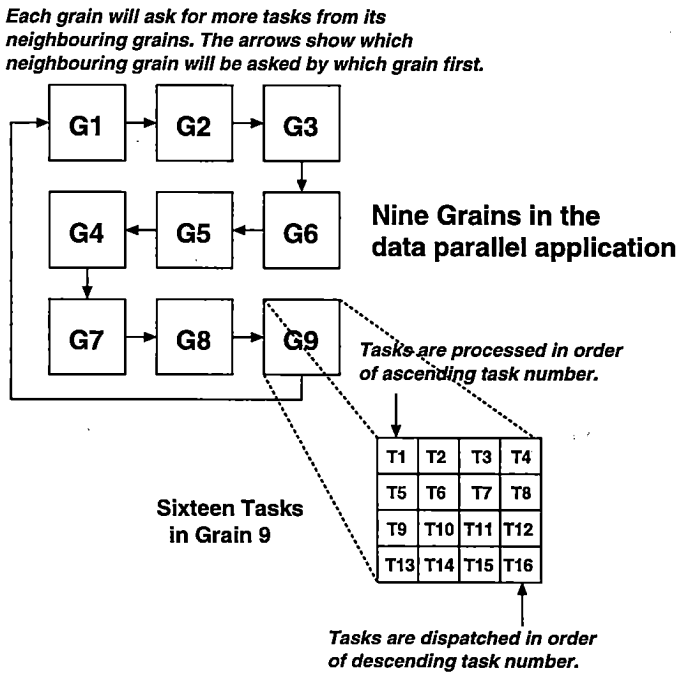


Figure 4.3: The LWD for Data-Parallel Applications

of ascending task number. A neighbourhood can be formed by adjacent grains, for example the nine grains in Figure 4.3 form a neighbourhood and each grain has eight neighbouring grains.

Figure 4.2 shows the possible conditional actions of the LWD for data-parallel applications; these actions are carried out by each grain of an application. On the one hand, if all the tasks of a grain are processed, the grain will inform its neighbouring grains about its completion and request its neighbouring grains for more unprocessed tasks until all its neighbouring grains are finished. The arrows between the grains shown in Figure 4.3 indicate which neighbouring grain will be asked by which requesting grain first. The requesting grain will then ask its next neighbouring grain in order of ascending grain number. For example, the order of request for G5 is G4, G6, G7, G8, G9, G1, G2, and G3. If the requesting grain has the highest grain number, it will ask the neighbouring grain with the lowest grain number for unprocessed tasks first. This order of request aims to reduce the chance of being requested by more than one grain at a time.

On the other hand, if a grain receives requests from its neighbouring grains while processing its own tasks, n unprocessed tasks will be dispatched to the requesting grain. The tasks are dispatched to its neighbouring grains in order of descending task number. As a result, workloads among the grains in a neighbourhood can be balanced locally.

4.5 Task Parallelism

4.5.1 Parallel Simulation

Discrete-event simulation modelling is one of the most common scientific applications that demands computationally intensive power from parallel machines. Parallel simulation is an example of a task-parallel application because the logical processes of the simulation can be defined by spatial decomposition. The potential gain in speed of the execution time of a simulation in workstation clusters is tremendous. Nevertheless, the conventional static schemes of allocating simulation workload into distributed workstations are inadequate because the assigned workload may change over time. An adaptive distributed simulation should also take account of the differences in workstation architectures and networking technologies among workstation clusters.

This section describes the LWD for the parallel simulator of an ATM network [Shu95] in workstation clusters. Since the scheme of the LWD for the parallel simulation is dependent on the computation and communication configurations of the simulation model, the complexity of building the LWD for the parallel simulation is generally higher than for the data-parallel applications.

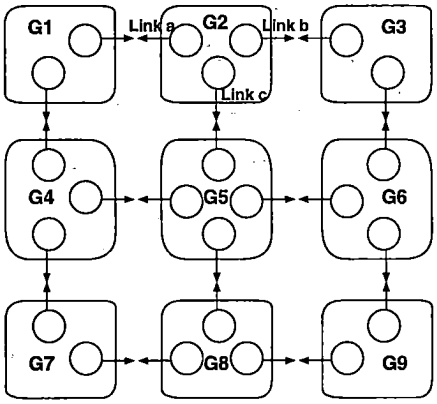
4.5.2 The ATM Simulator Model

Simulation is regarded as an important performance modelling tool for communication networks. However, the simulation of a broadband communication network, such as an ATM network, is extremely time-consuming owing to an enormous number of packet-level operations for broadband switches and communication links involved in performance modelling. Parallel and distributed simulation is therefore used to reduce the execution time. In previous work on speeding up the simulation of ATM or BISDN networks, dedicated multi-processor machines were used to execute parallel ATM network simulators [PC91, YKF91, BG92]. In a recent study [CG93], Chai and Ghosh carried out a distributed simulation of an ATM network on a network of SUN Sparcstation-1 workstations. However, the problem of load balancing was only tackled by manual adjustments.

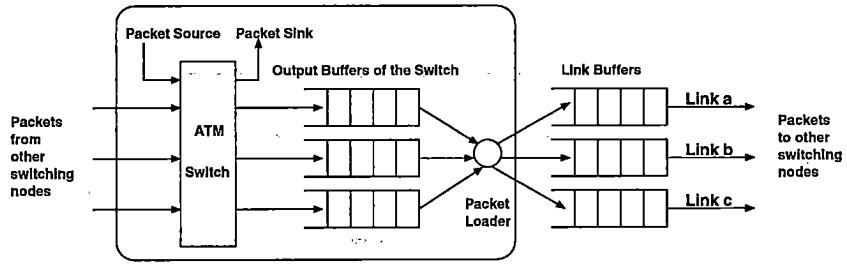
The parallel simulator (PSA) described in this work is designed for the purpose of evaluating the performance of the traffic flow control and call set-up algorithms for an ATM network proposed by Li and Shum [Li90, LS94].

An ATM network is made up of switching nodes and inter-node communication links. Figure 4.4a depicts an ATM network which consists of an array of nine switching nodes. As shown in Figure 4.4b, a generic model of a switching node (node G2 in Figure 4.4a) mainly consists of an ATM switch, a packet loader, output buffers, and link buffers. The link buffers model the buffering effect of communication links for transmitting packets.

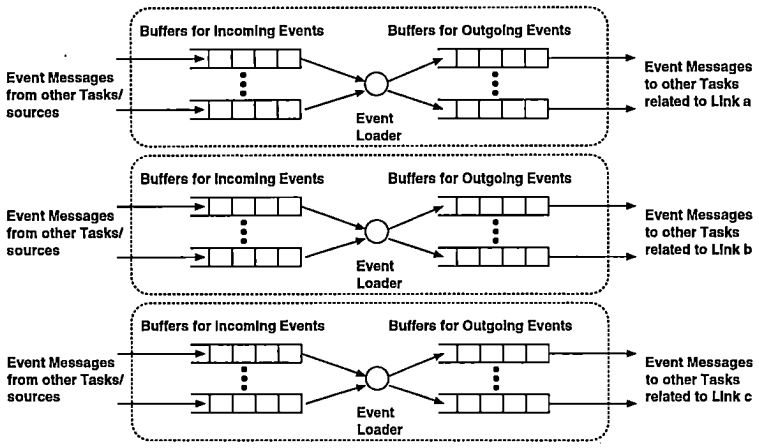
According to the reconfigurable software structure, the switching nodes and communication links of an ATM network can be decomposed spatially into logical processes which are hereafter referred to as tasks. Tasks are the smallest logical processes of the simulation model. Each task processes the events of multiplexing, demultiplexing, and buffering packets that are all related to a specific outgoing communication link as shown in Figure 4.4c. The tasks are then grouped into grains; each grain facilitates execution of the tasks on a workstation. For example, the tasks of the model shown in Figure 4.4a can be grouped into twenty-four



a. An array of nine switching nodes is decomposed into nine grains



b. A Generic Model for a Switching Node (G2)



c. The Decomposed Tasks of the Switching Node (G2)

Figure 4.4: Spatial Decomposition of an ATM Switching Network

grains, nine grains, three grains, and one grain.

Algorithms of *conservative time window* [Lub89] are applied to process event messages of the tasks in distributed workstations. The time window is the time period when events can be processed safely without incurring causality errors [Lam78]. The events in a time window are packed into time-stamped event messages before they are sent to other tasks. The basic operations of the conservative time window algorithm are described as follows: for every time window, each task exchanges time-stamped event messages locally with its neighbouring tasks according to the interconnection of the simulated network. A task is first synchronized with its preceding tasks for incoming event messages. Any pending events in the event-lists for that time window will then be processed. Finally, its outgoing event messages will be sent to its succeeding tasks. If the tasks are located at different grains, the exchange of event messages between them is through explicit message passing, otherwise event messages are transferred by internal data movement.

The target model for the simulation is an ATM Metropolitan Area Network (MAN) [Li90, LS94] which has the same topology as the network shown in Figure 4.4a. Basically, the size of a time window is determined by the propagation delay of the outgoing links, but the window size can be extended by the lookahead technique described in [Nic88]. Since propagation delay in MANs is very high, the number of events in each simulation window is large enough to provide a suitable granularity for the simulation on distributed workstations.

4.5.3 The Use of Virtual Tasks

Since the performance of the slowest grain will limit the overall performance of the parallel simulation, the speedup can be increased by sharing the workload of the slowest grain with its neighbouring grains. Figure 4.5 summarizes the simulation operations and the conditional actions of the LWD. The protocol of the LWD will only be deployed if any one of the conditions, shown in Figure 4.5, is satisfied. The conditional actions are embedded in the operations of the simulation. A *sender-initiated* [WM85] approach is adopted in which the task in a heavily loaded workstation initiates the LWD. The load metric for comparing loading is the relative processing speed of tasks which is the ratio of the average task execution time (t_{exec}) between grains. t_{exec} is calculated by dividing the execution time of a time window by the time window period. Its value can be normalized by the relative task computing demand in the simulation.

The Parallel simulation (Conservative Time Window) :

Do Parallel: for each Grain,
  Do Parallel: for each Task in the Grain,
    While simulation not completed,
      Begin
        Wait and receive incoming time-stamped event messages
          from its preceding tasks,
        Process the events for the next time window period,
          (Conditions for the local workload distribution are embedded),
        Send outgoing time-stamped event messages to succeeding tasks.
      End

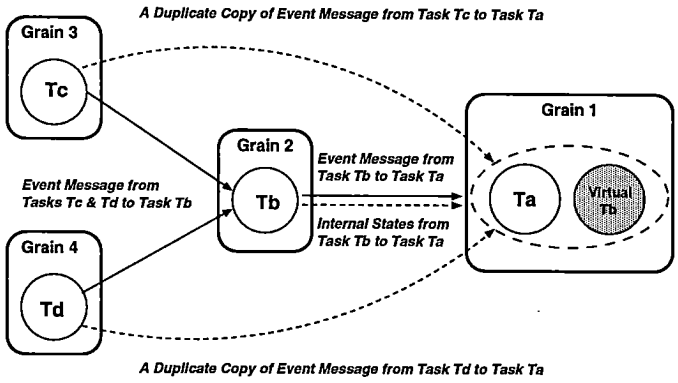
Conditional Actions of the local workload distribution :

Action 1
  If $\frac{t_{exec}}{t'_{exec}} \geq \gamma$ for more than η consecutive time windows,
    Set its *Sharing-task status.*
    Send its internal states to its takeover task.
Action 2
  If the *Sharing-task status* of its succeeding tasks is set,
    Send a duplicate copy of event message to the takeover task
      of its succeeding task.
Action 3
  If it is waiting for any task for execution and,
    If the internal states and incoming event messages are ready,
      Execute the virtual task.
Action 4
  If it is a virtual task, and is intercepted by the original task,
    Reset the *Sharing-task status* of the original task.
Action 5
  If it is intercepted by its corresponding virtual task,
    Send signals to its preceding tasks to deactivate all the incoming
      event messages.
    Deactivate itself.

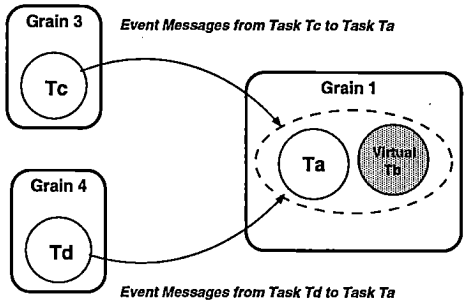
Definition of Symbols :

t_{exec} is its normalized task exec. time for a time window,
t'_{exec} is the normalized task exec. time for a time window of its takeover task,
γ, η are the constant great than or equal to one.

Figure 4.5: The Parallel Simulation and its Conditional Actions of the LWD



a. The Virtual Task of Tb is invoked



b. The Virtual Task of Tb has taken over the original Tb

Figure 4.6: The LWD for the Parallel Simulation

Both t_{exec} and a status vector called *Sharing-task status* will be exchanged between neighbouring grains during execution. They piggyback the outgoing event messages of the task to the succeeding task. Each bit of the Sharing-task status vector represents the status of a task. If a task's average execution time t_{exec} is greater than or equal to γ times of the average task execution time of its takeover task t'_{exec} for more than η consecutive time windows, it will set its Sharing-task status and send its internal states to one of its succeeding tasks, called the *takeover task*. The values of γ and η can be fine-tuned to adjust the sensitivity of the LWD to the varying workload owing to the simulation and the other users. On receipt of the Sharing-task status, the task's preceding tasks will send a duplicate copy of the event messages to its takeover task. If the grain of the takeover task is idle and all the internal states and event messages are ready, it will execute a *virtual task* to take over the original task. Once the virtual task is started, it will race with the original task for completion.

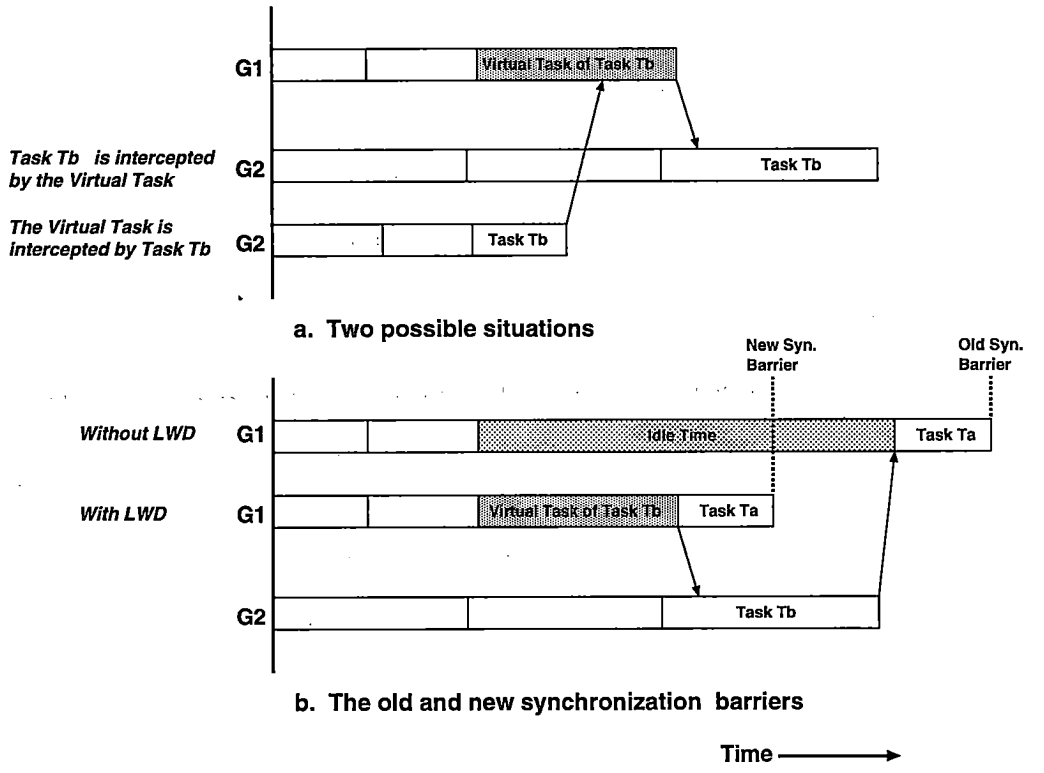


Figure 4.7: Timing Diagrams of the Grains for the LWD

Figure 4.6 depicts a simple example of performing the LWD. Tasks T_a, T_b, T_c, and T_d are all located in different grains. In the case of task T_b, it receives incoming event messages which come from tasks T_c and T_d (the preceding tasks) and sends outgoing event messages to task T_a (the succeeding task). If task T_b is placed in a workstation which has a low relative processing speed for tasks, task T_a will always wait for the incoming event messages to proceed. The transfer of duplicate copies of event messages and the internal states of task T_b at the time when its virtual task is invoked in the grain of task T_a is illustrated by Figure 4.6a. If the LWD is deployed successfully, the original task T_b will be taken over by its virtual task (Figure 4.6b).

Figure 4.7a shows two possible situations that may arise from the example of Figure 4.6, where tasks T_a and T_b are located in grains $G1$ and $G2$ respectively. If the virtual task finishes before the original task, it will send an intercepting signal to deactivate the original task, otherwise the virtual task will be deactivated if it is intercepted by the event message from the original task. All the event messages from the preceding tasks will be stopped if a task is deactivated and any out-of-date interception is discarded.

Figure 4.7b shows the effect of the LWD in shortening the synchronization barrier of $G1$. If the LWD is not applied, there will be a long idle time for $G1$ to wait for the incoming event message from $G2$. Instead of waiting idle, $G1$ executes a virtual task of T_b to race with the original T_b. The successful completion of the virtual task shortens the idle time while waiting for the synchronization barrier. It is possible that all the tasks of a grain are taken over by its neighbouring grains. In this case, the number of active grains varies during the simulation run.

4.6 Global Workload Distribution

4.6.1 Basic Conditional Actions

Besides sharing tasks with neighbouring grains, the tasks of an application can also be grouped into different pre-defined grain-sizes to enhance adaptability of resource requirement at runtime. As described in Section 3.4.2, these pre-defined groups of tasks are called *partitioning levels*. For the parallel simulation shown in Figure 4.4a, its tasks can be possibly grouped into four partitioning levels which consist of 1, 3, 9, and 24 grains. The basic conditional actions for the *Global Workload Distribution* (GWD) are grain merging, grain splitting and grain migration.

The grains of an application can split or merge into appropriate partitioning levels under different situations of resource availability. If more workstations become available, the grains can split to a higher partitioning level. On the other hand, the grains merge to a lower partitioning level if the workstations in which the grains are running become heavily loaded with external processes or are particularly slow. In addition to grain splitting and merging, the grains can also migrate to different workstations for execution in response to resource contention. The grain migration process can be carried out at the same time as grain splitting or grain merging. To perform these actions of the GWD, global synchronization among all the grains is required.

4.6.2 A Self-Tuning Mechanism

In contrast to the LWD, external information about the environment is required to avoid resource contention with other applications. An external agent process

called an *Application Reporter* (AR) is used to collect information about external resource availability and report the information to each application. Based on the external and internal information, the runtime configuration of an application can be self-tuned to enhance performance. This self-tuning mechanism enables an application to become adaptable to its environment through the GWD.

To collect the local information, each grain reports the average task execution time t_{exec} and the average communication and synchronization time t_{cs} of its tasks to the AR at every time interval t_{report}. For the workstations that do not have any resident tasks, the AR obtains their workstation load and bandwidth utilization from broadcast information in their local network. Therefore, any available workstations and workload imbalances in computation or communication can be identified. The availability of a workstation is determined by the number of active processes and the amount of free memory in the workstation.

The self-tuning mechanism is deployed whenever there is a need to improve execution performance. Grain merging is applied if there are not enough available workstations to maintain the present partitioning level and the unbalanced workload situation cannot be improved by the LWD alone. If more workstations are available, performance can be improved by grain splitting. Migration occurs if any grain is mapped into a different workstation. After a partitioning level and the required workstations are chosen, the grains are mapped into the workstations and the AR will inform every grain to invoke the global synchronization of the GWD.

4.7 Interplay between Local & Global Workload Distribution

The GWD and LWD are the key elements to support adaptive execution of parallel applications. The GWD helps to partition grains into a suitable partitioning level and relocate the grains from heavily loaded (or slow) machines to lightly loaded (or fast) machines. On the other hand, the LWD refines the size of the grains at each partitioning level to tackle local workload imbalances. Their combined effect facilitates automatic grain-size tuning at runtime.

For the parallel simulation, a network of nine switching nodes connected bidirectionally in a mesh can be decomposed into two partitioning levels as shown in Figure 4.8a and 4.8c. Grain merging and splitting of the GWD will be activated

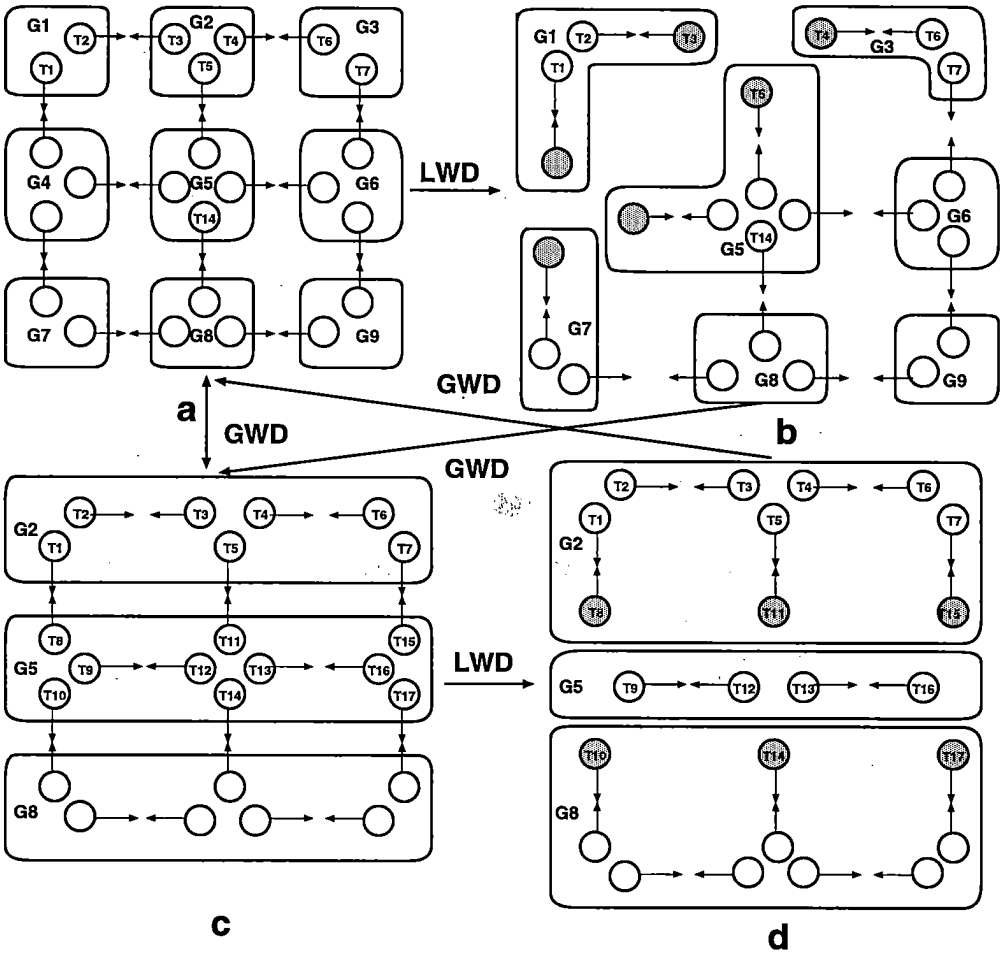


Figure 4.8: The Interplay between GWD and LWD for the Parallel Simulation

to alter substantially unbalanced workload situations. Figure 4.8 also shows that the GWD can adjust the grain-size of the simulation in a flexible manner. For example, the 3-grains in Figure 4.8c can be the result of grain merging from the 9-grains in Figure 4.8a or the 7-grains in Figure 4.8b.

Figures 4.8a and 4.8b depict the cases for the LWD when there are nine grains. If the workloads of $G2$ and $G4$ are unbalanced, their tasks are shared by their neighbouring grains, say the virtual tasks of $T3$, $T4$, and $T5$ of $G2$ are invoked by $G1$, $G3$, and $G5$ respectively. A similar situation occurs when there are three grains (Figure 4.8c and 4.8d). If the workload of $G5$ is unbalanced, the virtual tasks of $T8$, $T11$, and $T15$ of $G5$ are invoked by $G2$ and the virtual tasks of $T10$, $T14$, and $T17$ of $G5$ are invoked by $G8$.

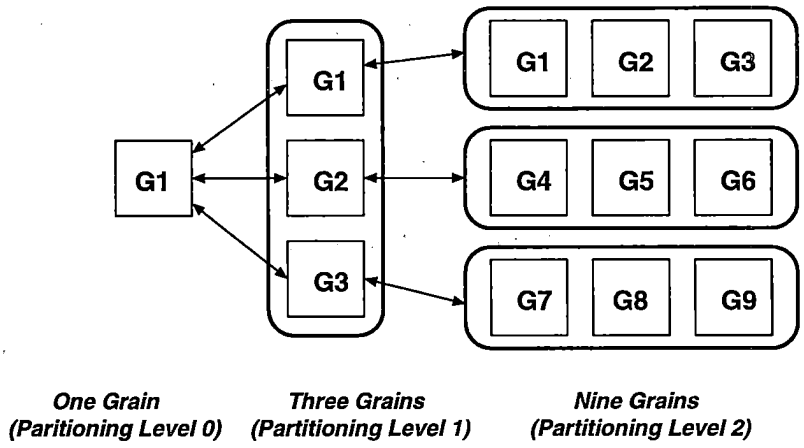


Figure 4.9: Grain Merging and Splitting for the Data-Parallel Applications

For the data-parallel applications, the same scheme of the LWD can be applied even if the number of grains changes owing to grain merging or grain splitting as shown in Figure 4.9. Again the GWD can work with the LWD at different partitioning levels.

In the above examples of data-parallel and task-parallel applications, the number of possible application configurations increased if the LWD and GWD work together. However, unlike data-parallel applications, the configurations of task-parallel applications are restricted by communication or data dependencies among their tasks.

The GWD or LWD does not in itself provide a complete solution for adaptive workload distribution. For the GWD, the predefined partitioning levels may not precisely suit all the possible unbalanced workload situations and the overhead of global synchronization may grow as the number of tasks increases; hence the LWD is introduced to balance the workload of a grain with its neighbouring grains more efficiently.

On the other hand, the LWD for the task-parallel applications is restricted by communication dependencies. For example, depending on the connectivity of the simulated network, a task will only exchange event messages with its preceding or succeeding tasks. If the workload of a task is unbalanced, a virtual task can only be invoked by its takeover task. Thus, the LWD cannot be deployed if the workload of the takeover task is also unbalanced. The GWD can then be applied to alleviate this kind of problem. Since the GWD is more costly than the LWD, it should be deployed less frequently. These two schemes of workload distribution

are therefore complementary to one another. Their limitations and strengths are summarized in Table 4.1.

	Limitations	*Strengths*
LWD	• Limited workload distribution	• Efficient local workload sharing • Automatic grain-size fine tuning • Scalability to large applications
GWD	• Limited/predefined partitioning • Global synchronization overheads	• Adaptation to significant workload imbalance by grain splitting and grain merging • Grain migration capability

Table 4.1: Limitations and strengths of the LWD and GWD

4.8 Implementation Notes

4.8.1 Approaches to Moving Workloads

As reported in [CKO+94], there are three basic approaches to implementing workload distribution for Unix workstations. The first approach [CCK+95] is to migrate a Unix process of a parallel application from one host to another. In this case, the application's execution states including its data, heap, stack, register context, and states in relation to the entire application are captured and transferred to another workstation. In the second approach, the object for migration is a smaller process called a User Level Process (ULP) [KCO+94], which defines a register context, stack, private data and heap space. The third approach, known as Adaptive Data Movement (ADM) [POW94], is an application-level methodology for supporting adaptive computation through data movement by the application.

Although the first and second approaches support a transparent migration for applications, the third approach can provide a more efficient application-oriented workload distribution. It can also support migration across heterogeneous platforms because the migration only requires data redistribution. The challenge to this approach, however, is the complexity of the programming.

The proposed approach is similar to ADM. Both of the approaches are based on an application-level methodology, however, ADM can only apply to data-parallel applications that are running in the *pool of tasks* paradigm. As illustrated previously by different data-parallel and task-parallel applications, our approach allows the tailor-made schemes of the LWD and GWD to work for different types of programming model. Moreover, the approach provides a basic support to tackle the complexity of the programming by structuring the workloads of applications. Finally, the ADM lacks the functions for applications to interface with other external resource managing agents. The importance of these functions is explained in Chapter 5.

4.8.2 Task-to-Grain & Grain-to-Workstation Mappings

Parallel Virtual Machine (PVM) [GBD+93] is used as the message passing library for communication in the example applications. The applications are written in a single program multiple data (SPMD) model meaning that all workstations run the same program. To facilitate inter-task communication, integer task identifiers (PIDs) are used for identifying physical locations of the PVM processes[1].

