# An Evaluation of Load Balancing Algorithms for Distributed Systems

by

Kouider Benmohammed-Mahieddine

Submitted in accordance with the requirements for the degree of
Doctor of Philosophy

The University of Leeds
School of Computer Studies
October, 1991

The candidate confirms that the work submitted is his own and that appropriate credit has been given where reference has been made to the work of others.

# Abstract

Distributed systems are gradually being accepted as the dominant computing paradigm of the future. However, due to the diversity and multiplicity of resources, and the need for transparency to users, global resource management raises many questions. On the performance level the potential benefits of the load balancing in resolving the occasional congestion experienced by some nodes while others are idle or lightly loaded are commonly accepted. It is also acknowledged that no single load balancing algorithm deals satisfactorily with the changing system characteristics and dynamic workload environment.

In modelling distributed systems for load balancing, optimistic assumptions of system characteristics are commonly made, with no evaluation of alternative system design options such as communications protocols. When realistic assumptions are made on system attributes such as communication bandwidth, load balancing overheads, and workload model, doubts are cast on the capability of load balancing to improve the performance of distributed systems significantly.

A taxonomy is developed for the components as well as the attributes aspects of load balancing algorithms to provide a common terminology and a comprehensive view to load balancing in distributed systems. For adaptive algorithms the taxonomy is extended to identify the issues involved and the ways of adding adaptability along different dimensions. A design methodology is also outlined. A review of related work is used to identify the most promising load balancing strategies and the modelling assumptions made in previous load balancing studies. Subsequently the research problems addressed in this thesis and the design of new algorithms are detailed.

A simulated system developed to allow an experimentation with various load balancing algorithms under different workload models and system attributes is described. Based on the nature of the file system structure and the classes of nodes processing speed involved, different models of loosely-coupled distributed systems can be defined. Four models are developed: disk-based homogeneous nodes, diskless homogeneous nodes, diskless heterogeneous nodes, and disk-based heterogeneous nodes. The nodes are connected through a broadcast transfer device.

A set of representative load balancing algorithms covering a range of strategies are evaluated and compared for the four models of distributed systems. The algorithms developed include a new algorithm called *Diffuse* based on explicit adaptability for the homogeneous systems. In the case of heterogeneous systems, novel modifications are made to a number of algorithms to take into account the heterogeneity of nodes speed. The evaluation on homogeneous systems is two-fold: an assessment of the effect of system attributes on the performance of the distributed system subject to these algorithms, and a comparison of the relative merits of the algorithms using different performance metrics, and in particular a classification of the performance of the Diffuse algorithm with regard to others in the literature. For the heterogeneous systems the performance of the adapted algorithms is compared to that of the standard versions and to the no load balancing case.

As a result of this evaluation, for a set of combinations of performance objectives, distributed system attributes, and workload environment, we identify the most appropriate load balancing algorithm and optimal values for adjustable parameters of the algorithm.

# Acknowledgements

I wish to express my deepest gratitude to Professor Peter Dew, Head of School, for his sound advice and constant encouragement. His patience will always be remembered.

Many colleagues in the School of Computer Studies helped me in this project. I would like to thank especially David Mallon, for proofreading many sections of this report, Mourad Kara, for his comments on the models investigated, and Neil Bowers, for his practical help on the UNIX environment.

I acknowledge the financial support of the National Institute of Electricity and Electronics (INELEC), Boumerdes, Algeria.

Finally, I dedicate this thesis to my wife Lalia, my daughter Souhila, and my sons Adel and Tarik. They were a constant reminder of the joy of living during the darkest moments of the PhD journey.

# Table of Contents

# CHAPTER 1

# Introduction

## 1.1. Motivation

With the advent of distributed systems, it is becoming possible to get the maximum out of a set of computing nodes through a dynamic workload redistribution to avoid the situation where some hosts are idle while others have multiple jobs queued up [Livny84]. The drive behind this load balancing is two-fold: efficiency and extensibility. The current advances in computer and communication technology make a multi-computer approach cheaper than a mainframe solution, of the same performance, provided all the computing resources are used efficiently. The extensibility aspect of a distributed system should provide for the addition of new processors as the user needs arise. A well designed load balancing scheme aims at accommodating both of these aspects. The goal of the design of such systems is to redistribute the global service demands generated at different workstations over the dynamically available computing resources.

The potential benefits of dynamic load redistribution to resolve the occasional congestion experienced by some nodes to improve the overall system performance, are commonly accepted [Zhou87] for distributed computer systems. However, no single load balancing algorithm deals satisfactorily with the various and rapidly changing system conditions, and the lack of up-to-date system state information. A load balancing algorithm consists of two elements: information and control. The information element exchanges and maintains information about the state of the distributed system. Different approaches as to what information, how much information is to be maintained, how often it is to be updated, and how large is the balancing region

involved, have been proposed. The load index and its measuring mechanism have also been the subject of many investigations. The control element uses this information to decide *when* it is advantageous to redistribute the load, *who* makes this decision, *which* process to transfer or migrate, and *where* to transfer a process to reduce congestion and improve performance.

A number of algorithms have been reported in the literature. They differ in the performance objectives sought, the nature of their information and control elements, the attributes of the system model used as a test bed, and the simplifying assumptions made to aid the analysis or the implementation of the simulation model. Under these models and assumptions substantial job mean response time improvements at acceptable costs are reported [Zhou88].

On the modelling for performance study, the distributed system is commonly assumed to include homogeneous computing nodes and to be based on either a shared file server or each node having its own local file system. An optimistic view is often taken on some essential system characteristics such as job transfer delays as pointed out in [Mirchandan89], load balancing overheads, and workload environment. No evaluation of system design alternatives (e.g. communication protocols) has been reported.

To gain accurate insights into the performance of load balancing in distributed systems, more realistic system characteristics need to be taken into account in terms of both load balancing overheads and system model attributes. We set out to examine the validity of the assumptions made on distributed system attributes in previous load balancing studies, and to assess the relative performance order of some common load balancing algorithms when more realistic models are assumed.

## 1.2. The Problem

In previous load balancing studies, a common approach is to use a simple model of the distributed system with assumptions such as large communication bandwidth, negligible load balancing overheads, homogeneous workload, and to search for complex load balancing algorithms whose viability is questionable [Eager86] and which might provide only little or no gain when evaluated on realistic systems. The general approach taken in this project is to evaluate the effect of distributed system attributes and workload model on the performance of representative load balancing strategies.

The model extensions are added to represent features that were assumed negligible or require a greater level of details. Other extensions not covered in common system models are added to model actual systems more accurately and reflect the different distributed system architectures. The variations involve the evaluation of the effects on load balancing performance of different system characteristics design options, such as communication protocols.

To determine the interdependence of system attributes and load balancing algorithm performance, we examine distributed systems along three paths 1) modelling of distributed system attributes including: *file system structure*, *nodes configuration*, *communication network*, 2) *workload nature* including: homogeneous users with homogeneous jobs, heterogeneous users with homogeneous jobs, and homogeneous users with heterogeneous jobs, and 3) the design and evaluation of *load balancing algorithms* that take into account realistic system attributes and non-negligible *load balancing overheads* introduced by load balancing activities.

