INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

Impact of Disk Access on the Performance of Load Sharing Policies in Distributed Systems

By

Jie Zhang, B.C.S.

A thesis submitted to
The Faculty of Graduate Studies and Research
In partial fulfillment of the requirements for the degree of

Master of Computer Science

Ottawa-Carleton Institute for Computer Science

School of Computer Science
Carleton University, Ottawa, Ontario
December 15, 1998

© copyright
1998, Jie Zhang

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-36882-3

The undersigned recommend to the Faculty of Graduate Studies
and Research acceptance of the thesis

“Impact of Disk Access on the Performance of Load Sharing Policies in Distributed Systems”

Submitted by

Jie Zhang, B.C.S.

in partial fulfillment of the requirements for
the degree of Master of Computer Science

Thesis Supervisor

Director, School of Computer

Carleton University
December 15, 1998

ABSTRACT

Load Sharing for distributed systems has been an active research area for many years. Many research efforts have demonstrated that load sharing algorithms produce considerable performance improvement. While most of the previous research provides considerable insight into various aspects of load sharing, the job I/O requirement and its effects on load sharing had not been investigated in great detail. This thesis is an attempt to fill the void in the existing literature. It studies the performance characteristics of three load sharing policies in two types of distributed systems.

The results show that job's I/O requirements have strong impact on system performance for all the policies. However, the performance difference between two I/O models is not noticeable unless the system load is high. Other factors, such as variance in job inter-arrival time, processor service time and the percentage of I/O jobs, etc., have great influence on system performance as well. Stable symmetrically initiated policy provides the best performance among the three policies. Global and local hierarchical policies perform similarly.

ACKNOWLEDGEMENT

I gratefully acknowledge the valuable assistance of my thesis supervisor, Dr. Sivarama P. Dandamudi, whose continued interest and guidance helped to improve my thesis quality. In addition, I express appreciation to all those who helped and supported me in the development of my thesis. In particular, to my family for their ongoing support.

TABLE OF CONTENT

ABSTRACT	III
ACKNOWLEDGEMENT	IV
TABLE OF CONTENT	V
LIST OF FIGURES.....	VIII
1 INTRODUCTION	1
1.1 INTRODUCTION	1
1.2 MOTIVATION.....	3
1.3 CATEGORIES OF LOAD SHARING POLICIES	5
1.4 LOAD SHARING VERSUS LOAD BALANCING.....	7
1.5 LOAD INDEX	7
1.6 SCHEDULING POLICIES	8
1.7 COMPONENTS OF LOAD SHARING POLICIES	9
1.7.1 <i>Transfer policy</i>	9
1.7.2 <i>Selection policy</i>	10
1.7.3 <i>Location policy</i>	10
1.7.4 <i>Information policy</i>	11
1.8 CONTRIBUTIONS OF THE THESIS	12
1.9 THESIS ORGANIZATION	13
2 RELATED WORK.....	14
2.1 INTRODUCTION	14

2.2	STATE BROADCAST ALGORITHM.....	14
2.3	BROADCAST IDLE ALGORITHM.....	16
2.4	POLL WHEN IDLE ALGORITHM.....	17
2.5	SINGLE COORDINATOR ALGORITHM	18
2.6	SENDER INITIATED ALGORITHM.....	19
2.6.1	<i>Random Sender Initiated</i>	20
2.6.2	<i>Threshold Sender Initiated</i>	21
2.6.3	<i>Shortest Sender Initiated</i>	22
2.7	RECEIVER INITIATED ALGORITHM	22
2.8	HIERARCHICAL RECEIVER INITIATED ALGORITHM.....	24
2.9	STABLE SYMMETRICALLY INITIATED ADAPTIVE ALGORITHM	28
2.10	STABLE SENDER INITIATED ADAPTIVE ALGORITHM.....	32
2.11	IMPACT OF JOB'S I/O ON TWO BASIC ALGORITHMS	33
3	POLICIES AND MODELS	35
3.1	INTRODUCTION	35
3.2	LOAD SHARING POLICIES	36
3.3	SYSTEM MODEL.....	38
3.4	WORKLOAD MODEL	40
3.5	I/O MODEL.....	40
4	PERFORMANCE IN HOMOGENEOUS DISTRIBUTED SYSTEMS	43
4.1	INTRODUCTION	43
4.2	PERFORMANCE ANALYSIS	44
4.2.1	<i>Performance as a function of System Load</i>	44
4.2.2	<i>Sensitivity to Variance in Inter-Arrival Times</i>	51
4.2.3	<i>Sensitivity to variance in processor service times</i>	54

4.2.4	<i>Sensitivity to variance in disk service times</i>	56
4.2.5	<i>Sensitivity to disk jobs</i>	58
4.2.6	<i>Sensitivity to probe limit</i>	61
4.3	SUMMARY	63
5	PERFORMANCE IN HETEROGENEOUS DISTRIBUTED SYSTEMS	65
5.1	INTRODUCTION	65
5.2	PERFORMANCE ANALYSIS OF TYPE I SYSTEMS.....	66
5.2.1	<i>Performance as a function of System Load</i>	67
5.2.2	<i>Sensitivity to variance in inter-arrival times</i>	72
5.2.3	<i>Sensitivity to variance in processor service times</i>	74
5.2.4	<i>Sensitivity to variance in disk service times</i>	76
5.2.5	<i>Sensitivity to disk jobs</i>	78
5.2.6	<i>Sensitivity to probe limit</i>	80
5.2.7	<i>Sensitivity to the degree of heterogeneity</i>	82
5.3	PERFORMANCE ANALYSIS OF TYPE II SYSTEMS	84
5.3.1	<i>Performance as a function of System Load</i>	85
5.3.2	<i>Sensitivity to the degree of heterogeneity</i>	87
5.4	SUMMARY	89
6	CONCLUSIONS	90
6.1	SUMMARY	90
6.2	FUTURE WORK.....	91
7	REFERENCES	92

LIST OF FIGURES

Figure 1.1 A locally distributed computer system.....	3
Figure 2.1 An example of a local hierarchical organization for an 8-node system ($B = 2$).	25
Figure 2.2 An example of a global hierarchical organization for an 8-node system ($B = 2$).	26
Figure 3.1 Two I/O models: (a) Non-interleaved (b) Interleaved.....	41
Figure 4.1 Performance sensitivity as a function of system load	45
Figure 4.2 Performance sensitivity as a function of system load by policy	45
Figure 4.3 Performance sensitivity as a function of system load	49
Figure 4.4 Performance sensitivity as a function of system load by policy	49
Figure 4.5 Performance sensitivity to inter-arrival time CV	52
Figure 4.6 Performance sensitivity to inter-arrival time CV by policy	53
Figure 4.7 Performance sensitivity to process service time CV	55
Figure 4.8 Performance sensitivity to inter-arrival time CV by policy	55
Figure 4.9 Performance sensitivity to disk service time CV	57
Figure 4.10 Performance sensitivity to disk service time CV by policy	57
Figure 4.11 Performance sensitivity to disk jobs.....	60
Figure 4.12 Performance sensitivity to disk jobs by policy.....	60
Figure 4.13 Performance sensitivity to the probe limit	62
Figure 5.1 Performance sensitivity as a function of system load	68
Figure 5.2 Performance sensitivity as a function of system load by policy	68
Figure 5.3 Performance sensitivity as a function of system load	70
Figure 5.4 Performance sensitivity as a function of system load by policy	71
Figure 5.5 Performance sensitivity to inter-arrival CV.	73

Figure 5.6 Performance sensitivity to inter-arrival CV by policy	73
Figure 5.7 Performance sensitivity to service time CV.....	75
Figure 5.8 Performance sensitivity to service time CV by policy	75
Figure 5.9 Performance sensitivity to disk service time CV.	77
Figure 5.10 Performance sensitivity to disk service time CV by policy	77
Figure 5.11 Performance sensitivity to disk jobs.....	79
Figure 5.12 Performance sensitivity to disk jobs by policy.....	79
Figure 5.13 Performance sensitivity to probe limit	81
Figure 5.14 Performance sensitivity to the degree of heterogeneity	83
Figure 5.15 Performance sensitivity to the degree of heterogeneity by policy	83
Figure 5.16 Performance sensitivity to offered system load	86
Figure 5.17 Performance sensitivity to offered system load by policy.....	86
Figure 5.18 Performance sensitivity to the degree of heterogeneity	88
Figure 5.19 Performance sensitivity to the degree of heterogeneity by policy	88

Chapter 1

Introduction

1.1 Introduction

The confluence of low-cost processor (e.g., microprocessor) and interconnect (e.g., local area network) technologies has spurred a great interest in locally distributed computer architectures, *locally distributed system*. The often-cited advantages of these systems include high performance, availability, and extensibility at low cost. However, in such locally distributed systems, each computing site, also called *node*, receives tasks in the form of jobs from external sources. These jobs are then processed there, and the outputs are returned to the external world. Little sharing of computational resources is provided. Therefore, the random arrival of jobs in such an environment can cause some nodes to be heavily loaded while others are idle or lightly loaded, resulting in poor overall system performance. Such occurrences are termed as *load imbalances*. Even in a homogeneous distributed system where all nodes are identical, at least one computer is likely to be idle while other computers are heavily loaded due to the statistical fluctuations in the arrival of jobs to computers and task-service-time requirements [18].

The purpose of *load sharing* is to improve the performance by appropriately redistributing the workload among the nodes. Load sharing is an important technique for performance enhancement in the design of high-performance distributed computing systems, and has been implemented in several experimental systems such as V [2], CONDOR [17] etc. Many studies, including [8], [13], and [18], have shown that load sharing has considerable potential to improve performance.

The dynamic load sharing policies differ in how they are implemented. Policies that store the state information in a single node and share that information with other nodes are called *centralized* policies. Policies that distribute state information across the nodes in the system are called *distributed* policies. Centralized policies have diminished fault-tolerance and potential bottleneck issues (although it may not be an issue in reasonably large distributed systems [25]), while distributed policies may cause performance problems in term of transmission.

The remainder of this chapter provides the brief overview of load sharing policies. It first describes the motivation for load sharing. Then, it presents different categories popularly defined in the load sharing policies. It also introduces load indexes in the design of load sharing algorithm¹. Finally, it discusses the components of load sharing policies.

¹ Terminology “algorithm” and “policy” are used interchangeably in this thesis.

1.2 Motivation

A locally distributed computing system consists of a collection of computing sites, also called *nodes*, connected by a local area communication network. Jobs generated at each node are processed locally. Let's consider a locally distributed system with the configuration shown in Figure 1.1. The random arrival of jobs in such a system can cause some nodes to be heavily loaded while others are lightly loaded, resulting in poor overall system performance.

Figure 1.1 A locally distributed computer system.

The situation shown in Figure 1.1 is likely to happen often when the computers differ in their processing power or workload generation. However, Livny and Melman [18] have shown that even in systems with homogeneous nodes, statistical fluctuation in

arrival and service patterns can cause situation in which jobs are waiting for service at some nodes while others are simultaneously idle. Such occurrences are termed as *load imbalances*. The overall system performance can potentially be improved by appropriate transfers of jobs from busy nodes to idle or lightly loaded nodes. This procedure is called *load sharing*. Many research efforts have demonstrated that load sharing is an important technique for performance enhancement in locally distributed systems, even a simple load sharing algorithm can produce considerable performance improvement [3], [4], [6], [8], [9], [13], [18], [21], [23], [24].

However, almost all previous studies have neglected the I/O requirements of jobs in evaluating the performance of various load sharing policies. In order to answer the question: is I/O behaviour of jobs important in load sharing, the impact of job's disk service demand needs to be investigated while evaluating different load sharing policies. Dandamudi and Hadavi had studied job's I/O impact on two basic load sharing policies [5]. The purpose of this thesis is to study the impact of job's I/O requirements on other load sharing policies. In this thesis, Local Hierarchical Receiver Initiated policy, Global Hierarchical Receiver Initiated policy and Stable Symmetrically Initiated policy are selected for further performance comparison, as they have been shown to provide better performance in previous studies [6], [7], [19], [23], [24]. Hierarchical policies minimize the drawbacks associated with the distributed and centralized policies and retain their advantages and stable symmetrical policy benefits from its two probing components.

1.3 Categories of Load Sharing Policies

Load sharing policies can be broadly characterized as *static*, *dynamic* or *adaptive*. Static policy uses only information about the average behaviour of the system with no knowledge about the current state. Some examples for static load sharing policies are: *Random Splitting* [22], each node distributes jobs randomly and uniformly to one of the N nodes with the probability $1/N$, where N is the number of nodes in the system. *Cyclic Splitting* [27], [1], each node assigns its i th arriving job to the node $i \bmod N$. Static load sharing policies are attractive because of their simplicity since there is no need to collect system state information. However, it is clear that the potential of static policies is limited by the fact that they do not react to the system state change.