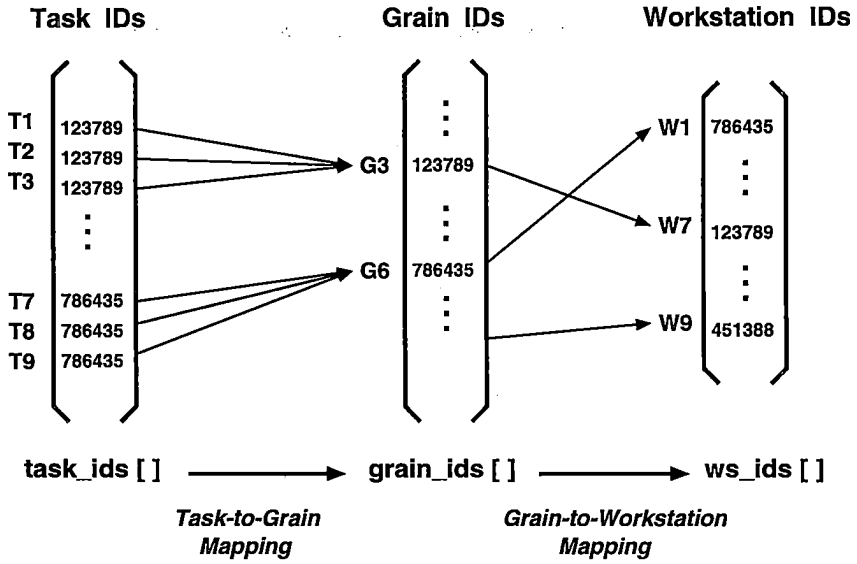


Figure 4.10: Mappings for the LWD and/or GWD

[1]The number of active PVM processes running on the workstations can be adjusted; the processes can be added or removed dynamically by pvm_addhosts() and pvm_delhosts() respectively.

Task-to-grain and grain-to-workstation mappings are used to relate the logical identification of different objects with the physical locations of execution. As shown in Figure 4.10, indices of the arrays are the the logical numbers of tasks, grains and workstations, and the contents of the arrays are the PIDs. Since each grain can only run on a workstation, the grain-to-workstation mapping is one-to-one. On the other hand, a grain can consist of more than one task, thus the task-to-grain mapping can be one-to-one or many-to-one. The PIDs of tasks, grains, and workstations are then referred to as task IDs, grain IDs, and workstation IDs respectively.

The workstation IDs are fixed at the time of program startup, but the grain and task IDs are updated whenever the task-to-grain or grain-to-workstation mappings varies. The task-to-grain and grain-to-workstation mappings are instigated by the conditional actions of the LWD and GWD. The initial grain-to-workstation mapping is determined by a simple assignment method in which grains are assigned to the most lightly-loaded workstation first. However, the grain-to-workstation mapping varies when grains migrate from one workstation to another. In addition, the task-to-grain mapping changes when grain splitting, merging, or the LWD occurs. Grain migration can also be carried out at the same time as grain merging or splitting. The mappings involved in the operations of the LWD and GWD are summarized in Table 4.2.

Workload Distribution		*Task-to-Grain Mapping*	*Grain-to-Workstation Mapping*
LWD		√	
GWD	*Grain Splitting*	√	
	Grain Merging	√	
	Grain Migration		√
	Splitting/Merging together with Migration	√	√

Table 4.2: Mappings involved in the operations of the LWD and GWD

4.8.3 Task Relocation

A task knows its next execution location from the new task ID generated from the mappings. For the LWD, tasks relocate between neighbouring grains. For the GWD, a new copy of the task IDs is sent by the Application Reporter (AR) to

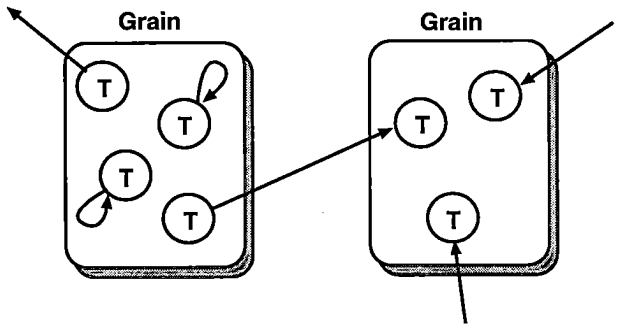


Figure 4.11: Intra-/Inter-Grain Relocation of Tasks

the grains. Since tasks in different grains check their new task IDs individually, the relocation can be carried out in parallel. This may lead to inter-grain and intra-grain relocation of tasks.

Figure 4.11 describes these two forms of task relocation. For intra-grain relocation, only the logical identification of a task needs to be updated from the task-to-grain mapping. For inter-grain relocation, tasks are relocated from one grain to another. The relocation of tasks may lead to grain migration, merging or splitting. If the grains of an application are merged to a lower partitioning level, the tasks in some of the grains are relocated to other grains. In grain splitting, some tasks of each grain are relocated to form new grains.

Functions[2] are written to set barriers for checking the conditions of the LWD and GWD. These functions probe for the control signals inside computational loops. If any control signal has been received, the conditional actions of the LWD or GWD will be invoked accordingly.

For data-parallel applications, the time for carrying out task relocation depends on the frequency of probing for control signals and the amount of program state that needs to be transferred between grains for data redistribution. Table 4.3 shows the average times taken for different operations of the GWD of the BMM in DEC3100 workstations. The operations include task relocation and communication between the AR and the grains. From the table, the values of the times[3] are proportional to task size because control signals are probed whenever a task is finished.

[2]Inside these functions, pvm_probe() is used to check the source and message labels of the incoming message.

[3]The times for grain splitting, merging, and migrating at partitioning levels one and two are similar.

Task Size	*Grain Splitting*	*Grain Merging*	*Grain Migration*
5x5	74.22	42.97	27.34
10x10	93.75	93.75	82.03
20x20	121.09	156.25	136.72
30x30	199.22	222.66	187.50
40x40	277.34	390.63	308.59

Table 4.3: Time Taken for the GWD of the BMM (in msec.)

For task-parallel applications, a global synchronization is required before any task relocation takes place so that consistency between communicating tasks can be maintained. After the synchronization, the internal states of the tasks will be transferred between grains. For the PSA, simulation event structures in the tasks will also be rebuilt after relocation. The average times taken for grain migrating, splitting, and merging of the PSA in DEC3100 workstations are 382.81 msec, 353.54 msec, and 452.43 msec respectively; each task in the grains has about 10Kbytes of internal state.

4.9 Experimental Results

The experiments are carried out on a range of workstations which consist of 50 DEC3100, 9 DEC Alpha, and 4 HP9000/700 workstations. All the workstations are connected by Ethernet on different segments. Among the high performance DEC Alpha workstations[4], there are one DEC3000/500, four DEC3000/400, and four DEC3000/300 and their memory capacities are also different. The CPU speeds of the DEC3100 workstations are relatively slow and they have smaller memory capacities[5]. The experiments are performed such that unexpected interference from other users is minimized so as to ensure a controlled environment. External workloads are generated artificially by computationally intensive loader processes.

[4]The SPEC marks (SPEC int92) of DEC3000/500, DEC3000/400, and DEC3000/300 are 84.4, 74.7, and 66.2 respectively.

[5]The SPEC mark (SPEC int92) and main memory size of the DEC3100 workstations are 7.1 and 16-24Mb respectively.

4.9.1 Adaptation of the Data-Parallel Applications

In this section, the experiments and results for testing the LWD for the blocked matrix multiplication (BMM) and the cycle-searching of a non-linear iterated function program (CSN) are presented. The data-parallel applications are run in different situations of workload imbalance caused by external loader processes and workstation heterogeneity. Tables 4.4 and 4.5 show the effects of external loader processes (LPs) on the performance of the BMM and the CSN. Since the matrix multiplication of the BMM can perform independently and deterministically, the speedup of the application increases quite linearly to the number of workstations. As can be observed from Table 4.4, the LWD improves performance by balancing the workload among workstations and does not incur heavy overheads.

For the CSN, since moving tasks from one grain to another may increase the chances of creating new cycles, extra time may be taken in searching and creating information on the new cycles. As a result, the improvement by the LWD shown in Table 4.5 is modest. The results also show that the actual speedup of the CSN is not noticeable until the data size increases to 300x300. Owing to the limitation of memory capacity in the DEC3100s, the program with this data size is executed on the other workstations and the speedup can be found in Table 4.6.

For mixed combinations of Alphas and HPs, the improvements in the performance of the applications by the LWD are presented in Table 4.6. The relative speedup of the BMM and CSN at different partitioning levels indicates that the applications are adaptable to the heterogeneity of workstations. Unlike the BMM, the CSN does not always deliver performance improvement because of the nondeterministic time for searching cycles.

4.9.2 Adaptation of the Parallel Simulator

In this section, the experimental results of performing the LWD on the parallel simulator (PSA) under different conditions are presented and evaluated. First, the effect of the time window size on the speedup of the PSA is evaluated. Secondly, the adaptability of the PSA to external workload interference is examined. The execution profile of the application exhibits a good speedup throughout its runtime under different conditions of resource availability. Finally, the effectiveness of grain-size determination is investigated when the application is executed on a cluster of heterogeneous workstations.

Data Size	*Number of Workstations*								
	9				*3*				*1*
	Without LWD		*With LWD*		*Without LWD*		*With LWD*		
	No LP	*1 LP*	*No LP*	*1 LP*	*No LP*	*1 LP*	*No LP*	*1 LP*	
480x480	60	117	61	69	177	353	179	221	530
600x600	116	232	118	132	345	538	347	402	1030
720x720	213	420	212	230	635	971	634	744	1878
780x780	268	488	269	285	803	1301	800	928	2384

Table 4.4: Performance of the BMM in a DEC3100 Cluster (in sec.)

Data Size	*Number of Workstations*								
	9				*3*				*1*
	Without LWD		*With LWD*		*Without LWD*		*With LWD*		
	No LP	*1 LP*	*No LP*	*1 LP*	*No LP*	*1 LP*	*No LP*	*1 LP*	
30x30	773	1435	830	1151	927	1662	1015	1209	1467
60x60	1374	2747	1464	1951	1438	2773	1486	2224	1712
120x120	1516	2503	1607	2073	1896	3513	2071	3006	2291
180x180	1710	3131	1880	2887	2101	4059	2207	3847	2740

Table 4.5: Performance of the CSN in a DEC3100 Cluster (in sec.)

Part. Level	*No. of Grains*	*Workstation Combinations*	*CSN*		*BMM*	
			No LWD	*W' LWD*	*No LWD*	*W' LWD*
			Relative Speedup	*Relative Speedup*	*Relative Speedup*	*Relative Speedup*
0	1	1 HP	1.00	-	1.00	-
		1 Alpha	2.56	-	1.31	-
1	3	2 Alphas, 1 HP	3.01	3.61	3.00	4.37
		1 Alphas, 2 HPs	2.92	3.68	3.97	3.97
		3 HPs	2.39	2.64	2.92	4.00
		3 Alphas	6.18	6.13	3.97	4.17
2	9	5 Alphas, 4 HPs	3.00	3.08	5.97	10.35
		4 Alphas, 5 HPs	3.23	3.71	6.00	11.18
		9 Alphas	8.95	7.63	9.05	11.79

Table 4.6: Effects of the LWD in a cluster of heterogeneous workstations (in relative speedup); data sizes for the CSN and BMM are 300x300 and 1200x1200 respectively.

4.9.2.1 Factors of Communication

The time window size of the simulation model is determined by the number of packets that can be stored in the communication link of the simulation network during packet transmission. In other words, the window size is proportional to the physical length of the links. For example, if the bandwidth of each unidirectional communication link is 424Mbps and the packets are 53 byte ATM cells, then the loading time for each packet is 1μs. If the length of the single-mode optical fibre communication link is 50km, then the size of the communication link buffer will be 250 cells. The granularity of the simulation increases with the length of the communication link.

Tables 4.7 and 4.8 show the relative speedup of the parallel simulation running on different workstations without workload distribution. The results show that the granularity has a direct influence on the performance of the simulation. When the size of the time windows increases, the speedup is improved.

Part. Level	*No. of Grains*	*No. of Tasks per Grain*	*Size of comm. link buffer* 250	500	1000	1500	2000
0	1	24	1.00	1.21	1.30	1.32	1.38
1	3	7, 10	1.96	2.73	3.27	3.52	3.66
2	9	2, 3, 4	4.36	6.31	8.13	8.93	9.38
3	24	1	7.96	11.44	18.30	21.52	22.88

Table 4.7: Effects of time window size on relative speedup (*DEC3100 Cluster*)

Part. Level	*No. of Grains*	*No. of Tasks per Grain*	*Size of comm. link buffer* 250	500	1000	1500	2000
0	1	24	1.00	1.14	1.23	1.24	1.29
1	3	7, 10	1.31	1.73	2.12	2.28	2.41
2	9	2, 3, 4	1.86	3.62	5.13	5.59	6.15

Table 4.8: Effects of time window size on relative speedup (*DEC Alpha Cluster*)

The number of tasks grouped in each grain at each partitioning level also affects the relative speedup. The predefined grain-sizes of the network consist of a partitioning of 1, 3, 9, and 24 grains. In the partitioning level that has 24 grains, there is only one task in each grain, so all the inter-task communication is done by message passing between workstations. The increase in the rate of the speedup is

far less than linear as the number of grains increases. This is because inter-grain communication is more expensive than intra-grain communication.

The relative speedup for the simulation running on DEC3100 and DEC Alpha workstation clusters is compared. Although the computing performance of a DEC Alpha workstation is much higher than that of a DEC3100, the relatively low communication performance of Ethernet limits the performance gain. The results also illustrate that the granularity of the same application will change when different workstations are used.

The above experiments are performed on workstations connected by Ethernet using the PVM communication primitive (PVM-Ethernet[6]). Besides the PVM-Ethernet, the simulation is also tested on three HP9000/700 workstations in two different cases: (1) PVM communicating over Ethernet using PVM Route Direct (PVM-Direct[7]); (2) PVM communicating over an ATM network via IP (PVM-IP/ATM).

By comparing the results in the PVM-Ethernet case, the benefit of using the PVM-Direct and the PVM-IP/ATM is small when the simulation window is large, that is when the computation-to-communication ratio is high. As the simulation window reduces to smaller sizes (say 100 packets), improvement in speedup is just less than 10% for PVM-IP/ATM and less than 20% for PVM-Direct. This shows that the bottleneck for communication performance is the overhead of supporting inter-process communication via PVM communication primitives but not the transmission delay of the network. This is due to the fact that the startup time per communication of PVM is much greater than the transfer time per byte[8].

[6]In the default situation of inter-process communication in PVM (datagram transmission), all PVM processes communicate via TCP/IP to a routing daemon within a node, and the routing daemon communicates to other daemons via UDP/IP.

[7]The PVM Route Direct option (stream transmission) can be used to allow a process to communicate to another process with UDP/IP directly without going through the routing daemon. As a result, the communication latency can be lowered.

[8]The communication time for transferring a message of m bytes can be modelled by a simple linear function [BR89]: $T_{comm} = \alpha + \beta m$, where α is the startup time and β is the transfer time per byte. The empirical values [SS94a] of α and β for PVM in 12-MIPS SUN Sparcstations interconnected by 10 Mbits Ethernet for datagram and stream transmissions are 4.527 msec, 0.0024 msec and 1.661 msec, 0.00157 msec respectively. In other words, the startup time for message transfer is a dominant factor especially when the message size is not large.

4.9.2.2 Adaptation to Workload Interference

As mentioned earlier, the granularity of the PSA increases when the size of communication buffer increases. The effectiveness of the LWD to improve performance for different granularities under workload interference is shown in Table 4.9. The results displayed in the table indicate that the LWD is very effective for the unbalanced situation when the application is running on nine workstations. This is because all the tasks in the heavily loaded grain are taken over by neighbouring grains when the partitioning level of the application equals to two.

Size of Comm. Link Buffer	*Number of Workstations*								
	9				*3*				*1*
	Without LWD		*With LWD*		*Without LWD*		*With LWD*		
	No LP	*1 LP*	*No LP*	*1 LP*	*No LP*	*1 LP*	*No LP*	*1 LP*	
250	165	318	179	158	374	1108	394	538	728
500	115	250	118	127	278	863	293	446	600
1000	92	224	95	106	223	759	224	460	553
2000	78	200	84	109	202	727	208	426	533

Table 4.9: Performance of the PSA in a DEC3100 Cluster (in sec.); each simulation take 0.2 million time units.

Figures 4.12 and 4.13 illustrate the adaptability of the PSA in a DEC3100 cluster and a DEC Alpha cluster under different situations of external workload interference. These figures show the relative speedup with and without workload distribution when different numbers of workstations are loaded with external loader processes. In the event that there is no workload distribution, the speedup drops sharply even when only one workstation is loaded with a loader process. However, if the LWD and GWD are deployed through self-tuning as described in Section 4.6.2, the speedup will only be degraded according to the number of workstations that are loaded. The number of active grains is different when the number of available workstations varies. If the number of workstations is sufficient and they are available when needed, grain migration can be carried out to maintain the maximum speedup to the migration level as shown in Figure 4.12. As the number of available workstations decreases, the simulation will either invoke the LWD or merge to a lower partitioning level. There is no indication of migration level in Figure 4.13 because migration can only be carried out within the nine workstations in the DEC Alpha cluster.

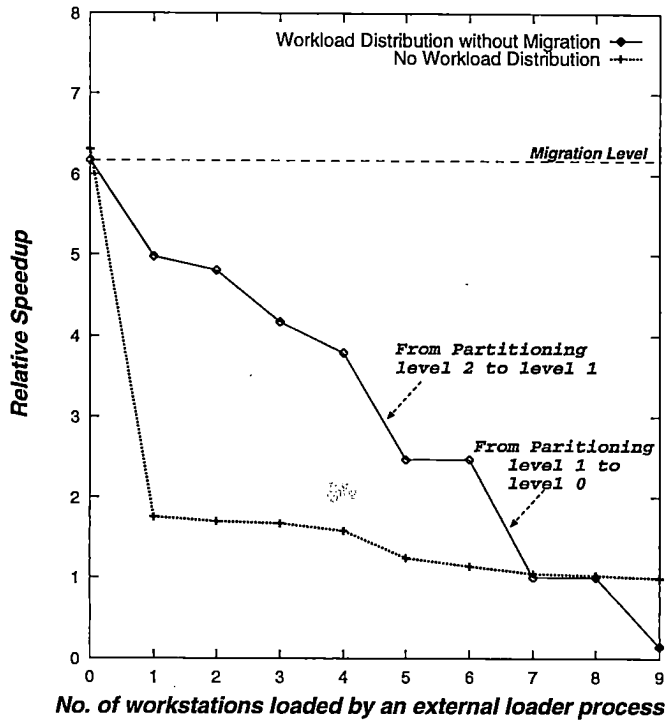


Figure 4.12: The LWD and GWD in a DEC3100 Cluster

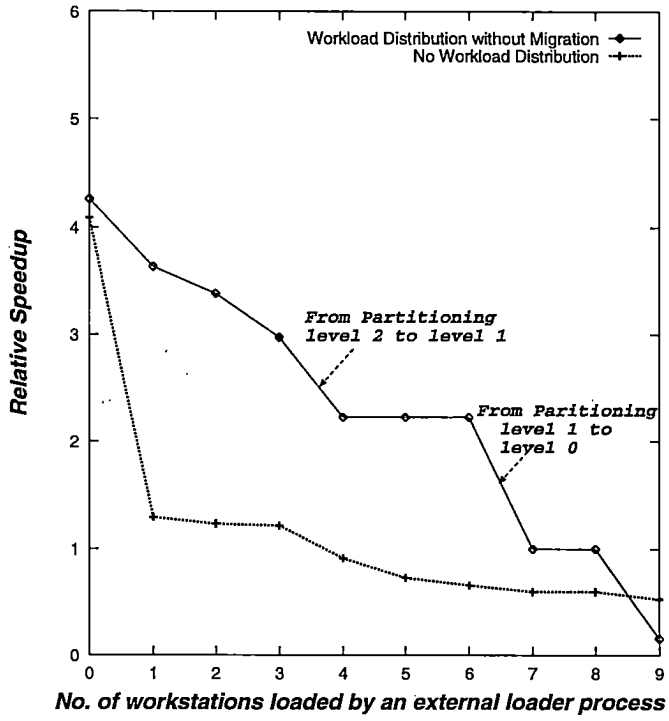


Figure 4.13: The LWD and GWD in a DEC Alpha Cluster

4.9.2.3 Heterogeneity

Table 4.10 examines the effect of using the LWD on the relative speedup of the simulation running on heterogeneous workstations. Since the memory capacities and processing speeds of the DEC Alpha workstations (Alphas) vary from machine to machine, the execution time of the parallel simulator in the Alphas is 1.73 - 2.40 times faster than the HP9000/700 workstations (HPs). The simulation is tested on different combinations of workstations at different partitioning levels. For the combination of 5 Alphas and 4 HPs, all the eight tasks from the 4 HPs are taken over by the virtual tasks in the Alphas because of the differences in computing speeds between Alphas and HPs. For the other combinations of 9 grains, a different number of tasks are taken over by the virtual tasks from the slower machines to the faster machines. The results also show that the simulation is adaptable to the heterogeneity among the nine Alphas.

No. of Grains	*Workstation Combinations*	*Without LWD*	*With LWD*	
		Relative Speedup	*Relative Speedup*	*No. of virtual tasks*
1	1 HP	1.00	-	-
	1 Alpha	1.73 - 2.40	-	-
3	2 Alphas, 1 HP	2.37	3.09	2
	1 Alpha, 2 HPs	3.13	3.05	0
	3 HPs	2.22	2.22	0
	3 Alphas	4.61	4.27	0
9	5 Alphas, 4 HPs	6.03	7.58	8
	4 Alphas, 5 HPs	4.35	6.71	10
	9 Alphas	8.39	9.79	2

Table 4.10: Adaptation to a cluster of heterogeneous workstations; size of communication link buffer is 1000.

4.10 Summary

Based on the reconfigurable software structure described in Chapter 3, the design and implementation of two complementary schemes for supporting adaptive execution have been presented in this chapter. The Local Workload Distribution (LWD) is designed to tackle local workload imbalances, and the Global Workload Distribution (GWD) aims to solve significant workload imbalances caused by external workload interference. The performance and adaptability of the LWD

and GWD for different data-parallel and task-parallel example applications have been examined. The experimental results show that a performance gain can be sustained when the applications are running in unbalanced situations owing to workstation heterogeneity and external workload interference, though the gain may be limited if the application is nondeterministic.

Chapter 5

Dynamic Space-Sharing through Competition

5.1 Introduction

Running parallel applications on the shared-network environment of workstation clusters brings a new challenge to existing operating systems in which processes are only scheduled locally by the kernels of the operation system on each workstation. Resource sharing is necessary among the applications that are running on the same clusters. Performance degradation will occur if applications are not aware of the coexistence of other applications when contending for resources. Although the total number of workstations in the clusters may be quite large, resource contention between applications will still be high if the number of high-performance workstations is smaller than the number of low-performance workstations.

The scheduling system for parallel computing on workstation clusters should be able to allocate resources among a number of competing applications from multiple users. The maximization of overall resource utilization and the minimization of the turnaround time for individual applications are two primary objectives to pursue. Without external scheduling mechanisms, both resource utilization and application turnaround time will be adversely affected. The effect of resource contention on applications is similar to the performance degradation owing to the interference of external loader processes reported in Section 4.9.

In this chapter, a prototype scheduling system for parallel applications called

Comedians is described and evaluated. In this system, parallel applications are regarded as adaptive computation agents that compete with one another for resources. The system provides supporting functions which include load monitoring and forecasting, application interfacing, and auctioning. The Comedians system aims to allocate workstations efficiently and fairly to parallel applications on a cluster of workstations. The parallel applications respond to the resource allocations through reconfiguration.

5.2 Space-sharing and Time-sharing

In Chapter 4, an application adapts to the environment by a self-tuning mechanism based on the information reported by its Application Reporter and grains. The self-tuning mechanism invokes Global Workload Distribution (GWD) which is used to alleviate the problems associated with substantial changes in resource availability. This is done by grain splitting, merging and migration. However, the self-tuning mechanism of each application does not co-ordinate with other applications to identify available resources. If other parallel applications are also running on the same clusters, the self-tuning mechanism is insufficient to guarantee a good resource utilization and a fair allocation among applications. A scheduling system is therefore required to provide better resource allocation to the applications. The conditional actions of the GWD should be deployed according to these allocations.

There is a volume of previous research on time-sharing and space-sharing approaches applied to scheduling on multiprocessor systems. However, only a few research systems applied these approaches to scheduling parallel jobs running on workstation clusters. In the time-sharing and space-sharing approaches, a set of jobs are sharing processors in the time and space domains respectively. The time-sharing approach is not appropriate for the environment of workstation clusters because of overheads in swapping and scheduling. Research on mixing time and space sharing in a workstation cluster environment has also been reported by Turner *et al.* [TNC94]. However, this approach requires careful control when mixing applications with different communication and computation characteristics, and does not guarantee performance gains.

Space-sharing is a commonly pursued strategy for job scheduling on partitionable multi-processor machines. In this strategy, the parallel jobs of an application are given exclusive access to a set of processors. There are three approaches to space-

sharing, namely static, adaptive, and dynamic. The static space-sharing approach can achieve low system overheads and simplicity by running parallel jobs on a fixed number of machines. For the adaptive space-sharing approach, the number of machines assigned for execution depends on the current system state at the time of application arrival. Among the approaches, only the dynamic space-sharing approach is adaptable to changes in workload interference and resource availability because the number of assigned processors can change during the execution of an application.

Regarding workstation clusters as a partitionable parallel computing machine, a new scheduling technique which adapts the idea of dynamic space-sharing is proposed. The design and implementation of an environment which supports dynamic space-sharing through competition is the centre-piece of the Comedians system. The inter-application competition is organized by this system using the mechanisms of auctioning and bidding.

5.3 Distinctive Features of the Comedians System

5.3.1 Related Competitive Approaches

Studies have shown that the microeconomic approach is an effective way to manage the resources of a distributed computer system. Algorithms are derived from this approach to allocate different types of resources, such as the processor and storage systems [DM88], communication channels [KSY85], file systems [KS86], and memory systems [CH93]. In addition, various studies have been conducted on the computational market for resource allocation of independent and parallel applications [FYN88], [MFGH88], [WHH+92]. These studies are those most related to the work presented in this chapter.

In [FYN88], microeconomic algorithms for load balancing in distributed computer systems were studied by Ferguson *et al.* However, only independent jobs were considered in their algorithms. Each job receives a fund upon entry to the system. This fund (money) is used to bid for the resources needed, according to a preference rule. A preference of one processor to another can be based on service time only, price only, or both. On the other hand, the processors auction their computation and communication resources with the objective of maximiz-

ing their profit. Ferguson *et al* have found that their microeconomic algorithms can achieve effective load balancing compared to a traditional non-competitive algorithm.

The other study is known as *Enterprise*; it is a market-like task scheduler for distributed computing environments suggested by Malone *et al.* [MFGH88]. In this study, the jobs (clients) ask for bids by announcing their requirements, and the idle processors (contractors) bid in response. The tasks, in turn, choose the contractor with the best bid. Again the work only involves independent jobs, and there is no market-like price control mechanism that considers the supply-demand element in the bidding process.

Waldspurger *et al.* [WHH+92] conducted a study on *distributed computational economy (Spawn)*. This is meant to support parallel as well as independent jobs. The processors auction the next available slice of processing time. Depending on the strategy, an auction may consider giving discounts, or raise the prices. The parallel application is represented by a tree of tasks. Each task is funded from the root, to bid in the auction. The experiments conducted were based on Monte Carlo applications which are suitable for decomposition into a number of tasks, if so required. They suggested the use of a simple sharing rule in place of an auction to determine prices, because of the increased overhead when the granularity of the tasks decreases.

Although the Spawn system [WHH+92] uses auctioning and bidding to schedule tree-structured applications, its mechanisms of auctioning and bidding are different from the Comedians system. Since the allocation of computational resources is the main objective of the Spawn system, the supply and demand of the resources determine the price and bids. In the Comedians system, every application is adaptable to workload imbalance as well as resource allocation. The Comedians system not only tackles the resource allocation problem, but also maximizes the execution speed of individual applications. Therefore, bids in the Comedians system are formulated in terms of runtime performance information supplied by applications. In addition, the frequency and the overheads of its auctions are much smaller than the Spawn system because applications bid for the sole execution rights of workstations (space-sharing) instead of time slices of workstations (time-sharing).

5.3.2 Other Scheduling Systems

As mentioned in Chapter 2, previous work on scheduling for parallel applications on workstation clusters appears in various references [LLM88, GS91, NR94, SOW94]. Nevertheless, they ignore the problem of fair allocation of resources to parallel applications running on the same workstations. The Comedians system addresses the problems of fair and efficient allocation by taking a different approach. With the motivation of increasing speedup, parallel applications in the Comedians system compete for workstations through auctions. No prior knowledge about the performance behaviour of applications is required because the values of bids are formulated by performance information collected at runtime.

Finally, one of the most distinctive features of the Comedians system is the capability of interfacing with the support for adaptive execution of parallel applications. The Comedians system provides a resource allocation service to the adaptive parallel applications and the applications respond to the allocations by workload distribution. The system forms an important part of an integrated framework to support adaptive parallelism in a multi-user environment.

5.4 The Comedians System

A prototype runtime system called *Comedians* [Shu96] (Competitive Environment for Distributed and Adaptive Applications) has been developed to allocate resources on a cluster of workstations to parallel applications through competition. The objectives of the Comedians system are to maximize the speedup of individual parallel applications and, at the same time, to allocate workstations efficiently and fairly to the applications. Individual applications respond to the allocations made by the Comedians system by the conditional actions of the GWD. The system can also work in conjunction with the LWD in a complementary way but can also work independently if either of them is not available. This work suggests an integrated solution to the issues of runtime adaptation and parallel application scheduling in a multi-user environment.