The actual system being modelled consists of a network of workstations interconnected by local area networks with individual, independent, sequential jobs

arriving at each autonomous processor. For a realistic distributed system model, the analytical approach is not suitable due to model complexity, whilst prototyping requires costly equipment/environment. A more appropriate approach for the study of load balancing performance on distributed computer systems is simulation. We carry out various performance studies of a set of representative load balancing algorithms on four simulated models of loosely-coupled distributed systems: diskless homogeneous nodes, disk-based homogeneous nodes, diskless heterogeneous nodes, and disk-based heterogeneous nodes. The objectives of this performance study are:

- To measure the effect of system attributes on the performance of load balancing and in particular to classify the performance of a novel algorithm called *Diffuse* with regard to others in the literature. These attributes include the file system structure and the communication bandwidth.
- To evaluate the effect of the workload model on the performance of load balancing algorithms.
- To measure the effect of the heterogeneity of nodes speed on the performance of standard load balancing algorithms and to assess the performance improvements made when adapted versions of these algorithms are used.

As a result of this evaluation, for a set of combinations of performance objectives, distributed system attributes, and workload environment, we identify the most appropriate load balancing algorithm and the optimal values for the adjustable parameters of the algorithm.

## 1.3. Thesis Organisation

In Chapter 2, a literature review on the issue of load balancing in distributed systems is presented. This is addressed at two levels: the modelling of distributed systems and the design of load balancing strategies for such environments. The results of the review are organised under a taxonomic structure for the algorithms components and attributes. A closer look at the adaptability attribute has been taken. A framework and design methodology for adaptive scheduling are developed for the case of a rapidly changing environment.

In Chapter 3, as the result of the review of previous performance studies, the research problems addressed in this thesis are defined. This includes the design of the Diffuse algorithm and the algorithm versions adapted to a heterogeneous system.

Chapter 4 describes the experimental system. First the purpose of the system design is defined and the experimental models and factors identified. Four models are considered: systems with homogeneous diskless nodes, systems with homogeneous disk-based nodes, systems with heterogeneous diskless nodes, and systems with heterogeneous disk-based nodes. Then the performance studies undertaken are outlined. The distributed systems on which this evaluation is targeted are modelled and a simulated implementation is described. This is followed by an overview of the simulation package and the simulation model calibration and validation.

In Chapter 5, the results of various performance studies undertaken on the four models to evaluate the effect of different system attributes and workload models are presented and compared to related work.

In Chapter 6, we summarise the results obtained and offer our conclusions. We then discuss future work related to this research.

# CHAPTER 2

## Survey of Load Balancing Algorithms in Distributed Systems

### 2.1. Introduction

In this chapter we review the literature that has been published in the area of interest to this thesis. This can be divided into five main areas.

(i) A motivation for dynamic load balancing in distributed systems is given. Then the previous work on the *modelling of distributed systems* and the *design of load balancing strategies* for such systems are surveyed.

(ii) The load balancing strategies reported in the literature are organised under a taxonomic structure to show how different researchers addressed the same load balancing issues. The algorithm attributes are also considered.

(iii) Special attention is given to a distributed system characterised by a rapidly changing environment. For such an environment adaptive scheduling is the way forward to maintain a consistent level of performance. The fundamentals of adaptive scheduling are described and a methodology for the design of adaptive load balancing algorithms is outlined.

(iv) A selection of load balancing algorithms is detailed. The criteria used is to select load balancing policies which include different algorithms components/attributes identified in the taxonomy and with most promising performance gain. Various algorithms included in the previous comparative studies are also described.

(v) Comparative studies of load balancing and load balancers implementation are reviewed. The results of this survey are summarised in Section 2.7.

## 2.2. Load Balancing in Distributed Systems

In a distributed system characterised by a resource multiplicity combined with a stochastic nature of the workload [Kleinrock85, Ezzat86, Cheriton88], there is a high probability for the occurrence of the 'wait while idle' state whereby some hosts in the pool are idle while other hosts have multiple jobs queued up [Livny84, Theimer85]. In his profiling of a network of workstations, Mutka has shown that processors are idle 70% of the time [Mutka87] while Livny showed that this probability depends on the system load and the size of the pool, and that load balancing can improve performance even in the case of homogeneous process arrival rates [Livny84]. This environment is characterised by changing and uneven loads on the hosts and a lack of up-to-date system state information. For an effective use of the resource multiplicity inherent in such systems, and to satisfy the diverse and sometimes conflicting users' performance expectations, the design of efficient distributed scheduling algorithms and mechanisms for processor allocation has been a research challenge for over a decade [Wang85, Casavant88, Goscinski90]. These algorithms deal with the global scheduling of system workload through local and remote job placement, while allocation of local resources is left to the local scheduling component.

Although the common objective of load balancing is to improve the performance of the computer system, the nature of the performance objective differs with the computing environment involved:

- For a *general purpose distributed computer system* based on a local area network, it is to reduce the average system response time with a minimum degradation of the performance for individual users. An alternative objective is greedy scheduling where each job is allocated to the node where it has the best response time regardless of the effect on other jobs [Bryant81, Stankovic84].

- For a *real time distributed system*, it is to provide a guaranteed response time.
- For a *parallel computer system*, it is to reduce the total execution time of a program composed of several modules.

In this review load balancing is addressed at two levels. On the distributed system level different architectural models and perspectives are surveyed. The communication model assumed is based on the broadcast device. On the algorithm level, first previous work on dynamic algorithms is reviewed then the approaches to the adaptability problem are considered.

### 2.2.1. Distributed Systems Modelling

The common approach adopted in computer science is modelling then building [Power89]. Three architectural models of distributed systems have been identified [Tanenbaum85, Coulouris88]: *workstation/server model* [Ezzat86], *processor pool model* [Needham82, Mullender86], and *integrated model* [Walker83]. In a workstation/server model single-user computers or workstations are provided. Most user needs are handled by his workstation, however expensive services such as file servers, high quality printers are shared. In the processor pool model application programs are executed within a set of computers managed by a processor service. The user needs only a terminal connected to the network to use the system. A *hybrid model* which combines features of both previous models has emerged to overcome the disadvantages in each. In the integrated model each computer can perform the role of server and application processor. The software at each computer is similar to that of a centralised multi-user system.

The communication network can be designed along two models [Livny84, Theimer88]: broadcast with the multicast as a variant, and point-to-point or store-and-forward. The store-and-forward model was first used in the wide area type of

networks and recently in mesh-connected parallel systems such as hypercubes and a pool of transputers. It is also popular as a general purpose model in the simulation of a pool of processors (e.g. Manhattan networks), and for analytical studies. The broadcast model is used for multicomputer systems and networks of workstations.