On the other hand, dynamic load sharing policies make their load distribution decisions by reacting to the system state changes dynamically. Therefore, dynamic load sharing policies tend to provide significant performance improvements over the static policies. Some examples for dynamic load sharing policies are: *Sender-initiated*, heavily loaded (local job queue length greater than threshold T) nodes randomly probes up to a maximum of probe limit P selected nodes to locate lightly loaded (local job queue length less than threshold T) nodes before transferring any job. *Receiver-initiated*, lightly loaded nodes search for heavily loaded nodes under the maximum probe limit P . If such a node is found, a job from remote node is transferred to this node. The attraction of dynamic policies is that they respond to system state, and so are better able to avoid bad transfer decision with unnecessary poor performance. They collect the considerable state information and try to make the “best” choice based on that information. One example is

the policy proposed by Krueger and Finkel [12]. It attempts to keep the queue length at each node near the system average queue length². However, since dynamic policies have to collect, store and analyze state information, they are necessarily more complex and incur more overhead than static policies. Fortunately, the research presented by Eager, Lazowska, and Zahorjan [8] has shown that extremely simple dynamic load sharing policies with very small amounts of overhead yield dramatic performance improvements.

Adaptive load sharing policies are a special class of dynamic policies. They adapt their activities by changing the load sharing policies or parameters of the policy based on the system state information. For example, typical adaptive policies might use the sender-initiated policy at low to moderate systems loads and the receiver-initiated policy at high system loads [23].

Due to the advantage of the dynamic load sharing policies³, most recent load sharing research has focused on dynamic policies. Therefore, the remainder of this thesis will concentrate on them.

² An implicit assumption of most proposed policies is that it attempts to balance the queue lengths at each node. In fact, such balancing is not required.

³ Dynamic load sharing policies studied in this thesis include adaptive load sharing policies.

1.4 Load Sharing versus Load Balancing

Load sharing has also been referred to as *Load balancing*. As we know, the purpose of load sharing is to improve system performance by appropriately redistributing workload among nodes. Hence, to obtain good performance, it would appear to be desirable to balance load among nodes in order to prevent the situation that one node is congested while another is lightly loaded.

The goal of load sharing algorithm is to maximize the rate at which a distributed system performs work when work is available. Load balancing algorithm goes one step beyond load sharing by attempting to equalize the loads at all nodes. Consequently, it may potentially reduce the mean and standard deviation of job response time compare with load sharing algorithm [13]. In addition, since load balancing always requires higher transfer rate than load sharing, the higher overhead incurred may potentially outweigh its performance improvement.

1.5 Load Index

Load index is a very important notion in the design of the dynamic load sharing policies. It is the performance measurement of a job at a particular node. Widely studied and used load indexes are: CPU queue length⁴, the average CPU queue length over some period, the amount of free memory, the context-switch rate, the system call rate, and CPU

⁴ “CPU queue length” and “Job queue length” are used interchangeably in this thesis.

utilization. It has been recognized that there are significant differences in the effectiveness of those load indexes, in particular, simple load indexes are especially effective. Kunz [15] found that performance of the load sharing policy can be considerably affected by the choice of load index and the most effective index is the CPU queue length.

1.6 Scheduling Policies

Two popularly used scheduling strategies are *preemptive* strategies and *non-preemptive* strategies. Non-preemptive strategies transfer⁵ only newly received jobs and hence do not require transferring the job's state. Examples in this category are: *First-Come-First-Serve (FCFS)*, and *Smallest Job First (SJF)*. Preemptive strategies involve transferring partially executed job. For example, *Round Robin (RR)* scheduling policy is a preemptive policy. This operation usually is very expensive since it is often difficult to collect a job's state which typically includes virtual memory image, process control block, unread I/O buffers and messages, file pointers, etc. On the other hand, some researches [10], [14] have shown that such job migration is strictly not necessary except in some extreme cases.

⁵ "Transfer" and "schedule" are used interchangeably in this thesis.

1.7 Components of Load Sharing Policies

Typically there are four components in the dynamic load sharing policies. They are a *transfer policy*, a *selection policy*, a *location policy*, and an *information policy*. Transfer policy and location policy are the most important policies among them.

1.7.1 Transfer policy

Transfer policy determines whether to process a job locally or remotely. Popular used transfer policies are *threshold* policy [18], [9], [8], [3]. The load index, job queue length, is used to express thresholds.

When a new job originates at a node, the transfer policy decides that the node is in one of following three states:

- *Receiver* state: if the job queue length of the node falls below a threshold T_l (lower bound threshold);
- *Sender* state: if the job queue length of the node exceeds a threshold T_h (upper bound threshold);
- *OK* state: if the job queue length of the node is in between T_l and T_h .

where $T_h \geq T_l$.

Threshold is a fundamental parameter in the load sharing algorithms and has considerable impact on the performance [15]. In general, when the threshold value is increased, the scope of the load sharing is decreased. When threshold $T_l = T_h = 1$, it

implies that whenever a job arrivals, if the system is busy, it would result in initiating load distribution activity. When threshold $T_l = T_h = \infty$, it implies that the system will never have load sharing activities since it will never reach sender or even OK state. Eager, Lazowska and Zahorjan's research [8] has shown that low thresholds are appropriate for low system loads; high system loads demand higher thresholds.

1.7.2 Selection policy

Selection policy is responsible for selecting a job for transfer after the transfer policy was determined that node is a sender. There are many factors that a selection policy have to consider when it selects a job, such as transfer overhead, job's lifetime and job system calls, etc. The simplest and most effective way to select a qualified job is to pick one of the newly originated jobs that caused the node to become a sender. Because the transfer involved in this approach is non-preemptive, such a job is relatively cheap to transfer.

1.7.3 Location policy

The location policy determines to which node a job selected for transfer should be sent. They have been widely implemented as either *centralized* or *distributed*. In the centralized policy, state information is stored in a single node called *coordinator*. Other nodes in the system have to contact this coordinator node to locate a suitable node for load sharing. The system state is collected by the information policy and used by the transfer policy. In the distributed policy, state information is distributed to all nodes in the

system. One approach to find a suitable node is through *polling*: A node polls another node on a random basis [9], [8] to find out whether it is suitable for load sharing. An alternative to polling is to broadcast a query seeking any node available for load sharing.

The centralized policies are potentially less reliable and uneasily remedied. It is obvious that the failure of a central coordinator may cause the load sharing activity to fail. In addition, the coordinator may easily become a bottleneck, limiting load sharing. It should be noted that some studies have shown that such a coordinator need not be a performance bottleneck for reasonably large distributed systems [25].

While the distributed policy eliminates these disadvantages, it may introduce more overheads as the state information may have to be transmitted across all the nodes in the system. Fortunately, previous studies had shown that those overheads can be substantially reduced by sampling only a few randomly selected nodes [8], [9]. The proposed hierarchical load sharing policy combines the merits of both centralized and distributed policies while minimizing the weakness of these policies [6].

1.7.4 Information policy

The information policy determines when the information should be collected and from where it is to be collected. Based on different timing, the information policy usually can be classified as *Demand-driven policies*, *Periodic policies* and *State-change-driven policies*.

Under demand-driven policies, a node collects the state information of other nodes only when it becomes either a sender or a receiver. Demand-driven policies may be

sender, receiver or symmetrically initiated and are innately dynamic since their actions depend on the system state.

Periodic policies collect state information periodically without adapting their rate of activity to the system state. They can be either centralized or distributed. Overheads incurred during periodic information collection at high system loads could continue to increase the system load and thus worsen the situation.

Nodes using state-change-driven policy broadcast information about their states whenever their states change by a certain degree. The state-change-driven policies differ from the demand-driven policies in that sending state information to the coordinator in centralized system or peers in distributed system rather than collect information about other nodes.

1.8 Contributions of The Thesis

The major contributions of the thesis are in Chapters 4 and 5. Through different experimental results presented in those two chapters, we will find that job's I/O requirement has a strong performance impact on the studied load sharing policies in two types of distributed systems. Overheads associated with job I/O service greatly deteriorate the performance of the interleaved I/O model as well as the non-interleaved I/O model.

The performance difference between pure CPU workload and the two I/O workloads is sensitive to several parameters, such as system load, inter-arrival CV,

service time CV, and heterogeneity, etc. The more jobs require disk service, the poorer the system performance.

Stable symmetrically initiated policy exhibits the best performance among the three policies, which is due to its two probing components and judicious use of the knowledge gained through probing. The poor performance of two hierarchical policies is because of their relatively inefficient hierarchical search and more probing overheads. Local hierarchical policy suffers in performance due to the delay of load sharing activities at low to moderate system loads. However, this delay becomes less critical when system load is high.

1.9 Thesis Organization

The remainder of this thesis is organized as follows: Chapter 2 introduces several representative dynamic load sharing algorithms. Chapter 3 describes the policies and models used in the experiments. Performance comparison of load sharing policies in homogeneous system is presented in Chapter 4. Chapter 5 provides the performance analysis of load sharing policies in both Type I and Type II heterogeneous systems. Conclusions are given in Chapter 6. References are listed in Chapter 7.

Chapter 2

Related Work

2.1 Introduction

During the past decade, many load sharing algorithms have been proposed and studied. This chapter describes several representative algorithms and related works that have presented in the literature. They will be explained in detail in order for us to see how the load sharing components fit together and how the choice of components affects the system stability.

2.2 State Broadcast Algorithm

The State Broadcast algorithm, along with the Broadcast Idle algorithm and Poll When Idle algorithm, was proposed by Livny and Melman [18]. It assumes that the distributed communication system uses message broadcast facility to broadcast messages to all of the nodes within the system. It also assumes that the expected time for transferring a message from peer to peer is the same for any pair of nodes in the system,

called *uniform distance*. The system itself ensures the equal opportunity of receiving a job among all the nodes.

Four parameters are defined in the algorithm, they are:

1. $ST = (n_1, \dots, n_i, \dots, n_n)$: is a vector that describes the state of the system, where n_i is the number of jobs in node i .
2. $SSV = (s_1, \dots, s_i, \dots, s_n)$: is a vector that contains the updated copy of the system state. It is maintained by information policy.
3. UBF: is an unbalanced factor that is defined as follows: maximum of
$$\begin{cases} (n_i - n_j)/n_j, & \text{for } i > 0 \text{ and } j \leq N \text{ if ST is unbalanced} \\ 0, & \text{otherwise} \end{cases}$$
, where N is the number of nodes in the system. The state of the system is considered as unbalanced if there are two servers, i and j , such that $n_i - n_j > 1$.
4. BT: is a parameter that controls the balancing threshold of the algorithms.

The state-change-driven information policy is used in the algorithm. Whenever a node changes its state, it broadcasts its new state information to other nodes in the system. The information policy ensures that each node has an updated and consistent copy of the system state vector SSV. However, the state vector ST may vary from node to node due to communication delays.

The transfer policy and location policy, also known as *control elements*, determines whether a node i has a waiting job to transfer (sender state) and to which node j the job should transfer based on following three conditions:

1. $s_i - s_j > 1 + (BT \cdot s_j)$.
2. $(s_i > s_k)$ or $(s_i = s_k \text{ and } i \geq k)$ for all $k = 1, \dots, N$.
3. $s_j \leq s_k$ for $k = 1, \dots, N$.

Non-preemptive selection policy transfers only newly received jobs. When more than one node has minimal number of waiting jobs, the destination node is selected randomly.

2.3 Broadcast Idle Algorithm

In Broadcast Idle algorithm, the information policy broadcast the state change information only when the node becomes idle. The received message triggers local control element to react as follows [18]:

1. If $n_i > 1$, go to step 2, else stop.
2. Wait D/n_i ⁶ units of time, where D is a parameter whose value depends on the properties of the communication system.
3. Broadcast a reservation message if no such message is broadcast during the timeout period⁷. Otherwise, if another node has succeeded in broadcasting the reservation message, then stop.

⁶ The algorithm uses state-dependent timeout period which gives better chance to heavily loaded nodes to transfer jobs to idle nodes.

⁷ The period between the same condition re-tested due to transmission collision and message retransmitted.

4. Wait for a reply message. If the node originally broadcast the idle message is still idle, the reply will be positive.
5. If the reply is positive and $n_i > 1$, then transfer a job to the idle node, else stop.

2.4 Poll When Idle Algorithm

The Poll When Idle algorithm is another dynamic algorithm proposed by Livny and Melman [18]. It differs from STD and BID in both information policy and control element. In this algorithm, polling is used rather than broadcast in the exchanging of the state information, thus the information policy is considered as demand-driven policy. When a node becomes idle, it polls a subset of the nodes. To do so, the location policy at node i does the following:

1. Randomly select a set of nodes $R = (a_1, \dots, a_r)$ and set $j = 1$.
2. Send a message to node a_j and wait for a reply.
3. When the reply message comes back, node a_j will either send one of its waiting job or an “empty queue” message.
4. If receiving a transferred job, then stop.
5. If the node is still idle and $j < R$, increment j and go to step 2, else stop.

In the algorithm, R works as a probe limit. It decides the maximum number of polling can be done for each probe. The selection policy only allows non-preemptive job transfer.

2.5 Single Coordinator Algorithm

In the single coordinator algorithm⁸, there is a central node, called coordinator, that collects state information about the whole system. Other nodes have to consult this specific node to locate a suitable node for load sharing. The transfer policy is a *threshold* policy. When a new job originated at a node, the transfer policy looks at the job queue length of the node. This job queue length counts both jobs currently being executed and jobs in the local queue waiting to be executed. If the job queue length is larger than threshold T , the transfer policy determines that the node is currently in the SENDER state, thus the new job is placed in the job transfer queue for possible remote execution. If the job queue length is equal to threshold T , the node is in the OK state. Otherwise, the node is in the RECEIVER state. In the latter two cases, the new job is placed in local queue and these jobs are not eligible for future transfer.