The Comedians system comprises two types of agent process that can offload the applications' work of identifying workload imbalances owing to external interference and negotiating resources in workstation clusters. The first type of agent process is called a *Cluster Scheduler* (CS). Each CS is a resource managing agent that manages a cluster of workstations and coordinates all the parallel

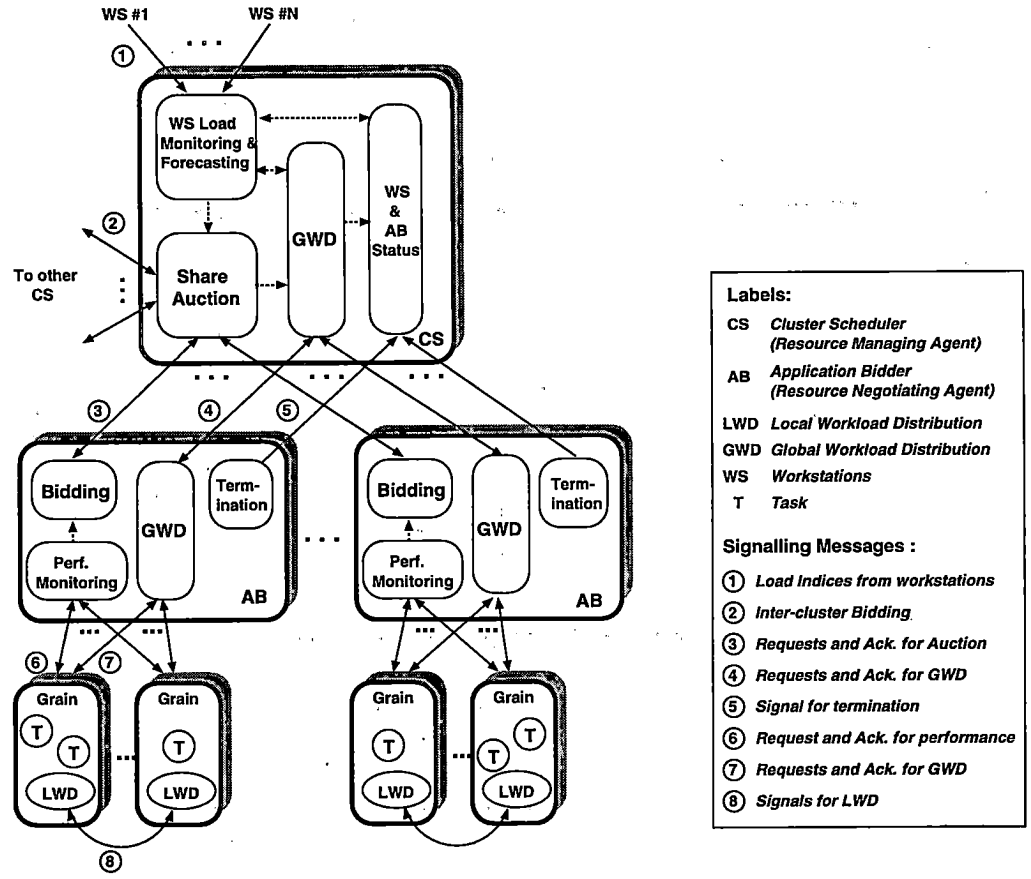


Figure 5.1: Functional Blocks of the Comedians System

applications that are running on the cluster. A CS is responsible for:

- Maintaining status information of the workstations and parallel applications;
- Monitoring and forecasting the availability of workstations in its cluster;
- Holding auctions among parallel applications and generating the auction results;
- Reporting the auction results which are the allocations of workstations to the bidding applications.

The other type of agent process is called an *Application Bidder* (AB). Each AB is a resource negotiating agent that provides an interface between the grains of an application and the coordinating CS. Its functions consist of:

- Monitoring and collecting the runtime performance of its grains;
- Sending the performance information to the CS that is holding an auction;
- Receiving the auction results from the CS and adapting the new allocation by enforcing the GWD.

Figure 5.1 depicts the functional blocks of the system and the signalling messages between the grains of applications. The Comedians system provides answers to when, where and how the grains of an application should be activated.

5.5 Load Monitoring and Forecasting

In the Comedians system, only available workstations will be allocated to parallel applications. It is therefore important to have an accurate measurement and definition of the availability of workstations. The typical definition of workstation availability [ML91, DO91, ADV+95] is based on a low load average and the lack of keyboard activity for a specified period of time. The Comedians system defines workstation availability not only by the current load measurements, but also by the predicted future load profile for a specified time period.

Workloads in the system are classified as independent workloads and parallel workloads. The parallel workloads come from parallel applications that are running on the Comedians system. If a workstation is occupied by a grain of an application, the workstation will be marked as unavailable. The independent workloads include serial jobs generated by the owners of workstations and other users. The independent workloads of each workstation in a cluster are collected periodically by the CS so that the current and future availability of each workstation can be estimated and forecasted.

The 5-minutes load average of a workstation $x_{d,h,m}$ reported by each workstation, is used to formulate the current load situation, where d denotes the day, h denotes the hour, and m denotes the number of the 5-minutes load average. The one-hour load average $X_{d,h}$ is calculated from the average value of the 5-minutes load average as $X_{d,h} = \frac{1}{12} \sum_{i=1}^{12} x_{d,h,i}$.

5.5.1 Load Forecasting Model

In [Mut92], the working habits of the users of privately owned workstations were predicted using daily correlation. The predicted workload of the hth hour of the dth day $\hat{Z}_{d,h}$, is formulated by equation 5.1. The constant value A represents the significance of the past predicted values. If A is equal to 1, the prediction becomes static. On the other hand, if A is equal to 0, the current predicated value is equal to the one-hour load average of the previous day. As noted by Mutka [Mut92], better prediction can be made if the days are distinguished between weekdays and weekends.

$$\hat{Z}_{d,h} = A \cdot \hat{Z}_{d-1,h} + (1 - A) \cdot X_{d-1,h}, \quad \text{where } 0 \leq A \leq 1 \tag{5.1}$$

However, we also observe that workload can be affected by *ad hoc* computing requirements. We therefore combine the daily correlation with an hourly correlation to generate better load prediction that can respond both to current fluctuations and daily trends of workstation load. Furthermore, the *Exponential Smoothing Method* [Gra77] is applied to formulate the future workload of a workstation for a forecasting period (s hours). The predicted values of the future workload and the current hourly workload can be calculated by equations 5.2 and 5.3 respectively. The forecasting period represents a time window for future workload formulation. For example, if h is the current hour, $\hat{Y}_{d,h+1}$, $\hat{Y}_{d,h+2}$, and $\hat{Y}_{d,h+3}$ are the future workloads of a workstation for the next three hours ($s = 3$).

$$\hat{Y}_{d,h+i} = B^{i+1} \cdot X_{d,h-1} + (1 - B^{i+1}) \cdot \hat{Z}_{d,h+i} \tag{5.2}$$

$$Y_{d,h} = C \cdot \hat{Y}_{d,h} + (1 - C) \cdot x_{d,h,m} \tag{5.3}$$
$$\text{where } 0 \leq i \leq s, \quad 0 \leq B, C \leq 1$$

The combination of the daily and hourly correlations is illustrated by Figure 5.2 in which the predicted workstation load of future hour $\hat{Y}_{d,h}$ is calculated from the load average of the previous hour and the predicted value owing to daily correlation. The value for the current hour $Y_{d,h}$ is formulated by the predicted value and current 5-minutes load average so that the most recent load information can be reflected.

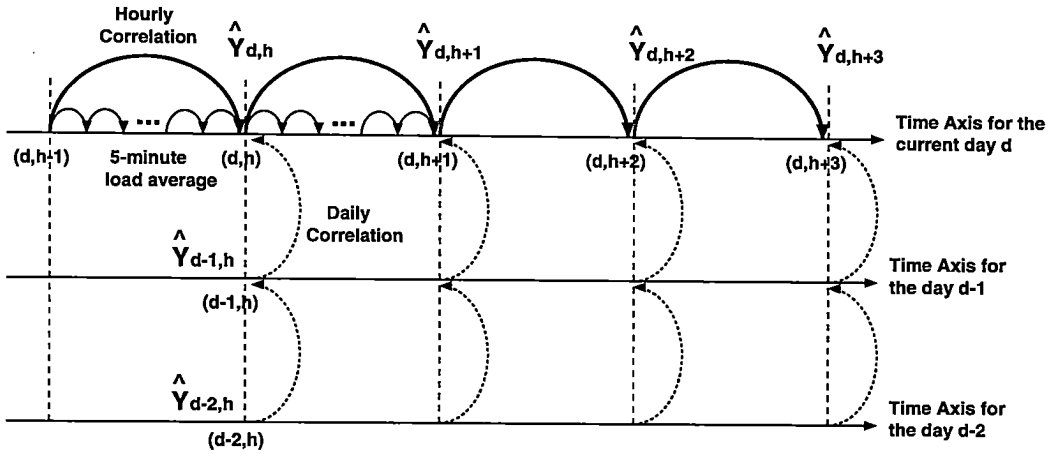


Figure 5.2: Workstation Load Forecasting

The relative importance of the daily and hourly correlations determines the values of A and B. The hourly correlation should be a more prominent factor than the daily correlation for publicly owned workstations. On the hand, for privately owned workstations, the daily correlation may be the dominant factor. The value of C shows the relative importance of the hourly prediction and the most recent load.

Since the definition of workstation availability depends on the current and future workstation load, the availability of a workstation can be deduced from the values of $Y_{d,h}$ and $\hat{Y}_{d,h+i}$. If these values are less than a threshold value, it is then said to be available. An average of the predicted values which is calculated by equation 5.4 can also be used to rank the availability of workstations. Similar forecasting techniques can be applied to different types of resource, such as memory and communication utilization.

$$\sum_{i=0}^{s} \hat{Y}_{d,h+i}/(s+1) \tag{5.4}$$

The forecasting model presented in this section can also work in conjunction with the *Social Contract* proposed by Arpaci *et al.* [ADV+95]. The Social Contract guarantees that an individual user will not be delayed more than a specified number of times during any day. If a user has been delayed the specified number of times, that user's workstation will not be used by parallel applications temporarily. The proposed forecasting model helps to provide the estimated current and future availability of workstations and the Social Contract minimizes possi-

ble interference from parallel workloads to users if misjudgements do occur. As a result, this modified definition of availability conforms more closely to actual usage and is less obtrusive than the other definitions, such as a fixed time period of low load average and low keyboard activity.

5.5.2 Trace Evaluation

To evaluate the sensitivity of the forecasting equations, the predicted values of hourly load are formulated from traces of load collected at a DEC5000/133 server workstation called *nene*. The traces consist of the 5-min load averages from nene. A week of traces are used to formulate a daily trend in each hour as shown in equation 5.1 and the value of the constant A is set to 0.6. The workloads of nene during the period of trace collection are mostly generated by independent jobs. For the rest of the evaluation in this section, the daily trend is applied to calculate the predicted current and future hourly load.

Figures 5.3 and 5.4 show the actual and predicted current hour load of nene on two weekdays; one is with a less bursty load and the other is with a more bursty load. As can be seen from these figures, the predicted current hour load follows closely the overall trend of the actual current hour when the value of the constant B is small. However, instantaneous response when using this value of B for fluctuating load is poor. This implies that the value calculated by equation 5.3 should be used for predicting the current hour load, because it can reflect the value of the most recent load.

For the future load for a period of time, Figures 5.5 and 5.6 illustrate the effects of changing the value of the constant B of equation 5.2. The predicted values of the future load are used to rank the future availability of workstations within a certain period of time. The respective periods of prediction on the day with a more bursty load shown in Figures 5.5 and 5.6 are two hours and five hours. The actual and predicted period means are compared to test the accuracy of the prediction. The actual period mean is smoothed out when the period of prediction is long. It is found that equation 5.2 can provide a good estimation of the future load when the value of the constant B is less than or equal to 0.3. From the results shown in this section, equations 5.1, 5.2, and 5.3 should be used together in predicting the current hour load and the future period load.

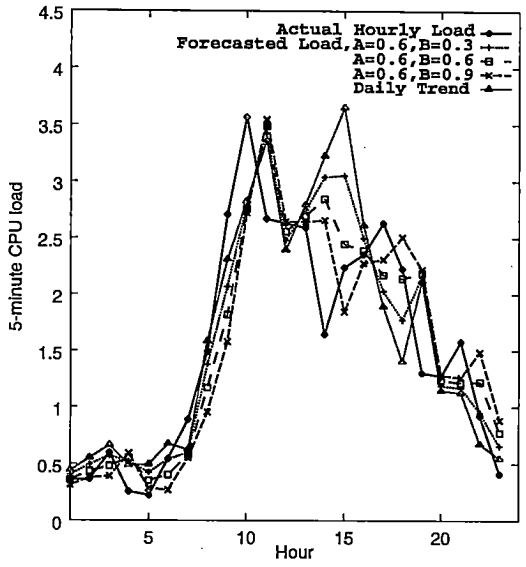


Figure 5.3: Actual Current Hour Load and Predicted Current Hour Load (On a day with less bursty load)

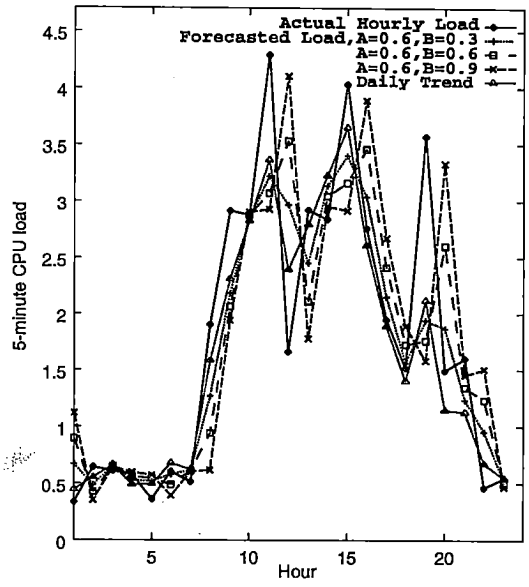


Figure 5.4: Actual Current Hour Load and Predicted Current Hour Load (On a day with more bursty load)

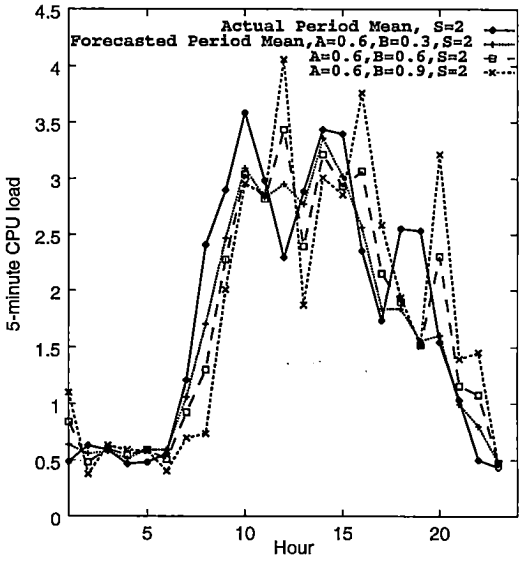


Figure 5.5: Actual Period Mean and Predicted Period Mean (Forecasting Period=2 hours)

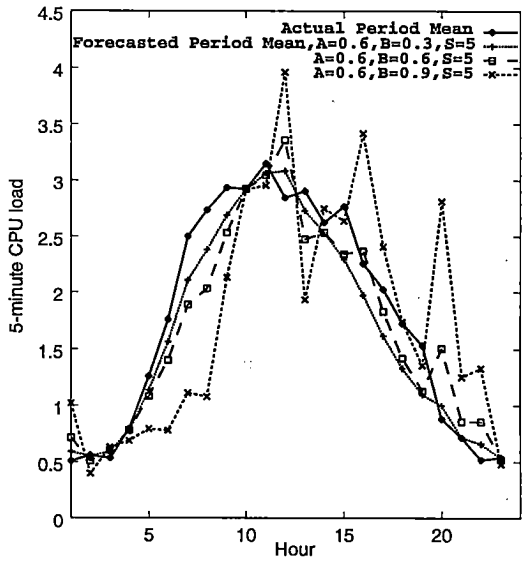


Figure 5.6: Actual Period Mean and Predicted Period Mean (Forecasting Period=5 hours)

5.6 Responses of Adaptive Parallel Applications

As mentioned in Section 4.6, the actions of switching between partitioning levels and relocating the grains during execution which are done by Global Workload Distribution (GWD) include grain splitting, grain merging, and grain migration. However, the full adaptability of applications can only be explored by the GWD when the capabilities for load monitoring, forecasting and external scheduling are provided. With this adaptability, not only can the resource utilization of workstation clusters be enhanced, but the fragmentation problem[1] can also be resolved, because the GWD provides preemptive mechanisms for an application to reconfigure itself during the lifetime of its execution.

In the Comedians system, the GWD is invoked by the CS that monitors the workstation cluster on which the application is running. The CS informs the application when any external workload interference is detected or any auction is being held. The AB of the application can then respond by enforcing the GWD.

In response to external workloads, grain migration or grain merging are invoked to release heavily loaded workstations. Grain merging is applied if there are not enough available workstations to maintain the present partitioning level. On the other hand, if there are sufficient newly available workstations, grain splitting will be deployed. New grains are created by redistributing the tasks from existing grains to the newly available workstations.

The LWD will be activated whenever there is no CS or no available workstations for migration. Moreover, it works in conjunction with the GWD by refining the grain-size of an application at different partitioning levels. Since the availability of a workstation is judged by the predicted values of the current and future load, if grains are merged to a lower partitioning level or grains are deactivated by the LWD due to the arrival of external workloads, then its partitioning level will not be restored or split to a higher level right after the external workloads have departed. This prevents the LWD and GWD from deploying too frequently. The GWD may also be the outcome of competition among parallel applications.

[1]Processor fragmentation occurs when the number of assigned processors is smaller than the number of processors needed to perform the current workload.

5.7 Auction and Bidding

5.7.1 Share Auction

An auction is a public sale of goods where people offer higher and higher bids until the goods are sold to the person who offers the highest bid. For resource allocation in computer systems, an auction is abstracted as a model for computational resource management. The concept of a *share auction* [Wil79] is used to resolve the competition of resources in a workstation cluster among parallel applications. In a share auction, each buyer bids for both a quantity and a price. The buyer with the highest price receives the quantity for which he bids at that price. If any product remains, the buyer with the second highest price takes the quantity he bids for, and so forth.

In the context of the Comedians system, the buyers are the Application Bidders (ABs) of the applications, the quantities are the desired number of workstations that the ABs want to occupy, and the prices are the bid values of the ABs. Although there are many other auction models for selling resources [BB92], all except the share auction have a single winner in each auction. In a share auction, resources can be allocated to multiple bidders in a single auction according to the bid values.

Share auctions are scheduled periodically, but the schedule period can be shortened by an increase or decrease in resource availability. The next auction will be rescheduled to an earlier time whenever a new application arrives. As shown in Figure 5.7, when a CS holds an auction, it announces the news of the auction to all the ABs of the existing applications. The ABs respond by requesting their grains to report the completion time of a predefined fraction of a task at the current partitioning level. The value of the completion time includes the average computing and communication time required to finish the fraction of the task. The measurement of the completion time is carried out at the iteration boundaries of the task. Then the ABs submit the average value of the collected completion times from all their grains to the CS; this average completion time is known as *unit task execution time.* The average value is used instead of the maximum value because it is the approximate value of the completion time when the LWD is applied. Upon receipt of the unit task execution time from the ABs, the CS will store this information and formulate bids on behalf of the applications.

In summary, the CS that is holding the auction has to update the following

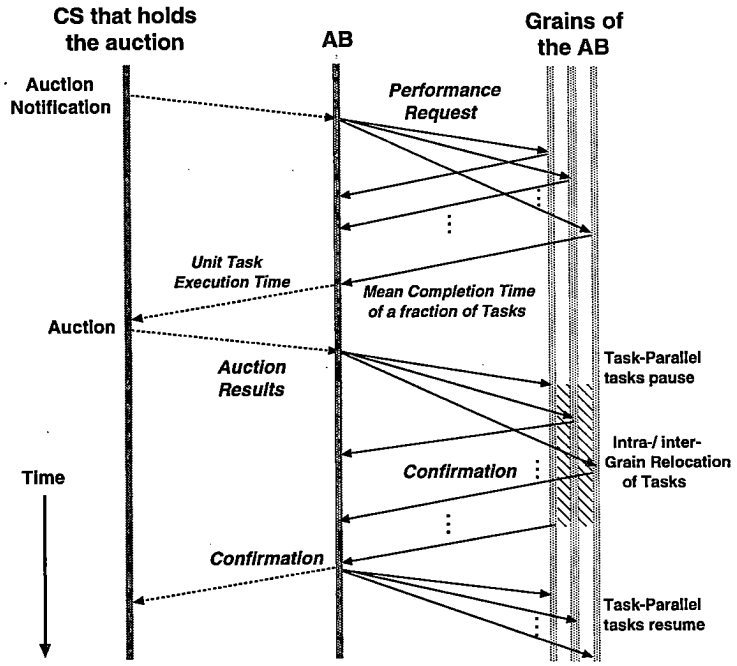


Figure 5.7: Interaction between Cluster Scheduler (CS), Application Bidder (AB), and Grains during an auction

information before the result of an auction is determined.

- The values of the unit task execution time of the participating applications.
- The current status of the applications that includes the current partitioning level and the workstations they are now running on.
- The current and future availability of the workstations of its cluster.

The auction results are multicast to the ABs after the auction has been resolved. The results of an auction sent to an AB contains a new partitioning level and a set of workstations that are available to the grains of the AB. The AB then invokes the GWD by transforming the auction results to task-to-grain and grain-to-workstation mappings. A new copy of the task IDs is multicast to its grains so that the tasks inside each grain can perform inter-grain or intra-grain relocation concurrently. The mechanisms of the relocation have been described in Section 4.8.3.

5.7.2 Bid Formulation

Through the share auction, workstations can be shared among the ABs in proportion to the values of their bids. The values of bids are formulated by the performance information of applications which is extracted at runtime from the grains of the applications. Therefore no prior knowledge about the runtime behaviour of applications is required. From the past and present performance information, the CS can formulate three types of bids: forward bid, backward bid, and potential forward bid.

The *forward bid* (FB_i) of an application AP_i is the ratio of the speedup of splitting to one higher partitioning level to the speedup of the current level. The bid value exhibits the ability to improve speedup by splitting to one higher partitioning level. This value can be formulated by the ratio of unit task execution times reported by the ABs as shown in equation 5.5, where $U_i(0)$ is the unit task execution time of AP_i at the lowest partitioning level, and $S_i(c)$ and $U_i(c)$ are the speedup and unit task execution time at the current partitioning level, respectively.

$$FB_i = \frac{S_i(c+1)}{S_i(c)} = \frac{U_i(0)/U_i(c+1)}{U_i(0)/U_i(c)} = \frac{U_i(c)}{U_i(c+1)} \tag{5.5}$$

The *backward bid* (BB_i), on the other hand, shows the loss in speedup of an application AP_i by releasing workstations to another application AP_f with the maximum forward bid to split. The bid value of application AP_i (equation 5.6) is obtained from the ratio of the unit task execution time at level $c-k$ to the unit task execution time at level c, where k is the level to which AP_i merges so that $\triangle N$ workstations can be freed. $\triangle N$ is the number of extra workstations demanded by AP_f for splitting; it is formulated by equation 5.7, where N_{c_f} is the number of workstations that AP_f requires at partitioning level c_f, and N_{avail} is the number of available[2] workstations.

$$BB_i = \frac{S_i(c)}{S_i(c-k)} = \frac{U_i(c-k)}{U_i(c)} \tag{5.6}$$

$$\triangle N = N_{c_f+1} - N_{c_f} - N_{avail} \tag{5.7}$$

[2]The definition of the availability of a workstation is described in Section 5.5.

The newly arrived and the first-time auction applications take priority over other applications when splitting because the reported unit task execution time at different partitioning levels helps the CS to formulate bids for their future bidding. The latest unit task execution time at each partitioning level of applications is recorded by the CS. This information is useful to formulate the forward and backward bids because both of the bids require previous information of unit execution time for their formulation.

For the applications that do not have known performance information about their higher partitioning levels, their *potential forward bids* will be used for bidding. A potential forward bid of an application AP_i is the ratio of the efficiency of the current partitioning level $E_i(c)$ to the efficiency of the previous partitioning level $E_i(c-p)$ (equation 5.8). The bid shows the potential of splitting an application to a higher partitioning level.

$$PB_i = \frac{E_i(c)}{E_i(c-p)} = \frac{S_i(c)/N_i(c)}{S_i(c-p)/N_i(c-p)} = \frac{U_i(c-p)/U_i(c)}{N_i(c)/N_i(c-p)} \tag{5.8}$$

5.7.3 Holding an Auction

When an application arrives, it will be informed by the CS via the AB about its initial allocation of workstations. The AB will activate its grains on the most available workstations at a partitioning level that is nearest but not equal to the maximum partitioning level N_{max}. The CS will then schedule the time for its first auction. The conditional actions for each auction are shown in Figure 5.8, where P denotes the set of all participating applications in an auction.

All the applications participating in an auction will go through these actions sequentially, but any application that has taken part in any one of these actions is removed from the set P. First, the first-time auction applications will have the highest priority to split using the available workstations or the workstations from the application with the minimum backward bid. Secondly, the application with the maximum forward bid will split to one higher level using the available workstations or the workstations from the application with the minimum backward bid. To ensure stability, no action will be taken if the maximum forward bid is less than the minimum backward bid by a factor of ψ, ($\psi > 1$). Finally, for the

Action 1: For the first-time auction AP_n, *where* $n \in P$
If $N_{avail} \geq N_c$, *where* N_c is the nearest number to N_{max}
AP_n splits to level c,
Else
If $\exists\ AP_b$, *s.t.* AP_b has $Min(BB_j)_{\triangle N}$ or $Min(PB_j)$, $j \in P$
AP_n splits to level c
AP_b merges from level c_b to $c_b - k$

Action 2: For the AP_i **with known** FB_i, *where* $i \in P$
Choose the AP_f with $Max(FB_i)$
If $N_{avail} \geq (N_{c_f+1} - N_{c_f})$
AP_f splits from level c_f to $c_f + 1$,
Else
If $\exists\ AP_b$, *s.t.* $Max(FB_i) > (\psi \cdot Min(BB_j)_{\triangle N})$, $\psi > 1$, $i \neq j$
AP_f splits from level c_f to $c_f + 1$
AP_b merges from level c_b to $c_b - k$

Action 3: For the AP_i **with unknown** FB_i, *where* $i \in P$
If $\exists\ AP_f$, *s.t.* AP_f has $Max(PB_i)$ and $(N_{avail} \geq (N_{c_f+1} - N_{c_f}))$
AP_f splits from level c_f to $c_f + 1$

Figure 5.8: The Conditional Actions for the Share Auction

applications with unknown forward bid, the application with the highest potential forward bid will split first. In addition to the above actions, an application will merge to a lower partitioning level if no gain in speedup has been reported after splitting.

For all the conditional actions shown in Figure 5.8, Action 1 enables all the newly arrived applications to obtain performance information for future bidding. Action 3 allows advancement of execution performance of the existing applications by using available workstations and Action 2 corrects any previous actions so that a better allocation can be obtained. In order to reduce the sensitivity of these actions, an application will only split to a higher partitioning level if the gain in speed is great enough.

The conditional actions are aimed at finding stable allocations for the applications in the Comedians system. The stable allocation can be termed as a *Pareto efficient allocation* [Var92], that is an allocation for which there is no way to make all applications better off. If the runtime performance of the applications

does not vary frequently, the applications will be settled in a Pareto efficient allocation because the actions can improve and correct the current allocation within a sensitivity level controlled by the parameter ψ.

5.8 Experimental Results

In this section, the experimental results for testing the Comedians system are presented. From the results, the effectiveness of the Comedians system is demonstrated. The three applications involved in the experiments are the block-matrix multiplication (BMM), the cycle-searching of a non-linear iterated function program for cryptography (CSN), and the parallel simulator of an ATM Network (PSA). These applications are the same data-parallel and task-parallel applications that are used for experimenting the adaptive workload distribution described in Chapter 4. In the experiments, the availability of workstations is measured by the current load of the workstations.

5.8.1 Parameter Tuning

The values of the following parameters of the system can be fine-tuned to get the right balance between sensitivity and efficiency.

- **The value of ψ**
  The value of ψ affects the sensitivity of invoking *Action 2* in the share auction. If the value is too small, excessive overheads will be incurred because applications overreact to this action by grain splitting and merging. On the other hand, the benefits of better allocation will be reduced if the value is set too high. A suitable value should be chosen to avoid both of the above cases. Throughout the experiments, the value of ψ is set to be 1.2.

- **The minimum inter-auction period**
  This minimum inter-auction period should be adjusted by the arrival and departure rate of parallel applications. For example, if the system becomes more heavily loaded and dynamic, the period should be shortened. In the experiments, no application will participate in an auction if the time between two auctions is less than 20 seconds.

- **The frequency of probing control signals**
  For invoking the GWD and collecting performance information, control signals are sent from the AB to the grains of its application. The frequency of probing the control signals of the AB in the grains can be adjusted to reduce overheads and keep a suitable level of sensitivity.