In this study we concentrate on loosely-coupled general purpose distributed systems. These systems consist of a collection of homogeneous or heterogeneous, autonomous processors connected by a local area network and operating in a cooperative fashion. This network of processors can be shared in two ways; either to improve system performance by relieving overloaded nodes through remote execution of part of their load on less loaded nodes, or by using a set of nodes for cooperative work on a single distributed computation. The nodes can be assumed to be publically or privately owned. In the case of a privately owned node, a priority for local processes is required if the local user is not to be penalised. Each processor in the distributed system is managed by a replicated copy of the system kernel with associated communication protocols and load distributing software (i.e. a distributed scheduler).

Distributed systems have been studied from several perspectives; based on the intended objective, different modelling approaches are appropriate. Mathematical modelling techniques have been used for formal specifications and verification [Hoare85, Broy87], and analysis based on queuing theory [Krueger87, Mitrani87]. When the objective is standardisation, a documentation such as ANSA standards [ANSA87] is used. In an approach similar to the PMS notation [Siewiorek82] (computer architecture context), a model based on the concept of BP processes [Jesty88] has been developed. It is a graphical representation used for an abstraction of the services offered by a distributed system to give a precise view of these services to the user and allow a comparison of different systems using a common terminology. Textbooks and system survey papers provide an informal description of distributed

systems [Coulouris88, Tanenbaum85]. The approach most favoured for performance evaluation purposes is simulation. Using simulation techniques the essential system features are abstracted and a model built to evaluate the system performance and behaviour, and test different solutions [Zhou88, Johnson88, Stankovic84a].

### 2.2.2. Load Balancing Strategies

There are so many aspects in the study of processor allocation schemes that it is of paramount importance to limit the class of algorithms and issues to be addressed, and make a precise description of the system to which they apply and the environment in which they will be tested. In this review, we concentrate on the policy aspects that deal with the collection of load information, process transfer, negotiation for an adaptive allocation of processes at the global level, based on the remote execution mechanisms described in the literature [Smith88].

Three approaches to workload scheduling on a computer system composed of multiple processors have been investigated: co-scheduling, clustering, and load balancing [Tanenbaum85]. In the clustering approach several communicating processes are assembled on the same node to reduce the communication overhead [Stankovic84]. In co-scheduling the opposite approach is taken [Ousterhout82]. The concept of distributed group is used. The members of the same group are spread over the network to exploit the concurrency among the nodes. These two approaches apply to task structure allocation. In the case of independent jobs scheduling, the favoured approach is to dynamically transfer jobs from heavily loaded hosts to lightly loaded hosts in order to improve the overall performance. The resulting form of computing is called load distributing [Eager86, Wang85, Alonso86]. It refers to both load sharing whose goal is to keep all computing nodes busy, and load balancing which attempts to have an equal load on all the nodes. The design of a load redistributing algorithm

depends on the performance objectives sought and the appropriate redistribution approach. The ultimate goal of these strategies is to minimise the system average and standard deviation of the response time with minimum adverse effect on individual users. The contributions to the design of load balancing algorithms can be organised into three categories:

**1) Static Algorithms**

These algorithms aim at finding an optimal assignment of tasks by clustering or co-scheduling, and is achieved by balancing the system loads periodically. They assume that the process behaviour is known and use graph theory models to attempt a fair distribution of the load [Efe82, Lo84]. The allocation decisions of the system components are based on pre-determined parameters. Early work on load balancing has been carried out along this approach but due to inherent drawbacks such as 1) the static nature of the algorithm does not allow these strategies to respond to short-term fluctuations in workload, 2) they require too much information such as arrival time and execution cost of each job or module to be implementable, and 3) they involve intensive computation to obtain the optimal schedule [Zhou87a]; the research effort has recently concentrated on the two other heuristic approaches which are implementable and achieve promising results. Quasi-static algorithms are a variant of this category. These algorithms ignore the current state of the system, but they tune their decision variables by adapting to slowly changing system characteristics such as the arrival rates of jobs[Green88].

**2) Dynamic Algorithms**

Here scheduling is seen as a job routing problem. These algorithms balance the loads upon the arrival of each job. This is achieved by a continuous assessment of the system load which is dynamic and unpredictable. The allocation of the job is done in real time following a fixed policy based on the recent system state information and

currently perceived system load imbalance [Zhou88] or base their decisions on statistical averages. Extensive research work has been done in this category [Zhou88].

**3) Adaptive Algorithms**

Scheduling in this approach can be interpreted as an adaptive control problem. These algorithms, like dynamic algorithms, balance loads upon the arrival of each job, but also balance loads whenever anomalies appear in the workload of the system or individual nodes. They exhibit more flexibility by adjusting their policy to match the dynamic system characteristics. In the literature some algorithms with different degrees and approaches of adaptability have been reported [Barak85, Shamir87, Krueger88]. To support adaptability, most of these algorithms use preemptive scheduling.

Although the term dynamic scheduling and adaptive scheduling have often been used interchangeably in the literature by grouping any policy that is not static under the heading of dynamic, there is a clear distinction between the two. A dynamic algorithm has a fixed policy in dealing with its dynamic environmental inputs, whereas an adaptive algorithm uses the environmental stimuli to modify the scheduling policy itself [Casavant88].

## 2.3. Taxonomy of Load Balancing Algorithms

Given the many dimensions involved in global scheduling of a distributed system, Casavant *et al.* [Casavant88] have addressed the needs for a taxonomy of the distributed scheduling algorithms reported. In the taxonomy, distributed scheduling is addressed as a resource management problem. Using this taxonomy the algorithms of interest in this study can be classified as global, dynamic, distributed, cooperative, sub-optimal, heuristic, adaptive, and have load balancing as a global objective. They may involve one-time assignment (non-preemptive) or dynamic reassignment (preemptive) of processes. They may also include some probabilistic components

[Bryant81, Hsu86, Chow86]. These algorithms are an example of distributed computation and involve the concepts of optimisation, adaptability and distributed decision-making. The load balancing approach was not fully investigated.

Three subclasses of these algorithms can be identified based on the communication model they assume: broadcast with various random and polling techniques which reduce the cost of indiscriminate broadcasting, point-to-point with various nearest neighbours techniques, and a third category of algorithms which are communication model independent (i.e. topology independent [Ni85]).

In this review we focus on the load balancing approach to resource management. A study of the load balancing literature reveals that a large number of design dimensions are involved in a load balancing algorithm and that there is no agreed upon terminology. In a previous work, Wang and Morris [Wang85] used mathematical techniques to categorise ten representative algorithms. The algorithms are categorised as *source-initiative* or *server-initiative*. A range of several information dependency levels involved are used to further classify the algorithms. Using a Q-factor (quality of load sharing) the performance of these algorithms is ranked. For negligible communication costs they show that at the same information level server-initiative algorithms outperform source-initiative algorithms.

In this work we take a broader view and cover several aspects of a load balancing scheme. A taxonomy can be made based on the approach taken to implement each component (i.e. load balancing activities) of the algorithm, or it can be based on the attributes of the algorithm which can apply to more than one component of the algorithm, and represent some general properties of the load balancing algorithm such as decentralisation, transparency, autonomy, scalability, and adaptability.