The algorithm uses State-change-driven information policy. Whenever the node changes its state due to the new job arrival or job finish, it sends this change in state to the central coordinator for updating states.

The selection policy used in the algorithm is non-preemptive policy, which means only newly received jobs are eligible for transfer. After transfer policy determines that a node is a receiver, receiver initiated location policy is triggered for load distribution. Typically, at job completion time, if a node changes its state to a receiver, it consults the

⁸ “Single coordinator algorithm” is also referred to a centralized policy.

coordinator to find a sender node. The coordinator will search its entire state list of all the nodes. If more than one sender nodes are found, one sender will be chosen at random. Then, the coordinator informs that sender node to transfer a job on behalf of the receiver. If no sender is found in the whole system, it informs the receiver with NOSENDER message.

As in the receiver-initiated policy, re-initiation of load distribution is necessary in order to improve system performance under certain system and workload conditions.⁹

2.6 Sender Initiated Algorithm

Under Sender Initiated algorithms, load sharing activity is initiated by an overloaded node, called *sender*, trying to send a job to an underloaded node, called *receiver*. Three simple, effective and fully distributed sender initiated algorithms were studied by Eager, Lazowska and Zahorjan [8].

All three algorithms use the identical transfer policy and selection policy. The selected transfer policy is a threshold policy based on the CPU queue length. The CPU queue length includes both the jobs already in service and the jobs waiting for service. There is no exchange of state information among the nodes is required in deciding whether to transfer a job since threshold policy only use local state information. A node is identified as a sender if a new job originating at the node makes the queue length exceed

⁹ Details will be discussed in Receiver Initiated Algorithm.

a threshold T . A node identifies itself as a receiver if accepting a job will not cause the node's queue length to exceed T . The selection policy only allow newly received jobs to be eligible for transfer.

The location policy is different among three algorithms, which will be reviewed as follows:

2.6.1 Random Sender Initiated

Random location policy is the simplest one that uses no information at all. With the Random policy a destination node is selected at random and the job is transferred to that node. There is no exchange of state information among the nodes required in deciding whether to transfer a job.

An issue that arises in considering the behaviour of the random policy is how the destination node should treat an arriving transferred job. The obvious answer to this is that a transferred job should be treated as a new arrival job, therefore it can again be transferred to another node. Unfortunately, this choice has the property of causing instability [8]: irrespective of average load of the system, it is guaranteed that the system will eventually enter a state in which the nodes are spending all their time transferring jobs, with no time spent processing them.

There are some control policies that can overcome instability. Some of them have been developed for a number of multiple access systems [16], [26]. One simple solution is to restrict the number of times that a job can be transferred by introducing a static *transfer limit* L_t . The destination node of the L_t th transfer of a job must process that job

regardless of its state. Despite its simplicity, this random sender initiated policy dramatically improves system response time relative to no load sharing [8].

2.6.2 Threshold Sender Initiated

The Threshold Sender Initiated policy tries to avoid useless job transfers by randomly polling a node to determine whether transferring a job would place it above threshold. The threshold location policy uses a small amount of information about potential destination nodes, although, like Random Sender Initiated policy, it has no knowledge of how to choose the “best” destination node. If the CPU queue length of the probed node will not exceed the threshold T after accepting a job, the job is transferred to that node and destination node must process the transferred job no matter what its state is when the job actually arrives. Otherwise, another node is selected at random and is polled in the same manner. A static probe limit L_p is used to ensure that the cost of executing the load sharing policy will not be prohibitive even in large networks. On the other hand, Eager, Lazowska and Zahorjan’s study had shown that the performance of this policy is surprisingly insensitive to the choice of probe limit [8]. If no suitable receiver node is found within the probe limit, the job is kept for local processing.

By avoiding useless job transfers, the threshold policy provides substantial performance improvement over the Random policy, which indicates that great benefit of using even a small amount of state information.

2.6.3 Shortest Sender Initiated

Unlike Random and Threshold policies, the Shortest Sender Initiated policy attempts to choose the best destination node for a job by acquiring additional system state information. Under Shortest location policy, a number of L_p (probe limit) distinct nodes are chosen at random and polled to determine their queue length. The node with the shortest queue is chosen as the destination for job transfer, unless its queue length is greater than or equal to threshold T , in which case the job is processed locally. The destination node must process the job regardless of its state at the time the job actually arrives.

The performance improvement obtained by using Shortest policy is not significantly better than that of the Threshold policy, indicating that using more state information is of little benefit [8].

2.7 Receiver Initiated Algorithm

In Receiver Initiated algorithm, load sharing activity is initiated from an underloaded node, *receiver*. A lightly loaded node attempts to receive a job from heavily loaded node. A Receiver Initiated algorithm studied by Livny and Melman [18], and Eager, Lazowska and Zahorjan [9] is reviewed in detail here.

The algorithm uses threshold transfer policy that makes decision based on the CPU queue length. When a new job arrives at a node, if the local queue length falls below the threshold T , the transfer policy identifies the node as a receiver and the job is placed in

the local job queue. Otherwise, a node is identified to be a sender and job is placed in the job transfer queue for possible transfer. In the former case, the demand driven information is triggered to collect the state information of sender nodes. Note that when the threshold is set to 0, this policy is identical to Livny's Poll When Idle policy.

The location policy is typically invoked when job completes. It selects a node at random and polls it to locate a node with a non-empty job transfer queue. If such a node is found, then the polled node transfers a job. Otherwise, another node is selected randomly and probing process is repeated at the same manner until either a sender node is found or a static probe limit L_p number of tries has failed to locate a sender. The selection policy can be either non-preemptive or preemptive.

A problem with this policy is that if all probes failed to locate a suitable sender, the node has to wait for the arrival of a local job, thus the processing power available at the receiver is completely lost. This poses severe performance problem when the load is heterogeneous [24] or when there is a high variance in job inter-arrival times [3], [4]. One simple technique to remedy this problem is to reinitiate load sharing activity after the a predetermined period if the node is still idle.

The Receiver Initiated algorithms are stable in term of system performance since even at high system load, a receiver is likely to find a suitable sender within a few polls. Consequently, polls are increasingly effective with increasing system load.

2.8 Hierarchical Receiver Initiated Algorithm

Load sharing algorithm can either be distributed or centralized. Distributed policies are scalable to large systems, but the performance of these policies is sensitive to variance in service times and inter-arrival times [3], [4]. Centralized policy provides near perfect load sharing, however, it may cause performance bottleneck, scalability and fault-tolerance problems. Dandamudi and Lo has proposed a Hierarchical Receiver Initiated algorithm [6] that provides performance closes to the centralized policy while inheriting the merits of the distributed policies.

Under this type of algorithm, nodes are organized into a logical control hierarchy. Rather than a single node maintaining the entire system state, a set of nodes is given this responsibility. The system is logically divided into clusters and each cluster of nodes will have a single node that maintains the state information of the nodes within the cluster. The state information on the whole system is maintained in the form of a tree where each tree node maintains the state information on the set of processor nodes in the sub-tree rooted by the tree node.

Figure 2.1 shows an example of hierarchical organization for 8 processors with a branching factor B of 2. As discussed before, each node can be in one of three states: overloaded (*sender*), normal load (*OK*) and underloaded (*receiver*). In the algorithm, +1, 0 and -1 are used to represent each of above state respectively. For example, Q4 maintains the state information of nodes N0 whose state is overloaded and N1 whose state is underloaded.

As we move up along the tree, the cluster size increases accordingly: Q2 maintains the state information of nodes N0, N1, N2 and N3. It may be noted from Figure 2.1 that only summary state information of sub-tree nodes is maintained in order to reduce state update frequency as well as the overhead of state updating at higher tree nodes. The summary metric used in this policy is the arithmetic sum of the state information. For instance, the summary state metric for Q4 is zero, indicating that the cluster represented by Q4 is in OK state; the summary state metric for Q6 is +1 rather than +2. Since this hierarchical policy greatly encourages local load balancing, it is also called *local hierarchical policy*.

Figure 2.1 An example of a local hierarchical organization for an 8-node system ($B = 2$).

While the local hierarchical has the advantage of reducing the number of updates and overhead caused by that, it also hides the fact that a sender may not be known beyond its cluster. For example, in Figure 2.1, the fact that node N0 is a sender will not be

discovered beyond the cluster of N0 and N1. Although the state information of node N0 will eventually be available to the other nodes if node N0 remains a sender state and N1 is no longer a receiver, the problem is that the nature of local hierarchical policy sometimes delays informing other nodes of the existence of a sender node by giving the priority to local load sharing. An alternative to this is a *global hierarchical* policy which is also proposed by Dandamudi and Lo. The global hierarchical policy differs from the local policy in the manner of information maintained in the hierarchical structure. Figure 2.2 gives an example of global hierarchical organization of an 8-node system with a branching factor of 2.

Figure 2.2 An example of a global hierarchical organization for an 8-node system ($B = 2$).

As shown in Figure 2.2, the hierarchical structure maintains only two state information: sender and non-sender, represented as 1 and 0 respectively. Thus, the semantics used to keep state information in the tree nodes are: 1 is stored if there is one or

more senders in that branch, otherwise 0 is stored. For example, in Q2, there is a value 1 that indicates left side branch has at least one sender. Global hierarchical policy overcomes local hierarchical policy's disadvantage by publicizing the existence information of a sender as widely as possible in order to allow any receiver at anywhere in the system to initiate a job transfer.

In the hierarchical structure, load sharing can be done at various levels. Therefore, root node does not have to be responsible for all load sharing requests in the system. The “tree” nodes can all be distributed to different system nodes so that no node is unduly overloaded with system state information. In a system with N nodes, the maximum number of tree nodes will be required is $N-1$ when the branching factor is 2. On top of the hierarchical structure, different load sharing policies can be implemented. Dandamudi and Lo proposed an algorithm that uses the receiver-initiated policy on top of the hierarchical structure [6]. Whenever a node becomes a receiver (job queue length below the threshold T^{l0}), after a job is completed, it consults its parent node in the hierarchy for a sender node. If the parent does not have any sender nodes in its sphere, it forwards the message to its parent tree node. Such a process is continued up the tree until either a tree node with a sender node is encountered or the root node is reached. In the latter case, if the root node does not have a sender node, it sends a “no job” message to the node that issued the request. If a tree node or the root node finds at least one sender branch, it sends the message downward along that branch until it reaches the sender node. Sender branch

¹⁰ Threshold T is set to 1 in the proposed Hierarchical Receiver-Initiated algorithm.

is selected at random if there are more than one sender branches. So does sender in the same sender branch.

When the sender receives the message and is still in the sender state, it transfers a job to the receiver node. Otherwise, a “false sender” message will be sent back to the receiver node. When the receiver node receive a “no job” or a “false sender” message and is still in the receiver state, it updates its state information in the hierarchy and waits for the corresponding re-initiation period and initiates another load distribution request if it remains in the receiver state at the end of the re-initiation period.

It had been shown that Hierarchical Receiver Initiated algorithm provides substantial performance improvements over the Sender Initiated algorithms and Receiver Initiated algorithms [6].

2.9 Stable Symmetrically Initiated Adaptive Algorithm

The Stable Symmetrically Initiated Adaptive algorithm [23], proposed by Shivaratri and Krueger, combines both sender-initiated and receiver-initiated components. It uses the information gathered during probing rather than discard it to keep track of the recent state of each node in the system. The purpose of that is to find a lightly loaded node when a heavily loaded node wishes to transfer jobs, and to find a heavily loaded node when a lightly loaded node wishes to acquire some work. It has shown that this algorithm is stable under all system loads [23].

A node's state, either sender or receiver or OK, is identified by its transfer policy and is returned to the location policy as the reply to a probe. This information is maintained independently at each node by a data structure that consists of a sender list, a receiver list and an OK list. At the beginning, all nodes are considered as receivers and the receiver list at each node is initialized with all nodes except the node itself. The sender list and OK list are initialized empty.

The threshold transfer policy used to make load distribution decisions is based on the CPU queue length. A node identifies itself as a suitable sender whenever its queue length exceeds the specified threshold T , as a receiver whenever its queue length is less than T and as OK whenever its queue length equal to T . The location policy has two components: the sender-initiated component and the receiver-initiated component. The sender-initiated component is triggered when a node becomes a sender, the receiver-initiated component is triggered when a node becomes a receiver.

During sender-initiated negotiation, a sender node tries to find a suitable receiver. To do so, it does the following:

1. The sender probes the node at the head of its receiver list to determine whether it is still a receiver.
2. If the probed node identifies itself as a receiver, probed node returns its ID to the sender and the sender transfers the new job to it. Otherwise, the probed node's ID is removed from the receiver list and is put at the head of the OK list if it is OK or the sender list if it is a sender.

3. Probing stops if a suitable receiver is found, or the number of probe limit is reached, or if the receiver list is empty. If probe fails, the job is processed locally.

Meanwhile, the probed node reacts as follows:

1. On the receipt of a probe message from a sender, the probed node removes the sender node's ID from the list it is presently in, puts it at the head of sender list. (Note: if the sender was already in the sender list, this step simply changes its position in the list).
2. Reply the sender indicating whether the probed node is a receiver, sender or OK.