- **Exemption conditions for auctions**
  Each application in the Comedians system is regarded as an autonomous agent; it can join or exempt itself from an auction according to its current situation. An application will be exempted from an auction under the following conditions.

  - When the time of auction is near the end of its computation;
  - When it had already reacted and committed to a predetermined number of auctions;
  - When there is not enough time for measuring a reliable unit task execution time at the current partitioning level.

  The first two exemption conditions grant the application a higher priority for maintaining the current allocation of resources. The last condition guarantees that the application has enough time for formulating bids based on the measured unit task execution time. For some nondeterministic applications such as the CSN application mentioned in Chapter 4, more time may be required for measuring the unit task execution time.

5.8.2 Dynamics of Competition

To test the dynamics of competition in the Comedians system, different numbers of example applications are run on a cluster of DEC3100 workstations. All the applications can be partitioned into one grain (level 0), three grains (level 1), and nine grains (level 2). Tables 5.1-5.4 present the results of the auctions in different scenarios. The workstation load collection is performed by the Cluster Scheduler at every 20s. The *action taken* by each auction refers to the actions shown in Figure 5.8.

Tables 5.1 and 5.2 show two similar scenarios in which three applications are competing for resources in a cluster of twenty workstations. In the first scenario, the applications arrive at different times. The first two applications (BMM1 and

Time of Auction (in sec.)	*No. of occupied workstations*			*Action Taken*	*Comments*
	BMM1	*BMM2*	*PSA1*		
60	3	-	-	-	BMM1 arrives and splits to level 1
100	9	3	-	Action 1	BMM1 splits by using avail. ws, BMM2 arrives and splits to level 1
160	9	9	1	Action 1	BMM2 splits by using avail. ws, PSA1 arrives and runs at level 0
200	3	9	3	Action 1	PSA1 splits by using BMM1's ws, BMM1 merges, BMM1's BB=2.914, BMM2's BB=3.020
240	9	9	1	Action 2	BMM1 splits by using PSA1's ws PSA1 merges, BMM1's FB=3.010, PSA1's BB=2.865, Note that this operation will not occur, if $\psi \geq \frac{BMM1's\ FB}{PSA1's\ BB} = 1.05$

Table 5.1: Three applications in a cluster of twenty workstations (Scenario One)

Time of Auction (in sec.)	*No. of occupied workstations*			*Action Taken*	*Comments*
	BMM1	*BMM2*	*PSA1*		
60	3	3	3	-	All applications arrive, Each takes 3 avail. ws
100	1	9	9	Action 1 Action 1	BMM2 splits by using avail. ws PSA1 splits by using BMM1's ws, BMM1 merges
140	3	9	3	Action 2	BMM1 splits by using PSA1's ws, PSA1 merges, BMM1's FB=3.213, BMM2's BB=3.109, PSA1's BB=2.172
260	9	9	-	Action 3	PSA1 terminates and frees 3 ws BMM1 splits by using avail. ws

Table 5.2: Three applications in a cluster of twenty workstations (Scenario Two)

Time of Auction (in sec.)	*No. of occupied workstations*				*Action Taken*	*Comments*
	BMM1	*BMM2*	*PSA1*	*PSA2*		
60	3	-	-	-	-	BMM1 arrives and splits to level 1
100	9	3	-	-	Action 1	BMM1 splits by using avail. ws BMM2 arrives and splits to level 1
140	9	9	3	-	Action 1	BMM2 splits by using avail. ws PSA1 arrives and splits to level 1
180	3	9	9	-	Action 1	PSA1 splits by using BMM1's ws
220	9	9	3	-	Action 2	BMM1 splits by using PSA1's ws, PSA1 merges, BMM1's BB=2.878, BMM2's BB=3.205
280	9	9	3	3		PSA2 arrives and splits to level 1
320	3	9	3	9	Action 1	PSA2 splits by using BMM1's ws
360	9	9	3	3	Action 2	BMM1 splits by using PSA2's ws, PSA2 merges, BMM1's FB=2.907, PSA1's FB=2.146, BMM2's BB=3.187, PSA2's BB=1.323

Table 5.3: Four applications in a cluster of twenty five workstations (Scenario Three)

Time of Auction (in sec.)	*No. of occ.ws*		*No. of LP*	*Action Taken*	*Comments*
	PSA1	*CSN1*			
80	1	1	10	-	PSA1 and CSN1 arrive and takes 1 ws each
140	3	1	6	Action 1	PSA1 splits to level 1 by using avail. ws
180	3	3	6	Action 1	CSN1 splits to level 1 by using avail. ws
340	3	9	0	Action 3	CSN1 splits to level 2, PSA1's PF=3.171, CSN1's PF=3.301

Table 5.4: Two applications in a cluster of twelve workstations with Loader Processes (Scenario Four)

BMM2) split to the highest partitioning level using available workstations in the cluster. After the arrival of the third application (PSA1), the application splits to a higher level and forces one of the other applications to merge because the first-time auction application has a higher priority than the other to split. The BMM1 is chosen to merge to a lower level because it has a lower backward bid than the BMM2. If the value of ψ is greater than or equal to 1.05, the allocation of workstations will become stable, otherwise the PSA1 will merge to level 0 and the BMM1 will be restored to level 1 in the next auction.

In the second scenario, all the applications arrive at the same time and each application takes three available workstations for execution. In their first auction, all applications request splitting, but only the BMM2 gets the available workstation to split. The PSA1 splits by forcing the BMM1 to merge. In the next round of auctioning, the BMM1 wins back the workstations from PSA1 because it has the highest forward bid and its forward bid is greater than the backward bid of the PSA1. The allocation of workstations becomes stable until the PSA1 terminates and the BMM1 can then split further to level 2.

In the third scenario, there are four applications in a cluster of twenty five workstations. Table 5.3 illustrates how a stable allocation of workstations is made among the applications. The applications arrive one by one so that the first and the second applications (BMM1 and BMM2) can split to the highest level by using available workstations in the cluster. However, the third and the fourth applications (PSA1 and PSA2) do not have enough available workstations to do so; they have to compete for workstations through auctions. The BMM1 wins the auctions because it has the highest forward bid and the PSA2 has to merge because it has the smallest backward bid.

Table 5.4 shows the fourth scenario in which the PSA1 and CSN1 are competing in a cluster of twelve workstations. Each application can initially gets one workstation for execution because external loader processes (LPs) are running on the rest of the workstations. When four workstations become available, the applications then split to level 1. When all the external loader processes are finished, the CSN1 wins the newly available workstation because it has a higher potential forward bid than the PSA1. However, as illustrated in the experiments in Sections 4.9.1 and 4.9.2.3, the speedup performance of the PSA1 is better than that of the CSN1, so the PSA1 will win back the workstations if the measured speedup of the CSN1 later drops. To obtain a more reliable bid value that can reflect the actual speedup of a nondeterministic application, the time between measuring unit task execution time should be sufficiently long.

5.9 Summary

Since resource contention between parallel applications can lead to performance degradation, a new scheduling system called *Comedians* is proposed to maximize the speedup of individual applications and to allocate workstations efficiently and fairly to the applications based on dynamic space-sharing. The design, implementation, and testing of the Comedians system are presented in this chapter. The system facilitates adaptive execution of applications by providing load monitoring and forecasting, Global Workload Distribution, and mechanisms for auction and bidding.

The effectiveness of the system is tested and the results of the experiments exhibit the dynamics of competition between parallel applications in different scenarios. The results of the experiments show that stable or Pareto efficient allocations can be obtained through the three simple conditional actions of the share auction if the runtime behaviour of applications does not vary frequently. Moreover, the system provides a useful testbed for studying the dynamics of competition between parallel applications.

Chapter 6

The Comedians on Heterogeneous Clusters

6.1 Introduction

Workstations in a distributed network are seldom homogeneous; they are often different in processing speed, memory capacities, and communication bandwidth. Nevertheless, workstations can be grouped into clusters according to their different characteristics. Management of these clusters can only be scalable if a distributed control scheme is applied. To fully deploy workstation cluster computing, the aggregate computing power of heterogeneous workstations must be used by parallel applications as a single virtual machine. This chapter describes an extension of the Comedians system [SM96] to facilitate a competitive environment for parallel applications on multiple and heterogeneous workstation clusters. The extension enhances the system to tackle the job scheduling problem with respect to heterogeneity between the clusters.

6.2 Execution States of Adaptive Parallel Applications

Akin to the problem of process mapping in multiprocessor systems, the number of possible task-to-grain and grain-to-workstation mappings in heterogeneous clusters is large and the optimal mapping problem is NP-complete [Cof74, LK78,

GJ79]. The optimal solution can only be worked out in some special cases [Hu61, CG72, Sto77, Bok81]. In a shared network environment, dynamic workload and workstation heterogeneity should also be taken into consideration when a mapping is selected. It is therefore more crucial to find a quick and efficient way of searching for suitable mappings at runtime.

To enable fast workstation allocation, the number of possible task-to-grain and grain-to-workstation mappings is restricted. The mapping problem is simplified by representing the sizes of the grains as partitioning levels, and by grouping workstations into clusters according to their processing speeds. As defined in Section 3.4.2, each partitioning level of an application is a predefined grouping of tasks. A parallel application will only switch from one partitioning level to another partitioning level[1]. The grains of an application can also be executed in workstations across heterogeneous clusters. As a result, the possible mappings of a parallel application are limited by the dimensions of partitioning level and cluster type. The representations that are used to identify the runtime configurations of applications are referred to as *execution states* (ES). An example of this two-dimensional state space of execution states of an application is shown in Figure 6.1.

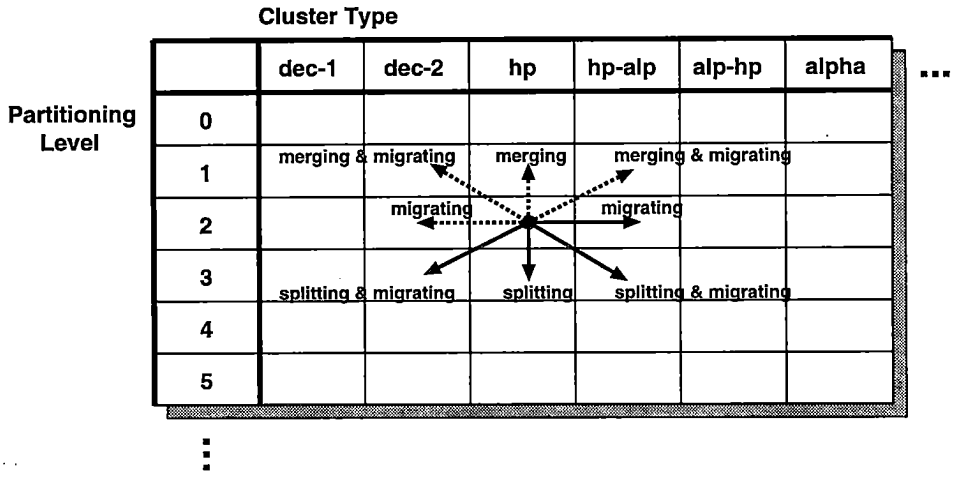


Figure 6.1: The State Space of Execution States of an Application

If an application is running on two or more different clusters, the application is said to be running on a *pseudo-cluster*. For example, if an application is running on HP and DEC Alpha workstations, we may call its cluster type *hp-alpha* if

[1]In addition, the size of the grains between two partitioning levels can be refined by changing task-to-grain mappings invoked by the Local Workload Distribution.

the number of occupied HP workstations is greater than that of DEC Alpha workstations, otherwise *alpha-hp*.

Parallel applications can only change their execution states by coordinating with the Comedians system. Figure 6.1 also shows all the possible movements of an execution state. These movements are the outcomes of the Global Workload Distribution (GWD) which are the results of auctions or responses to workload interference through the Comedians system. To change the execution state of an application, an application can either migrate its grains to workstations in other clusters without changing its partitioning level (the horizontal movements of ES), or split/merge its grains to different partitioning levels (the vertical movements of ES). Grain migration can also be carried out at the same time as grain splitting and merging (the diagonal movements of ES). The solid arrows shown in Figure 6.1 represent the preferable movements of ES because splitting grains to a higher partitioning level and migrating grains to a faster workstation cluster can normally enhance performance until a saturation point is reached.

6.3 System Architecture

Figure 6.2 depicts the system architecture that supports the execution of adaptive parallel applications on heterogeneous workstation clusters. The Comedians system serves as a resource management medium between the clusters and the applications. An application can run on more than one workstation cluster. One of the clusters is specified as the local cluster and the others as the remote clusters. All the up-to-date information of the application is recorded in the CS of the local cluster (local CS); this information is updated whenever the execution state of the application changes. Consistency and efficiency in updating the status of applications are the two main design goals of the Comedians system operating on multiple clusters.

An application has to register with the workstation clusters in which it will execute. However, the CS can refuse the registration of the application if its cluster is overloaded. As shown in Figure 6.3a, an application will first register at its local CS and the local CS will then send the registration message to its remote CS. Similarly, when an application terminates, the termination signal will be sent to the local CS and the local CS will then send it to the remote CS.

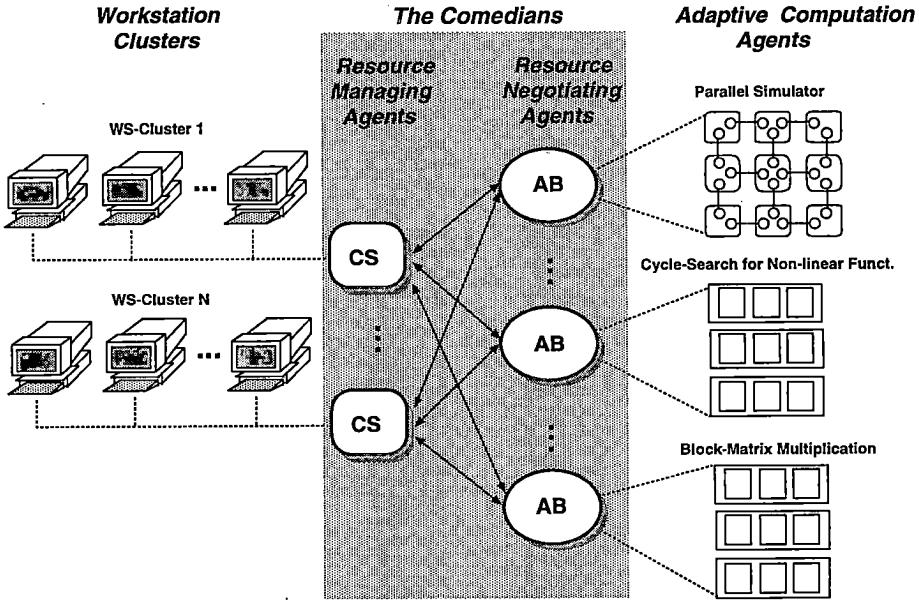


Figure 6.2: System Architecture that supports Adaptive Parallel Applications

6.3.1 Responses to External Workloads

Distributed Cluster Schedulers (CSs) invoke the GWD responding to dynamic workload in individual workstations. Since the grains of a registered application will only run on available workstations, two parallel applications will not share the same workstations for execution. If external processes from the workstations' owners or other users arrive at the workstations in which the grains of a parallel application are currently running, the performance of both the external processes and the parallel application may suffer greatly. Therefore, the CS will inform the AB of the application to migrate its grains from the heavily loaded workstations to other available workstations. If there are not enough workstations available for migration, the grains of that application will be merged to a lower partitioning level releasing the heavily loaded workstations. In either of the above cases, the application will acknowledge the CS that initiates the migration or merging; the CS will then inform the local CS and other CSs that are involved in this operation (Figures 6.3b and 6.3c).

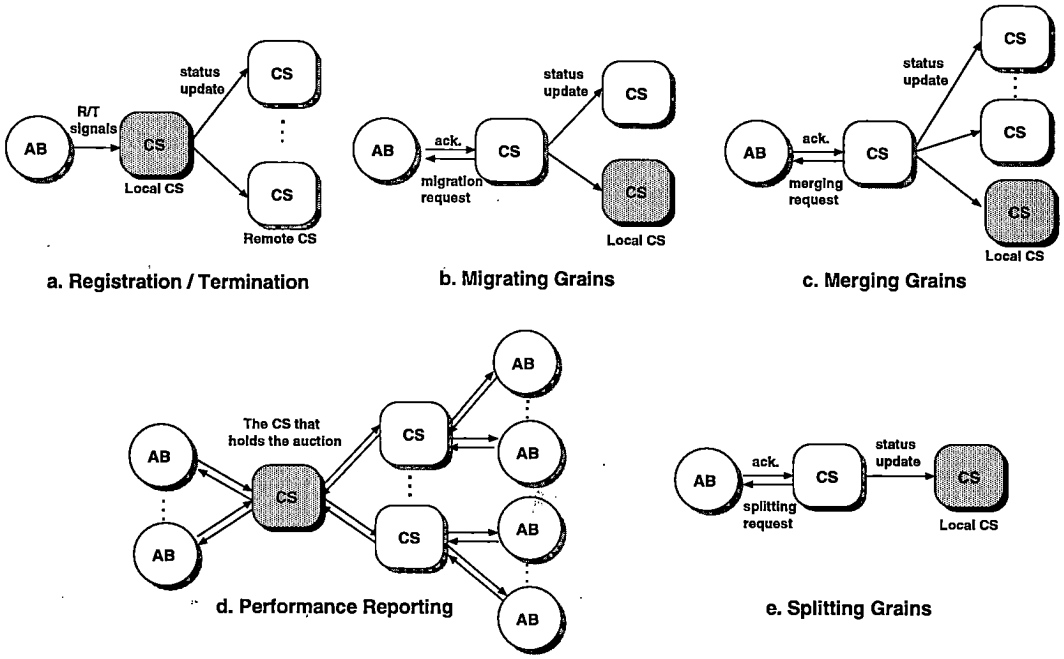


Figure 6.3: Interaction between ABs and local and remote CSs

6.3.2 Holding a Share Auction on Heterogeneous Clusters

Figure 6.4 illustrates the actions involved in a share auction across multiple clusters. To hold an auction, the CS first broadcasts the news of the auction to all the ABs that have been registered in the cluster. It also sends requests to related CSs for the number of available workstations in the remote clusters. If the CS that holds the auction is not the local CS of the application, the auction notification will be sent to the local CS before sending to the AB. Similarly, the current unit task execution time will be collected by the local CS first and then it is forwarded to the CS that holds the auction (Figure 6.3d). The local CS should always keep the most up-to-date status of the application.

Similar to the measurement defined in Section 5.7.1, the value of the unit task execution time $U_i(x, y)$ of application AP_i at cluster type x and partitioning level y is the mean completion time of a predefined fraction of a task $u_g(x, y)$ collected from the grains of the application. The value of $u_g(x, y)$ includes the average computing and communication time required to finish the fraction of a task.

Upon receipt of all the unit task execution times from registered applications, the CS will record and store the information. An auction will then be held and the result will be sent to the involved ABs. After confirmation from the ABs has

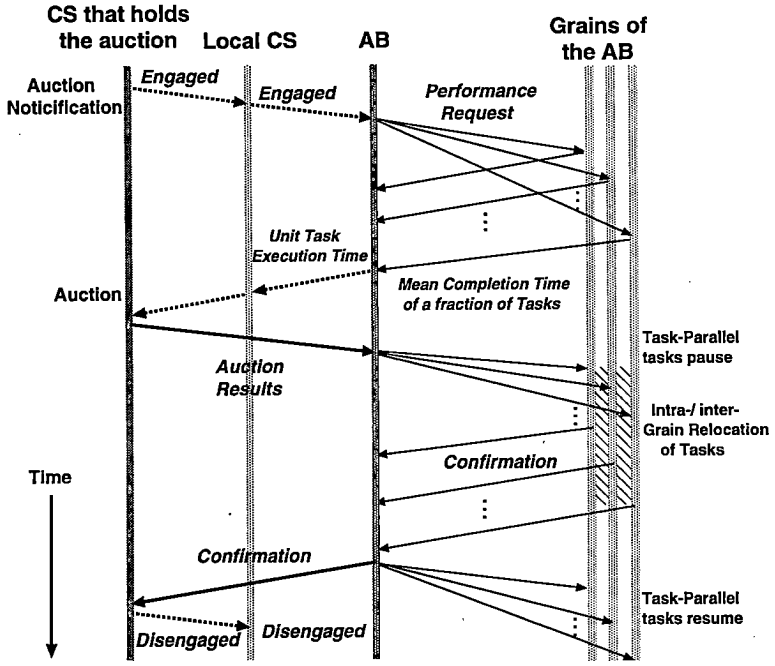


Figure 6.4: Share Auction across Multiple Clusters

been received, all the CSs are disengaged and the status of the applications in the local CSs are updated. For sending and receiving of the auction results and confirmation from the ABs, the local CS will be bypassed.

Once a CS holds an auction or a CS/AB agrees to participate in an auction, it then switches from a *disengaged* status to an *engaged* status. If a CS/AB has been engaged, it will not take part in another auction or other grain movement. After the result of the auction is received by all the involved ABs, the CSs/ABs will then be disengaged. This ensures the status information of applications is updated in an orderly and consistent way. If an auction is turned down by an engaged local CS, or the CS that holds the auction has been engaged by another CS, the time of the next auction will be rescheduled randomly within a predefined reauction period.

6.3.3 Bidding on Heterogeneous Clusters

From the past and current records of performance information, the CS may formulate three types of bids, namely forward bid, backward bid, and potential forward bid. These bids enable applications with different performance records

to compete for workstations. Formulation of bids and auction results are modified based on the details described in Chapter 5.

The *forward bid* (FB_i) of an application AP_i is the ratio between the speedup of an execution state (x_f, y_f) and the speedup of the current execution state (x_c, y_c). The bid value exhibits the ability to improve speedup by moving forward to another better execution state which has been visited previously. This value can be formulated by the ratio of unit task execution times reported by the ABs as shown in equation 6.1, where $U_i(x_l, 0)$ is the unit task execution time of AP_i at its local cluster and its lowest partitioning level, $S_i(x_c, y_c)$ and $U_i(x_c, y_c)$ are the respective speedup and unit task execution time at the current execution state.

$$\begin{aligned} FB_i &= \frac{S_i(x_f, y_f)}{S_i(x_c, y_c)} = \frac{U_i(x_l, 0)/U_i(x_f, y_f)}{U_i(x_l, 0)/U_i(x_c, y_c)} \\ &= \frac{U_i(x_c, y_c)}{U_i(x_f, y_f)} \end{aligned} \tag{6.1}$$

The *backward bid* (BB_i), on the contrary, shows the loss in speedup of an application AP_i by releasing workstations to application AP_f with the maximum forward bid. The bid value of AP_i (equation 6.2) can be formulated by the ratio of the unit task execution time at a lower previous execution state (x_b, y_b) to the unit task execution time at the current execution state (x_c, y_c). AP_i moves to the execution state (x_b, y_b) so that AP_f can move forward to the execution state (x_f, y_f) by using the workstations freed by it.

$$BB_i = \frac{S_i(x_c, y_c)}{S_i(x_b, y_b)} = \frac{U_i(x_b, y_b)}{U_i(x_c, y_c)} \tag{6.2}$$

For the applications that do not have known performance information about their higher execution states, their *potential forward bids* will be used for bidding. A potential forward bid of an application AP_i is the ratio of efficiencies between $E_i(x_l, y_c)$ and $E_i(x_l, y_{c-p})$, where x_l is the local cluster type, y_c is the current partitioning level, and y_{c-p} is the lowest partitioning level of which the unit task execution time is known or can be estimated. $N_i(y_c)$ is the number of occupied workstations at the current partitioning level. The bid value (equation 6.3) exhibits the potential of AP_i to split to a higher execution state.

$$
\begin{aligned}
PB_i &= \frac{E_i(x_l, y_c)}{E_i(x_l, y_{c-p})} \\
&= \frac{S_i(x_l, y_c)/N_i(y_c)}{S_i(x_l, y_{c-p})/N_i(y_{c-p})} \\
&= \frac{U_i(x_l, y_{c-p})/U_i(x_l, y_c)}{N_i(y_c)/N_i(y_{c-p})}
\end{aligned} \tag{6.3}
$$

In formulating the potential forward bid, if the actual value of the unit execution time is not known, its estimated value can be calculated by equation 6.4, where P_{x_c}/P_x is the estimated ratio of CPU processing speeds of the workstations between the current cluster x_c and the cluster type x. The ratio[2] of P_{x_c}/P_x is initially estimated by the CPU processing speeds of the workstations in a cluster. The ratio will later be updated by the actual performance of the application, that is, the unit task execution times collected at the next round of the auction.

$$
U_i^{est}(x, y) = \frac{P_{x_c}}{P_x} \cdot U_i(x_c, y) \tag{6.4}
$$

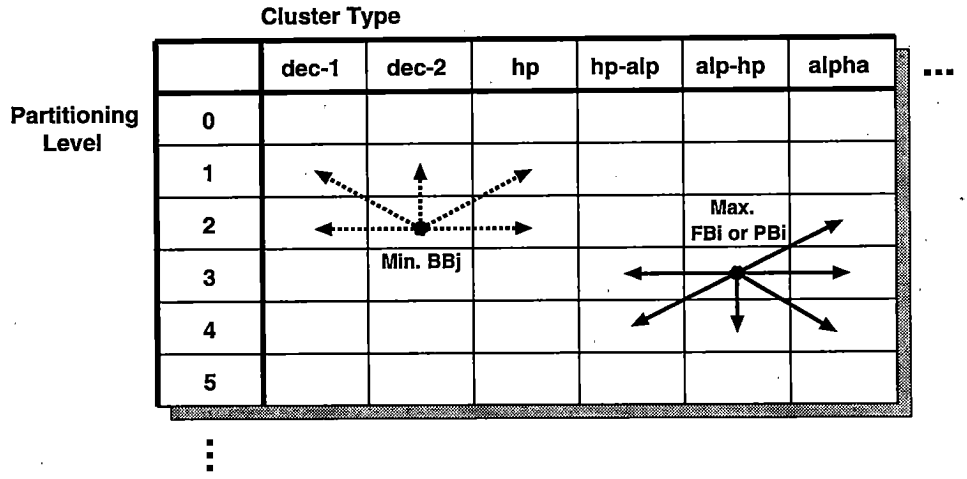


Figure 6.5: An Example of Bidding

After an application has registered in its local and remote CSs, it will first run on the available workstations of its local cluster at a partitioning level that is nearest but not equal to the highest level it can reach. Its local CS will then schedule

[2]This ratio can have different values for integer operations and floating point operations.

the time for the first auction. The results of each auction are the outcomes of the following three conditional actions. All the applications participating in an auction will go through these actions sequentially, but each application can only take part in at most one of these actions. Figure 6.6 describes all the conditional actions taken by applications involved in an auction, where $ES_i(x, y)$ represents the execution state of an application AP_i at cluster x and partitioning level y.

- **Action 1**
  First-time auction applications will try to split to a higher partitioning level in the local cluster using available workstations. If the number of the available workstations (N_{avail}) is not enough for splitting, the application with the minimum backward bid will release its workstations through grain migration (Figure 6.3b) or grain merging (Figure 6.3c). For the first-time auction applications, the only information known to the CS that holds the auction is the unit task execution time at the current execution state $U_i(x_c, y_c)$. Therefore they take priority over other applications when splitting in the local cluster because the reported unit task execution times at different execution states help the CS to formulate forward and backward bids for future bidding.

- **Action 2**
  For applications with known forward bids, the application with the maximum forward bid will move forward to a higher execution state either by grain splitting or grain migration. For grain splitting, the CS that holds the auction sends a splitting request to the AB of the application. If this CS is not the local CS of the application, updated status information for the application will also be sent to the local CS (Figure 6.3e). The application will take the available workstations and/or the workstations released by the application with the minimum backward bid through grain migration or grain merging. Figure 6.5 shows an example where an application with the maximum forward bid moves to a higher execution state by forcing another application with the minimum backward bid to a lower execution state. To reduce sensitivity, no action will be taken if the maximum forward bid is less than the minimum backward bid by a factor of ψ, and the value of the maximum forward bid must be greater than one.

- **Action 3**
  For applications with unknown forward bids, the application with the maximum potential forward bid will move to a higher execution state. If a

Action 1: For the first-time auction AP_n, *where* $n \in P$

If $N_{avail} \geq N_c$, *where* N_c is the nearest number to N_{max}

$AP_n : ES_n(x_0, y_0) \xrightarrow{s} ES_n(x_0, y_f)$,

Else

If $\exists\ AP_b$, *s.t.* AP_b has $Min(BB_j)_{\triangle N}$ or $Min(PB_j)$, $j \in P$

$AP_n : ES_n(x_0, y_0) \xrightarrow{s} ES_n(x_1, y_f)$,

$AP_b : ES_b(x_1, y_1) \xrightarrow{mm} ES_b(x_b, y_b)$,

Action 2: For the AP_i with known FB_i, *where* $i \in P$

Choose the AP_f with $Max(FB_i)$

If N_{avail} in local and/or remote clusters is enough for AP_f to move to a better execution state

$AP_f : ES_f(x_0, y_0) \xrightarrow{sm} ES_f(x_f, y_f)$,

Else

If $\exists\ AP_b$, *s.t.* $Max(FB_i) > (\psi \cdot Min(BB_j)_{\triangle N})$, $\psi > 1$, $i \neq j$

$AP_f : ES_f(x_0, y_0) \xrightarrow{sm} ES_f(x_f, y_f)$,

$AP_b : ES_b(x_1, y_1) \xrightarrow{mm} ES_b(x_b, y_b)$,

Action 3: For the AP_i with unknown FB_i, *where* $i \in P$

If $\exists\ AP_f$, *s.t.* AP_f has $Max(PB_i)$ and N_{avail} in local and/or remote clusters is enough for AP_f to move to a better execution state

$AP_f : ES_f(x_0, y_0) \xrightarrow{smm} ES_f(x_f, y_f)$.