### 2.3.1. Components of a Load Balancing Algorithm

The performance of a load balancing system depends on four factors: the load index, the load balancing algorithm, the workload environment, and the underlying distributed system attributes. In this review we concentrate on the first two factors.

A load balancing algorithm consists of a number of components which interact in various ways to redistribute the users' submitted jobs among the nodes of a distributed system. The objective is to improve the system performance by sharing the whole computing power available. Three main components can be identified: information gathering policy, transfer control policy, and location/negotiation control policy. The policies within the components are inter-related, fixing one would limit the options within the others. It is to be noted that Johnson [Johnson88] surveyed similar components but with more restricted dimensions for each.

**1) Local Load Measure**

At each node a mechanism must be provided to give a good estimation of the current local load. There are two important aspects to be considered here: a load metric which has a close correlation with the performance objective pursued, and the measuring mechanism that must give a quick and efficient evaluation of the local load state.

The local load measure alternatives described in the literature include: "load average" metric provided by the UNIX BSD 4.2 uptime(1) command, a specialised idle process for load estimation based on CPU utilisation [Ni85, Stumm88], virtual load value which is the sum of actual load and the number of processes currently in transit averaged over a period of time longer than the time necessary to migrate an average process [Krueger84], evaluation of remaining service time using probability functions [Bryant81], using a linear combination of all main resource queues [Ferrari85, Zhou87]

(e.g. CPU, paging/swapping, and file I/O queues averaged over 4 seconds period), number of ready processes on a processor during a time interval [Barak85]. Zhou has shown that the CPU queue length is a good load index and has a close correlation to the mean response time [Zhou86, Zhou87a]. This is supported by the work of Kunz [Kunz91]. Although it is the favoured load metric, the CPU queue length is not an adequate load indicator when the processing node possesses a multiplicity of resources which affect the performance. This requires a combined queues length index. When the processors of the system have different processing speeds, a scaled CPU queue length is more appropriate because for the same arrival rate, the CPU utilisation level depends on the processor speed.

**2) Information Policy**

This component is responsible for the exchange and maintenance of the information about other individual or groups of nodes such as load level, nature of workload or the average load over the entire system. It is also responsible for the frequency of the state information update, the ways to exchange this information among the various nodes, the numbers of nodes involved in the exchange, and the amount of information made available to the decision makers. It must maintain consistent information about the global state at the distributed points of control [Casavant87].

Load balancing strategies can be categorised based on the amount of global information used, and the global information update technique. Three levels of global information can be used: local indicators only with no global state information for *random* policies, information about subset of nodes or partial system state information for *polling* policies, and information about all the nodes or system-wide state information for *broadcast* policies.

Two broad categories of global information update technique can be assumed: *periodic* update of tables of information using broadcast mechanism or *on demand* update of information based on bidding or polling techniques. An example of the latter technique is the interrogation of neighbours or random set of nodes only when the node becomes idle.

The type of the information moved between the nodes depends on the nature of the exchange. For a system state update the load index is used (e.g. CPU queue length, nature of workload, node characteristics) while for job transfer it depends on the file system structure. For a diskless node system all that is exchanged is the job name, path, input and output files names. For a disk-based node system the complete job file as well as the input files needed are transferred. At the end of the remote execution the results files are returned to the originating node.

A large and diverse number of information policies are reported in the literature. Each policy is usually reported as having a better performance than the no load balancing case or when compared under some restrictive system assumptions (e.g. negligible load balancing overheads) to other policies. It is not possible to cover them all in this review, however a sample of the most common approaches is described below.

i) Broadcast Approach

A systematic load exchange is done whenever the load of a node changes. This assumes a broadcast communication medium is available [Livny84].

ii) Load Distribution Vector

A load vector of a specific size is periodically updated and sent to randomly chosen nodes where a mapping of vectors is done [Barak85] . This results in a propagation of jobs similar to a "gas diffusion" process. An alternative to the

periodic broadcast is a restricted update where the information is exchanged with neighbours only when the load crosses the low or high water-mark [Ni85].

iii) Global System Load

When a node does not receive a reply from a node in a complementary state, it assumes that all nodes are overloaded, updates its perception of global load and broadcasts it to all nodes [Krueger84]. This is more adaptive to system extreme load conditions. An alternative global load view can be represented by a collection of distances of each processor from lightly loaded processors [Lin87]. This is applicable to point-to-point network topologies.

iv) Polling Approach

The information is requested from neighbours or randomly polled nodes only when the node becomes overloaded or idle [Eager86, Theimer88].

**3) Transfer Policy**

This component decides when it is beneficial to transfer a process from the local workload and selects which process to transfer/migrate. The overloaded node chooses heuristically an advantageous process (for example; long lived process, availability of specific resources at specific nodes) to transfer, based on the local information, the remote information maintained locally, or acquired during the negotiation with other nodes. The transfer policy is also responsible for requesting the transfer of work from other nodes when the local node is about to become idle. The transfer policy is the minimum component needed to implement a load balancing strategy (e.g. Random algorithm [Eager86]). Several aspects of the transfer policy can be explored:

*a) Node initiating the load balancing process?*

There are three ways to specify the condition of the node that initiates the load redistribution process: sender node (overloaded) attempting to push jobs, receiver node

(underloaded) attempting to pull jobs, or a dynamic switching between the two pushing or pulling jobs whenever appropriate. The latter case is called symmetrically-initiated load balancing.

*b) When is it advantageous to transfer or to receive processes?*

i) Load balancing triggering events:

The event that triggers the load redistribution can be a newly arriving job (*exogenous* event) or a process completion/resumption or a periodic invocation based on system clock to correct imbalances through process migration [Ezzat86, Johnson88] (*endogenous* event).

ii) Load imbalance indication:

Several types of indicators of the load imbalance have been used. They correspond to a threshold level or an imbalance gap crossing. A one level static local load threshold [Eager86] or two levels (low and high water-mark values) [Alonso88, Ni85] are the most commonly assumed. A load difference bias relative to peer nodes [Stankovic84a] or a dynamic global average load value [Krueger84] are alternative indicators. Barak [Barak85] used an implicit load difference through a periodic examination of estimated response time on another node. An alternative global load view is the inter-node load distance within gradient surface [Lin87] where the imbalance is represented by a set of distances between the nodes.

*c) Type of transfer?*

A job can be allocated to a remote node before it starts execution (non-preemptive transfer or initial placement). When the job is transferred to a remote node after it has started execution, it is said to have been migrated (preemptive transfer).

*d) Which process to transfer?*

Different approaches to the selection of the process to transfer have been reported. For non-preemptive transfers the newly arriving job is chosen. When an executing job can be migrated, a long-lived process is worth transferring. Krueger [Krueger84] has devised a scheme where many factors are taken into account in the selection of the process to migrate: least often transferred process, a process that has executed for a minimum amount of time, or a process with a small migration size (e.g. < 100 Kbytes). In a workload with several classes of jobs the restriction to transfer only from the class of long jobs can be made.