During receiver-initiated negotiation, a receiver tries to find a suitable sender. To do so, it does the following:

1. Probe a selected node to determine whether it is a sender. The node selected is based on following order:
 - 1). Head to tail in the sender list. The most up-to-date information is used first.
 - 2). Tail to head in the OK list. The most up-to-date information is used first in the hope that the node has become a sender.
 - 3). Tail to head in the receiver list. The most up-to-date information is used first.
2. If the probed node responds that it is a sender, the receiver node removes the sender node from the list it is presently in and puts it at the head of the sender list if it is still a sender after transferring a job or the OK list if it is no longer a sender after the transfer. If the probed node is not a sender, the receiver node removes the probed node from the list it is presently in and puts it at the head of the receiver list if it is a receiver or the OK list if it is OK.

3. Probing stops if a sender is found, if the receiver is no longer a receiver, if the number of probing limits is reached or all the nodes that might be considered for probing have been probed.

At the same time, the probed node reacts as follows:

1. On the receipt of probing message, removes the receiver node from whatever the list it is in and puts it at the head of the receiver list.
2. If the probed node is a sender, transfer a job to the receiver and informs it the state after the transfer. If the probed node is not a sender, return a message indicating its state.

Non-preemptive selection policy is used by both sender-initiated and receiver-initiated components. The information policy of this algorithm is demand driven since probing starts when a node becomes either a sender or a receiver.

Stable Symmetrically Initiated policy shows improved performance over a wide range of system loads and preserves system stability. At low system loads, receiver-initiated probes are frequent. Although they tend to be useless, frequent probes keep the receiver list accurately reflecting the system's state, which makes future sender-initiated probes succeed within a few polls. At high system loads, the sender-initiated component is deactivated, leaving only receiver-initiated load sharing, which is effective at such loads.

2.10 Stable Sender Initiated Adaptive Algorithm

For systems that are unable to transfer partly-executed job, Shivaratri and Krueger proposed Stable Sender Initiated Adaptive Algorithm [23] to meet such a requirement. This algorithm uses the same sender-initiated component as Stable Symmetrically Initiated algorithm but has a modified receiver-initiated component. Obviously, non-preemptive selection policy is used in this algorithm as well. Since Stable Sender Initiated algorithm is very similar to the Stable Symmetrically Initiated algorithm, here only the difference will be reviewed.

In the Stable Sender Initiated algorithm, the state information is maintained by a *state vector* array. Each node keeps track of which list it belongs to at all the other nodes in the system. For instance, $statevector_i[nodeid]$ tells which list node i belongs to at the node indicated by $nodeid$.

Initially all nodes belong to the receiver lists at all other nodes. Sender-initiated negotiation is identical to that of Stable Symmetrically Initiated algorithm with the following additional steps:

1. At a sender node, after the sender probes a selected node, the sender updates its state vector to show that the sender belongs to the senders list at the selected node.
2. At a probed node, it updates its state vector based on the reply it sent to the sender node to reflect which list it will belong to at the sender.

In order to avoid excessive and wasteful state change information messages, a potential receiver informs only those nodes whose receiver list do not contain it. To do so, two steps are needed:

1. At a receiver node, it updates its state vector to reflect its current state at only those nodes that are misinformed about its current state and informs those nodes.
2. At an informed node, it removes the receiver node from whatever list it is in, and puts it at the head of receiver list.

Finally, the reason for this algorithm's stability is the same as that for the Stable Symmetrically Initiated algorithm.

2.11 Impact of Job's I/O on Two Basic Algorithms

Dandamudi and Hadavi had studied the impact of job's I/O service demand on the performance of sender-initiated and receiver-initiated load sharing policies [5]. In both policies, the same threshold-based transfer policy is used. A destination node is selected at random and the job is transferred to that node if a suitable target node is found within P_t probes. Otherwise, the job is processed locally. There is no exchange of state information required among nodes.

Job's I/O service is modeled in two ways: non-interleaved I/O and interleaved I/O. In the non-interleaved model, two I/O operations are performed. One is at the beginning and the other is at the end of job execution. In the interleaved model, the total

I/O service demand is equally divided among n I/O operations, interleaving with equally distributed CPU demand.

The performance evaluation is done in the homogeneous distributed system. The results of the study have shown that exact I/O model has a more significant impact on receiver-initiated policies when the inter-arrival time variance is high. Variance in disk service times impacts both sender-initiated and receiver-initiated policies similarly. The performance difference between interleaved and non-interleaved I/O models increases with the proportion of jobs with I/O requirements.

Chapter 3

Policies and Models

3.1 Introduction

The past few years have experienced a steady growth in research of load sharing in distributed systems. While most of previous research have concentrated on analyzing different load sharing policies based on CPU-intensive jobs, the problem of job's I/O requirements and its effects on load sharing had not been investigated in great detail until recently [5]. In that paper, the impact of job I/O requirements on the performance of sender-initiated and receiver-initiated load sharing policies as well as the performance sensitivity to difference system and workload parameters had been examined. This thesis continues to study the impact of job's disk service demand on other three load sharing policies and evaluated the performance difference among them. It also extends the analysis to include both homogeneous and heterogeneous distributed systems. The objective of the thesis is not to determine the absolute performance of particular load sharing policies but rather to assess the relative advantage of varying degrees of sophistication.

3.2 Load Sharing Policies

Three load sharing policies are studied in the thesis. They are Local Hierarchical Receiver Initiated policy, Global Hierarchical Receiver Initiated policy and Stable Symmetrically Initiated policy.

All of three load sharing policies share the same non-preemptive selection policy, which allows only newly received job to be eligible for transfer. The adopted static transfer limit, L_t , in this thesis is set to 1. Thus, when a job is transferred to the destination node, the node must accept and process the transferred job even if the state of that node has changed since then.

Three load sharing policies have identical transfer policies. They use a threshold transfer policy. When a job arrives at a node, the transfer policy places the new job in the job queue for local execution if the number of jobs already in service or waiting for service is less than the threshold T . Otherwise, the job is placed in the job waiting queue for possible remote execution.

Local Hierarchical Receiver Initiated policy (LH): As discussed in Section 2.8, each node can be in one of three states: sender (+1), OK (0) and receiver (-1). The hierarchical structure contains only summary state information. The summary metric used is the arithmetic sum of the state information of the tree nodes below it. Tree nodes are distributed uniformly across the system nodes. The information policy used in the algorithm is state-change-driven policy as the node updates its state information in the hierarchy whenever its state changes. The receiver-initiated location policy is implemented on top of the hierarchical structure. This policy is triggered when a node

becomes a receiver, it consults its parent tree node for a suitable sender. The message is forward up the tree until either a tree node with a sender branch is found or the root node is reached. Finally, the receiver will receive a transferred job from the sender or a message indicating “no job” or “false sender”. In the latter case, probe will be reinitiated for a predetermined time if the node is still idle.

Global Hierarchical Receiver Initiated policy (GH): It uses the exact same information policy and location policy as Local Hierarchical Receiver Initiated policy, but differs in the state information maintained in the hierarchy. Each node in the global hierarchy can only be either sender (+1) or OK (0). The intention is to publicize the existence of a sender node in the system as widely as possible rather than encourages the local load sharing. On top of the hierarchical structure, the receiver-initiated load sharing policy is implemented.

Stable Symmetrically Initiated policy (SYM): As described in Section 2.9, in this algorithm, the location policy has two components: the sender-initiated component and the receiver-initiated component. The sender-initiated component is triggered when a node becomes a sender, while the receiver-initiated component is invoked when a node becomes a receiver. The location policy utilizes the information gathered during probing to keep track of the recent state of each node in the system. The state information is maintained at each node by three lists: sender list, OK list and receiver list. Three lists change with the state of the system as the node receives probes and reply messages from other nodes. The information policy is demand driven as probing starts when a node becomes either a sender or a receiver.

3.3 System Model

The network of nodes used in the experiments is based on inexpensive off-the-shelf components. The system runs PVM (*Parallel Virtual Machine*) under Linux. The current configuration consists of 6 Pentium and 3 Pentium Pro nodes, each with 32 MB of memory. These nodes are interconnected with a 10 Mbps “service” network as well as with a 100 Mbps 100 Base TX high-speed network. The 10 Mbps network is used for booting the system, providing the basic communication and remote file service through NFS (*Network File System*). The 100 Mbps network is connected with a switch that has a 2 Gbps backplane. Both the 10 Mbps network and 100 Mbps network can be used for high level communication under PVM.

The Parallel Virtual machine (PVM) system uses explicit message-passing model to allow programmers to exploit distributed computing across a wide variety of computer types. It makes a collection of computers appear as one large *virtual* machine. PVM transparently handles all message routing, data conversion, and task scheduling across a network of incompatible computer architectures. Tasks access PVM resources through a library of standard interface routines.

To make a reasonably comprehensive study that is tractable within the scope of this thesis, the following assumptions are made about the distributed environment. The jobs are individually executable, are logically independent of one another, and can be processed by any node.

A locally distributed system is represented by a collection of $N = 32$ single processor nodes and $D = 8$ disk nodes. The communication among the nodes is through

the PVM system, which includes sending and receiving probing messages, transferring job and job departure for disk service. Each job has its unique id across the system. Only job header that includes job id is transferred to another node during load sharing. Then, the destination node is able to retrieve transferred job's executable code under the NFS system.

The CPU would give preemptive priority to communication activities, such as packing and unpacking data and sending a probe message etc, over the processing of jobs. The processor scheduling policy is first-come/first-served (FCFS). The mean response time is used as the chief performance metric to compare the performance of different load sharing policies in this thesis. The mean response time is defined as average job completion time.

In this thesis, two types of distributed systems are studied: homogeneous system and heterogeneous system. In homogeneous system, all the nodes are identical and the workload at each node is similar. In heterogeneous system, two types of systems are considered: in type I heterogeneous system, the nodes are identical in terms of processing capabilities and speeds, but the job arrival rates are different. In type II heterogeneous system, nodes have different job arrival rate as well as processing rate. Both types of heterogeneous systems contain two classes of nodes, where a class is based upon similar job inter-arrival rates and/or similar processing rate [21]. All the nodes in a particular class are assumed to be identical.

3.4 Workload Model

The system workload is represented by the following parameters: The job arrival process at each node is characterized by a mean inter-arrival time $1/\lambda$ and a coefficient of variation C_a , where λ is the rate of arrival of new jobs. Jobs are characterized by a processor mean service time $1/\mu$ and a coefficient of variation C_s , where μ is the job service rate. The Job I/O service demand is characterized by a mean $1/\mu_{io}$ with a coefficient of variation C_{io} . The simulation study uses a two-stage hyper-exponential model [11] to generate inter-arrival time, processor service time and disk service time CVs greater than 1. $CV = 1$ indicates Poisson distribution.

The offered system load is defined as job's inter-arrival rate divided by job's processor service rate λ/μ . Since mean job processor service rate $\mu=1$ is used for all the experiments, the offered system load is equal to job's inter-arrival rate λ in this thesis.

3.5 I/O Model

Given the parameters that characterize the total job I/O service demand that is still a question of how this service demand is distributed within a job. This thesis models I/O service demand in two ways [20]: non-interleaved I/O (NIL) and interleaved I/O (INL).

In the non-interleaved I/O model, two I/O operations are performed: one at the beginning and the other at the end of job execution (see Figure 3.1a). The total I/O service demand of a job is equally divided between the two operations. The first I/O

operation can be interpreted as reading input data, for example, and the second I/O operation can be seen as writing the result data to the disk. This type of I/O model is appropriate for applications that process files sequentially.

In the interleaved I/O model, disk service is interleaved with CPU service (see Figure 3.1b). The total I/O service demand of a job is equally divided among n I/O operations. Consequently, the total CPU service is also equally distributed as shown in the Figure 3.1b. This type of model is suitable for applications that require random file access. The interleave times n is an input parameter to the simulation model. In this thesis, it is fixed at 5 due to the convenience and simplicity. I/O job joins at the front of the queue once it finishes disk service.

Figure 3.1 Two I/O models: (a) Non-interleaved (b) Interleaved

A fraction α is a parameter used to classify the jobs arriving at each node. The α fraction is assumed to be IO-jobs (i.e., jobs with some disk service requirement) and the remaining $(1-\alpha)$ fraction of the jobs are CPU-intensive. Thus, by changing α value, the model can be used to simulate pure CPU-intensive workload ($\alpha = 0$), pure IO-intensive

workload ($\alpha = 1$), or mix workload ($0 < \alpha < 1$). In this thesis, $\alpha = 0.5$ (half CPU-intensive jobs and half IO-intensive jobs) is used in most of the simulation experiments. However, the performance sensitivity to this parameter is studied in the thesis as well.

Three workloads¹¹ with different job I/O requirements are studied in the thesis. They are pure CPU-intensive jobs, 50% CPU-intensive jobs with 50% non-interleaved I/O jobs, 50% CPU-intensive jobs with 50% interleaved I/O jobs. The results of these experiments are presented in next two chapters to make better performance comparison among above three workloads and look at the impact of jobs disk service requirements.

¹¹ “Workload” and “models” are used interchangeably in this thesis.

Chapter 4

Performance in Homogeneous Distributed Systems

4.1 Introduction

This chapter presents the simulation experiment results and discusses I/O impact on performance characteristics of three load sharing policies: Local Hierarchical Receiver Initiated policy (LH), Global Hierarchical Receiver Initiated policy (GH) and Stable Symmetrically Initiated policy (SYM) in the homogeneous distributed environment. Unless otherwise specified, the following default parameter values are used for all the tests in this chapter.