Changing Execution States by GWD Operations :

$\xrightarrow{s}$: Grain Splitting

$\xrightarrow{mm}$: Grain Merging and/or Grain Migrating

$\xrightarrow{sm}$: Grain Splitting and/or Grain Migrating

$\xrightarrow{smm}$: Grain Splitting or Grain Migrating or Grain Merging if the grains is merging from a cluster of slow machines to a cluster of fast machines.

Figure 6.6: The Conditional Actions taken by Forward APs and Backward APs

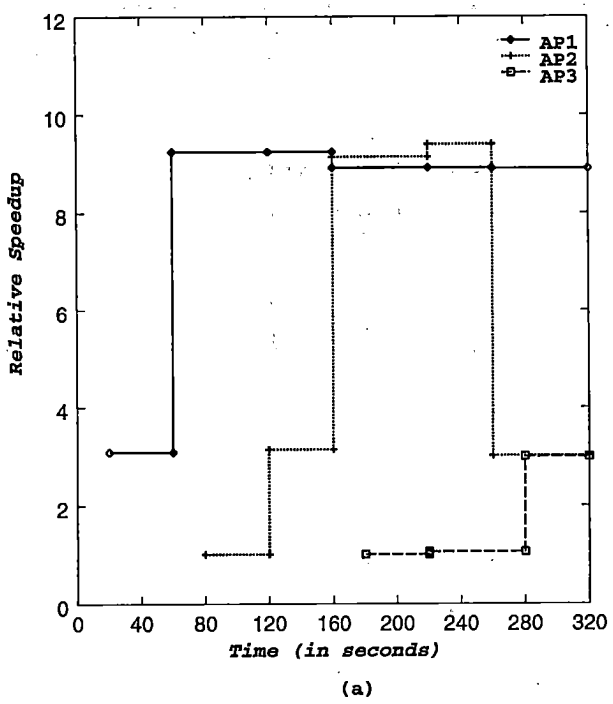
potential forward bid is selected, the possible results include grain merging, grain migration and grain splitting (Figure 6.5). Grain splitting and grain migration for this action will only take place on the available workstations in the cluster that carries out the auction. No application will merge its grains except when the application with the maximum potential forward bid merges its grains from slow workstations in a cluster to faster workstations in another cluster.

6.4 Performance Profiles

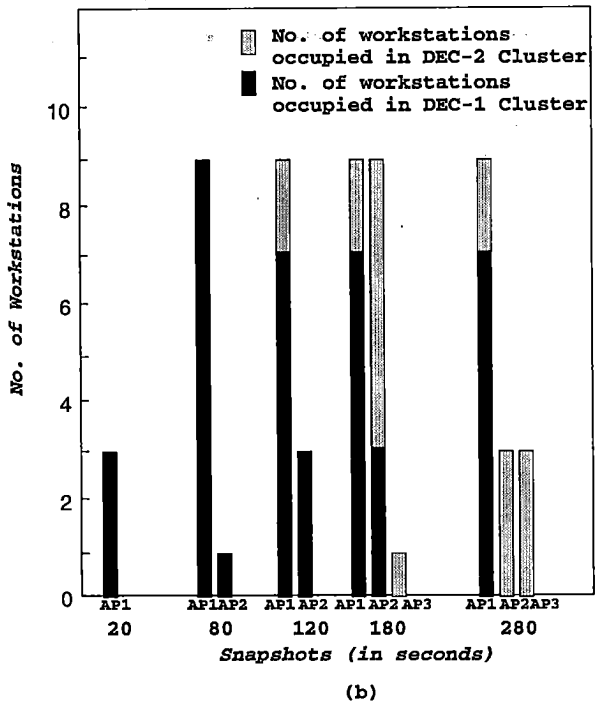
In this section, the experiments and results for examining the performance profiles of competing applications on multiple and heterogeneous workstation clusters are presented. The experiments are carried out on three workstation clusters which consist of DEC Alpha, HP9000/700, and DEC3100 workstations. The configurations of hardware used in the experiments here and the experiments described in Section 4.9 are similar. The parallel applications used in the experiments consist of the block-matrix multiplication (BMM) and the parallel simulator of an ATM Network (PSA).

The interaction of parallel applications in two scenarios are investigated by experiments. In the first scenario, three BMM programs (AP1, AP2 and AP3) are competing in two workstation clusters (DEC-1 and DEC-2 cluster) and each of the clusters consists of ten DEC3100 workstations. The DEC-1 cluster is the local cluster of AP1 and AP2 and the DEC-2 cluster is the local cluster of AP3. AP1 and AP2 arrive one by one at the DEC-1 cluster and they occupy workstations in the DEC-2 cluster when there are not enough available workstations for splitting in the DEC-1 cluster. A stable allocation for the applications is obtained after AP3 splits to level one in the DEC-2 cluster. Figures 6.7a and 6.7b show the relative speedup and the snapshots of the applications. The speedup values are relative to the execution time of the applications in a DEC3100 workstation at partitioning level 0.

In the second scenario, two BMM programs (AP4 and AP6) and one PSA program (AP5) are invoked in three clusters. These clusters include six DEC Alpha workstations (Alpha cluster), five HP workstations (HP cluster) and thirty DEC3100 workstations (DEC-3 cluster). This scenario is common in many academic and corporate environments where there is a large pool of slow machines and a relatively smaller number of fast machines. It is quite obvious that all



(a)



(b)

Figure 6.7: Relative speedup and snapshots of the experiments in the first scenario: there are two clusters, each has 10 DEC3100.

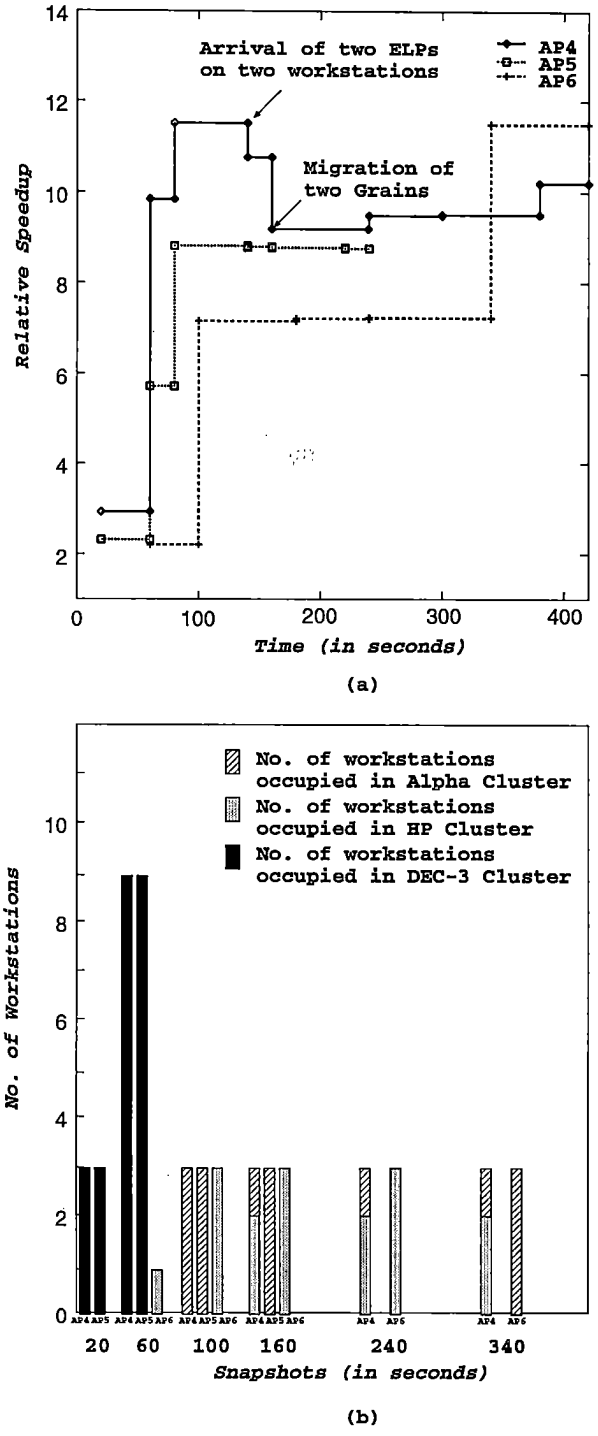


Figure 6.8: Relative speedup and snapshots of the experiments in the second scenario: there are three clusters, one has 6 DEC Alpha, the second one has 5 HP, and the third one has 30 DEC3100.

applications will compete for the fastest workstations in the Alpha cluster.

Figures 6.8a and 6.8b present the relative speedup and the snapshots of the applications in this scenario. The DEC-3 cluster is the local cluster of AP4 and AP5 and the HP cluster is the local cluster of AP6. The applications arrive nearly at the same time on their local cluster and compete for more and faster workstations. After AP4 and AP5 have split to level two on the DEC-3 cluster by *Action 2* of the share auction, they merge to level one on the Alpha cluster in order to gain better values of speedup. When external loader processes are invoked in two workstations of the Alpha cluster on which the grains of AP4 are running, the grains are migrated to two available workstations on the HP cluster. After AP5 has terminated, AP6 migrates from the HP cluster to the Alpha cluster because it has a higher forward bid value than AP4.

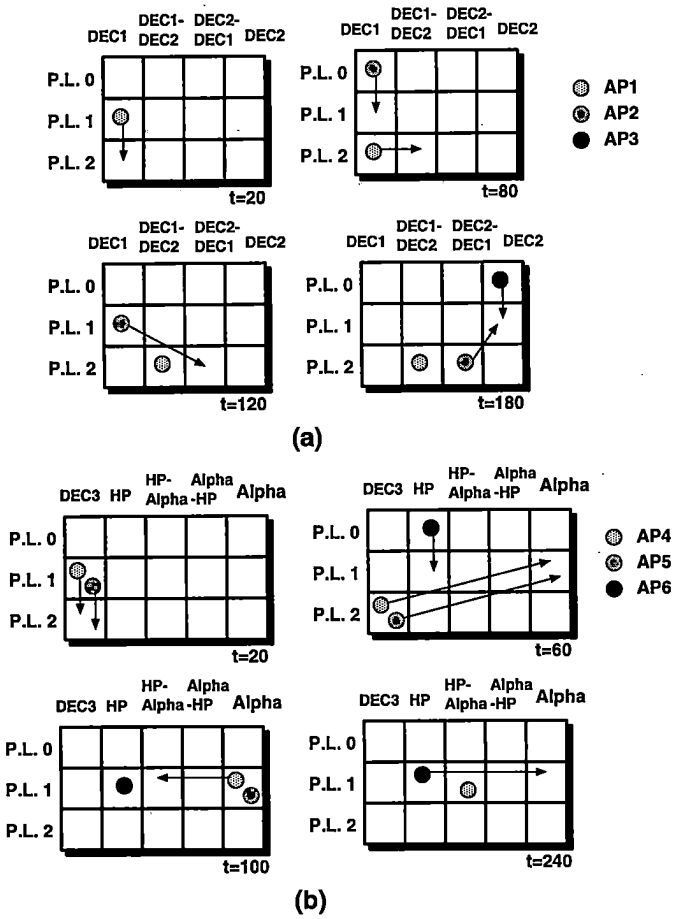


Figure 6.9: (a) Profiling Execution States in the First Scenario; (b) Profiling Execution States in the Second Scenario

To monitor the status of all applications in the Comedians system, the execution states of applications can be collected from all the Cluster Schedulers. Figures 6.9a and 6.9b show the execution profiles of applications involved in scenario one and two after each auction has taken place. The dots represent the execution states of the applications and the arrows show the transitions from the current execution state to the next execution state in the state space of cluster types and partitioning levels. For example in Figure 6.9a, the AP2 in the first scenario has visited three different execution states before settling down at the stable allocation.

Since the execution states of each application are recorded by its local Cluster Schedulers, the ratio of P_{x_c}/P_x and the unit task execution time at the execution states that have been visited will become the performance history of the application. The information can be reused for the next execution of the same or similar applications on the same clusters. This information is especially useful for formulating bids when the applications start running on the clusters. By using the performance history, the time for reaching a stable allocation of workstations can be shortened.

6.5 Summary

To facilitate an efficient mechanism for finding suitable application configurations in a multi-user, multi-parallel program environment, the concept of execution state is introduced in this chapter. The reason for using execution states to represent application status is to reduce substantially the number of possible configurations of an application because partitioning level and cluster type are the only parameters to represent an execution state and the values of these parameters are reasonably small. Based on this concept, the Comedians system is extended to utilize the aggregate computing power of heterogeneous workstation clusters efficiently.

This chapter also presents the design and implementation of the share auction that is used to allocate workstations in heterogeneous clusters to applications. Experimental results show that the share auctions held by distributed Cluster Schedulers can achieve stable allocations among competing applications.

Chapter 7

Combining Parallelism with Replication

7.1 Introduction

The concept of *replication*, which is used to manage applications in the Comedians system, is presented in this chapter. Replicated applications are defined as an organized group of related applications that coexist in workstation clusters. The benefits of applying the concept to parallel applications are also discussed. The objective of this work is to provide runtime support for a multi-user, multi-parallel program environment. This chapter is organized as follows. First, a new structure that manages replicated applications as coalitions is described. Secondly, an efficient fault-tolerant model using replicated applications is suggested. Finally, a case study of running replicated parallel simulators in the Comedians system is used to demonstrate the importance of replication.

7.2 From Replication to Coalition

7.2.1 Management of Replication

Workstation clusters will become an indispensable computing resource only if they can satisfy the diverse computing needs of different users. The need to manage different applications is especially great when the number of applications

running on workstation clusters is large. In a competitive environment envisaged by the Comedians system, replicated applications owned by the same user can form a *coalition*. The coalition partners share a common mission – that is to serve the computing needs of their owners. The purposes of forming a coalition of applications are as follows:

1. **Priority Setting:** Among the members of a coalition, the urgency of completion can be defined by setting the priority of running on newly available workstations. A better use of priority can improve resource utilization.

2. **Competition:** An application will not compete with other coalition partners for resources. In other words, an application will not split to a higher partitioning level and/or migrate to faster machines (a higher execution state) at the expense of forcing other coalition partners to merge to a lower partitioning level and/or migrate to slower machines (a lower execution state).

3. **Shared Knowledge:** Knowledge about the current and past execution performance of coalition partners can be shared to facilitate better initial workstation assignments and resource allocations. For example, the shared knowledge can be used to formulate improved bids.

4. **Monitoring:** The amount of resources occupied by a coalition or by a user can be identified and monitored. Resources should be allocated to each user fairly and no user should consume an unlimited amount of resources. A budgetary control system can be applied for this purpose.

7.2.2 Weak and Strong Coalitions

To identify applications owned by individual users, the applications are labelled by replication numbers. The replication numbers add a new dimension to the execution state of an application which can be represented by a notation $ES_i(p, c, r)$, where p is the partitioning level, c is the cluster type, and r is the replication number. The replication number is composed of a user identification and a replication identification. Besides the current status, the execution states also indicate the ownership of individual applications.

Coalitions can be further classified as different types to exercise different management policies. Among the applications owned by a user, the applications

Applications owned by User A:

Weak Coalition

Strong Coalition

$\{ES_i(p_1, c_1, r_1), \ldots, ES_j(p_1, c_2, r_2), \ldots, \overbrace{[ES_k(p_n, c_1, r_n), \ldots, ES_{k+m}(p_n, c_1, r_{n+m})]}, \ldots\}$

The Status Information of Applications recorded by Cluster Schedulers:

CS1 - User A: $\{ES_i(p_1, c_1, r_1), ES_{i+1}(p_2, c_1, r_1), \ldots,$

$[ES_k(p_n, c_1, r_n), \ldots, ES_{k+m}(p_n, c_1, r_{n+m})]\}$

User B: $\{ES_u(p_1, c_1, r_1), ES_{u+1}(p_2, c_1, r_1), \ldots,\}$

$\vdots$

CS2 - User A: $\{ES_j(p_1, c_1, r_1), ES_{j+1}(p_2, c_1, r_2), \ldots,\}$

User B: $\{ES_v(p_1, c_1, r_1), ES_{v+1}(p_2, c_1, r_1), \ldots,$

$[ES_s(p_n, c_1, r_n), \ldots, ES_{s+m}(p_n, c_1, r_{n+m})]\}$

$\vdots$

$\vdots$

Figure 7.1: The applications owned by a user and the execution states recorded by Cluster Schedulers

with similarities can be classified as special coalitions. The competition for resources between different coalitions can be carried out according to the types of the coalition. Two types of coalition are defined namely *weak coalition* and *strong coalition*. Weak coalition represents individual groups of programs that are owned by a common owner. Strong coalition consists of related applications that are working co-operatively for a common objective.

For example, the identical and independent runs of a simulation with different input parameters or random number seeds can be executed concurrently to form a strong coalition. Applications in a strong coalition are also members of a weak coalition of the same user.

Figure 7.1 shows the execution states of the applications owned by a user and

the execution states recorded by Cluster Schedulers (CSs). In this example, *user A* owns a number of applications that are grouped into coalitions and are running on two clusters. The execution states inside curly brackets represent the applications of a weak coalition and the execution states inside square brackets comprise the applications of a strong coalition. Although each user owns a number of applications, the execution states of applications are hidden from the user. In other words, users will not know how and where their applications are running because the applications adapt to the environment automatically. All the execution states are maintained by the local CSs of the Comedians system that update the execution states whenever the applications reconfigure.

7.3 Fault Tolerance through Replication

Fault tolerance is one of the most important issues for distributed computing because distributed nodes are subject to system or hardware failure. Parallel computing in workstation clusters is also vulnerable to failure especially for long running applications. Recent solutions [LFS93, PL96, KRS96] to the fault tolerance problem on workstation clusters are based on total or partial recovery from the previous checkpointed states of applications.

A simple yet efficient fault-tolerant model that applies the idea of replicated execution is presented. Replicated execution [SS94b] provides fault tolerance by having a number of processes running the same program concurrently. The concept of using replicated execution for fault tolerance have been applied to distributed operating systems [Ng90] and distributed systems [OL88, MPS89]. In this section, replicated execution is used to provide fault tolerance for adaptive parallel applications in the Comedians system.

7.3.1 The Fault-Tolerant Model

In the simplest case, replicated copies of an application run on different workstations concurrently until one of them finishes; the fastest application will then inform the other replicated applications to terminate (Figure 7.2a). The final result of the application is fault-proof as long as at least one of the applications can complete its computation. However, the obvious disadvantage of this method is the requirement of exceedingly large amounts of resources to support the redundancy.

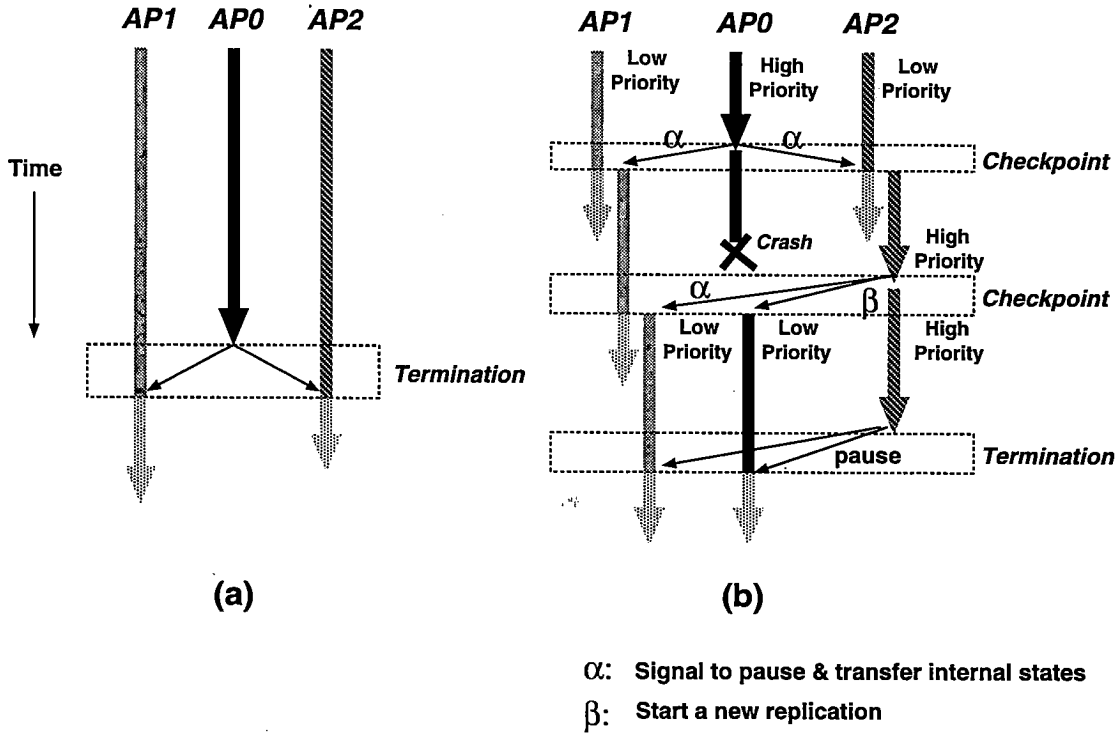


Figure 7.2: (a) Replicated Parallel Applications without Checkpoints; (b) Replicated Parallel Applications with Checkpoints

To minimize the demand of workstation resources for fault tolerance, the replicated applications form a strong coalition in the Comedians system. Only one of them in the coalition has a high priority to obtain resources. When the competition for workstations is keen, only the high priority application can compete with other applications for splitting to higher partitioning levels or running on fast workstations. The other low priority coalition partners will not consume excessive resources.

Figure 7.2b describes an example of running three replicated applications concurrently. Only one of the applications is initially set as the high priority application. Checkpoints are introduced to synchronize the progress of the replicated applications. The coalition partners race with one another until one of them reaches the checkpoint; the fastest application will stop other applications and transfer its internal states to them. As a result, the best progress of the applications can be maintained at every checkpoint by reloading checkpointed states from the fastest application to the other applications.

At each checkpoint, the grains of the fastest application will be synchronized and a ready signal for a checkpointed state transmission will be sent to the local CS via the Application Bidder. The local CS will then request the grains of the other applications to pause and get ready for the receipt of the checkpointed states.

If a failure is detected at one of the grains of the high priority application, one of the low priority applications will switch to high priority. If a low priority application fails, no action needs to be taken until the next checkpoint. At the next checkpoint, the above synchronization procedures will be instigated and a new low priority replicated application will be restarted by the latest checkpointed states. The idea of racing between coalition partners is similar to the racing between grains in the scheme of the Local Workload Distribution for task-parallel applications described in Section 4.5.3.

7.3.2 Costs and Benefits

This fault-tolerant model for cluster computing is quite different from the conventional approaches to support fault tolerance. The benefits of this model are listed as:

1. Checkpointed states do not require expensive disk reads and writes because the states are transferred between applications.

2. No recovery or rollback from checkpoints or core dump files [LS92] is required. The only recovery procedure that needs to be performed when a high priority application fails is to invert the priority of one of the remaining applications. If the number of remaining applications is more than one, then it is optional whether to restart a new replicated application at the next checkpoint.

3. The consumption of workstation resources is dependent upon resource availability. When availability is low, the low priority applications can reconfigure to lower their execution states. Otherwise, the replicated applications will race with one another and maintain the best progress by transferring checkpointed states.

4. The fault-tolerant model can easily be implemented on top of the infrastructure of the Comedians system. The Cluster Schedulers help to detect process failures and to synchronize grains at each checkpoint.

The costs associated with this model are as the following:

1. Extra workstations are still needed to run low priority replicated applications. Despite this requirement, it is common in many situations that slow machines greatly outnumber the fast machines. Therefore, the low priority applications can always run on the slow but available workstations. In an extreme case, each low priority application can run on the minimum number of workstations that can satisfy the memory requirement of the application.

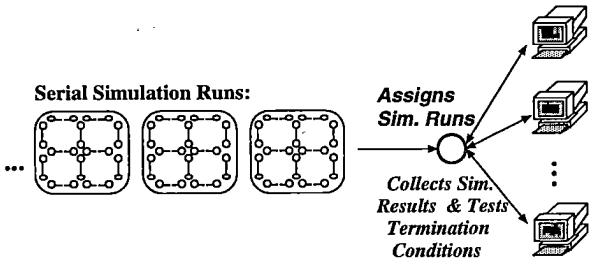
2. There are overheads in global synchronization and communication of all the replicated applications. However, the cost of the global synchronization and communication is still significantly lower than the cost of reading and writing checkpointed states on disks. Moreover, the need for global synchronization is common for all the fault-tolerant methods using consistent checkpointing [CL85, LNP91].

7.4 A Case Study of a Strong Coalition

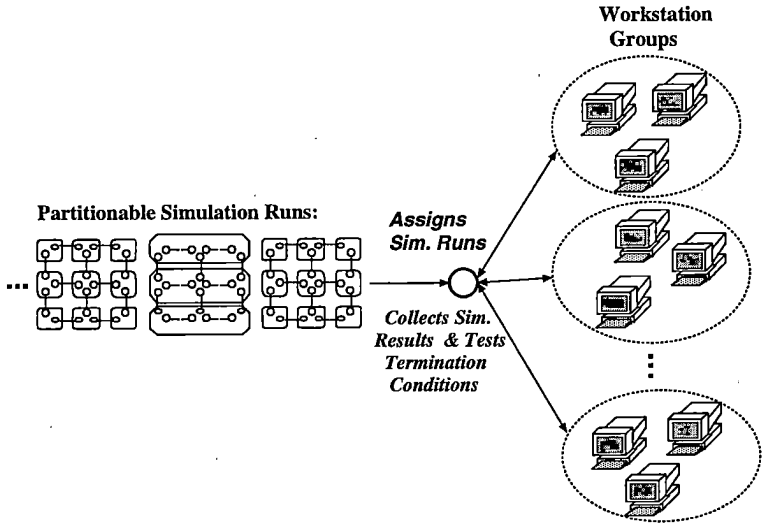
A case study is set up to test the usefulness of forming a strong coalition in the Comedians system. Although the example applications used in this case study are the replication of a parallel discrete-event simulation, most of the derived ideas could be extended to other kinds of applications. The analysis and experimental results of this case study show that the strong coalition of parallel simulations provides an effective way of speeding up simulation modelling.

7.4.1 The Replicated Parallel Simulator Model

Many discrete event simulations are regarded as computationally intensive. To reduce the turn-around time of simulation, parallelism is introduced to perform simulation operations in multi-processor or multi-computer machines. Parallel simulation is a promising approach to exploiting potential parallelism in many simulation models. In a parallel simulation, a model is decomposed into logical processes and then the logical processes execute in distributed processors. A number of parallel simulation techniques [Fuj90, NF94] have been devised to formulate statistical results efficiently and accurately.



(a) The Replicated Serial Simulator (RSS) Model



(b) The Replicated Parallel Simulator (RPS) Model

Figure 7.3: The RSS and RPS Models

Another approach to parallelism applied to simulation is to run multiple serial simulation programs on multiple processors in parallel and average the results at the end of the runs. This approach is referred to as replicated serial simulator (RSS) [BI87, Hei88, Lin94], which belongs to a wide class of parallelism known as serial program, parallel subsystem (SPPS) [Pfi95]. The major advantage of this approach is to provide a simple implementation to reduce the overall turnaround time of multiple simulation runs. Figure 7.3a shows the RSS model in which the replication of serial simulation runs are assigned to individual workstations.

However, the RSS model may not be adequate if the time and computational complexity of the simulation is too demanding to be executed serially. For example, the simulation of a broadband communication network, such as the simulation of an ATM (Asynchronous Transfer Mode) network as described in Section 4.5.2,

is extremely time-consuming because a great number of packet-level operations is involved in performance modelling. An effective approach [SL96] to speeding up the simulation of an ATM network on workstation clusters is proposed in this section. In this approach, multiple simulation runs are performed by replicated parallel simulators (RPSs) concurrently. This approach aims at combining the benefits of the parallel simulation approach and the RSS approach, which are reductions in the turnaround time of each simulation run and the overall turnaround time of all the runs. By analogy with the SPPS, this parallelism can be termed as parallel program, parallel subsystem (P^3S).

As shown in Figure 7.3b, partitionable simulation runs are assigned to different groups of workstations. Since the execution platform of the simulation is in a shared-network environment, the RPSs must compete with other applications for resources. The size of the workstation groups varies according to the availability of workstations. An RPS must be reconfigurable at runtime and adaptable to interference from dynamic workloads generated by other applications or other RPSs. Moreover, the approach of RPSs demands computational power from a large number of workstations. Thus availability is an important factor to promote the deployment of RPSs.

For both the RSS and RPS models, simulation stops when the reported simulation results of the runs satisfy a termination condition, for example when a desirable confidence interval of output results is obtained. If the termination condition is not satisfied, new simulation runs will continue to be assigned for execution.

7.4.2 Performance Evaluation

Figure 7.4 briefly depicts the relation between the overall turnaround time, parallelism, and replication of the RPS model. To make the evaluation tractable, the RPSs are assumed to be running on a dedicated homogeneous workstation cluster. The parallelism axis represents the number of workstations on which RPSs are running, and the replication axis denotes the number of running RPSs. For a given number of workstations (W), parallelism (P) and replication (R) can be applied concurrently to minimize the overall turnaround time.