*e) Number of successive transfers of a process?*

When a job is not guaranteed execution after a transfer, different categories of queues are used to maintain process information: locally generated, remote transfer and number of moves, migrated and number of moves.

**4) Negotiation Policy**

Once a node has decided that it is a suitable transfer client (overloaded i.e. that is wishing to get rid of some of its load) or it is a potential transfer server (lightly loaded or idle i.e. that is looking for work), it engages in a pairing process. This process consists of a search for a transfer partner, a node in a complementary state. There are two aspects to the negotiation policy: among which set of nodes is the partner to be looked for (balancing region), how to search the load balancing region and which rule to use in selecting that partner (partner selection rule). When a system load vector is

used this policy is called location policy.

*a) Balancing region of a node?*

Among the alternative balancing region sizes are the neighbours for a point-to-point model, the collection of idle nodes, a cluster of nodes for a multi-domain network, and the entire system for a single domain network.

*b) Partner selection rule?*

The load imbalance indicator is used by the partner selection rule to pick-up the the complementary partner. The rule used can vary from the simple random probability distribution to an inspection of a load vector. The load vector inspection involves the selection of the node with the minimum value among a set of values or using a rule based on the blackboard concept [Kara89], where each node periodically checks the load vector and if it finds itself having the heaviest load it transfers a job to the node with the lightest load, and updates the vector. This process is repeated until the system is balanced (i.e. the load difference falls within the interval $\delta$).

For sender-initiated algorithms some of the rules used are: strictly random selection [Eager86], polling based on fixed load threshold level or load difference, shortest queue length, or finding an idle node. An alternative to the random polling is a cyclic probing of neighbours [Bryant81, Stumm88]. For the bidding algorithm [Stankovic84], the node with the winning bid (i.e. the shortest load) is selected.

Below are specified the rules used for two receiver-initiated algorithms. Zhou [Zhou87]. describes an algorithm where the underloaded node registers reservations for work at the others. These reservations are stored in a LIFO stack. The rule used here is to send the newly arrived job to the node which made the most recent reservation (i.e. on top of the stack). Ni *et al.* propose an algorithm where the rule used is to select the node with the highest draft-age which represents a node that needs most help [Ni85].

### 2.3.2. Attributes of a Load Balancing Algorithm

In addition to the basic components that constitutes a load balancing algorithm, the latter can be synthesised through the identification of some high level attributes that characterise these components. The attributes include the load redistribution objective, the decision-making structure, the transparency, the autonomy, the scalability, and the adaptability. Each of these attributes is described below.

**1) Load Redistribution Objective**

Different objectives can be pursued when performing a load redistribution within a distributed system. The term *load sharing* is used when the objective is to keep all the nodes busy; *load balancing* is used when the objective is not only to keep the nodes busy but also to attempt an equalisation of the load over all the nodes. When the nodes are privately owned and their sharing is allowed only with the approval of the owner we use the term *restricted sharing*. The computing power is sharable only during a specific period of time based on the discretion of the node owners [Alonso88].

A finer objective is the type of load imbalance that the algorithm attempts to resolve. For a steady state imbalance the jobs are transferred between the nodes so that the arrival rates approach the mean arrival rate. Transient imbalance is resolved by assigning each new job to the node with the least number of jobs.

**2) Decision-making Structure**

The load balancing algorithms can be distinguished based on the decision-making structure used to implement the different components. This structure can be centralised, hierarchical, decentralised, or a hybrid form. The centrally based algorithms such as Central [Zhou87] suffer from the reliability problem due to a potential central point of failure and the potential bottleneck of the central node. Some of these problems have been addressed by the hierarchical structure proposed by Van Tilborg [Tilborg84]. But

to deal with the autonomous nature of the nodes in a distributed system, fully decentralised properties are needed [Stankovic82]. Most of the algorithms considered in this review fall under the decentralised category.

**3) Transparency**

The implementation of the load balancing scheme can be made transparent to users. The can be achieved by assuming non-selective transfer of jobs or by providing a system interface that identify automatically the jobs eligible for transfer. In this case, submission of the users jobs need not be accompanied with specific information about the nature of the jobs to be used in the load balancing algorithm. The users interaction with the system is not affected by the presence of the load balancing scheme.

**4) Autonomy**

A load balancing algorithm that has an autonomy attribute does not infringe the control of the job allocation at individual nodes. The Random and Shortest algorithms described in Section 2.5.1 override the autonomy property of the computing nodes because once a node is selected either randomly or based on the shortest queue rule it cannot refuse a transferred job. This can result in severe overloading.

**5) Scalability**

As the number of nodes in the distributed system grows and the range of workload fluctuations increases, scalability problems can arise. In order to cope with the communication and scheduling overhead resulting from the increased load distribution effort, a number of principles are to be observed during the design of these algorithms to make them more scalable [Barak87]:

*Symmetry:*

All nodes in the system should be allowed to play an equal role.

*Customer-server protocols:*

Each customer-server interaction should involve at most two nodes (one-to-one).

*Partiality:*

Every decision should be based on information from a bounded subset of the other nodes.

*Use of randomness:*

The set of nodes with which a node interacts is chosen at random.

**6) Adaptability**

Dynamic factors such as system load, network traffic, and the availability of computing nodes which characterise a distributed system, have a direct effect on the system performance. To maintain the global scheduling scheme tuned to the variations in the environment, even when the system conditions change drastically, the scheme must include an adaptability feature. It is a mechanism built into the algorithm that uses the environmental parameters for a dynamic selection of the components of the global scheduling strategy:

-degree of sharing to aim for
-type of process transfer to invoke
-load conditions of the node initiating the load distribution process [Hong88]
-dynamic adjustment of algorithm scheduling parameters
(i.e. relaxation of parameters for the information, transfer, or negotiation component).

It also provides a decision-making procedure to control these modifications, for example to increase the load distribution effort in the case of a wide load imbalance or to reduce it when the load of all nodes is so heavy or so light that no improvement can be achieved by such effort [Ramamritha87]. The reported approaches to adaptability can be classified into two categories.

In the first category the adaptability is included within the basic structure of the algorithm (information, transfer, and negotiation). This aims at taking into account the various system parameters and the history of the system's behaviour, in performing process scheduling [Stankovic84] or to allow migration (preemptive transfer) of process whenever anomalies in the load distribution occur [Barak85]. The load balancing

process is activated only when necessary [Stankovic84a, Shamir87] (i.e. when the system load is below a particular minimum threshold) by turning off all parts of the load balancing algorithm except for the monitoring of the load. The algorithm described in [Krueger84] is based on the average load of the entire system, with each node aiming at keeping its load within an acceptable range from the system average. When the communication device becomes overloaded, the load balancing negotiations are slowed down. Other algorithms use dynamic mechanisms to estimate their most sensitive parameters [Ezzat86, Pulidas88]. We describe this type of adaptability as being *inherent* to the basic load balancing algorithm.