The distributed system contains $N = 32$ identical processor nodes and $D = 8$ disk nodes interconnected by a 10 megabit/second Ethernet-like communication network. The communication, such as probing, job transferring and job departure for disk service, is through PVM. The size of a probe message is 16 bytes. The job header is 40 bytes. The mean job service rate μ is 1 per time unit and disk service rate μ_{io} is 2.5. The percentage of IO-jobs α is fixed at 50%. The IO-jobs are either all non-interleaved (NIL) or all

interleaved (INL). Threshold T for all policies is fixed at 2. A static probe limit P_t of 3 is used for symmetrically initiated policy in all the experiments except in Section 4.2.6 which will present the performance sensitivity of varying the probe limit. Hierarchical policies use the default branching factor $B = 2$ in this thesis.

Batch strategy has been used to compute confidence intervals. Each experiment uses at least 30 batch runs that each contains 3,000 to 6,000 jobs varied based on offered system load and coefficient of variation (CV). This strategy generated 95% confidence intervals that were less than 1% of the mean response times when the system utilization is low to moderate and less than 5% for moderate to high system utilization (in most cases, up to about 90%).

4.2 Performance Analysis

4.2.1 *Performance as a function of System Load*

The motivation of this study is to determine the performance sensitivity of the mean system response time to variations in offered the system load and the effect of job I/O requirement of the three load sharing policies.

Figure 4.1 and 4.2 are graphs of mean response time versus system load with the impact of job I/O requirements for three policies. The results in Figure 4.1 and 4.2 correspond to an inter-arrival CV (C_a), service time CV (C_s) and disk service time CV (C_{io}) of 1.

Figure 4.1 Performance sensitivity as a function of system load ($N=32$ nodes, $\mu=1$, λ is varied, $C_a = C_s = C_{io}=1$, $\mu_{io}=2.5$, $T=2$, $P_l=3$, $B=2$, all other parameters are set to their default values).

Figure 4.2 Performance sensitivity as a function of system load by policy (Parameters are set the same values as in Figure 4.1).

It can be observed from the data shown in the Figure 4.1 and 4.2 that job's I/O requirements have substantial performance impact for all three policies. In general, the pure CPU-intensive workload provides the best performance among the three different workloads. This is because job's I/O requirement involves additional overheads, such as job's departure cost and disk contention, which causes longer average job response time and poorer overall system performance. The more job's disk service demands, the worse system performance. When system load increases, the number of I/O jobs increases. Increased job's I/O requirements quickly deteriorate the performance of non-interleaved and interleaved I/O models. As a result, the performance difference among three workloads increases with the system load.

At low to intermediate system loads (up to about 70%), the overheads caused by disk service do not greatly deteriorate the overall system performance as we can see from Figure 4.1 and 4.2. This is because extra processing capacity is available at those loads. Therefore, both non-interleaved model and interleaved mode does not significantly affect the performance of all three load sharing policies at low to moderate system loads.

Figure 4.1 indicates that stable symmetrically initiated policy¹² outperforms both local and global hierarchical receiver initiated policies under all three workloads, though the performance difference is marginal at low system load. The performance difference between symmetrical policy and two hierarchical policies becomes bigger when system load moves to high. At low system load, a potential receiver node is unlikely to find a

¹² "Stable symmetrically initiated policy" is interchanged with "Symmetrically initiated policy" or "Symmetrical policy" in this thesis.

sender since few nodes are overloaded. A potential sender node, however, has little trouble finding a receiver node since many nodes are lightly loaded. Thus, receiver-initiated negotiation generally fails. However, in stable symmetrically initiated policy, the failure of receiver-initiated negotiation has the positive effect of keeping the receiver lists accurate at other nodes, sender-initiated negotiation generally succeeds in finding a receiver after only a few probes. At moderate loads, both senders and receivers in stable symmetrically initiated policy are able to find partners.

When the system loads become higher, most of nodes are likely to be always at sender state, resulting in reduced receiver-initiated probe and load sharing activities, hurting all three policies' performance. Nevertheless, symmetrical policy benefits from its both sender-initiated and receiver-initiated components, precisely using the system state information gathered during probes and effectively locating receivers. As a result, it increases the load sharing activities as much as possible and remains system performance stability. While in two hierarchical policies, locating a sender usually involves sending more messages through the hierarchy, generating more overheads, further worsening the performance.

Global hierarchical policy performs marginally better than local hierarchical policy, especially at low to moderate system loads. One main reason is because local hierarchical receiver initiated policy encourages local load sharing, which could potentially degrade load sharing activities as the state information of the sender nodes will not be available beyond those tree branches that have at least one receiver node. The net effect of that is the local hierarchical policy sometimes hides the fact of the existence of senders while some receiver nodes are looking for them, resulting in fewer and

delayed load sharing activities. In global hierarchical policy, sender state information is published widely in the hierarchical tree so that a receiver is able to locate an existing sender anywhere in the hierarchy, increasing load sharing activities. However, the delay in local hierarchical policy becomes less critical when system load becomes high since more nodes are at sender states. As a result, the performance of local hierarchical policy is close to global hierarchical policy.

Overall, stable symmetrically initiated policy remains system stability across the range of system loads, even when the system load is extremely high. In such situation, an initial high rate of unsuccessful sender-initiated probes results in the removal of nodes from the receiver lists, finally causing the receiver lists become empty. Thus, sender-initiated probing is cut off at high system loads and only receiver-initiated probing is performed, which does not lead to instability.

Figure 4.3 and 4.4 show the mean response time of the three load sharing policies as a function of offered system load. These results correspond to an inter-arrival CV (C_a) and service time CV (C_s) of 4. The disk service time CV (C_{io}) remains 1.

As shown in the Figure 4.3 and 4.4 that job I/O requirements continue to have substantial impact on system performance especially at high system load for the same reason discussed before. In all the policies, pure CPU model provides the best performance across the spectrum of system loads. Non-interleaved I/O model performs better than interleaved I/O model especially at high system loads. At low to moderate system loads, the difference between two I/O models is not noticeable due to the availability of extra processing capacity.

Figure 4.3 Performance sensitivity as a function of system load ($N=32$ nodes, $\mu=1$, λ is varied, $C_a=C_s=4$, $C_{io}=1$, $\mu_{io}=2.5$, $T=2$, $P_l=3$, $B=2$, all other parameters are set to their default values).

Figure 4.4 Performance sensitivity as a function of system load by policy (Parameters are set the same values as in Figure 4.3).

Figure 4.3 and 4.4 also suggests that higher inter-arrival and service times CVs have a great impact on all policies since the mean response time of all policies in all workloads is much higher than that in Figure 4.1 and 4.2. Stable symmetrically initiated policy, again, provides low mean response time over both local and global hierarchical policies and maintains excellent system stability for the whole range of system loads.

The performance difference between symmetrical policy and two hierarchical policies for each model becomes bigger than that shown in Figure 4.1. The reason is because the higher inter-arrival CV implies clustered nature of job arrivals at each node, which reduces receiver-initiated probing and fosters sender-initiated probing. Consequently, the poor performance of two hierarchical policies results even at low system load due to reduced load sharing activities. The graph shows that two hierarchical receiver initiated policies are very sensitive to inter-arrival CV. In addition, sender-initiated probing is more sensitive to service times CV than receiver-initiated probing as larger jobs monopolize the processing power, causing increased sender-initiated probe rates. Section 4.2.3 will further discuss the impact of this parameter. Since sender-initiated policy is effective at low to moderate system load, extra probing increases load sharing activities, resulting improved performance in symmetrical policy. At high system load, the sender-initiated search is cut off due to the empty receiver list at each node, left only receiver-initiated probes. As the state lists quite accurately reflect system's state in symmetrical policy, probing usually is more effective than random search in two hierarchical policies, resulting in reduced probe overheads and better performance.

Global hierarchical policy performs slightly better than local hierarchical policy at low to moderate system loads. This result comes from more load sharing activities in the

global hierarchical policy. When the system load becomes high, the delay of in the local hierarchical policy in making the sender state information globally available becomes unimportant. Thus, the performance of the local hierarchical policy is similar to that of the global hierarchical policy.

4.2.2 Sensitivity to Variance in Inter-Arrival Times

This section considers the effect of varying inter-arrival CV (C_a) on the performance of three load sharing policies. Offered system load is set to 80% for all the nodes. The service time CV (C_s) and disk service time CV (C_{io}) are set to 1. Figure 4.3 depicts the results of varying C_a between 0 and 4 for three policies under three workloads.

It can be observed from the Figure 4.5 and 4.6 that the impact of variance in inter-arrival CV is much more significant on both local and global hierarchical receiver-initiated policies than on symmetrically initiated policy. The performance of two hierarchical policies deteriorates quickly with the increase of inter-arrival CV while symmetrically initiated policy remains stable for the whole range of tested inter-arrival CV. The reason is that receiver initiated probing is more sensitive to the inter-arrival CV while sender-initiated probing is relatively less sensitive. Higher inter-arrival CV means that the job arrival process at each node is clustered. For example, assume that 4 jobs have arrived into the system as a cluster. Then, the receiver-initiated policy at node will not start the probe process until those four jobs are completed. Thus, increasing job inter-arrival CV basically decreases the receiver-initiated load sharing activities at each node

as the probability of finding a node with no job is low, moving the receiver-initiated based system towards no load sharing system [4]. This fact severely hurts the performance of the two hierarchical policies since both of them use receiver-initiated policy, resulting in quick performance decline. The performance of the local and global hierarchical policies is similar as can be seen from the graph.

Figure 4.5 Performance sensitivity to inter-arrival time CV (C_a) ($N=32$ nodes, $\mu=1$, $\lambda=0.8$, C_a is varied, $C_s = C_{io}=1$, $\mu_{io}=2.5$, $T=2$, $P_l=3$, $B=2$, all other parameters are set to their default values).

On the other hand, clustered arrival of jobs fosters increased load sharing activities in sender-initiated based system. Thus, the probe rate and the job transfer rate increase with inter-arrival CV. This explains why symmetrically initiated policy yields better performance over both local and global hierarchical policies. By using sender

initiated probing, symmetrical policy is able to maintain certain level of load sharing even when inter-arrival CV is high, resulting in good performance and system stability. In addition, probes in symmetrical policy usually are more efficient due to its accurate state lists, resulting in less probing overheads.

Figure 4.6 Performance sensitivity to inter-arrival time CV by policy (Parameters are set the same values as in Figure 4.5).

Figure 4.5 and 4.6 also show that job's I/O requests have strong impact on system performance for all three policies. Pure CPU model performs the best among three workloads since there is no disk service overheads. The performance difference between two I/O models is more noticeable in two hierarchical policies than in the symmetrical policy as the inter-arrival CV increases. When the inter-arrival CV becomes bigger, receiver-initiated load sharing activities are greatly reduced. Without load sharing, the more job's disk service overheads, the worse the performance, which resulting in huge performance gap between interleaved and non-interleaved models in two hierarchical policies.

4.2.3 Sensitivity to variance in processor service times

This section considers the impact of service time CV (C_s) on the performance of the three load sharing policies. Offered system load is set to 80% for all the nodes. The inter-arrival time CV (C_a) and disk service time CV (C_{io}) are set to 1. Figure 4.7 and 4.8 shows the results of varying C_s between 0 and 4 for the three policies under three workloads.

Compare Figure 4.7 and 4.8 with two figures in Section 4.2.2, it can be noted that both local and global hierarchical policies are much less sensitive to service time CV than inter-arrival CV. This is due to the nature of FCFS scheduling policy. Under FCFS policy, with the increasing of service time CV, larger jobs tend to monopolize the processing power, causing increased job queue length. As a result, sender-initiated probing activity increases since nodes become senders more often. Since hierarchical policies use receiver-initiated policy on top, they exhibit less sensitivity to service time CV. The performance difference between local hierarchical policy and global hierarchical policy is insignificant as it can be seen in Figure 4.7.

Although symmetrically initiated policy has sender-initiated component, it is still stable and exhibits better performance over both local and global hierarchical policies. Besides the fact that symmetrical policy benefits from its receiver initiated component which is less sensitive to service time CV, it is also effective in controlling unnecessary probing due to its judicious use of the knowledge gained by probing even when service time CV is high. As a result it exhibits reduced sensitivity to the service time CV. In

contrast, in both local and global hierarchical policies, more probing messages are needed in order to locate suitable senders, causing more probing overheads.

Figure 4.7 Performance sensitivity to process service time CV (C_s) ($N=32$ nodes, $\mu=1$, $\lambda=0.8$, C_s is varied, $C_a = C_{io}=1$, $\mu_{io}=2.5$, $T=2$, $P_l=3$, $B=2$, all other parameters are set to their default values).

Figure 4.8 Performance sensitivity to inter-arrival time CV by policy (Parameters are set the same values as in Figure 4.7).

Job's I/O requirements continue to play substantial role in influencing system performance as shown in Figure 4.7 and 4.8. Pure CPU model performs much better over non-interleaved and interleaved I/O models in all three load sharing policies. Non-interleaved I/O model performs marginally better than interleaved I/O model. However, the difference increases with increasing of service time CV. At high service time CVs, larger jobs occupy the CPU time, resulting in average longer job waiting time. Since I/O jobs have to rejoin the job queue each time after finishing disk service, they normally have longer response time than non-IO jobs. In interleaved I/O model, IO jobs depart for disk service more than in non-interleaved I/O model, resulting in longer response time for those jobs in interleaved model than in non-interleaved model. Therefore, performance of interleaved model deteriorates.