From Figure 7.4, the parallel speedup will be saturated when there is no improvement of speedup by increasing the number of processors because of communication overheads and other bottlenecks. The replication speedup will also be saturated when the number of machines exceeds the number of simulation runs

(N). The working region (the shaded area) contains all the possible execution states of RPSs ($W \leq P \cdot R$) excluding all the areas where saturation can occur. The *working frontier* is the line ($W = P \cdot R$) on the edge of the working region where maximum utilization of workstations can be achieved.

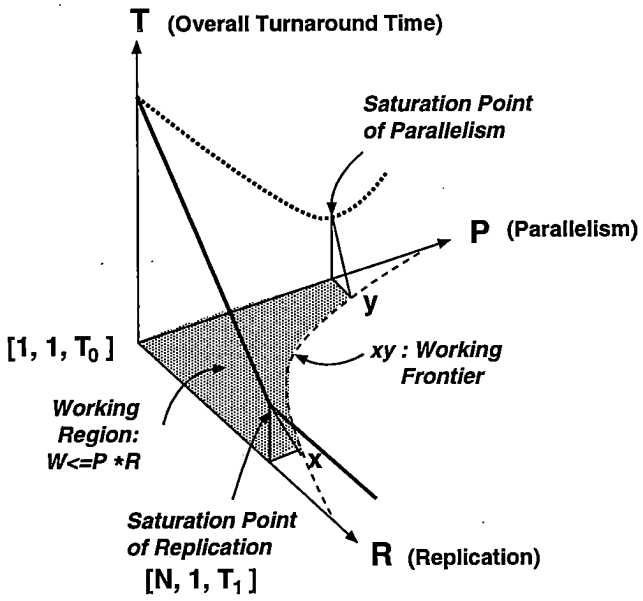


Figure 7.4: Overall Turnaround Time of RPSs

Like the RSS, the *first N replications initiated* (FNI) scheduling policy [BI87, Hei88, Lin94] must be applied to the RPSs to obtain statistically accurate simulation results. In this scheduling policy, N results of simulation runs are recorded from the first N replications initiated, where the value of N is determined by the termination condition of the simulation. The overall turnaround time for performing N replications by RPSs using the FNI scheduling method can be estimated by equation 7.1. The *End Effect*[1] (H_p) is the time the last R RPSs have to wait until the RPS with the longest execution time completes.

[1]Suppose the execution times of the RPSs are i.i.d. random variables T_i ($1 \leq i \leq R$) with an exponential distribution and means equal to 1. Then the expected value for the longest residual execution time is (cf., [BI87, Lin94]):

$$E\left[\max_{1\leq i\leq R} T_i\right] = R\int_{t=0}^{\infty} te^{-t}(1-e^{-t})^{R-1}dt = \sum_{1\leq i\leq R}\frac{1}{i} = H_p$$

The overall turnaround time shown in Figure 7.4 excludes the contribution of the *End Effect*. If the *End Effect* is excluded, the overall speedup owing to parallelism and replication is

$$
\begin{aligned}
T &= \frac{N}{S_p(P) \cdot S_r(R)} + \text{ End Effect} \qquad (7.1)\\
&\approx \frac{N}{S_p(P) \cdot R} + H_p \quad \text{as } S_r(R) = \frac{N}{\lceil \frac{N}{R} \rceil} \approx R,\\
&\geq \frac{N}{S_p(P) \cdot \frac{W}{P}} + \sum_{1 \leq i \leq R} \frac{1}{i}\\
&= \frac{N}{E_p(P) \cdot W} + \sum_{1 \leq i \leq R} \frac{1}{i}
\end{aligned}
$$

Remark No. 1: It can be deduced from the result of equation 7.1 that the overall turnaround time can be minimized if the number of available workstations (W) and the efficiency of parallelism ($E_p(P)$) are maximized. Therefore, the best execution state of the RPSs should lie on the working frontier with the maximum efficiency of parallelism.

If $N < W$, the *saturation point of replication* (Figure 7.4) will be the limiting factor. There are always enough workstations to support both parallelism and replication.

If $N \geq W$, there is a choice between using the RSS and RPS model. If the time and computational complexity is too demanding for the simulation to run serially, owing to the memory limitation in a single workstation say, then parallelism can bring superlinear performance, that is, over 100% efficiency of parallelism. In these cases, the RPS model should be deployed. Otherwise, the RSS model is a better choice because it does not require inter-machine communication so it can always deliver 100% efficiency of parallelism.

For a competitive environment, the number of available workstations can vary during the lifetime of an application. As shown in Figure 7.5, the working frontier shifts according to the value of W. If the value of W changes, the RPSs will have to adjust their execution states on the working frontier dynamically at runtime.

formulated as the product of parallel speedup and replication speedup:

$$S_{overall} = S_p(P) \cdot S_r(R)$$

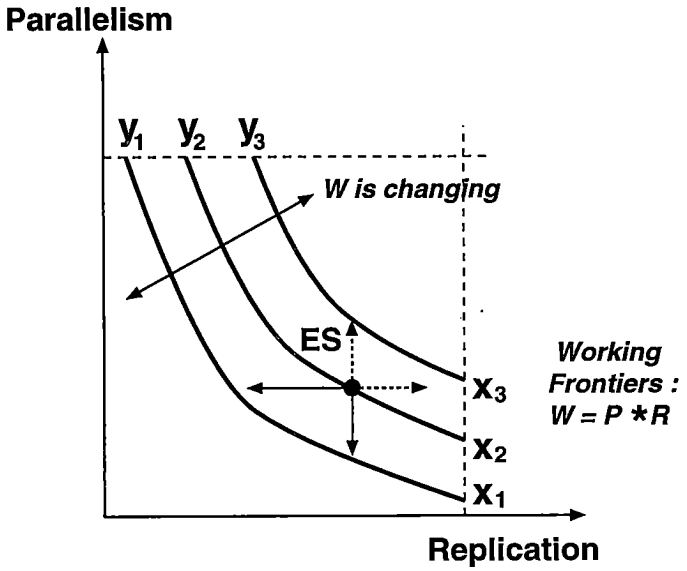


Figure 7.5: The Execution State of RPSs in a Competitive Environment

Remark No. 2: Another advantage of the RPS model over the RSS model is its adaptability to resource availability. Besides the capability of deploying the LWD described in Chapter 4, an RPS can change its execution state through the Comedians system. The Cluster Schedulers (CSs) of the Comedians system can measure the runtime efficiency $E_p(P)$ of the RPSs and provide them with suitable allocations to change the attributes of the execution states. The attributes are parallelism, cluster type and replication.

If more workstations become available, the grains of an RPS can split to a higher partitioning level, otherwise the grains merge to a lower partitioning level or migrate to other workstations if the workstations on which the grains are running are heavily loaded with external processes.

In addition, the number of the RPSs can be altered by restarting a different number of RPSs, but it is less responsive than the GWD because once an RPS is started, it will run to completion. On the other hand, the RSSs can only respond to the environment by program restart or migration.

7.4.3 The RPSs & the Comedians System

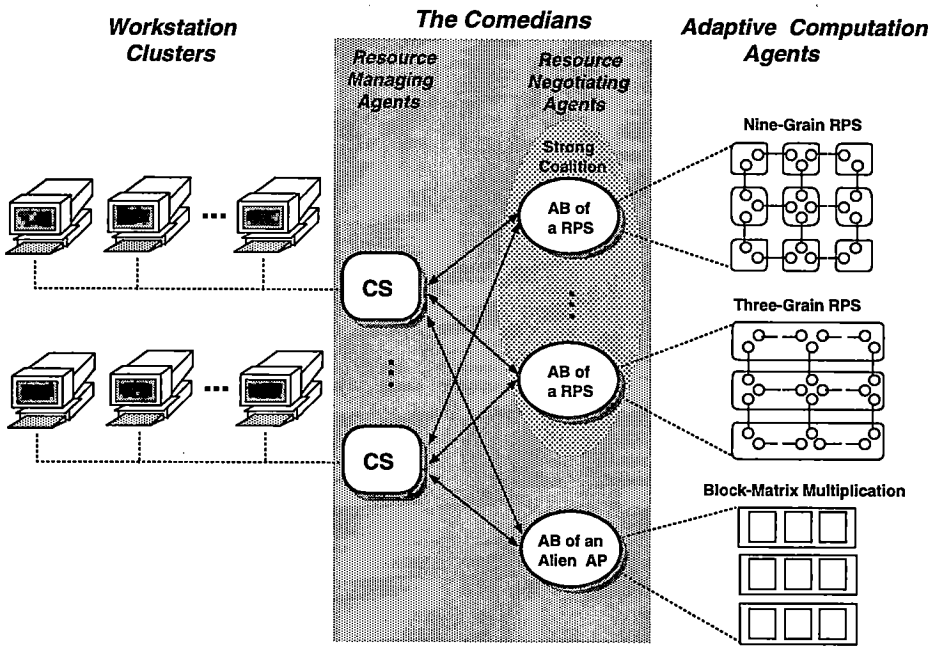


Figure 7.6: The System View of RPSs

The adaptive execution of RPSs is supported by the Comedians system. As shown in Figure 7.6, the Comedians system provides the resource management functions in workstation clusters. The competition between RPSs and other applications is solved by the auctions held by CSs. The RPSs of the same user form a strong coalition. Other parallel applications of other users that are running on the Comedians system are regarded as *alien* applications to the coalition. On the other hand, the alien applications regard the coalition as independent parallel applications. For example, the parallel block-matrix multiplication program shown in Figure 7.6 is an alien application to the coalition of the RPSs.

If the number of simulation runs is greater than the number of RPSs, the RPSs must be able to restart themselves. Figure 7.7 depicts the interaction between the CSs, AB and grains of an RPS during a simulation run restart. If the grains of the RPS have completed the present simulation run, the AB will check whether the termination condition of simulation is satisfied. If the termination condition is satisfied, the AB will inform the grains to terminate; otherwise the grains will wait for a new allocation of workstations before restarting the next simulation run. The AB also sends a message to its local CS about the completion of its present run. The local CS then allocates available workstations to the AB and informs

remote CSs to update the status of the RPS. The next simulation run restarts when the grains receive a new copy of task-to-grain and grain-to-workstation mappings from the AB.

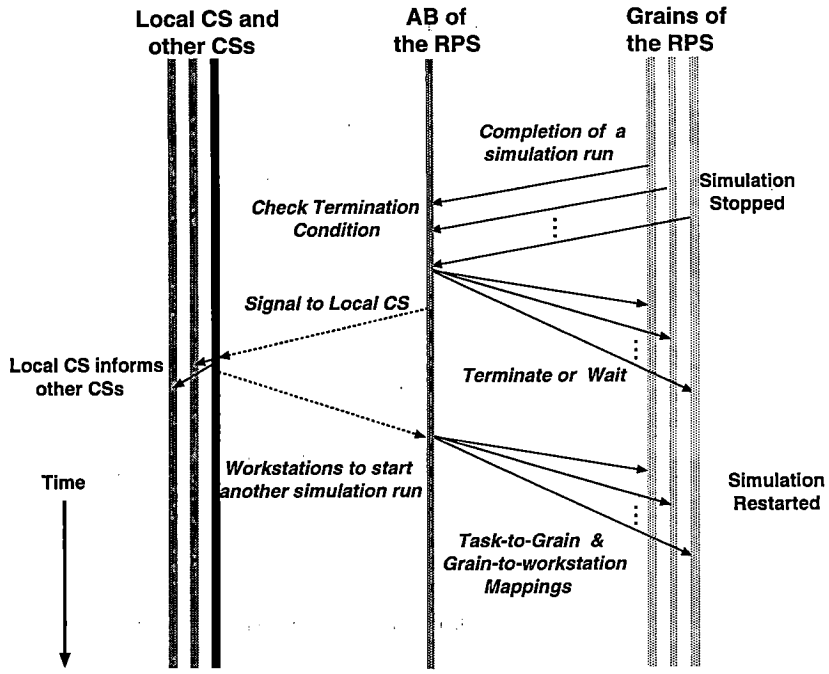


Figure 7.7: Interaction between the Cluster Schedulers (CSs), Application Bidder (AB), and Grains of an RPS during a simulation restart.

7.4.4 Scheduling Policies

The rules of competition are determined by the scheduling policies of the RPSs. As mentioned earlier in Section 7.4.2, the *first N replications initiated* (FNI) scheduling policy is applied by the RPSs to obtain statistically accurate simulation results. Since the FNI does not specify any control over workstation allocation, an RPS will compete with all the other applications in the Comedians system based on its runtime performance. To harness the dynamic computational resource of workstation clusters, special scheduling policies are proposed to be built on top of the FNI. These policies are enforced by the CSs of the Comedians system. Three scheduling policies are tested and compared in this case study; they are FNI-SA *(Static Assignment)*, FNI-DA *(Dynamic Assignment)*, and FNI-DAC *(Dynamic Assignment with Coalition)*.

In the FNI-SA policy, an RPS can only be executed on the workstations of its local cluster, but different RPSs can be assigned to different local clusters. On the other hand, the RPSs in the FNI-DA policy can be executed on the workstations of multiple clusters dynamically. The FNI-DA and the FNI-DAC policies are basically the same except that the FNI-DAC policy allows the RPSs from the same user to become coalition partners.

Since RPSs from the same user have a single objective – the completion of all simulation runs, the RPSs should not compete with one another for workstations. Instead the RPSs with earlier initiated simulation runs should have higher priority than the others to get resources because the simulation terminates as soon as the first N replications finish their runs. This special condition allows the RPSs to be executed more efficiently on workstation clusters by forming a coalition. If a coalition is formed, other parallel applications that are running on the Comedians system are regarded as alien applications to the coalition.

There are two requirements for being a member of a strong coalition of RPSs:

- An RPS will not compete with other RPSs that are in the same coalition. In other words, an RPS will not split to a higher partitioning level at the expense of forcing other coalition partners to merge to a lower partitioning level or to migrate to slower workstations.
- Among coalition partners, the earliest initiated replication has the highest urgency to finish. Therefore, an RPS has a priority over other coalition partners to run on newly available workstations if its simulation run was initiated earlier than the others.

7.4.5 Experimental Results

In this section, the experiments and results for testing and evaluating the proposed scheduling policies are presented. The experiments are carried out on DEC3100 and DEC Alpha workstation clusters in which all the workstations are connected by Ethernet in different segments. The simulation model for the RPSs is the same as the model of the parallel simulator of an ATM network (PSA). The RPSs can be partitioned into one grain (partitioning level 0), three grains (partitioning level 1), and nine grains (partitioning level 2). Different simulation runs of the RPSs have different sets of random number seeds for simulation operations in the grains.

The scheduling policies are tested in two scenarios. In the first scenario, simulations are running on two homogeneous DEC3100 workstation clusters; each cluster consists of twenty-four workstations. Figure 7.8 shows the relative speedup of the FNI-SA and the FNI-DA policies for different numbers of RPSs. The figure records the maximum, the mean, and the minimum relative speedup of the RPSs for finishing ten simulation runs; each run performs a simulation of half a million time units. The values of the speedup are relative to the execution time of an RPS in a DEC3100 workstation at partitioning level 0.

7.4.5.1 Comparison between Static and Dynamic Assignments

The performance of the FNI-SA and the FNI-DA policies is similar until the number of RPSs becomes five. In this case, three RPSs are running on the first cluster and two RPSs are running on the other. There are not enough workstations for all the RPSs to split to the highest partitioning level. As indicated in Figure 7.8, the speedup is improved by applying the FNI-DA policy in this situation because the RPSs can relocate their grains to the remote cluster for execution. The discrepancy of the speedup between the maximum and the minimum is great when four RPSs are running, because two of the RPSs perform one more simulation run than the other RPSs. Discussion will later concentrate on the minimum speedup (or the maximum turnaround time) of the RPSs because it determines the overall turnaround time of the simulation.

Figures 7.9 and 7.10 are used to compare the FNI-SA and the FNI-DA policies when five RPSs are running on the two clusters of DEC3100 workstations and the length of simulation time or the number of simulation runs varies. As can be seen from Figure 7.9, the benefit of reducing the overall turnaround time using the FNI-DA policy increases gradually when the simulation time is lengthened. Figure 7.10 shows the difference of these two policies for different numbers of simulation runs: each simulation run performs half a million simulation time units.

These two policies are also tested in the situation when an alien application is running on one of the clusters. The alien application is a block-matrix multiplication program (BMM) which has the same partitioning levels as the RPSs. Figure 7.11 depicts the relative speedup of the simulation when different numbers of RPSs are running with the alien application. When the number of RPSs increases to three, the relative speedup of RPSs for the FNI-SA drops because two RPSs and the alien application are competing for workstations on the same cluster. The

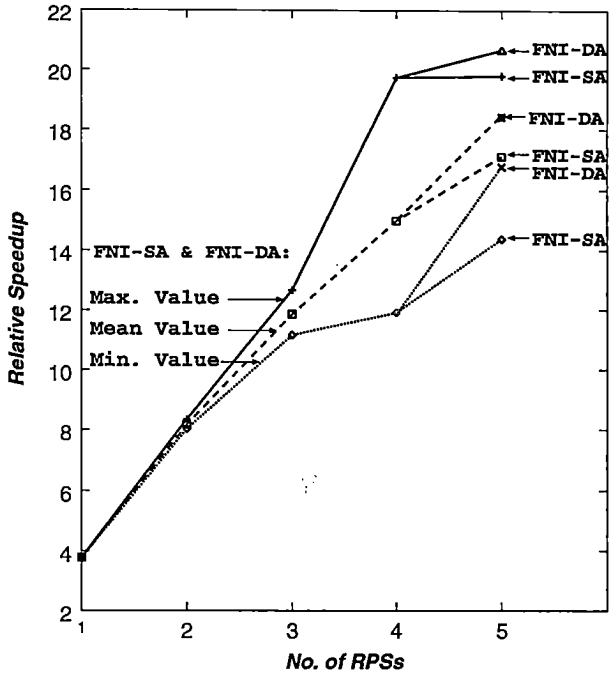


Figure 7.8: Relative Speedup obtained when simulation runs are performed by different numbers of RPSs (Scenario One: two DEC3100 clusters; each cluster has 24 workstations.)

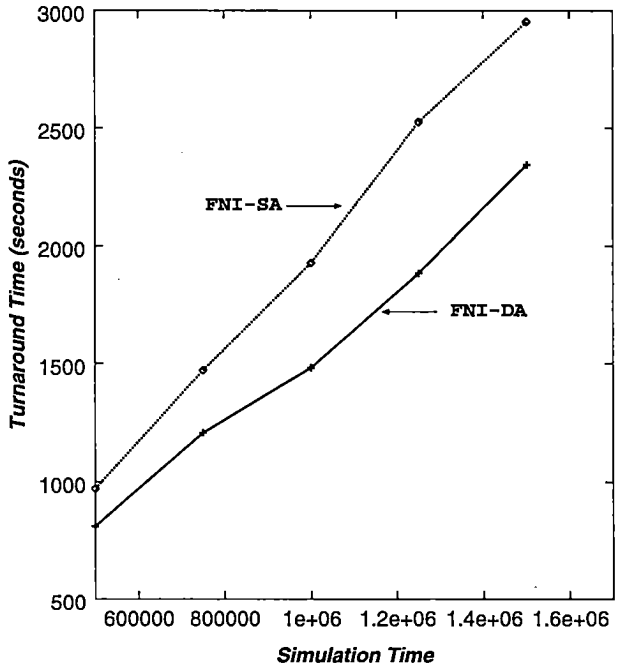


Figure 7.9: Performance of the FNI-SA and FNI-DA policies for different simulation times (Scenario One: two DEC3100 clusters; each cluster has 24 workstations.)

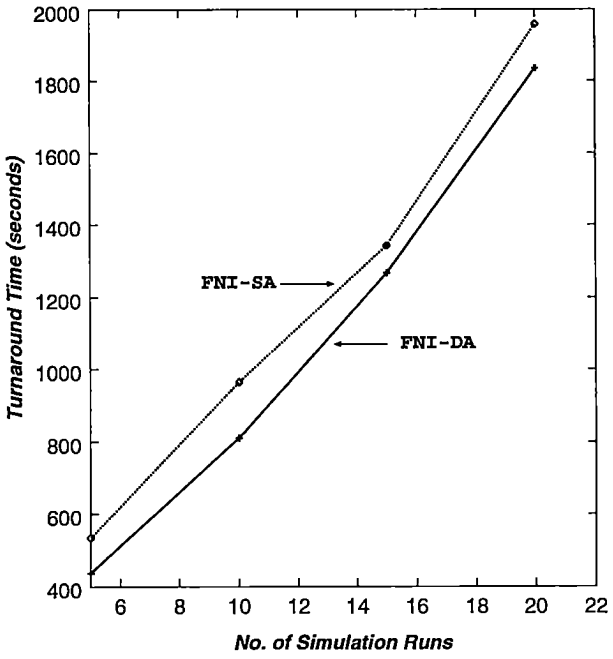


Figure 7.10: Performance of the FNI-SA and FNI-DA policies for different numbers of simulation runs (Scenario One: two DEC3100 clusters; each cluster has 24 workstations.)

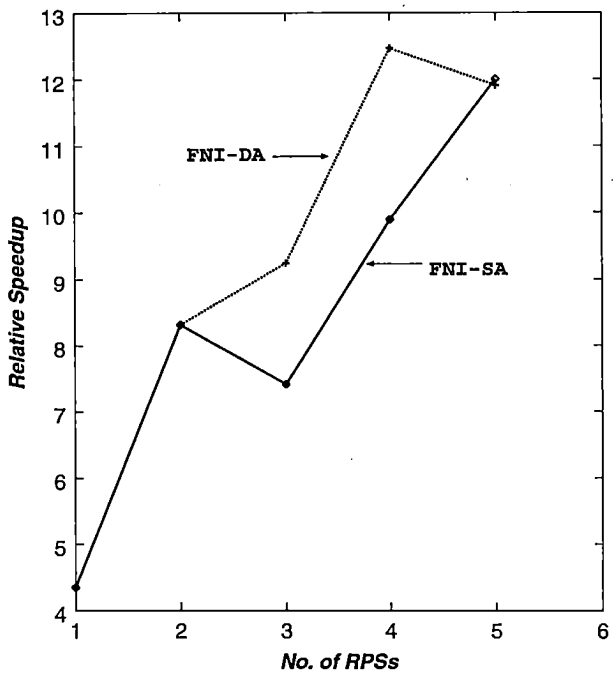


Figure 7.11: Relative Speedup of the FNI-SA and FNI-DA policies when there is one alien application running on the first cluster (Scenario One: two DEC3100 clusters; each cluster has 24 workstations.)

results show that the FNI-DA policy can sustain performance gain by allowing one of the RPSs to migrate its grains to the remote cluster when there are not enough workstations available in its local cluster. However, when the number of RPSs is greater than or equal to five, both clusters become overcrowded.

7.4.5.2 The Effect of Coalition Formation

In the second scenario, the simulation is performed on two heterogeneous workstation clusters – one cluster of thirty DEC3100 workstations and the other cluster of six DEC Alpha workstations. Since the speedup of an RPS at partitioning level one in DEC Alpha workstations is higher than the speedup at partitioning level two in DEC3100 workstations, the RPSs will compete for DEC Alpha machines even at the expense of merging to a lower partitioning level. As described in Section 7.4.4, the FNI-DAC policy is a modified version of the FNI-DA policy, which is proposed to facilitate the coalition of the RPSs. Experiments are conducted to compare the FNI-DA policy with the FNI-DAC policy. Figures 7.12 through 7.15 depict the performance of the policies when there are three running RPSs.

Figures 7.12 and 7.13 illustrate the execution time of the three coalition partners of RPSs for different periods of simulation run. The shaded and white time slots depict the allocations of DEC Alpha and DEC3100 clusters respectively. Since the six DEC Alpha workstations can only accommodate two RPSs at partitioning level one at the same time, one of the RPSs has to execute in the DEC3100 cluster. When an RPS running on the DEC Alpha cluster terminates, the RPS with the earliest initiated time will migrate from the DEC3100 cluster to the DEC Alpha cluster.

The *End Effect* of the RPSs will become significantly important in this setting of heterogeneous clusters because the difference in processing speed between DEC3100 and DEC Alpha workstations is quite great. The formation of a coalition can reduce the End Effect because the earliest initiated RPS can always execute in the faster cluster. The figures show that the RPSs run on the DEC Alpha cluster alternately.

Owing to the competition for workstations in the Comedians system, it is possible that some RPSs are constantly excluded from running on DEC Alpha machines. As a result, the overall turnaround time will be restricted by the slowest RPS. This special case of the FNI-DA policy is compared with the FNI-DAC policy. Figures 7.14 and 7.15 suggest that the difference in the overall turnaround time

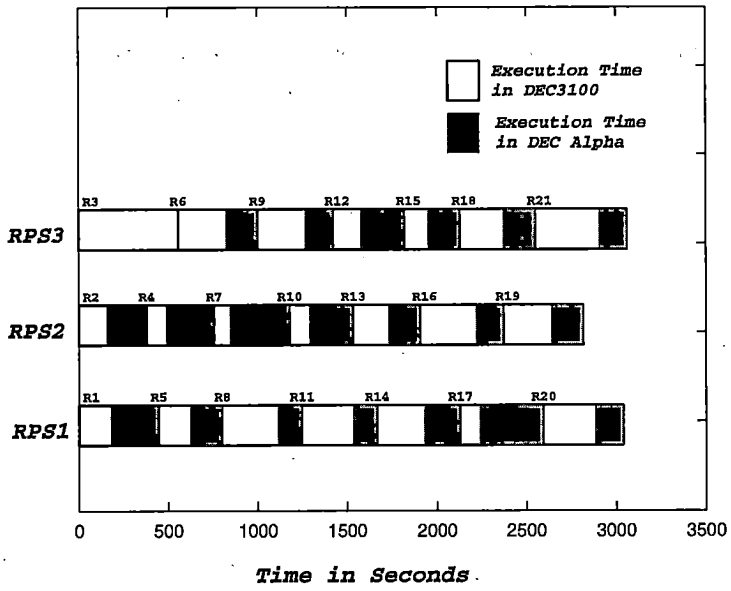


Figure 7.12: The execution time of three RPSs in DEC3100 and DEC Alpha clusters; each simulation run takes 0.75 million time units (Scenario Two: two heterogeneous clusters; one has 30 DEC3100 workstations, the other has 6 DEC Alpha workstations.)

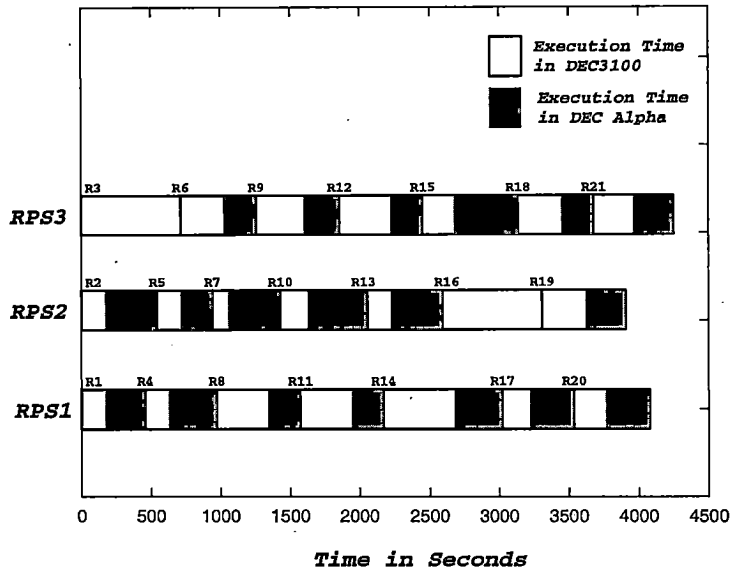


Figure 7.13: The execution time of three RPSs in DEC3100 and DEC Alpha clusters; each simulation run takes 1 million time units (Scenario Two: two heterogeneous clusters; one has 30 DEC3100 workstations, the other has 6 DEC Alpha workstations.)

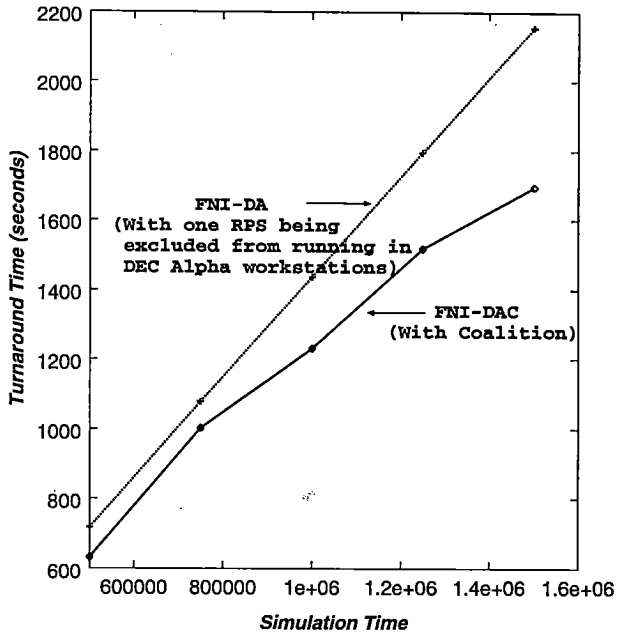


Figure 7.14: Performance of the FNI-DA and FNI-DAC policies for different simulation times (Scenario Two: two heterogeneous clusters; one has 30 DEC3100 workstations, the other has 6 DEC Alpha workstations.)