The second approach to making a scheduling algorithm adaptive to the system dynamic characteristics and workload conditions, is to dynamically assess the system environment and adjust the global scheduling strategy accordingly. Ramamritham *et al.* [Ramamritha87] propose a meta-level controller for a distributed real-time system, which is a more predictable environment. Based on the current system conditions, it selects the algorithm(s) used for task scheduling on a node, the algorithm(s) for cooperation among the nodes, and the values of the scheduling parameters used in the chosen algorithm(s). The load distributing algorithm PollGen [Krueger87a] includes the possibility of dynamically switching from a load sharing to a load balancing objective, and having the load distribution process initiated either by the overloaded node or the idle node. We describe this category of algorithms as having an *explicit* adaptability mechanism.

An extended review of the adaptability attribute is given in the following section. This includes an outline of the issues involved, the concept of tolerance of a scheduling algorithm, and the adaptability dimensions.

## 2.4. Adaptive Load Balancing

In this section, a computing environment which is rapidly changing is described and the motivation for adaptive load balancing given. The tuning of scheduling in a distributed system can be implemented manually or automatically.

- Manual: It is performed by the system manager to adjust the system parameters to long term fluctuations of the environment.
- Automatic: It is performed by an on-line scheduling scheme. Depending on the magnitude of the fluctuations involved a dynamic or an adaptive strategy is appropriate. To deal with short term fluctuations in a rapidly changing environment, an algorithm that can dynamically switch its policy is required. The resulting scheme can be identified as an adaptive distributed scheduler.

In computing environments where the system characteristics do not fluctuate too much (e.g. homogeneous workload), and resource consumptions can be estimated (i.e. transaction processing or real time systems), the dynamic approach using initial job placement alone can provide significant improvement at a lower cost [Zhou87a], and thus does not justify the design of more complex load balancing algorithms [Eager86]. The computing environment of interest in this section is characterised by rapidly changing and unpredictable system state characteristics and workload (e.g. workstation-based distributed system [Ezzat86, Mutka87]). In these environments the load distribution is not homogeneous in nature and its magnitude can vary significantly over time. This makes it impossible to devise one single load balancing policy that performs well in all the circumstances. A new class of algorithms that adapt to changes in the system environment and are robust across a wide range of conditions is recognised as the most promising [Krueger87]. Among the changes that might occur and for which the scheduling strategy must adapt are the number of nodes available, the

variations in jobs arrival rates (e.g. bursty jobs), the distribution of process size and service demands, and the utilisation level of the communication network [Ramamritha87].

Each of these parameters affects the choice of the appropriate level of load sharing to aim for, and the suitable scheduling strategy. The scheduling scheme for such an environment must adjust automatically the tunable parameters specific to each algorithm and/or switch to a more appropriate policy as the situation changes. These types of schemes are truly adaptive as they react immediately to anomalies, allowing the system to be always operating close to its optimal point. However, there is a potential price to pay in performance degradation due to the scheduling overhead, unless the adaptive strategy includes a mechanism by which ineffective load balancing activities are minimised. The adaptive algorithm must take into account the changing parameters to 1) provide for dynamic modifications to the components of the scheme, and 2) contain an adequate decision-making procedure to control these modifications. Apart from algorithmic adjustments, simple adjustments to the variable parameters specific to each policy (e.g. dynamic threshold calculation [Hac87]) can improve the performance significantly when the system load fluctuates [Zhou87a].

The essential static characteristics of the system for which adaptive scheduling algorithms are to be developed, have been outlined in Section 2.2. Here we concentrate on the rapidly changing aspects of the system the workload patterns and intensities as well as the dynamic characteristics of the system itself (number of active nodes, extreme node load, global system load nature and level, characterisation of scheduling algorithm overhead and communication delays), how they affect the algorithm performance, and the need for adaptability. These random and dynamic changes make a fixed load balancing policy which improves performance at some time, whilst at other times is inefficient but can even degrade the performance and cause system instability.

To address this problem both dynamic parameter adjustments or policy changes are required.

The adaptive load balancing algorithms are potentially more complex than their dynamic counterparts because to adapt the scheduling strategy to dynamic system conditions involves more system conditions monitoring, and CPU overhead for the dynamic adjustment of parameters and policies. It also involves more process transfer cost for the preemptive migrations made necessary in some adaptability cases. These activities put more strain on both the CPU through overhead computation costs, and the communication medium through extra message traffic leading to extra CPU queue delays for the transferred jobs. For a suboptimal performance a compromise between potential improvement of load balancing actions and the performance degradation incurred by the overhead costs must be reached. In the comparison of the merits of different algorithms both the computation costs and the communication costs must be evaluated and included in the performance assessment. The CPU overhead which includes the handling of message traffic during negotiation between nodes, the algorithm execution costs added to both sending and receiving nodes, and the excess delay (i.e. wait time) caused to other local processes must be calculated. The communication costs (i.e. packing, transmission and unpacking of data) are caused by an increase in message traffic over the communication medium due to the information exchange/negotiation messages, and the transfer rates increase.

In subsequent sections first the adaptability issues and approaches involved are identified. Then a framework for the design of an adaptive scheduling algorithm is described. This involves the tolerance concept [Krueger88], the adaptability dimensions, and some implementation considerations. The contribution of this review consists of a global view to the adaptability problem in load balancing algorithms and an outline of a design methodology.

### 2.4.1. Adaptability Issues

In the reported research on load balancing algorithms, no systematic view to the design of adaptive algorithms was taken with the exception of the work of Krueger [Krueger88]. Most algorithms have not been designed with adaptability to a widely changing environment as a specific goal. The approach usually taken is to include some flexibility into one of the algorithm components through a dynamic evaluation of some scheduling parameters. The resulting algorithms adapt only to limited type and magnitude of system changes.

There are two types of adaptability which can be included within a load balancing scheme: *inherent*, which is built within the basic components of the algorithm, and *explicit*, which involves global parameters and policy switching. The adaptability can be detailed further by identifying the type of actions taken. Among these actions are:

- dynamic process placement based on local information and probabilistic functions
- parameters dynamically estimated [Pulidas88] to reflect new system conditions
- parameters dynamically adjusted or ignored
- dynamic policy switching

Each of these type of actions corresponds to a level of adaptability and involves different overhead costs and algorithm complexity. The choice of the appropriate level depends on the magnitude and duration of the system and workload fluctuations.

*a) Algorithms with inherent adaptability*

Based on the choices made (i.e. algorithm components and attributes) in building the load balancing algorithm, different levels of adaptability and steadiness of the performance rate can be maintained.

*b) Algorithms with explicit adaptability*

The fundamental approach for this category of algorithms is to add adaptability mechanisms to dynamic algorithms, based on the concepts of tolerance of algorithm,

adaptability dimensions, and adaptive scheduling strategy. The relevance of the approach of adding explicit adaptability to load balancing algorithms was demonstrated through the PollGen algorithm example [Krueger88], which includes some adaptability mechanisms. As shown in Section 2.3, there is a multi-dimensional parameter and policy space in a load balancing algorithm. This is addressed further in Section 2.7.3.