4.2.4 Sensitivity to variance in disk service times

This section examines the impact of varying disk service time CV (C_{io}) on the performance of three load sharing policies. Offered system load is set to 80% for all the nodes. The inter-arrival time CV (C_a) and service time CV (C_r) are set to 1. Figure 4.9 and 4.10 shows the results of varying C_{io} between 1 and 4 for the three policies under non-interleaved I/O model and interleaved I/O model.

The important observation from Figure 4.9 and 4.10 is that disk service time CV affects all the policies similarly. For the same workload model, the performance difference among three policies is marginal. Overall, symmetrically initiated policy performs slightly well than both local and global hierarchical policies for each workload.

Figure 4.9 Performance sensitivity to disk service time CV (C_{io}) ($N=32$ nodes, $\mu=1$, $\lambda=0.8$, C_{io} is varied, $C_a = C_s=1$, $\mu_{io}=2.5$, $T=2$, $P_l=3$, $B=2$, all other parameters are set to their default values).

Figure 4.10 Performance sensitivity to disk service time CV by policy (Parameters are set the same values as in Figure 4.9).

The reason is that, when disk service CV (C_{io}) increases, the jobs depart the disk in a clustered fashion causing the jobs to return to CPU for service in a cluster manner. This causes the same effect as the inter-arrival CV, which decreases the receiver-initiated probe rate and increases the sender-initiated probe rate. Since the sender-initiated probing is very limited at high system load in symmetrically initiated policy, the increase of disk service CV has almost the same impact on all three policies, resulting in similar performance deterioration. The slight performance difference is mainly due to the difference of probing effectiveness among the three policies. In the two hierarchical policies, locating a sender needs more probes through the hierarchy. While in symmetrical policy, a fairly accurate state lists make the probe more effective and reduce probing overheads.

Increased disk service CV decreases receiver-initiated load sharing activities since it has similar effect as increased inter-arrival CV. Interleaved I/O model is more sensitive to reduced load sharing because there are more overheads associated with job I/O service than in non-interleaved model. As a result, the performance difference between two I/O models increases with increasing of disk service CV.

4.2.5 Sensitivity to disk jobs

In all the experiments described as far, the parameter α is fixed at 50%, which means an equal proportion of CPU-intensive jobs and IO-jobs. This section studies the sensitivity of the mean system response time to variations in the proportion of IO-jobs. Offered system load is set to 80% for all the nodes. Inter-arrival time CV (C_a), service

time CV (C_s) and disk service time CV (C_{io}) are set to 1. Figure 4.11 and 4.12 present the results of varying fraction α between 10% and 60% for the three policies under non-interleaved I/O model and interleaved I/O model.

Some key observations from Figure 4.11 and 4.12 are as follows:

- I. The mean response time of each policy increases with the proportion of IO-jobs. This result supports the previous observation that job's I/O requirements significantly impact the system performance. The more jobs require disk service, the higher the mean response time.
- II. The difference in response time between non-interleaved and interleaved models for each load sharing policy increases with the increase of IO-jobs. When α is less than 40%, both models in each policy seem to provide similar response times. In such case, extra processing capability is available to satisfy job's I/O requests. When α is greater than 40%, the performance gap becomes bigger as the percentage of IO-jobs increases. The main reason is because the overhead of job departure for disk service increases dramatically as the percentage of IO-job increases, which quickly drains the CPU processing capability, resulting in severe performance deterioration. On the other hand, more I/O jobs indicate long job waiting time in the job queue. As a result, nodes are often in the sender state rather than in the receiver state. Thus, both sender-initiated load sharing and receiver-initiated load sharing activities are reduced due to likely empty receiver list and rare receiver state respectively, moving all three policies towards no load sharing.

Figure 4.11 Performance sensitivity to disk jobs ($N=32$ nodes, $\lambda=0.8$, $\mu=1$, $C_a = C_s = C_{io}=1$, $\mu_{io}=2.5$, fraction of disk jobs α is varied; $T=2$, $P_l=3$, $B=2$, all other parameters are set to their default values).

Figure 4.12 Performance sensitivity to disk jobs by policy (Parameters are set the same values as in Figure 4.11).

III. Symmetrically initiated policy exhibits marginally better performance than both local and global hierarchical policies. The poor performance of two hierarchical policies results from their relatively inefficient hierarchical search and more probing overheads. On the other hand, symmetrically initiated policy seems relatively sensitive to disk jobs. When disk jobs is over 60%, for the reasons explained in II, both sender-initiated probing and receiver-initiated probing in symmetrical policy are reduced, resulting sharp performance decrease.

4.2.6 Sensitivity to probe limit

The data presented in previous experiments assumed a probe limit of 3 for the symmetrically initiated policy. This section discusses the impact of the probe limit on the performance of the symmetrically initiated policy under the three workloads. Offered system load is set to 80% for all the nodes. The inter-arrival time CV (C_a), service time CV (C_s) and disk service time CV (C_{io}) are set to 1. The probe limit is varied between 1 to 10. Figure 4.13 shows the sensitivity of the mean response time against the probe limit for the symmetrically initiated policy.

From Figure 4.13, it can be noted that there is substantial performance difference between two I/O models and pure CPU model. Non-interleaved I/O model has smaller response time than the interleaved I/O model. The performance difference exists for the reasons discussed in the previous sections.

Figure 4.13 Performance sensitivity to the probe limit ($N=32$ nodes, $\mu=1$, $\lambda=0.8$, P_l is varied; $C_a = C_s = C_{io}=1$, $\mu_{io}=2.5$, fraction of disk jobs $\alpha = 50\%$; $T=2$, $B=2$, all other parameters are set to their default values).

The marginal benefit that results from increasing the probe limit shows that the lists in the symmetrically initiated policy accurately reflect the system state. Since most of its load transfer attempts are successful within 2 probes, further increases in probe limit do not result in performance improvement. On the other hand, setting probe limit too high would result in marginal performance deterioration (see two I/O models in Figure 4.13) due to the increased probe overhead and the uselessness of further probing. It is clear that small probe limits are appropriate.

4.3 Summary

In this chapter, we have studied the performance characteristics of three load sharing policies under three different workloads in the homogeneous distributed system.

The results in this chapter demonstrated that job's I/O requirement has strong performance influence on all three policies. Due to the nature of job disk service demand, overhead caused by job departure and delay for disk service dramatically deteriorate the system performance, resulting in poor mean job response time compared to that in the pure CPU model. The performance difference between pure CPU model and two I/O models increases with the proportion of system load. Interleaved I/O model normally has worse performance than the non-interleaved I/O model, mainly because of the overheads associated with job disk service in the interleaved model are much more than those in the non-interleaved model.

Two hierarchical policies exhibited more sensitivity to inter-arrival CV because of their receiver-initiated component. Stable symmetrical policy reduces its sensitivity to processor service time by well controlling the unnecessary probes. Disk service time CV affects the performance of three policies similarly.

Probe limit is another fundamental parameter that affects the system performance. However, results presented here indicate that small probe limits are appropriate. There is no significant performance improvement when the probe limit is increased; instead, it might cause performance deterioration because of probe overhead and the ineffectiveness of further probing.

Overall, symmetrically initiated policy provides better performance than both local and global hierarchical policies in the experiments presented in this chapter. Besides the reasons explained before, the poor performance of two hierarchical policies is also due to their relatively inefficient hierarchical search and more probing overheads. Local hierarchical policy suffers in performance because of the delay in publicizing the existence of senders and relatively fewer load sharing activities. However, this delay becomes less important when system load is high. Symmetrically initiated policy benefits from its judicious use of the knowledge gained through probing and its two probing components.

Chapter 5

Performance in Heterogeneous Distributed Systems

5.1 Introduction

This chapter considers the I/O impact on performance characteristics of three load sharing policies in the two types of heterogeneous systems. Three policies studied here are Local Hierarchical Receiver Initiated policy (LH), Global Hierarchical Receiver Initiated policy (GH) and Stable Symmetrically Initiated policy (SYM). In order to reduce the complexity of the experiments, it uses two types of heterogeneous systems [21]. In type I system, nodes have identical processing rates but the external job arrival rates are different. In type II system, nodes have different processing rates as well as external job arrival rates. Unless otherwise specified, the following default parameter values are used for this chapter's experiments.

The distributed system contains $N = 32$ identical processor nodes and $D = 8$ disk nodes interconnected by a 10 megabit/second Ethernet-like communication network. The communication, such as probing, job transferring and job departure for disk service, is

through PVM. The size of a probe message is 16 bytes. The job header size is 40 bytes. Disk service rate μ_{io} is 2.5. The percentage of IO-jobs α is fixed at 50%. Section 5.2.5 will discuss the impact of the number of IO-jobs. The IO-jobs are either all non-interleaved (NIL) or all interleaved (INL). Threshold T for all policies is fixed at 2. A static probe limit P_l of 3 is used for symmetrically initiated policy in all the experiments except in Section 6.2.6 which will present the performance sensitivity of varying the probe limits. Hierarchical policies use the default branching factor $B = 2$.

Batch strategy has been used to compute confidence intervals. Each experiment uses at least 30 batch runs with 3,000 to 6,000 jobs for each batch which based on offered system load and coefficient of variation (CV). This strategy produced 95% confidence intervals that were less than 1% of the mean response times when the system utilization is low to moderate and less than 5% for moderate to high system utilization (in most cases, up to about 90%).

5.2 Performance Analysis of Type I Systems

In type I system, nodes have identical processing rates but the external job arrival rates are different. For the following experiments, nodes are divided into class 1 and class 2 that contain 6 and 26 nodes respectively ($N_1=6$, $N_2=26$). Section 5.2.7 will discuss the impact of node distribution between class 1 and class2. The average job service rate is 1 per time unit for both classes ($\mu_1=\mu_2=1$). Note that the offered system load for class i

nodes is given by λ_i/μ_i . Since $\mu_1=\mu_2=1$, offered system load for class i nodes is equal to λ_i .

5.2.1 Performance as a function of System Load

This section discusses the performance as a function of offered system load for all three policies under the three workloads. In this experiment, the λ_1 is fixed at 0.9 and λ_2 is varied to see the performance impact of system load. The results shown in Figure 5.1 and 5.2 correspond to an inter-arrival CV (C_a), service time CV (C_s) and disk service time CV (C_{io}) of 1 for both classes.

The results presented in Figure 5.1 and 5.2 show that I/O has strong impact on the system performance for all three policies. In general, pure CPU model has the best performance among three workloads. Non-interleaved I/O model performs better than interleaved I/O model. However, the difference between them is marginal at low to moderate system load. The performance difference among the three workloads becomes noticeable with the increase of system load. This is mainly because when system moves to more balanced state from highly unbalanced state, a probing node needs to work harder to find a suitable transfer partner independent of the policy, resulting in more probing overheads. In addition, frequent job's disk services add extra burden on processing power, resulting in performance deterioration.

Figure 5.1 Performance sensitivity as a function of system load. ($N=32$ nodes, $N_1=6$, $N_2=26$, $\mu_1=\mu_2=1$, $\lambda_1=0.9$, λ_2 is varied to vary system load on class 2 nodes; $C_a=C_s=C_{io}=1$, $\mu_{io}=2.5$, fraction of disk jobs $\alpha=50\%$; $T=2$, $B=2$, $P_1=3$, all other parameters are set to their default values).

Figure 5.2 Performance sensitivity as a function of system load by policy (Parameters are set the same values as in Figure 5.1).

It also can be observed from Figure 5.1 that symmetrically initiated policy performs the best among three policies under each workload. When system load is low to moderate, due to the large heterogeneity between class 1 nodes and class 2 nodes, a receiver node in class 2 is able to locate a sender node in class 1 easily and a sender can also easily find a receiver. Since symmetrically initiated policy has both sender-initiated and receiver-initiated components, it greatly increases load sharing activities by effectively using them, resulting in good performance. While both local and global hierarchical policies only use receiver-initiated policy, the probing rate is less than that in the symmetrical policy, relatively poor performance results.

When system load is high, the difference in system load between two classes decreases. However, the heterogeneity continues to provide some room for a few successful sender-initiated probes in the symmetrical policy, generating more load sharing activities than the two hierarchical policies. In addition, probing is more effective in the symmetric policy than in the two hierarchical policies. This fact becomes more important when system is under a more balanced state because each node has to work harder to find a transfer partner. As a result, symmetrical policy outperforms both local and global hierarchical policies. The poor performance of two hierarchical policies results from their fewer loads sharing activities and relatively inefficient hierarchical probing.

The performance difference between local and global hierarchical policies is marginal. Local hierarchical policy suffers in performance due to the delay in publicizing sender information. This delay becomes less critical at higher system loads.

Figure 5.3 and 5.4 present the results of mean response time as a function of offered system load, for the fixed inter-arrival CV (C_a) and processor service time CV (C_s) of 4 for both classes. Disk service time CV (C_{io}) is remaining at 1 for both classes.