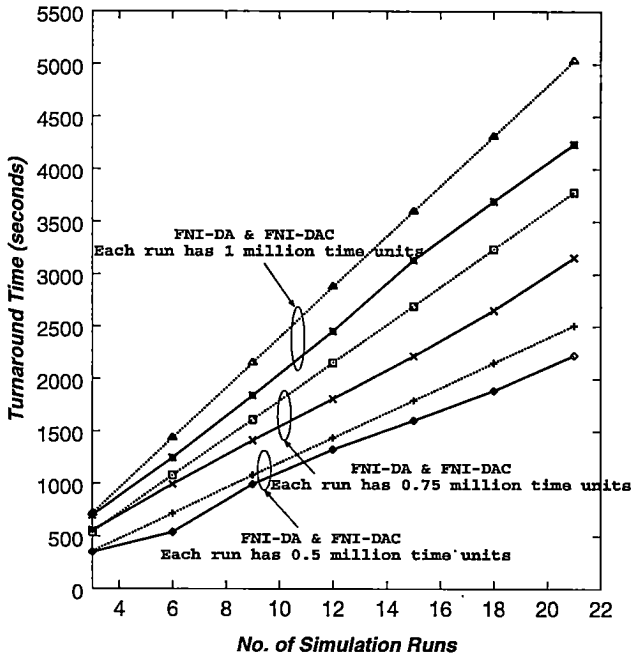


Figure 7.15: Performance of the FNI-DA and FNI-DAC policies for different numbers of simulation runs (Scenario Two: two heterogeneous clusters; one has 30 DEC3100 workstations, the other has 6 DEC Alpha workstations.)

between these two policies rises slowly when the length of simulation time per run or the number of simulation runs increases. It is found that the FNI-DAC policy generates a better overall performance on heterogeneous workstation clusters because the priority of workstation allocation of the FNI-DAC policy is determined by the order of initiation.

7.5 Summary

Replication was found to be a useful technique to complement parallelism for a multi-user, multi-parallel program environment. Replicated applications owned by a user can be grouped as a weak coalition or a strong coalition to facilitate better resource management. Moreover, an efficient way of supporting fault tolerance can be provided by replication.

RPSs (Replicated Parallel Simulators) were suggested as part of a case study of a strong coalition to speed up the simulation of an ATM network. It can be observed from performance evaluation that the RPSs can greatly enhance performance for computationally intensive simulation. The design and implementation of a runtime system which harnesses the aggregate computing power of workstation clusters by the RPSs in the Comedians system were presented.

Scheduling policies were proposed to be built on top of the FNI (First N replications initiated) policy so as to provide better resource allocations for the simulation runs of the RPSs. The policies were tested and evaluated by experiments in the scenarios of homogeneous and heterogeneous workstation clusters. The experimental results show that the overall performance of the simulation can be enhanced significantly by applying replication to parallel simulation modelling. The results also indicate that the overall performance can further be improved if (i) the RPSs can relocate their grains to remote clusters dynamically; (ii) the RPSs can form a coalition so that the priority of workstation allocation decreases with the order of initiation.

Chapter 8

Conclusion and Further Work

This dissertation has presented a framework to support adaptive parallelism on heterogeneous workstation clusters. The framework comprises software support for adaptive workload distribution and system support for parallel application scheduling. By managing replicated applications based on the framework, a multi-user, multi-parallel program environment was created. This chapter concludes the dissertation and suggests scope for further work.

8.1 Summary

Chapter 3 described two different metrics to structure parallel workloads in a dynamic environment. The chapter proposed a reconfigurable software structure that provides the basis of adaptability to dynamic resource availability and workload imbalances. The structure allows applications to decompose into tasks and the tasks can be grouped into grains. The performance of applications can be optimized by suitable task-to-grain and grain-to-workstation mappings.

In Chapter 4, the schemes of workload distribution, namely Local Workload Distribution (LWD) and Global Workload Distribution (GWD), were designed for distributing workloads based on the above reconfigurable structure. The LWD distributes workloads locally between neighbouring grains and the GWD resolves significant workload imbalances by grain migration, splitting, and merging. The schemes were tested on different examples of data-parallel and task-parallel applications in situations subjected to workload imbalances arising from dynamic workload interference and heterogeneity.

Chapter 5 presented a system called *Comedians* that can maximize the speedup of individual parallel applications and, at the same time, allocate workstations fairly and efficiently to the applications. There are two types of agent processes in the Comedians system, namely Cluster Scheduler (CS) and Application Bidder (AB). The CS manages the workstations in a cluster and the AB provides an interface between an application and the CSs. The mechanisms of allocation are auctions and bidding which are driven by competition between applications. No prior knowledge about the runtime behaviour of applications is required because the values of bids are formulated according to the execution times of grains collected by the ABs. Each application responds to external workload and auction results through dynamic reconfiguration. The experimental results showed that stable or Pareto efficient allocations can be achieved through competition in different scenarios.

The functions of the Comedians system were enhanced in Chapter 6 so that workstations in heterogeneous clusters can be utilized by parallel applications as a single virtual machine. Auction and bidding are carried out across multiple clusters. Execution states were introduced to represent the status of running applications in a simple and efficient way.

Finally, Chapter 7 introduced the concept of combining parallelism with replication. Coalitions are formed to organize replicated parallel applications. The concept enables better management of applications and facilitates efficient fault tolerance. A case study was carried out to realize the concept by implementing special scheduling policies for the model of replicated parallel simulators. The effectiveness of the scheduling policies was demonstrated by experiments performed with the Comedians system.

In brief, the design, implementation, and evaluation of the following novel and related ideas were presented in this dissertation:

1. A software structure and schemes of adaptive workload distribution that optimize the performance of different types of parallel application by dynamic reconfiguration at runtime.

2. A scheduling system that facilitates fair and efficient resource allocation to parallel applications on heterogeneous workstation clusters through competition.

3. The management of replicated parallel applications by coalition formation.

4. A multi-user, multi-parallel program environment on heterogeneous clusters.

8.2 Further Work

- **Automatic generation of the reconfigurable software structure**
  In the current implementation, it is a programmer's responsibility to define boundaries between tasks and grains, and to choose a suitable workload distribution scheme for a particular parallel programming paradigm. The burden of programmers could be reduced by a special compiler that can generate boundaries for possible reconfigurations and apply a suitable workload distribution scheme automatically.

- **Large scale experiments of auction and bidding**
  The maximum number of workstations involved in the current experiments for the Comedians system is about sixty. Further experimental work on studying the dynamics of application competition could be conducted on a larger network with higher heterogeneity in machines and connections. A theoretical study of the interaction of adaptive parallel applications in the Comedians system could also be investigated.

- **Cluster computing for next-generation applications**
  The examples of parallel applications used for illustrations and experiments in this dissertation were mainly scientific computational problems. The benefit of solving new applications by workstation clusters could be investigated. These applications may require high computational power and the exploration of large data sets through the Internet, such as Web servers, video servers, and data-mining engines. To meet the dynamic demand of these applications, a variable number of high-performance workstations interconnected with a high-speed network can work cooperatively as a highly scalable and available *virtual server*.

- **Relation between efficient data transfer and resource scheduling**
  For real-time applications, response time is an important factor for performance. The time for data transfer such as parallel I/O latency and communication protocol overheads should be minimized. New scheduling algorithms for cluster computing could be derived to hide the data transfer time by workload and I/O optimization.

- **Distributed Artificial Intelligent Agents**
  In the dissertation, parallel applications in the Comedians system can be regarded as adaptive computation agents for intensive scientific computing. Besides scientific computing, applications that demand multiple requirements of resources such as database searching and continuous media processing will require reasoning power to handle highly complicated resource negotiation. On the other hand, the Comedians system could be modified to provide resource arbitration of CPU time, communication bandwidth, and storage capacity between these applications. The challenge of this system is to organize applications that demand diverse but correlated resource requirements.

8.3 Conclusion

This dissertation has examined an integrated framework to implement a multi-user, multi-parallel program environment on workstation clusters. Experimental studies have shown that the framework enables adaptive parallel applications to harness the resources of heterogeneous workstation clusters as a single virtual machine.

Three distinct but complementary functions have been set up to achieve adaptive parallelism, namely the adaptability, resource allocation, and coalition formation of parallel applications. First, the dissertation has shown that both data-parallel and task-parallel applications which are based on the reconfigurable software structure can reconfigure adaptively according to resource availability by applying suitable tailor-made workload distribution schemes. Secondly, the Comedians system has been shown to be effective and flexible in allocating workstations to competing applications through auctioning and bidding. Based on the results of auctions, individual applications can respond to different allocations by dynamic reconfiguration. Finally, fault tolerance and overall performance enhancement capabilities in managing replicated parallel applications through coalition formation have been demonstrated.

The functions of adaptability and resource allocation have separate control over parallel applications so that detailed design of individual functions can be implemented and refined independently. However, they work co-operatively to facilitate a unified support for adaptive parallelism.

Appendix A

Partitioning and Mapping using Market Price

A.1 Introduction

The algorithms of partitioning and mapping that use market price as a metric for structuring workloads are presented in this appendix. Applications are led through two phases: partitioning and mapping. The partitioning phase includes two stages: granulation and merge. The granulation stage involves computing the proportion of the application and the number of grains for each workstation cluster; the merge stage groups the tasks into grains. In doing so, the workstation cluster size, processing, interference, and communication prices incurred by the individual workstations need to be considered. This partitioning phase is similar to the clustering algorithms proposed in [GE76, Efe82, BNG92]. These clustering algorithms minimize interprocess communication cost by grouping processes into clusters. However none of these algorithms evaluate the computation and communication capacity using dynamic information, such as the market price, that can reveal the current supply of and demand for the resources.

Following the partitioning phase, each selected workstation cluster will have a number of grains to be properly mapped to the constituent workstations. The mapping phase produces the best possible distribution based on the price determined by market rules. It, too, involves two stages: initial assignment and dynamic grain exchange between the workstations to improve the initial assignment. The components of the price involve computation, communication, and

other factors that may be present in market rules, and which would normally show variations because of the supply of and the demand for the resources concerned.

The algorithms of the partitioning and mapping phases can be re-activated whenever the computation or communication requirements of the application change or the computation or communication capacity of the workstation cluster varies. In the latter case, the prices can reveal the availability of resources dynamically.

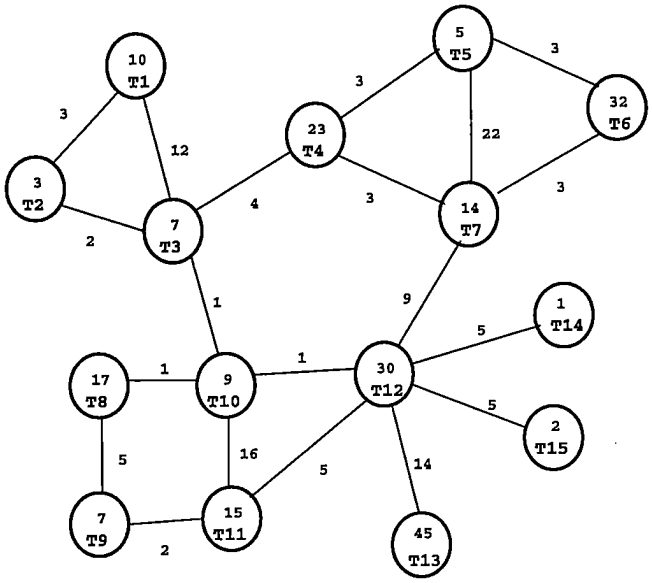


Figure A.1: The Task Graph of an Application

Let the application be represented by a triple $\Theta = (T, E, C)$ where T is a set of tasks, E is the set of task execution times, and C is the set of inter-task communication requirements. The DCS is a set of clusters, $\Gamma = \{\Gamma_i | i \in n\}$. A workstation cluster is a set of workstations, $\Gamma_i = \{W_{i,j}, | i \in n, j \in m_i\}$. The workstations are related to each other by the price of unit data communication between them. Obviously, the price would be infinity if they were not related at all. Figure A.1 shows an example of the connected task-graph of an application.

The following list documents all the symbols used in the rest of the appendix:

C_{ij}	Communication requirement (no. of bytes) between two tasks
C_T	Total communication requirement among tasks of an application
E_i	Task Execution requirement (no. of instructions)
E_T	Total task execution requirement of an application
N_T	Total number of tasks
N	Total number of clusters
W_T	Total number of available workstations
M_i^a	No. of available workstations (ws) in a workstation cluster
P_i^p	unit processing price of a workstation cluster
P_i^t	unit communication price of a workstation cluster
$P_{i,j}^e$	unit communication price between two ws in different clusters
G	Total number of grains
g_i	No. of grains in a workstation cluster
r_i	Total grain-size in a workstation cluster
t_p^u	Maximum expected processing time of an application
t_p^l	Minimum expected processing time of an application
t_t	Maximum expected communication time of an application
t_e	Maximum expected inter-cluster communication time
X_i	ws processing capacity (instr./sec.) offered by a workstation cluster
Y_i	ws communication capacity (bytes/sec.) offered by a workstation cluster
$Z_{i,j}$	inter-cluster communication capacity (bytes/sec.) offered by two workstation clusters

A.2 Partitioning Algorithm

A.2.1 Granulation of the Application

Since each grain is run on a single workstation, the size of a grain (grain-size) is an indication of the capacity offered by the hosting workstation. The higher the capacity, the larger will be the grain-size. When an application is mapped onto a number of workstation clusters, the algorithm shown in Figure A.2 computes the total grain-size (r_b) and the number of grains (g_b) for a workstation cluster b. The prelude to the algorithm is to compute the normalized total price for each workstation cluster, which is a function of computation and communication capacities of the cluster. The workstation clusters (Γ^s) are sorted in ascending order of their normalized total prices as indicated at the bottom end of the algorithm.

First, for each workstation cluster b (initially, b=1), the algorithm chooses g_b, between the lower and upper bounds imposed by the application and workstation

Cluster_Grain (b, r_b) : initially the cluster number b=1, the grain size $r_1 = 1$

Begin

$M_b^l \longleftarrow \lceil \frac{E_T}{X_b \cdot t_p^u} \cdot r_b \rceil$

$M_b^u \longleftarrow \lceil \frac{E_T}{X_b \cdot t_p^l} \cdot r_b \rceil$

$g_b' \longleftarrow M_b^u \cdot f(P_b^p)$

While $[g_b' > (\frac{N_T \cdot Y_b \cdot t_t}{C_T} \cdot g(P_b^t))]$

If $(g_b' > M_b^l)$

$g_b' \longleftarrow g_b' - 1$

Else

Termination : fail to satisfy execution requirements

If $(g_b' > M_b^a)$

If $(b = N)$ or $[g_b' > (\frac{N_T \cdot Z_{b,b+1} \cdot t_e}{C_T} \cdot h(P_{b,b+1}^e))]$

Termination : fail to satisfy execution requirements

Else

$r_{b+1} \longleftarrow r_b \cdot (1 - \frac{M_b^a}{g_b'})$

$r_b \longleftarrow r_b \cdot \frac{M_b^a}{g_b'}$

$g_b \longleftarrow M_b^a$

Cluster_Grain $(b+1, r_{b+1})$

Else

$g_b \longleftarrow g_b'$

End

Definition of Symbols :

$\Gamma^s = \{\Gamma_i^s | i \in n\}$, and Γ_i^s are arranged in sorted normalized total price order :

$$(\frac{E_T}{X_i} \cdot P_i^p + \frac{C_T}{Y_i} \cdot P_i^t) \leq (\frac{E_T}{X_{i+1}} \cdot P_{i+1}^p + \frac{C_T}{Y_{i+1}} \cdot P_{i+1}^t)$$

$f(P_b^p)$, $g(P_b^t)$, and $h(P_{b,b+1}^e)$ are the utilization rate of average processing power, communication capacity, and inter-cluster communication capacity, and $P_{b,max}^p$, and $P_{b,max}^t, P_{b,max}^e$ are the maximum prices of the respective resources.

$$f(P_b^p) = \frac{P_b^p}{P_{b,max}^p}, \quad g(P_b^t) = \frac{P_b^t}{P_{b,max}^t}, \quad h(P_{b,b+1}^e) = \frac{P_{b,b+1}^e}{P_{b,max}^e}$$

Figure A.2: Granulation of the Application

execution data. Second, it checks whether the inter-workstation communication requirement is also satisfied. Since the average communication cost is proportional to g_b, the communication constraint can be represented as $Y_b \cdot g(P_b^t) \geq \frac{C_T \cdot g_b}{t_t \cdot N_T}$. If the communication requirement is not satisfied, the value of g_b will be reduced, otherwise it will check whether g_b is greater than the number of available workstations in the workstation cluster. If the computation and communication capacities are insufficient, the algorithm will split the application into grains of appropriate sizes to be allocated to this cluster. Grains can be assigned to other workstation clusters only if the inter-cluster communication requirement is satisfied. The algorithm runs recursively until the entire application is granulated.

A.2.2 Merging Grains

The second stage of the partitioning algorithm is to merge tasks into G grains for each Γ_b^s according to the values of g_b. The merging algorithm, described in Figure A.3, starts with selection of nuclei for each grain. The nuclei are the most demanding tasks in terms of computation and communication costs (Φ). Each nucleus may merge with a number of non-nucleus tasks. With each merge, the computation and communication costs of the merging non-nucleus tasks are added to those of the nucleus. Out of the T tasks, G nuclei are chosen. From $b = 1$ to $b = N$, g_b nuclei are allocated to Γ_b^s. The nuclei will be selected one by one, and the one with the highest value of Φ will be selected first.

The merging decision is determined by the value of, so called, *decoupling force* (*df*) between a nucleus and a non-nucleus task. The *df* between t_i and t_j, consists of two components as shown in *Definition of Symbols* section of the merging algorithm in Figure A.3. The first term indicates the computation cost of merging two tasks. The second term is the difference between the communication cost of running two tasks that have been merged and the communication cost of running two separated tasks.

The interference costs account for the cost of incompatibility of task pairs [Lo88]. I^p is called the *processor-based interference cost* incurred by process switching and synchronisation; I^{c1} is the interference cost incurred by interprocess communication. In addition, when two communicating tasks are executed on the same processor, the overhead in multiplexing/demultiplexing the aggregate communication data should also be considered as illustrated in (Figure A.4). I^{c2} represents this overhead which possesses the following property :

$$\sum_{l}^{n_i+n_j-2} I^{c2}(i,l) \geq \sum_{l}^{ni} I^{c2}(i,l) + \sum_{l}^{nj} I^{c2}(j,l) \tag{A.1}$$

Higher values of df represent greater advantage in separating the tasks, thus, favouring parallel execution. Therefore, merging two tasks with the minimum values of df will maximize the benefit of concurrency, under the computation and communication constraints of the workstation clusters. For a task t_i, it will merge with a nucleus t_j if the decoupling force $df(i,j)$ is the minimum.

In summary, the decoupling force plays an important role in the merging algorithm. There are three distinctive advantages of using it as a metric in merging a task with a nucleus:

1. **Load Balancing**
   If only computation cost is taken into consideration, a task will merge with a nucleus with the smallest computation cost. In this case, a task is more likely to merge with a smaller nucleus or a nucleus allocated to a more lightly-loaded cluster than otherwise.

2. **Minimization of Communication Overheads**
   If two tasks are merged, the cost of inter-workstation communication between them can be saved. However, if tasks are assigned to the same workstation, they also share the same communication facilities, such as the communication processor. Conversely, the tasks assigned to different processors will not compete for the same communication facilities. Nevertheless, the inter-workstation communication can be expensive. The difference between merging and parallel execution is evaluated by the decoupling force.

3. **The Relative Importance of Processing and Communication**
   The prices of computation and communication are used to measure the relative importance of computation and communication. Use of the decoupling force gives appropriate weight to balancing the processing load as against minimizing communication overheads.

Figure A.5 shows an example of merging the application shown in Figure A.1 into grains. In this example, the results of the granulation algorithm on the application shown in Figure A.1 is $\{g_i\} = \{4, 3, 1\}$, meaning that four grains, three grains and one grain will be mapped into the first, second, and third cluster respectively. These clusters are the elements of the set Γ^s. The grains will then be allocated to workstations in three workstation clusters by the mapping algorithm.

Merging

Begin

Select G tasks from T to be nuclei with highest values of Φ,
among them select g_b nuclei to Γ_b^s,

For every $t_i \in T$ except the nuclei

For every nucleus t_j allocated to Γ_b^s

Calculate decoupling force $df(i,j)$

Merge t_i into t_j with the minimum value of $df(i,j)$,
(break ties by smaller value of E_j)

Select g_b nuclei to be allocated to Γ_b^s as grains

End

Definition of Symbols :

$$G = \sum_{i=1}^{N} g_i,$$

$$\Phi = E_i \cdot P_b^p + (\sum_j C_{ij}) \cdot P_b^t,$$

$$df(i,j) = [(E_i + E_j) + I^p(i,j)] \cdot P_b^p +$$

$$\{[\sum_l^{n_i+n_j-2} I^{c2}(i,l) + I^{c1}(i,j)] -$$

$$[\sum_l^{ni} I^{c1}(i,l) + \sum_l^{nj} I^{c2}(j,l) + C_{ij}]\} \cdot P_b^t$$

where I^p is processor-based interference cost,
I^{c1} is interference cost due to intertask communication,
I^{c2} is interference cost due to multiplexing and demultiplexing,
n_i is the number of communication links of task t_i

Figure A.3: Merging Grains

A.3 Mapping Algorithm

A.3.1 Initial Assignment

After the merging process, the mapping algorithm that assigns the allocated grains into workstations will be executed on one of the workstations in each workstation cluster, called the *cluster controller*. The algorithm (Figure A.6) has two parts: initial mapping and dynamic exchange. In the first part, the grain

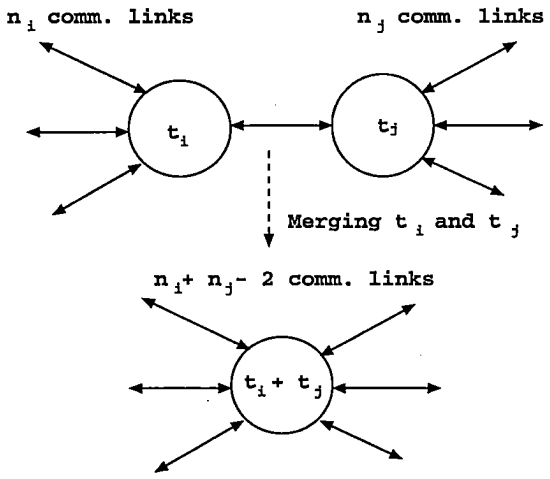


Figure A.4: Merging of two tasks

with the maximum total cost of computation and communication is assigned to the workstation with the minimum total price. The processing price of the workstation ($P^p_{b,x}$) is then updated according to the proportion of consumption as in Equation A.2.

$$P^p_{b,x} = P^p_{b,x} \cdot (1 + \frac{X_b}{X^T_b}) \tag{A.2}$$

where X^T_b is the total computation capacity of a workstation in Γ^s_b.

After the first assignment, the next adjacent grain with the maximum total cost is assigned to the workstation with the minimum total price. Besides the processing price, the link communication prices ($P^t_{b,\alpha\beta}$) along the path between the workstations to which the two adjacent grains are assigned are also updated by Equation A.3. This mapping process is repeated until all the grains have been mapped.

$$P^t_{b,\alpha\beta} = P^t_{b,\alpha\beta} \cdot (1 + \frac{Y_b}{Y^T_b}) \tag{A.3}$$

where $W_{b,\alpha}$ and $W_{b,\beta}$ are the adjacent workstations along the path between workstations $W_{b,x}$ and $W_{b,y}$ to which t_i and t_j are assigned respectively, and Y^T_b is the total communication capacity of a workstation in Γ^s_b.

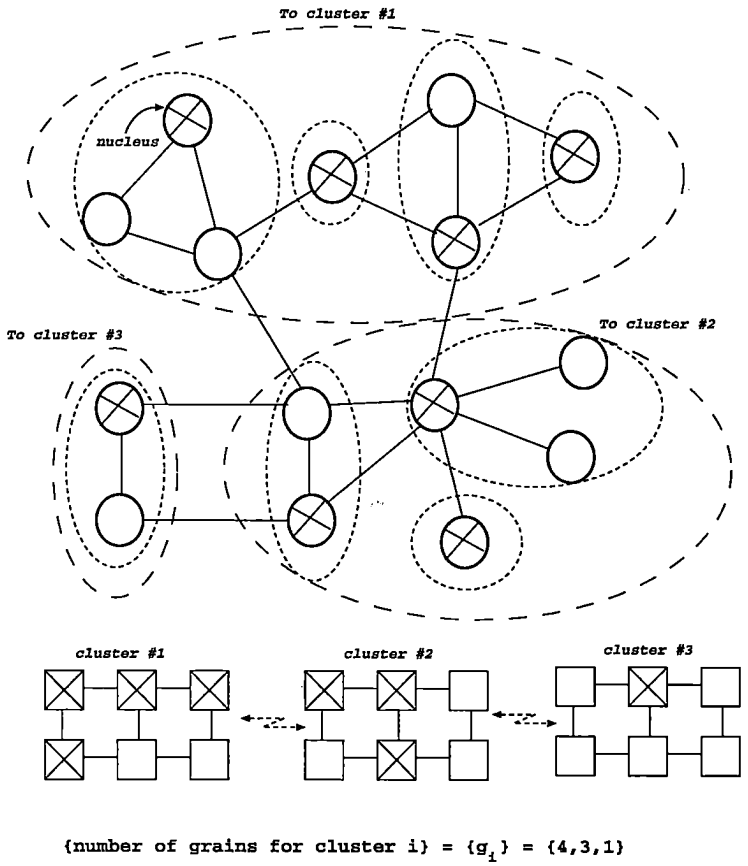


Figure A.5: An Example of the Merge Stage

A.3.2 Dynamic Exchanges

The second part of the mapping algorithm involves dynamic exchanges to improve the initial mapping. First, the cluster controller searches for the maximum difference between the cost of grain execution in the cluster. If the difference is greater than ϵ and the number of dynamic exchanges for this mapping is less than η, then evaluation for grain exchange continues. The values of ϵ and η are specified constants. Secondly, if the maximum gain in a grain exchange ($\Psi_{x,y}$) is greater than ϵ, then the grain exchange will be performed. After each exchange, the processing and communication prices related to the exchange are updated accordingly. The algorithm repeats until all the differences of the total prices are not greater than ϵ, or the number of grain exchanges is equal to η. Figure A.7 depicts the formation of the four grains that are allocated to cluster number 1 and the mapping process of these grains into workstations.

Initial assignment

Begin

Select a grain t_i with $\text{Max}(P_b^p \cdot E_i + (\sum_j C_{ij}) \cdot P_b^t)$

Assign t_i to $W_{b,x} \in \Gamma_b^s$ with $\text{Min}(P_{b,x}^p \cdot E_i + (\sum_j C_{ij}) \cdot \text{Max}(P_{b,xz}^t))$,

where $W_{b,z} \in \Gamma_b^s$ is an adjacent nodes of $W_{b,x}$

Update processing price of $W_{b,x}$ by applying Eqn. A.2

While there are any grain t_j which have not been assigned

Select t_j adjacent to t_i with $\text{Max}(P_b^p \cdot E_i + C_{ij} \cdot P_b^t)$

Assign t_j to $W_{b,y} \in \Gamma_b^s$ with $\text{Min}(P_{b,y}^p \cdot E_j + C_{ij} \cdot P_{b,xy}^t)$

Update processing prices of $W_{b,y}$ by applying Eqn. A.2

Update link communication prices along $\langle W_{b,x}, W_{b,y} \rangle$

to which t_i and t_j are assigned by applying Eqn. A.3

If all the adjacent grains of t_i have been assigned,

Select the ws to which t_i is assigned as $W_{b,x}$,

and the assigned grain with $\text{Max}(P_b^p \cdot E_i + (\sum_j C_{ij}) \cdot P_b^t)$ as t_i.

End

Dynamic Exchange

Begin

While in the current mapping,

there exists $(Max(\Phi_{i,x}) - Min(\Phi_{j,y}) > \epsilon)$ & (No. of Exchange $< \eta$) s.t.,

$t_i, t_j \in T$ and $W_{b,x}, W_{b,y} \in \Gamma_b^s$,

If $\text{Max}(\Psi_{x,y}) > \epsilon$

Perform task exchange between $W_{b,x}$ and $W_{b,y}$

Update processing prices of the exchanged tasks by applying Eqn. A.2

Update link communication prices along $\langle W_{b,x}, W_{b,y} \rangle$ by applying Eqn. A.3

Else

Termination.

End

Definition of Symbols :

$$\Phi_{i,x} = E_i \cdot P_{b,x}^p + \sum_k (C_{ik} \cdot P_{ik,xz}^v)$$

$$\Psi_{x,y} = (\Phi_{i,x} - \Phi_{j,x}) + (\Phi_{j,y} - \Phi_{i,y})$$

where $\Phi_{i,x}$ is the total price of executing t_i in $W_{b,x}$,

$P_{ik,xz}^v$ is the price on the path $\langle W_{b,x}, W_{b,z} \rangle$, if t_i was assigned to $W_{b,x}$,

t_k is the adjacent grain to t_i,

$\Psi_{x,y}$ is the gain incurred, if the tasks t_i on $W_{b,x}$ and t_j on $W_{b,y}$ are switched.