These are some of the issues related to the adaptability of an algorithm that need to be investigated:

1) Definition of a stable and balanced system
2) To which algorithm dimension(s) and/or component(s) is adaptability to be added?
3) Trade-offs in the design of an adaptive scheme:
   -complexity of algorithm and range of adaptability
   -responsiveness and accuracy of adaptability
   -extent of variability in distributed systems and performance gain
4) How to quantify adaptability?
   e.g. improvement in response time, quality of host selection

### 2.4.2. Tolerance of a Scheduling Algorithm

Given the fact that processors in a distributed system are autonomous and communicate only through message-passing mechanisms [Chandras90], the best load balancing algorithm cannot escape overhead costs (i.e. load redistribution actions cost), both in terms of computation costs and communication delays, and uneven periods of load distribution (i.e. periods of unbalanced states). Adaptive scheduling can be expressed as finding the right balance between two conflicting issues. The first issue is the minimisation of the overhead cost by using the estimate costs established for the system environment (encouraging only the cost-effective actions). The second issue is the reduction of the duration and magnitude of these undesirable states. The concept of algorithm tolerance is suggested by Krueger [Krueger88] for adding an explicit adaptability to a load balancing algorithm. One way to reach this balance is to

distribute the load in degrees [Krueger87a]. Based on the system conditions a load sharing objective with or without anticipatory transfers, load balancing objective is activated or no load redistribution at all. To give more flexibility for the scheduling algorithm to adapt to the changing environment, it must also be allowed to deviate from its main strategy by varying within a range for each scheduling parameter and policy option. The magnitude and duration of these deviations can be specified as the tolerance of the algorithm. The adaptability mechanism is used by the algorithm to tune its strategy (i.e. taken corrective actions) within the algorithm tolerance according to variations in the environment and maintain an acceptable level of performance. Three types of tolerance can be identified:

*a) Minimum tolerance:*

This corresponds to the ideal case where no periods of unbalanced states occur. A balanced load on all the machines at all times is maintained. Unfortunately this would be achieved with excessive costs and may even result in performance degradation as in the situation where processes are transferred from nodes with few processes to an idle node. The costs of the transfers can far outweigh the gain in load balancing.

*b) Heuristic values:*

These values are obtained though experimentation and can achieve adequate results. For example instead of using the strictest load difference of one between two nodes in order to perform a transfer, it is more sensible to use the higher difference of three, which gives more gain to outweigh the load distribution overhead cost. However, these results are not acceptable when we are dealing with a widely changing environment.

*c) Adaptive values:*

Here not all the parameters or policies are fixed. The sensitive features are varied to allow, based on the system state, a dynamic adjustment of the tolerance of the algorithm to be carried out to optimise the performance. The design of adaptive load balancing algorithms, in addition to the classical components of the dynamic load balancing algorithms, involves the provision of an adaptability mechanism which can be implemented by:

1) On-line estimation of parameter changes that require adaptability of strategy (i.e. current system state)
2) Including mechanisms for modification of the values within the tolerance of the algorithm (i.e. dynamic manipulation of algorithm parameters and policies)
3) Establishing the rules of the adaptive scheduling strategy (i.e. when to adjust, what, and how)
4) Providing appropriate decision-making procedure to control these modifications

### 2.4.3. Adaptability Dimensions

Instead of striving for the minimum tolerance, we examine how the algorithm components can be adjusted and the scheduling strategy tuned to maintain performance and stability. The adaptability of a load balancing algorithm can be explored along two paths *parametric tuning* with three types of parameters involved: scalar, timing, and threshold, and *policy switching* with four types of policies involved: condition of the node initiating the load redistribution process, type of process transfer, load redistribution objective, and basic algorithm component options.

#### 1) Parametric Tuning

The parameters of a load balancing algorithm that can be tuned can be classified into three types: threshold, timing, and scalar. This adjustment can occur within any of the components of the load balancing scheme: load measure, information policy,

transfer policy, and negotiation policy.

Threshold

These parameters put limits on the level of usage of a resource being managed. To identify a suitable partner, a node uses the load difference level to justify the performance gain of a remote execution. Other threshold parameters include local load threshold, difference between local load and global average load value, limit on the number of successive process transfers, size of the subset of nodes that exchange information or negotiate process transfer with a given node. Instead of being fixed to an average value an adaptive threshold is evaluated dynamically (wherever necessary or periodically). Different relationships have been used to evaluate the threshold value. Hac *et al.* [Hac87] used the formula $T=f(Nr/P-1)$ where Nr is the number of active processes and P the processor capacity. Lee [Lee86] linked the threshold to the job arrival rate, while Pulidas [Pulidas88] linked it to the flow of jobs on the network, the incremental delay information, and the minimum incremental delay. Others used the job transfer cost or transfer device utilisation level.

Timing

These parameters determine how often the load redistribution actions will be performed, for example slowing down the scheduler activity for periodic policies [Stankovic85] or the tuning of the amount of idle time [Hac87]. They depend on the static and dynamic loading of the system. This involves the specification of temporal relationships between negotiation sessions, process transfer or the exchange of information etc.

Scalar

The scheduling parameters of an algorithm can be assigned a weight (e.g. node

speed processing factor [Castagnoli86] ) to emphasise their static or dynamic importance in a decision function or to modify the weight of a decision based on static or dynamic local conditions (e.g. bidding approach [Stankovic84]). For example negotiation, transfer, or remote information policies may include probabilistic values.

**2) Policy Switching**

To cope with a changing environment, the scheduling algorithm involves many policies. These policies can be classified further according to their nature and the options available. They are invoked dynamically for example the initiation of transfer can be performed by either the overloaded or the underloaded node. The choice can be based on the system load. Other policies include the degree of sharing, the type of transfer, the algorithm specific policies: information, transfer, negotiation. All these parameters can be fixed or tuned dynamically to provide an adaptive scheduling environment.

a) Node initiating the load redistribution process

The node that initiates the load distribution process can be an overloaded node seeking to reduce its load by migrating some of its local processes to a lightly loaded or idle node. It may also desire to transfer newly arriving processes to a complementary node. The algorithms based on this approach are called *sender-initiated* and are commonly used for dynamic load balancing. They do not require preemptive scheduling. The initiation of the load distribution process by the idle or lightly loaded node is the second alternative. In this case the node is searching for an overloaded node for the purpose of relieving it of part of its load. These algorithms are called *receiver-initiated* and in most cases assume the availability of preemptive migration of processes, because there is a small probability that at the time the idle node interrogates the overloaded node, a new external job arrives.