Figure 5.3 Performance sensitivity as a function of system load. ($N=32$ nodes, $N_1=6$, $N_2=26$, $\mu_1=\mu_2=1$, $\lambda_1=0.9$, λ_2 is varied to vary system load on class 2 nodes; $C_a = C_s = 4$, $C_{io}=1$, $\mu_{io}=2.5$, fraction of disk jobs $\alpha = 50\%$; $T=2$, $B=2$, $P_l=3$, all other parameters are set to their default values).

As can be expected, higher inter-arrival and processor service time CVs have significant impact on the performance for all policies. At low to moderate system loads, the performance difference between non-interleaved and interleaved I/O models is small. When system load is high, interleaved I/O model shows worse performance than the non-interleaved I/O model. One reason is that more overheads are involved in the interleaved model. Another reason is when the degree of heterogeneity reduces each node have to

work harder to find a transfer partner, resulting in fewer load sharing activities and more probing overheads involved with each search. Since interleaved I/O model has more overheads, poor performance yields. All the overheads become critical when system is heavily loaded.

Figure 5.4 Performance sensitivity as a function of system load by policy (Parameters are set the same values as in Figure 5.3).

It is also can be seen from Figure 5.2 that symmetrically initiated policy performs better than both local and global hierarchical policies across the range of system loads. Local hierarchical policy has very close performance to the global hierarchical policy. When system load is low to moderate, the heterogeneity is high. Thus, sender-initiated component in symmetrical policy works effectively so that it increases load sharing activities. In the two hierarchical policies, there is only receiver-initiated component. This, therefore, diminishes load sharing activity.

5.2.2 Sensitivity to variance in inter-arrival times

This section discusses the impact of variance in inter-arrival CV on the performance of three policies. Service time CV (C_s) and disk service time CV (C_{io}) are set to 1 for both classes. Inter-arrival time CV (C_a) is varied. Figure 5.5 and 5.6 depict the results of mean response time against inter-arrival CV (C_a).

Figure 5.5 and 5.6 clearly indicate that the hierarchical policies are more sensitive to inter-arrival CV than the symmetrical policy. Since increased inter-arrival CV implies clustered nature of job arrivals at a node, it greatly reduces the receiver-initiated probe rate, thus, reduces the load sharing activities. Since there are only 6 out of 32 nodes that are lightly loaded, reduced receiver-initiated probing in two hierarchical policies results delayed and fewer job transfers from 26 heavily loaded nodes. This moves the system based on the two hierarchical policies towards no load sharing, causing performance deterioration. In addition, slow arrival rate at class 1 nodes also delays job transfer since receiver-initiated probes are usually originate there. On the other hand, symmetrically initiated policy doesn't show such sensitivity to inter-arrival CV since sender-initiated negotiation works well in such heterogeneity.

Figure 5.5 Performance sensitivity to inter-arrival CV (C_a) ($N=32$ nodes, $N_1=6$, $N_2=26$, $\mu_1=\mu_2=1$, $\lambda_1=0.1$, $\lambda_2=0.9$; C_a is varied from 0 to 4; $C_s = C_{io}=1$, $\mu_{io}=2.5$, fraction of disk jobs $\alpha = 50\%$; $T=2$, $B=2$, $P_l=3$, all other parameters are set to their default values).

Figure 5.6 Performance sensitivity to inter-arrival CV by policy (Parameters are set the same values as in Figure 5.5).

5.2.3 Sensitivity to variance in processor service times

This section presents the impact of variance in processor service time CV (C_s) on the performance of three policies. Inter-arrival CV (C_a) and disk service time CV (C_{io}) are set to 1 for both classes. Service time CV (C_s) is varied. Figure 5.7 and 5.8 show the results of mean response time against processor service time CV (C_s) under the three workloads.

Comparing Figure 5.7 and 5.8 with two figures in Section 5.2.2, it can be noted that both local and global hierarchical policies exhibit less sensitive to service time CV than the inter-arrival CV, which indicates receiver-initiated probing is less sensitive to receiver time CV. This is because under FCFS node scheduling, when service time CV increases, larger jobs monopolize the processing power, causing increased sender-initiated probing rather than receiver-initiated probing.

The poor performance of both local and global hierarchical policies is due to delayed job transfers and relatively inefficient searches in the hierarchy. In contrast with two hierarchical policies, symmetrically initiated policy is able to control the unnecessary sender-initiated probing generated by increased service time CV based on its fairly accurate state lists. As a result, it reduces sensitivity to service time CV, keeping system performance stable across the range of tested service time CV. In addition, probes are more efficient than those in the two hierarchical policies.

Figure 5.7 Performance sensitivity to service time CV (C_s) ($N=32$ nodes, $N_1=6$, $N_2=26$, $\mu_1=\mu_2=1$, $\lambda_1=0.1$, $\lambda_2=0.9$; C_s is varied from 0 to 4; $C_a=C_{io}=1$, $\mu_{io}=2.5$, fraction of disk jobs $\alpha=50\%$; $T=2$, $B=2$, $P_l=3$, all other parameters are set to their default values).

Figure 5.8 Performance sensitivity to service time CV by policy (Parameters are set the same values as in Figure 5.7).

5.2.4 Sensitivity to variance in disk service times

This section demonstrates the impact of variance in disk service time CV (C_{io}) on the performance of three policies under two I/O workloads. Inter-arrival CV (C_a) and processor service time CV (C_s) are set to 1 for both classes. Disk service time CV (C_{io}) is varied. Figure 5.9 and 5.10 present the results of mean response time against disk service time CV (C_{io}) under the three workloads.

As it is shown on Figure 5.9 and 5.10, disk service time CV affects all three policies similarly. Symmetrically initiated policy performs better than both the local and global hierarchical policies in both non-interleaved and interleaved I/O models. In addition to the reasons discussed before, increased disk service time CV is another reason that reduces receiver-initiated probe rate. As the disk service time CV increases, jobs depart for disk service in a clustered fashion causing the jobs to return to CPU for service in a clustered manner. This affects the receiver-initiated probing the same way as increasing inter-arrival CV (see Section 5.2.2). Reduced receiver-initiated probe activities cause longer mean response time. Global hierarchical policy provides marginal better performance than local hierarchical policy.

Since overall system load is high, the sender-initiated probes are limited, although not cut off totally, in symmetrically initiated policy. Therefore, increased disk time CV has similar impact on the symmetrical policy as on the two hierarchical policies. However, symmetrically initiated policy still benefits from its limited sender-initiated load sharing, exhibiting better performance over the two hierarchical policies.

Figure 5.9 Performance sensitivity to disk service time CV (C_{io}) ($N=32$ nodes, $N_1=6$, $N_2=26$, $\mu_1=\mu_2=1$, $\lambda_1=0.1$, $\lambda_2=0.9$; C_{io} is varied from 1 to 4; $C_a = C_s=1$, $\mu_{io}=2.5$, fraction of disk jobs $\alpha = 50\%$; $T=2$, $B=2$, $P_l=3$, all other parameters are set to their default values).

Figure 5.10 Performance sensitivity to disk service time CV by policy (Parameters are set the same values as in Figure 5.9).

Figure 5.9 and 5.10 also suggest that the difference between non-interleaved and interleaved I/O models in all three policies increases with the disk service time CV. This is because increased disk CV reduces more receiver-initiated probes in the interleaved I/O model than in non-interleaved I/O model as, during each I/O job processing, there are more receiver-initiated probes fostered by job departures in the interleaved model than in the non-interleaved model. As a result, more overheads in the interleaved model greatly harm the system performance.

5.2.5 Sensitivity to disk jobs

All the data presented in the previous experiments assumed an equal proportion of CPU-intensive jobs and IO-jobs (i.e. the parameter α is set to 50%). This section considers the impact of variance in number of IO-jobs on the performance of three policies under two I/O workloads. Inter-arrival time CV (C_a), service time CV (C_s) and disk service time CV (C_{io}) are set to 1 for both classes. Figure 5.11 and 5.12 show the results of the fraction α varied between 10 and 60.

It can be observed from Figure 5.11 and 5.12 that the mean response time for all three policies increases with the proportion of IO-jobs. This result demonstrates that job's I/O requirements significantly impact the system performance. The more jobs require the disk service, the higher the mean response time.

Figure 5.11 Performance sensitivity to disk jobs. ($N=32$ nodes, $N_1=6$, $N_2=26$, $\mu_1=\mu_2=1$, $\lambda_1=0.1$, $\lambda_2=0.9$; fraction of disk jobs α is varied; $C_a = C_s = C_{io}=1$, $\mu_{io}=2.5$; $T=2$, $B=2$, $P_l=3$, all other parameters are set to their default values).

Figure 5.12 Performance sensitivity to disk jobs by policy (Parameters are set the same values as in Figure 5.11).

The actual I/O model does not significantly affect the performance until IO job fraction α is great than 50 as it is shown in Figure 5.11 and 5.12. When α is less than 50, increased receiver-initiated probe activities result from more frequent job departures for disk service in the interleaved I/O model. This generates more load sharing activities between two class nodes, resulting in close or sometime even better performance then its non-interleaved counterpart in all three policies. When α is greater than 50, high percentage IO-jobs make mean job waiting time in the queue longer, obviously resulting in longer mean response time.

The better performance of symmetrically initiated policy results from its useful sender-initiated and receiver-initiated components. There is marginal performance difference between local hierarchical policy and global hierarchical policy.

5.2.6 Sensitivity to probe limit

Experiments of the previous sections in this chapter assume a probe limit of 3 for symmetrically initiated policy. This section examines the effect of varying the probe limit on the performance of the symmetrically initiated policy. Inter-arrival time CV (C_a), service time CV (C_s) and disk service time CV (C_{io}) are set to 1 for both classes. Figure 5.13 depicts the results of varying probe limit P_l between 1 and 10 for symmetrical policy under non-interleaved I/O model and interleaved I/O model.

The data presented in Figure 5.13 shows that the performance of symmetrically initiated policy do not improve beyond a probe limit of 2. This is mainly because three state lists (sender, OK and receiver lists) in symmetrically initiated policy accurately

reflect the system state so that most of its load transfer attempts are successful within 2 probes. Further increases in probe limit do not improve performance. It also can be easily noted from Figure 5.7 that setting probe limit too high would result in marginal performance deterioration (see two I/O models in Figure 5.7) caused by increased probe overheads and the uselessness of further probing. It is clear that small probe limits are appropriate.

Figure 5.13 Performance sensitivity to probe limit. ($N=32$ nodes, $N_1=6$, $N_2=26$, $\mu_1=\mu_2=1$, $\lambda_1=0.1$, $\lambda_2=0.9$; P_l is varied; $C_a = C_s = C_{io}=1$, $\mu_{io}=2.5$; $T=2$, $B=2$, $\alpha=50\%$, all other parameters are set to their default values).

Pure CPU model performs better than both two I/O models in Figure 5.13, which indicates the overheads associated with job I/O service affects the system performance. Under such heterogeneity, the actual I/O model does not significantly affect system performance since receiver-initiated and sender-initiated components can both work

effectively. Extra overheads from more job departures in the interleaved I/O model are compensated for by the increased load sharing activities resulting from increased receiver-initiated probing.

5.2.7 Sensitivity to the degree of heterogeneity

This section considers the impact of degree of heterogeneity on the performance for all three policies by varying the number of nodes in class 2. The remaining nodes belong to class 1 group. Inter-arrival time CV (C_a) and service time CV (C_s) are set to 4 for both classes. Disk service time CV (C_{io}) are set to 1 for both classes. Figure 5.14 and 5.15 show the results of varying number of class 2 nodes from 1 to 31. Since class 1 nodes are lightly loaded, the mean response time increased as N_2 increases.

It can be observed from Figure 5.14 and 5.15 that, job's I/O requirements continues to affect the system performance in all three policies. Pure CPU model yields better performance over both non-interleaved and interleaved I/O models in all circumstance. Between latter two, non-interleaved I/O model performs better than the interleaved model. The difference between the two I/O models increases with the increase in the number of class 2 nodes. This is because, when number of class 2 node increases, the overall system load increases. Therefore, the overhead associated with job I/O service greatly deteriorates the system performance. Since interleaved model involves more job departure for disk service, it has more overheads, poor performance results.

Figure 5.14 Performance sensitivity to the degree of heterogeneity. ($N=32$ nodes, N_2 is varied, $N_1=32-N_2$, $\mu_1=\mu_2=1$, $\lambda_1=0.1$, $\lambda_2=0.9$; $C_a=C_s=4$, $C_{io}=1$, $\mu_{io}=2.5$; $T=2$, $B=2$, $P_i=3$; $\alpha=50\%$, all other parameters are set to their default values).

Figure 5.15 Performance sensitivity to the degree of heterogeneity by policy (Parameters are set the same values as in Figure 5.14).

Symmetrically initiated policy outperforms both local and global hierarchical policies for the whole range of varied class 2 nodes. When the number of class 2 nodes N_2 is smaller, the probability of finding a receiver is high since most of the nodes are idle. Thus, receiver-initiated probes generally fail. Sender-initiated probes, however, often succeed based on accurate receiver list in the symmetrical policy. More load sharing results in better performance. When the number of two class nodes are close, both receiver-initiated and sender-initiated probes have equally opportunity to find suitable senders and receivers. Since the symmetrical policy has both components, it generates more load sharing activities, resulting in a better performance. When number of class 2 nodes is higher, receiver-initiated probes are more efficient since there are more senders than receivers. Nevertheless, since the number of lightly loaded class 1 nodes where receiver-initiated probes originated reduces, the receiver-initiated load sharing activities are reduced. This fact becomes more important in the two hierarchical policies that only have receiver-initiated probe component. Consequently, their performance deteriorates rapidly. Since symmetrically initiated policy has two probing components, it benefits from dynamically using both of them so that it is able to remain stable.