Figure A.6: Mapping Algorithm

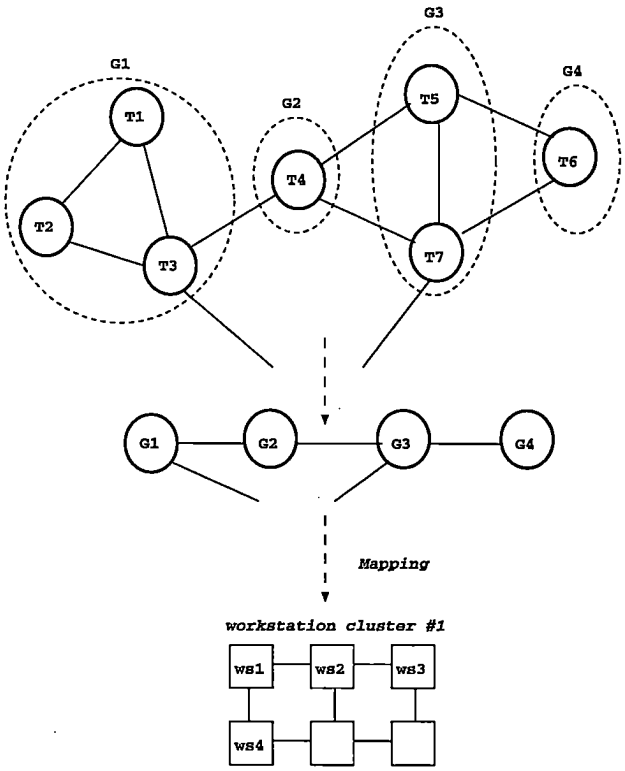


Figure A.7: An Example of the Mapping Algorithm Executed in Cluster #1

A.4 The Complexity of the Algorithms

For the partitioning phase, the function *Cluster_Grain*(b, r_b) of the granulation algorithm will be executed recursively until the values of g_b are all formulated for N workstation clusters. Thus the time complexity of the granulation algorithm is $O(N)$. Since the decoupling force of the merging algorithm has to be calculated between every non-nuclei task and every nucleus, the complexity is bounded by $O(G \cdot N_T)$, where G is the total number of grains and N_T is the total number of tasks (usually $N_T \gg G$).

For the mapping phase, the time complexity of the initial assignment is $O(G)$. To examine the requirement of grain exchange, the calculation of total cost for grain execution demands $O(\eta \cdot G^2)$ number of operations. However, the time complexity of these operations can be reduced to $O(g_b^2)$ if all cluster controllers perform dynamic exchange for mapping grains onto their workstations in parallel.

A.5 Summary

This appendix provides solutions to the problem of partitioning and mapping in a dynamic environment, but with simplification and flexibility provided by the market rules which are often translated into market price. The approach includes two main phases: partitioning and mapping. In each phase, low complexity algorithms are developed to enable fast reiteration. The algorithms could allow dynamic adjustments, effected by the changes in the workstation clusters and/or the applications. All that is required is to incorporate the changes into the new auction and bidding prices of the commodities (computation and communication capacities).

Bibliography

[ABL+95] J. Arabe, A. Beguelin, B. Lowekamp, E. Seligman, M. Starkey, and P. Stephan. Dome: Parallel programming in a heterogeneous multi-user environment. Technical report, CMU-CS-95-137, School of Computer Science, Carnegie Mellon University, 1995. *(Cited on page 14).*

[ACD+96] C. Amza, A.L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel. Treadmarks: Shared memory computing on networks of workstations. *IEEE Computer*, pages 18–28, February 1996. *(Cited on page 10).*

[ACP94] T.E. Anderson, D.E. Culler, and D.A. Patterson. A case for NOW (Networks of Workstations). Technical report, UC Berkeley, 1994. *(Cited on page 7).*

[ADV+95] R.H. Arpaci, A.C. Dusseau, A.M. Vahdat, L.T. Liu, T.E. Anderson, and D.A. Patterson. The interaction of parallel and sequential workloads on a network of workstations. In *Proc. of Sigmetric'95/Performance'95*, pages 267–278, 1995. *(Cited on page 63).*

[AJ88] R. Agrawal and H. V. Jagadish. Partitioning techniques for large-grained parallelism. *IEEE Trans. on Computers*, 37(12):1627–1634, 1988. *(Cited on page 20).*

[AYHI96] D. Andresen, T. Yang, V. Holmedahl, and O.H. Ibarra. SWEB: Towards a scalable world wide web server on multicomputers. In *Proc. of 10th International Parallel Processing Symposium, IEEE IPPS'96*, pages 850–856, 1996. *(Cited on page 6).*

[BB92] E.K. Browning and J.M. Browning. *Microeconomic theory and applications.* New York: Fourth Edition, Harper-Collins Publishers Inc., 1992. *(Cited on page 69).*

[BDG+91] A. Beguelin, J. Dongarra, A. Geist, R. Manchek, and V.S. Sunderam. Solving computational grand challenges using a network of heterogeneous supercomputers. In *Proc. Fifth SIAM Conference on Parallel Processing*, 1991. *(Cited on page 6).*

[BG92] A. Bhimai and S. Ghosh. Modeling and distributed simulation of complex Broadband ISDN networks under overload on loosely-coupled parallel processors. In *Int. Conf. Comm., IEEE Comm. Society*, pages 1280–1284, 1992. *(Cited on page 33).*

[BI87] V.C. Bhavsar and J.R. Issac. Design and analysis of parallel Monte Carlo algorithms. *SIAM Journal on Scientific and Statistical Computing*, 8:s73–s95, 1987. *(Cited on pages 104 and 106).*

[BKM89] M. Bozyigit, U. Kalaycioglu, and M. Melhi. Load balancing in dense distributed systems. In *4th International Symposium on Computer and Information Sciences*, pages 345–361, Cesme, October 1989. *(Cited on page 20).*

[BL92] R. Bulter and E. Lusk. User's guide to the P4 programming system. Technical report, ANL-92/17, Argonne National Laboratory, 1992. *(Cited on page 8).*

[BNG92] N.S. Bowen, C.N. Nikolaou, and A. Ghafoor. On the assignment problem of arbitrary process systems to heterogeneous distributed computer systems. *IEEE Trans. on Computers*, 41(3):257–273, 1992. *(Cited on pages 20,123).*

[Bok81] S.H. Bokhari. A shortest tree algorithm for optimal assignments across space and time in a distributed processor system. *IEEE Trans. on Software Engineering*, 7(6):583–589, 1981. *(Cited on pages 11 and 82).*

[BR89] L. Bomans and D. Roose. Benchmarking the ipsc/2 hypercube multiprocessor. *Concurrency: Practice and Experience*, 1(1):3–18, September 1989. *(Cited on page 51).*

[CCK+95] J. Casas, D. Clark, R. Konuru, S. Otto, R. Prouty, and J. Walpole. MPVM: A migration transparent version of PVM. Technical report, CSE-95-002, Computer Sci. & Eng. Dept., OGI, 1995. *(Cited on pages 14 and 43).*

[CCO+95] D.L. Clark, J. Casas, W. Otto, R.M. Prouty, and J. Walpole. Scheduling of parallel jobs on dynamic, heterogeneous networks. Technical report, OGI, 1995. *(Cited on page 15).*

[CFGK95] N. Carriero, E. Freeman, D. Gelernter, and D. Kaminsky. Adaptive parallelism and Piranha. *IEEE Computer*, January 1995. *(Cited on page 14).*

[CG72] E.G. Coffman and R.L. Graham. Optimal scheduling for two-processor systems. *Acta Informatica*, 1:200–213, 1972. *(Cited on pages 11 and 82).*

[CG89] N. Carriero and D. Gelernter. Linda in context. *Communications of ACM*, 32(4), April 1989. *(Cited on page 10).*

[CG92] N. Carriero and D. Gelernter. Coordination languages and their significance. *Communications of ACM*, 35(2), Feb. 1992. *(Cited on page 9).*

[CG93] A. Chai and S. Ghosh. Modeling and distributed simulation of a Broadband-ISDN network. *IEEE Computer*, 26(9):37–51, September 1993. *(Cited on page 33).*

[CH93] D.R. Cheriton and K. Harty. A market approach to operating system memory allocation. Technical report, URL: http://www-dsg.standford.edu/Publications.html, Standford University, 1993. *(Cited on page 59).*

[CHI] CHIMP, Edinburgh Parallel Computing Center. ftp://ftp.epcc.edu.ac.uk/pub/chimp/release/chimp.tar.z. *(Cited on page 8).*

[CKO+94] J. Casas, R. Konuru, S. W. Otto, R. Prouty, and J. Walpole. Adaptive load migration systems for PVM. In *Proc. of the Supercomputing '94*, 1994. *(Cited on page 43).*

[CL85] K.M. Chandy and L. Lamport. Determining global states of distributed systems. *ACM Trans. on Computer Systems*, 3(1):63–75, Feb. 1985. *(Cited on page 103).*

[Cof74] E.G. Coffman. *Computer and Job-Shop Scheduling Theory.* New York: Wiley, 1974. *(Cited on pages 11 and 82).*

[CS93] M. Calzarossa and G. Serazzi. Workload characterization: A survey. *Proceedings of IEEE*, 81(8):1136–1150, August 1993. *(Cited on page 17).*

[DM88] K.E. Drexler and M.S. Miller. Incentive engineering for computational resource management. In Huberman B.A., editor, *The Ecology of Computation.* Amsterdam:North-Holland, 1988. *(Cited on page 59).*

[DO91] F. Douglis and J. Ousterhout. Transparent process migration: Design alternatives and the Sprite implementation. *Software - Practice and Experience*, 21(8), August 1991. *(Cited on page 63).*

[DOSW96] J.J. Dongarra, S.W. Otto, M. Snir, and D. Walker. A message passing standard for MPP and workstations. *Communications of the ACM*, 39(7):84–90, July 1996. *(Cited on page 9).*

[DPCS95] P.W. Dowd, F.A. Pellegrino, T.M. Carrozzi, and S.M. Srinidhi. Geographically distributed computing: ATM over the NASA ACTS Satellite. In *IEEE MILCOM'95*, October 1995. *(Cited on page 13).*

[DSP+95] P.W. Dowd, S.M. Srinidhi, F.A. Pellegrino, T.M. Carrozzi, D.L. Guglielmi, and R. Claus. Impact of transport protocols and message passing libraries on cluster-based computing performance. In *IEEE MILCOM'95*, October 1995. *(Cited on page 7).*

[EASS95] G. Edjlali, G. Agrawal, A. Sussman, and J. Saltz. Data parallel programming in an adaptive environment. Technical report, CS-TR-3350, University of Maryland Inst. for Advanced Comp. Studies, 1995. *(Cited on page 14).*

[EBBV95] T. Eicken, A. Basu, V. Buch, and W. Vogels. U-Net: A user-level network interface for parallel and distributed computing. In *ACM SIGOPS'95*, pages 40–53, 1995. *(Cited on page 7).*

[Efe82] K. Efe. Heuristic models of task assignment scheduling in distributed systems. *IEEE Computer*, 15(6):50–56, 1982. *(Cited on page 123).*

[FJL+88] G.C. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D.W. Walker. *Solving Problems on Concurrent Processors.* Prentice Hall, 1988. *(Cited on page 18).*

[FK94] J. Flower and A. Kolawa. Express is not just a message passing system: current and future directions in Express. *Parallel Computing*, 20:597–614, 1994. *(Cited on page 8).*

[For91] Fortran 90, international organization for standardization and international electrotechnical commission. [ISO/IEC 1539:1991 (E)], 1991. *(Cited on page 10).*

[For93] High Performance Fortran Forum. High Performance Fortran Language Specification. *Scientific Programming*, 2(1-2):1–170, 1993. *(Cited on page 10).*

[Fox89] G.C. Fox. 1989– the first year of the parallel supercomputer. In *Proc. of the Fourth Conference on Hypercubes, Concurrent Computers, and Applications*, pages 1–37, 1989. *(Cited on page 30).*

[FS93] R.F. Freund and H.J. Siegel. Heterogeneous Processing. *IEEE Computer*, pages 13–17, June 1993. *(Cited on page 7).*

[Fuj90] R. Fujimoto. Parallel discrete event simulation. *Comm. of the ACM*, 33(10):30–53, October 1990. *(Cited on page 103).*

[FYN88] D. Ferguson, Y. Yemini, and C. Nikolaou. Microeconomic algorithms for load balancing in distributed computer systems. In *8th Int. Conf. on Distributed Computing Systems*, pages 491–499, 1988. *(Cited on page 59).*

[GBD+93] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM 3.0 User's Guide and Reference Manual*, Feb. 1993. *(Cited on pages 8 and 44).*

[GC91] Grand challenges: High performance computing. Committee on Physical, Mathematical, and Engineering Sciences, National Science Foundation, Washington, D.C., 1991. *(Cited on page 6).*

[GE76] V.B. Gylys and J.A. Edwards. Optimal partitioning of workload for distributed systems. In *Proc. COMPCON Fall'76*, pages 353–357, 1976. *(Cited on page 123).*

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York: Freeman, 1979. *(Cited on page 11 and 82).*

[GLS94] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI - Portable Parallel Programming with the Message-Passing Interface.* MIT Press, 1994. *(Cited on page 8).*

[Gra77] C.W.J. Granger. *Forecasting Economic Time Series.* Academic Press Inc., 1977. *(Cited on page 64).*

[GS91] T.P. Green and J. Synder. DQS, a distributed queueing system. In *Workshop on Heterogeneous Network-based Concurrent Computing,* October 1991. *(Cited on pages 15 and 61).*

[Hag86] R. Hagmann. Process Server: Sharing processing power in a workstation environment. In *6th International Conference on Distributed Computing Systems,* pages 260–267, 1986. *(Cited on page 6).*

[Hei88] P. Heidelberger. Discrete event simulations and parallel processing: Statistical properties. *SIAM Journal on Scientific and Statistical Computing,* 9(6):1114–1132, November 1988. *(Cited on pages 104 and 106).*

[Hu61] T.C. Hu. Parallel sequencing and assembly line problems. *Oper. Res.,* 9:841–848, November 1961. *(Cited on pages 11 and 82).*

[KBM94] E.D. Katz, M. Bulter, and R. McGrath. A scalable HTTP server. *Computer Networks and ISDN Systems,* 27:155–164, 1994. *(Cited on page 6).*

[KC91] P. Krueger and R. Chawla. The stealth distributed scheduler. In *11th International Conf. on Distributed Computing Systems,* pages 336–343, 1991. *(Cited on page 5).*

[KCO+94] R. Konuru, J. Casas, S. W. Otto, R. Prouty, and J. Walpole. A user-level process package for PVM. In *Proc. of 1994 Scalable High-Performance Computing Conference,* Knoxville,TN, 1994. *(Cited on pages 14 and 43).*

[KR95] M. Kaddoura and S. Ranka. Runtime support for parallelization of data-parallel applications on adaptive and nonuniform computational environments. Technical report, School of Computer and Information Science, Syracuse University, 1995. *(Cited on page 14).*

[KRS96] F. Knop, V. Rego, and V. Sunderam. Fail-safe concurrency in the EcliPse system. *Concurrency: Practice and Experience,* 8(4):283–312, May 1996. *(Cited on page 100).*

[KS86] J.F. Kurose and R. Simha. A microeconomic approach to optimal file allocation. In *6th Int. Conf. on Distributed Computing Systems*, pages 28–35, 1986. *(Cited on page 59).*

[KSY85] J.F. Kurose, M. Schwartz, and Y. Yemini. A microeconomic approach to optimization of channel access policies in multiaccess networks. In *5th Int. Conf. on Distributed Computing Systems*, pages 70–77, 1985. *(Cited on page 59).*

[Lam78] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Comm. of the ACM*, 21(7):558–564, July 1978. *(Cited on page 35).*

[LC96] M. Lauria and A. Chien. MPI-FM: High performance mpi on workstation clusters. Technical report, Dept. of Computer Science, University of Illinois at Urbana-Champaign, 1996. *(Cited on page 7).*

[LFS93] J. Leon, A.L. Fisher, and P. Steenkiste. Fail-safe PVM: A portable package for distributed programming with transparent recovery. Technical report, Tech. Rep. CMU-CS-93-124, School of Computer Science, Carnegie Mellon University, 1993. *(Cited on page 100).*

[LH95] C.K. Lee and M. Hamdi. Parallel image processing on a network of workstations. *Parallel Computing*, 21:137–160, 1995. *(Cited on page 6).*

[LHD+94] M. Lin, J. Hsieh, D. Du, J. Thomas, and J. MacDonald. Distributed network computing over local ATM networks. Technical report, Tech. Rep. TR-94-17, Computer Science Department, University of Minnesota, 1994. *(Cited on page 7).*

[Li90] S.-Y.R. Li. Algorithms for flow control and call set-up in multi-hop Broadband ISDN. In *Proc. INFOCOM*, June 1990. *(Cited on pages 35 and 33).*

[Lin94] Y.-B. Lin. Parallel independent replicated simulation on a network of workstations. In *Proc. of 8th Workshop on Parallel and Distributed Simulation*, pages 73–80, 1994. *(Cited on pages 104 and 106).*

[LK78] J.K. Lenstra and A.H.G.R. Kan. Complexity of scheduling under precedence constraints. *Oper. Res.*, 26:22–35, January 1978. *(Cited on pages 11 and 82).*

[LLM88] M.J. Litzkow, M. Livny, and M.W. Mutka. Condor - A Hunter of idle workstations. In *8th Int. Conf. on Distributed Comp. Systems*, pages 104–111, 1988. *(Cited on pages 6, 15, and 61).*

[LNP91] K. Li, J.F. Naughton, and J.S. Plank. Checkpointing multicomputer applications. In *Proc. of the Tenth Sym. on Reliable Distributed Systems*, pages 2–11, Sept.. 1991. *(Cited on page 103).*

[Lo88] V.M. Lo. Heuristic algorithms for task assignment in distributed systems. *IEEE Trans. on Computers*, 37(11):1384–1397, 1988. *(Cited on page 127).*

[LS92] M.J. Litzkow and M. Solomon. Supporting checkpointing and process migration outside the Unix kernel. In *Proc. of the Winter Usenix Conference*, 1992. *(Cited on page 102).*

[LS94] S.-Y.R. Li and K.H. Shum. Distributed algorithms for flow control & call set-up in a broadband packet network. In *Proc. of 12th European Fibre Optic Communications and Networks*, pages 182–186, 1994. *(Cited on pages 35 and 33).*

[Lub89] B. D. Lubachevsky. Efficient distributed event-driven simulations of multiple-loop networks. *Comm. ACM*, 32(1):111–123, January 1989. *(Cited on page 35).*

[MA86] C.E. McDowell and W.F. Applebe. Processor scheduling for linearly connected parallel processors. *IEEE Trans. Computer*, 35(7):632–638, July 1986. *(Cited on page 11).*

[MFGH88] T.W. Malone, R.E. Fikes, K.R. Grant, and M.T. Howard. Enterprise: a market-like task scheduler for distributed computing environments. In *The Ecology of Computation.* Amsterdam: North-Holland, 1988. *(Cited on page 59).*

[MIC96] P.G. Meisl, M.R. Ito, and I.G. Cumming. Parallel synthetic aperture radar processing on workstation networks. In *Proc. of 10th International Parallel Processing Symposium, IEEE IPPS'96*, pages 716–723, 1996. *(Cited on page 6).*

[MJS+94] R.L. Martino, C.A. Johnson, E.B. Suh, B.L. Trus, and T.K. Yap. Parallel computing in biomedical research. *Science*, 265:902–908, August 1994. *(Cited on page 13).*

[ML87] M.W. Mutka and M. Livny. Profiling workstations' available capacity for remote execution. In *12th IFIP WG &.3 Symposium on Computer Performance, Brussels, Belgium*, 1987. *(Cited on page 5).*

[ML91] M.W. Mutka and M. Livny. The available capacity of a privately owned workstation environment. *Performance Evaluation*, 12(4):269–84, July 1991. *(Cited on page 63).*

[MPS89] S. Mishra, L.L. Peterson, and R.D. Schlichting. Implementing fault-tolerant replicated objects using Psync. In *Proc. of the 8th Sym. on Reliable Distributed Systems*, 1989. *(Cited on page 100).*

[Mut92] M.W. Mutka. Estimating capacity for sharing in a privately owned workstation environment. *IEEE Trans. on Software Engineering*, 18(4):319–328, April 1992. *(Cited on page 64).*

[NF94] D. Nicol and R. Fujimoto. Parallel simulation today. *Annals of Operations Research*, 53:249–286, December 1994. *(Cited on page 103).*

[Ng90] T.P. Ng. Design and implementation of a reliable distributed operating system - Rose. In *Proc. of the 9th Sym. on Reliable Distributed Systems*, 1990. *(Cited on page 100).*

[Nic87] D. A. Nichols. Using idle workstations in a shared computing environment. In *Eleventh ACM Symposium on Operating Systems Principles*, pages 5–12, 1987. *(Cited on page 6).*

[Nic88] D. Nicol. Parallel discrete-event simulation of FCFS stochastic queuing network. *SIGPLAN Notice*, 23(9):124–137, September 1988. *(Cited on page 35).*

[NQ93] N. Nedeljkovic and M.J. Quinn. Data-parallel programming on a network of heterogeneous workstations. *Concurrency: Practice & Experience*, 5(4):257–268, June 1993. *(Cited on page 14).*

[NR94] B.C. Neuman and S. Rao. The Prospero Resource Manager: A scalable framework for processor allocation in distributed systems. *Concurrency: Practice and Experience*, 6(4):339–355, June 1994. *(Cited on pages 15 and 61).*

[NRS92] H. Nakanishi, V. Rego, and V. Sunderam. Superconcurrent simulation of polymer chains on heterogeneous networks. In *Proc. of*

the Fifth High-Performance Computing and Communications Conference: Supercomputing'92, 1992. *(Cited on page 7).*

[NS88] D. Nicol and J. H. Saltz. Dynamic remapping of parallel computations with varying resource demands. *IEEE Trans. on Computer*, 37(9), September 1988. *(Cited on pages 12).*

[NS92] H. Nakanishi and V.S. Sunderam. Superconcurrent simulation of polymer chains on heterogeneous networks. In *Proc. IEEE Supercomputing Symposium*, 1992. *(Cited on page 6).*

[NT93] M.G. Norman and P. Thanisch. Models of machines and computation for mapping in multicomputers. *ACM Computing Surveys*, 25(3):263–302, 1993. *(Cited on page 11).*

[OL88] B.M. Oki and B.H. Liskov. Viewstamped replication: A new primary copy method to support highly-available distributed systems. In *Proc. of the 7th ACM Sym. on Principles of Distributed Computing*, 1988. *(Cited on page 100).*

[PC91] C. I. Phillips and L. G. Cuthbert. Concurrent discrete event-driven simulation tools. *IEEE J. Select. Areas in Comm.*, April 1991. *(Cited on page 33).*

[Pfi95] G. F. Pfister. *In Search of Clusters: The Coming Battle in Lowly Parallel Computing.* New Jersey: Prentice Hall PTR, 1995. *(Cited on page 104).*

[PL95] J. Pruyne and M. Livny. Parallel processing on dynamic resources with CARMI. In *Proc. of Workshop on Job Scheduling Strategies for Parallel Processing, IEEE IPPS'95*, 1995. *(Cited on page 15).*

[PL96] J. Pruyne and M. Livny. Managing checkpoints for parallel programs. In *Proc. of Workshop on Job Scheduling Strategies for Parallel Processing, IEEE IPPS'96*, 1996. *(Cited on page 100).*

[POW94] R. Prouty, S. Otto, and J. Walpole. Adaptive execution of data parallel computations on networks of heterogeneous workstations. Technical report, CSE-94-012, Computer Sci. & Eng. Dept., OGI, 1994. *(Cited on pages 14 and 43).*

[PWA+93] D.H. Porter, P.R. Woodward, S. Anderson, J. MacDonald, K. Chin-Purcell, R. Hessel, D. Perro, I. Zacharov, J. Ryan, L. Wildra, and

M. Galles. Attacking a grand challenge in computational fluid dynamics on a cluster of silicon graphics challenge machines. In *Proc. of Cluster Workshop*, 1993. *(Cited on page 6).*

[Qui94] M.J. Quinn. *Parallel Computing: Theory and Practice.* McGraw-Hill International, 1994. *(Cited on page 12).*

[Sah84] S. Sahni. Scheduling multipipeline and multiprocessor computers. *IEEE Trans. Computer*, 33(7):637–645, July 1984. *(Cited on page 11).*

[SB94] K.H. Shum and M. Bozyigit. A load distribution through competition for workstation clusters. In *Proc. of the Ninth International Symposium on Computer and Information Sciences*, pages 810–817, Antalya, November 1994. *(Cited on page 20).*

[SCMS82] J.A. Stankovic, N. Chowdhury, R. Mirchandaney, and I. Sidhu. An evaluation of the applicability of different mathematical approaches to the analysis of decentralized control algorithms. In *IEEE COMPSAC*, pages 62–69, 1982. *(Cited on page 11).*

[Shu95] K.H. Shum. Adaptive distributed simulation for computationally intensive modelling. In M. Merabti et al., editor, *Proc. of the 11th U.K. Performance Engineering Workshop for Computer and Telecommunication Systems.* Springer-Verlag, 1995. *(Cited on page 33).*

[Shu96] K.H. Shum. Adaptive distributed computing through competition. In *Proc. of the International Conference on Configurable Distributed Systems*, Maryland, May 1996. IEEE Computer Society. *(Cited on page 61).*

[SL96] K.H. Shum and S.-Y.R. Li. Runtime support for replicated parallel simulators on workstation clusters. In *Proc. of Euro-Par'96, Lecture Notes in Compter Science*, Lyon, August 1996. Springer-Verlag. *(Cited on page 105).*

[SM96] K.H. Shum and K. Moody. A competitive environment for parallel applications on heterogeneous workstation clusters. In *Proc. of the Heterogeneous Computing Workshop (HCW'96), IEEE IPPS'96, presented in the joint session with the 2nd Workshop on Job Scheduling Strategies for Parallel Processing*, Hawaii, April 1996. *(Cited on page 81).*

[SNM+95] P. Stolorz, H. Nakamura, E. Mesrobian, R.R. Muntz, E.C. Shek, J.R. Santos, J.Yi, K. Ng, S.-Y. Chien, C.R. Mechoso, and J.D. Farrara. Fast spatio-temporal data mining of large geophysical datasets. In *Proc. of the First International Conference on Knowlege Discovery and Data Mining, AAAI Press*, pages 300–305, 1995. *(Cited on page 6).*

[SOW94] K.A. Saqabi, S.W. Otto, and J. Walpole. Gang scheduling in heterogeneous distributed systems. Technical report, OGI, 1994. *(Cited on pages 15 and 61).*

[SS84] J.A. Stankovic and I.S. Sidhu. An adaptive bidding algorithm of processes, clusters, and distributed groups. In *Int. Conf. Distributed Comput. Syst.*, pages 49–59, May 1984. *(Cited on page 11).*

[SS94a] B.K. Schmidt and V.S. Sunderam. Empirical analysis of overheads in cluster environments. *Concurrency: Practice and Experience*, 6(1):1–32, February 1994. *(Cited on page 51).*

[SS94b] M. Singhal and N.G. Shivaratri. *Advanced Concepts in Operating Systems.* New York: McGraw-Hill, 1994. *(Cited on page 100).*

[Sto77] H.S. Stone. Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Trans. on Software Engineering*, 3(1):85–93, 1977. *(Cited on pages 11 and 82).*

[Str95] V. Strumpen. Coupling hundreds of workstations for parallel molecular sequence analysis. *Software-Practice and Experience*, 25(3):291–304, March 1995. *(Cited on page 13).*

[THM+94] C. Trefftz, C.C. Huang, P.K. McKinley, T.Y. Li, and Z. Zeng. Design and performance evaluation of a distributed eigenvalue solver on a workstation cluster. In *Proc. of 14th International Conference on Distributed Computing Systems*, pages 608–615, 1994. *(Cited on page 6).*

[TNC94] S.W. Turner, L.M. Ni, and B.H.C. Cheng. Time and/or space sharing in a workstation cluster environment. In *Proc. of Supercomputing'94*, pages 630–639, 1994. *(Cited on page 58).*

[Tur93] L.H. Turcotte. A survey of software environments for exploiting networked computing resources. Technical report, MSSU-EIRS-ERC-

932. Engineering Research Center, Mississippi State University, 1993. *(Cited on page 8).*

[Var92] H.R. Varian. *Microeconomic Analysis.* Third Edition, New York: Norton International Student Edition, 1992. *(Cited on page 73).*

[WHH+92] C.A. Waldspurger, T. Hogg, B.A. Huberman, J.O. Kephart, and W.S. Stornetta. Spawn: A distributed computational economy. *IEEE Trans. on Software Engineering*, 18(2):103–117, February 1992. *(Cited on pages 15, 59, and 60).*

[Wil79] R. Wilson. Auctions of shares. *Quarterly Journal of Economics*, 93:675–89, 1979. *(Cited on page 69).*

[WM85] Y.-T. Wang and R. J. T. Morris. Load sharing in distributed systems. *IEEE Trans. Computers*, C-34, March 1985. *(Cited on page 35).*

[YKF91] T. Yokoi, N. Kishimoto, and Y. Fujii. ATM network performance evaluation using parallel and distributed simulation technique. In *Proc. of the Teletraffic and Datatraffic, ITC-13*, pages 815–820, 1991. *(Cited on page 33).*