5.3 Performance Analysis of Type II Systems

In type II system, nodes have different processing rates as well as different external job arrival rates. For the following experiments, nodes are divided equally between the two node classes ($N_1 = N_2 = 16$). Impact of node distribution will be studied in section 5.3.2. The offered system load is maintained the same for both classes. The job arrival

rate of class 1 is set at half of that for class 2 nodes ($\lambda_1 = \lambda_2/2$) and the processing rate of class 1 nodes is maintained at half of that for class 2 nodes ($\mu_1 = \mu_2/2$) as well. Thus, class 2 nodes are twice as fast as class 1 nodes. μ_1 is set to 0.5 and μ_2 is set to 1.0.

5.3.1 Performance as a function of System Load

This section presents results of mean response time against offered system load in type II heterogeneous system. Inter-arrival time CV (C_a) and service time CV (C_s) are set to 4 for both classes. Disk service time CV (C_{io}) are set to 1 for both classes. λ_2 is varied. Figure 5.16 and 5.17 shows mean response time as a function of the offered system load for the three policies under the three workloads.

It can be observed from Figure 5.16 and 5.17 that symmetrically initiated policy outperforms both the local and global hierarchical policies over the whole range of system load. Since the job arrival rate of class 2 nodes is twice as fast as that of the class 1 nodes, they have a higher probability of being in the sender state than class 1 nodes. Thus, it is easier to find sender nodes in class 2, resulting in more jobs being transferred to class 1 nodes. These job transfers can be achieved through receiver-initiated probing in the two hierarchical policies and both sender-initiated and receiver-initiated probing in symmetrically initiated policy. Obviously, since more load sharing activities occur in the symmetrically initiated policy, it has the lowest mean response time.

Local hierarchical policy performs worse than global hierarchical policy when system load is low for the reasons discussed before. All other observations also hold for the type II systems.

Figure 5.16 Performance sensitivity to offered system load. ($N=32$ nodes, $N_1=N_2=16$, $\lambda_1=\lambda_2/2$, λ_2 is varied; $\mu_1=0.5$, $\mu_2=1.0$; $C_a=C_s=4$, $C_{io}=1$, $\mu_{io}=2.5$; $T=2$, $B=2$, $P_l=3$; $\alpha=50\%$, all other parameters are set to their default values).

Figure 5.17 Performance sensitivity to offered system load by policy (Parameters are set the same values as in Figure 5.16).

5.3.2 Sensitivity to the degree of heterogeneity

This section describes the impact of the degree of heterogeneity on system performance. Figure 5.18 and 5.19 shows the mean response time against number of class 1 nodes (N_1). The number of class 2 nodes N_2 is given by $32 - N_1$. Job arrival rate for two classes are set to 0.4 and 0.8 respectively ($\lambda_1=0.4$, $\lambda_2=0.8$). Inter-arrival time CV (C_a) and service time CV (C_s) are set to 4 for both classes. Disk service time CV (C_{io}) are set to 1 for both classes. Number of class 1 nodes N_1 is varied to vary the heterogeneity. Since class 1 nodes are slow nodes, the mean response time increases with the number of class 1 node N_1 increases.

The results presented in Figure 5.18 and 5.19 show that the performance of all three policies is affected by job's I/O requests. Pure CPU model has lowest mean response time over interleaved and non-interleaved I/O models. Non-interleaved I/O model outperforms interleaved I/O model because of less job I/O service overheads.

Two hierarchical policies are sensitive to the number of class 1 nodes N_1 . Their performance decreases with the number of slow nodes in class 1. The poor performance of both local and global hierarchical policies is due to relatively ineffective random probes and more probing overheads. Also as high inter-arrival CV reduces receiver-initiated probing rate, resulting in fewer load sharing activities in two hierarchical policies. Symmetrically initiated policy seems to show no sensitivity to the number of slow nodes. It provides the best performance and remains system stability for the whole range of tested condition.

Figure 5.18 Performance sensitivity to the degree of heterogeneity. ($N=32$ nodes, N_1 is varied, $N_2=32-N_1$, $\lambda_1=0.4$, $\lambda_2=0.8$; $\mu_1=0.5$, $\mu_2=1.0$; $C_a=C_s=4$, $C_{io}=1$, $\mu_{io}=2.5$; $T=2$, $B=2$, $P_l=3$; $\alpha=50\%$, all other parameters are set to their default values).

Figure 5.19 Performance sensitivity to the degree of heterogeneity by policy (Parameters are set the same values as in Figure 5.18).

5.4 Summary

In this chapter, we have studied the performance characteristics of three load sharing policies under three different workloads in the two types of heterogeneous distributed systems.

As in homogeneous systems, job's I/O requirements have strong impact on all policies' performance in both type I and type II heterogeneous systems. Unavoidable overheads associated with job disk service, such as transfer overheads, disk contention and waiting time in the job queue, etc., contribute to the poor performance of two I/O models when compared to the pure CPU model.

High inter-arrival CV, service time CV and disk service time CV also greatly affect the performance of the three load sharing policies. Two hierarchical policies show sensitivity to the high inter-arrival CV due to the job arrival in a clustered fashion. Stable symmetrically initiated policy is able to control unnecessary probing at high service time CV. As a result, it exhibits reduced sensitivity to service time CV. Disk service CV affects the three policies similarly. The small probe limits are proved to be appropriate.

Overall, symmetrically initiated policy outperforms both local and global hierarchical policies due to its effective probes and judicious use of the knowledge gained through probing.

Chapter 6

Conclusions

6.1 Summary

This thesis studied the impact of job's I/O requirements on the performance of three different load sharing policies. The workload models used in the simulation tests are pure CPU model, non-interleaved I/O model and interleaved I/O model.

Chapter 1 introduced the background and motivation of load sharing in distributed systems. It identified categories of load sharing policies, discussed characteristics of load sharing and addressed the different components of typical load sharing policies.

In Chapter 2, a broader range of dynamic load sharing policies were presented and discussed to give an overview of research activities in load sharing areas. The load sharing policies and system and workload models used in the experiments were detailed in Chapter 3.

Chapter 4 presented the performance results of the three load sharing policies in homogeneous system over a variety of parameters. Three different workloads were used

for most of the experiments. A summary of the results was given at the end of this chapter.

Chapter 5 presented the performance results of the three load sharing policies in two types of heterogeneous system. Three workload models were examined there as well. A summary was given at the end.

6.2 Future Work

This thesis studied I/O impact on the performance of three different load sharing policies in both homogeneous and heterogeneous systems. Three workload models are used to compare the performance difference among these policies. However, all the policies assumed non-preemptive selection policy. Further work is needed to investigate the performance difference when preemptive selection policy is used.

Hierarchical policies studied in this thesis used a branch factor of 2. It has been recognized that performance of the hierarchical policies is sensitive to the branching factor of the hierarchy [6]. More research is needed to cover the gap left and to further analyze the impact of the branch factor on the performance of the hierarchical policies under the two I/O workloads.

Future work also includes studying other types of jobs, such as parallel jobs and memory intensive jobs, besides CPU-intensive jobs and IO-intensive jobs.

Chapter 7

References

- [1] K. Agrawala and S. K. Tripathi, "On the optimality of semidynamic routing schemes", *Inform. Processing Lett.*, Vol. 13, No. 1, pp. 20-22, Oct. 1981.
- [2] D. R. Cheriton, "The V kernel Distributed Operating System", *Commun. ACM*, vol. 1, pp. 105-115, Feb. 1985.
- [3] S. P. Danadamudi, "Sensitivity Evaluation of Dynamic Load Sharing in Distributed Systems", *IEEE Concurrency*, Vol. 6, pp. 62-72, Jul.-Sept. 1998.
- [4] S. P. Dandamudi, "Performance Impact of Scheduling Discipline on Adaptive Load Sharing in Homogeneous Distributed Systems", *IEEE International Conference on Distributed Computing Systems*, pp. 484-492, Vancouver, May 1995.
- [5] S. P. Dandamudi and H. Hadavi, "Impact of I/O on Dynamic Load Distribution in Homogeneous Distributed Systems", *International Journal of Computers and their Applications*, Vol. 5, pp. 150-164, Sept. 1998.
- [6] S. P. Dandamudi and M. Lo, "Hierarchical Load Sharing Policies for Distributed Systems", Tech. Rep. SCS-96-1, School of Computer Science, Carleton University, Ottawa, Canada, 1996.
- [7] S. P. Dandamudi and M. Lo, "A Comparative Study of Adaptive and Hierarchical Load Sharing Policies for Distributed Systems", *Proc. International Conference on Computers and Their Applications*, Honolulu, Hawaii, Mar. 1998.
- [8] D. L. Eager, E. D. Lazowska and J. Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems", *IEEE Transactions on Software Engineering*, SE-12, No. 5, pp. 662-675, May 1986.

- [9] D. L. Eager, E. D. Lazowska and J. Zahorjan, "A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing", *Performance Evaluation*, Vol. 6, No. 1, pp. 53-68, Mar. 1986.
- [10] D. L. Eager, E. D. Lazowska and J. Zahorjan, "The Limited Performance Benefits of Migrating Active Processes for Load Sharing", *ACM Sigmetrics Conference*, pp. 63-72, 1988.
- [11] H. Kobayashi, "Modeling and Analysis: An Introduction to System Performance Evaluation methodology", Addison Wesley, 1981.
- [12] P. Krueger and R. A. Finkel, "An adaptive load balancing algorithm for a multicomputer", Department of Computer Science, University of Wisconsin, Madison, Tech. Rep. 539, Apr. 1984
- [13] P. Krueger and M. Livny, "The Diverse Objectives of Distributed Scheduling Policies", *Proceedings of 7th International Conference on Distributed Computing Systems*, pp. 242-249, Sept. 1987.
- [14] P. Krueger and M. Livny, "A Comparison of Preemptive and Non-Preemptive Load Distributing", *IEEE International Conference on Distributed Computing Systems*, pp. 123-130, 1988.
- [15] T. Kunz, "The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme", *IEEE Transactions on Software Engineering*, Vol. 17, No. 7, pp. 725-730, Jul. 1991.
- [16] S. S. Lam and L. Kleinrock, "Packet switching in a multiaccess broadcast channel: Dynamic control procedures", *IEEE transactions on Communication*, Vol. COM-23, pp. 891-904, Sept. 1975.
- [17] M. J. Litzkow, M. Livny and M. Melman, "Condor – A Hunter of Idle Workstations", *Proc. 8th International Conference on Distributed Computing Systems*, pp. 104-111, Jun. 1988.
- [18] M. Livny and M. Melman, "Load Balancing in Homogeneous Broadcast Distributed Systems", *Proc. ACM Computer Network Performance Symp.*, pp.47-55. Proceedings printed as a special issue of *ACM Performance Evaluation Rev.*, Vol. 11, No. 1, 1982, pp. 47-55, 1982.
- [19] M. Lo and S. P. Dandamudi, "Performance of Hierarchical Load Sharing in Heterogeneous Distributed Systems", *Proc. International Conference on Parallel and Distributed Computing Systems*, Dijon, France, 1996.

- [20] S. Majumdar and Y. M. Leung, "Characterization of Applications with I/O for Processor Scheduling in Multiprogrammed Parallel Systems", *Proc. IEEE Symp. Parallel Distributed Processing*, pp. 298-307, Dallas, 1994.
- [21] R. Mirchandaney, D. Towsley and J. Stankovic, "Adaptive Load Sharing in heterogeneous Distributed Systems", *J. Parallel and Distributed Computing*, Vol. 9, pp. 331-346, 1990.
- [22] L. M. Ni and K. Abani, "Nonpreemptive Load Balancing in a class of local area networks", *Proc. Computer Networking Symp.*, Dec. 1981.
- [23] N. G. Shivaratri and P. Krueger, "Two Adaptive Location Policies for Global Scheduling", *Proc. 10th International Conference on Distributed Computing Systems*, IEEE CS Press, Los Alamitos, CA, Order No. 2048, pp. 502-509, 1990.
- [24] N. G. Shivaratri, P. Krueger and M. Singhal, "Load Distributing for Locally Distributed Systems", *IEEE Computer*, Vol. 25, No. 12, pp. 33-44, Dec. 1992.
- [25] M. M. Theimer and K. A. Lantz, "Finding Idle Machines in a Workstation-Based Distributed System", *IEEE International Conference on Distributed Computer Systems*, pp. 112-122, 1988.
- [26] F. A. Tobagi and L. Kleinrock, "Packet switching in radio channels: Part IV – Stability considerations and dynamic control in carrier sense multiple access", *IEEE Transactions on Communication*, Vol. COM-25, pp. 1103-1119, Oct. 1977.
- [27] T. Yum, "The design and analysis of a semidynamic deterministic routing rule", *IEEE Transactions on Communication*, Vol. COM-29, pp. 498-504, Apr. 1981.

IMAGE EVALUATION TEST TARGET (QA-3)

APPLIED IMAGE, Inc.
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989

© 1993, Applied Image, Inc., All Rights Reserved