It must be a resident or running process that has to be migrated. In a study by Eager *et al.* [Eager85], it is shown that under low system loads the sender-initiated algorithms perform better than the receiver-initiated algorithms, the latter performs better under heavy system loads. A third alternative is to have either the sender or the receiver node initiate the load distribution process. These types of algorithms are called *symmetrically-initiated* [Krueger87a] and have more potential for adaptability.

b) Type of process transfer

The transfer of a process for remote execution can be done before it begins execution on the local node (i.e. non-preemptive scheduling) or even while it is running on the local node. In this case it is interrupted and sent, along with its image including the changes which occurred due to execution [Smith88] to another node for remote execution. In both types of transfer, the results of execution are sent back to the originating node if no shared file system is used. There is a substantial cost involved in migrating a running process. However, preemptive algorithms have more potential to adapt to dynamic changes in system conditions (e.g. process completions or resumptions) than the non-preemptive algorithms because the latter cannot transfer processes after they have begun their execution. They deal only with newly arriving processes. Before a new process arrives no load anomalies can be corrected.

c) Load redistribution objective

Based on the system conditions and the performance objectives sought, different degrees of load distribution can be implemented [Krueger87]. When all the nodes in the pool are idle or lightly loaded, or all heavily loaded; there is no performance gain in trying to distribute the load. If the load distribution goal is to maximise the rate at which work is performed by the system by making sure no node is idle

while processes are waiting for service at other nodes (i.e. work conservation scheduling), then load sharing is the solution. This assumes that keeping all the nodes busy results in a better mean job response time. Load balancing extends the load sharing objective by aiming at allocating a near equal number of jobs to each node in the system. In addition to mean response time, the mean and standard deviation of the wait ratio (i.e. wait time per unit of service) are to be minimised. Load balancing reduces both wait time and wait ratio [Krueger87]. This implies a fairness of scheduling, but may degrade performance in some cases as in a system with heterogeneous node capacities. By allowing the load balancing algorithm to select the degree of distribution to aim for, adaptability to wide ranging system conditions can be achieved.

### 2.4.4. Implementation Considerations

The general purpose of the adaptability attribute is to get around the lack of global state information or out of date information which characterises distributed systems, and the cost of its maintenance. This can be achieved by using approximate information, and successive dynamic adjustments of the scheduling strategy to the system environment. To implement an adaptive strategy and control the adaptive components for a load balancing algorithm, the rules which link the current system conditions to the appropriate scheduling parameters and policies must be identified. Then a decision-making procedure to dynamically apply those rules must be established.

Although the centralised approach presents the advantage of scalability in implementing an adaptive multiple-options scheme [Zhou88], it is rejected for the classical disadvantages of centralised systems, namely the central controller may become a bottleneck and have an adverse effect on the complete system. Another

reason for choosing decentralised scheduling is that it is less complex to implement, compared to its centralised counterpart [Theimer88]. It involves the following steps:

*Step 1:* On-line evaluation of dynamic changes in conditions which drive the decision-making process (e.g. current system load).

*Step 2:* Use built-in rules (heuristics) and current system state for the selection of scheduling strategy components: the algorithm dimension(s) to be affected, the policies and parameters to be affected (information, transfer, negotiation), and the choice of appropriate level of adaptability.

*Step 3:* Perform on-line modifications of scheduling strategy using the mechanisms for manipulation of values of parameters within their tolerance or switching to an appropriate algorithm option.

*Step 4:* Perform the load balancing actions.

Some of the choices to be made during the development of this scheme include periodic invocation or on-demand adjustments, whether to memorise and use past decisions or to base the decision on the currently perceived system state only, how to control the algorithm modifications in a decentralised environment and to what extent are these decisions affected by the accuracy of the local view of the global state. Since the adaptive decision-making procedure will be implemented for several basic load balancing policies, it has to be decided which parts are to be embedded in the scheduling algorithm itself and which parts are to be embedded in the distributed kernel.

### 2.4.5. Design Methodology for Adaptive Scheduling Algorithms

For a dynamic system tuning, made necessary by the wide and unpredictable fluctuations in the distributed system, algorithms with adaptability feature are to be developed. The design of such algorithms can be pursued along two approaches. One

approach is to design new algorithms based on novel adaptive models or policies with inherent adaptability. The alternative approach is to add explicit adaptability to dynamic algorithms based on representative strategies and deriving adaptive algorithms (e.g. symmetrical GLOBAL_AVG [Johnson88] and PollGen [ Krueger88]) by combining the best policies of existing dynamic algorithms and allowing a dynamic switching of these policies based on the system conditions.

A methodology for adding adaptability to a load balancing algorithm involves the following steps:

1) Performance objectives specifications
   (response time, balance factor, stability, minimum cost)
2) Changing environment characteristics specification
   (extent of variability for system and workload)
3) Identify structure of load balancing strategy and communication model assumed
4) Identify load balancing algorithm components involved
   (Information, Transfer, Location/Negotiation)
5) Establish basic load balancing activities cost (computation and communication)
6) Identify dimension(s) of algorithm to which adaptability is to be added
7) Identify adaptable features for each component:
   -dynamically estimated parameters
   -tunable parameters
   -adjustable policies
8) Derive algorithm structure based on added dimensions
   (i.e. combined policies)
9) Establish relationships between current system state and scheduling strategy
   (i.e. which policy(ies) and parameter(s) to adjust and when) taking into account:
   -performance objectives
   -load balancing activities costs
   -current system state and extent of variability
   -adaptable features of algorithm
   (Provide a decision-making procedure to tune parameters and policies)
10) Construct full adaptive algorithm structure
11) Test algorithm performance against no load balancing case or other algorithms

This design methodology could be used for the design of adaptive load balancing strategies. It consists of adding an explicit adaptability feature to a load balancing algorithm through a combination of different policies or by adding a mechanism for an automatic tuning of the algorithm parameters.

## 2.5. Description of a Selection of Algorithms

In this section, the algorithms selection criteria are outlined, then a set of load balancing algorithms where most components and attributes defined in previous sections are represented, is described. Last other algorithms included in reported load balancing studies and referred to in Section 2.6, are detailed.

A dynamic algorithm uses fixed algorithm parameters and the same policy (e.g. when a new job arrives and finds the node overloaded, the current system load vector is checked and the job is sent to the node with the shortest queue). The job placement decision is based on the current system state (e.g. threshold-type information, inter-nodes load imbalance information). When the load balancing algorithm places the users jobs using the current perception of system load distribution, and also adjusts its policies to reflect the needs of load redistribution, it becomes an adaptive algorithm.

Both aperiodic algorithms where the arrival of a job (process creation) or a job departure (process completion) trigger the load balancing process, and periodic algorithms which are timer-driven, are considered in this study. The algorithms developed below include both categories of load sharing and load balancing objectives, however we use the more general *load balancing* denomination. The choice of these algorithms has been motivated by the need to cover a range of information and control policies identified in Section 2.3.

For a consistent description of the load balancing strategies a common terminology is defined. The load level at a sending node is indicated by load.i, and by

load.j at the receiving node. The algorithms considered are threshold driven according to the queue length of processes. Four threshold parameters are used and whose crossing corresponds to:

| | |
|---|---|
| $T_{si}$ | the load balancing strategy is activated by the sending node |
| $T_{sa}$ | an acceptance of a transfer is indicated by the sending node |
| $T_{ri}$ | the load balancing strategy is activated by the receiving node |
| $T_{ra}$ | an acceptance of a transfer is indicated by the receiving node |

All the strategies considered are based on non-preemptive transfer policies. Below is detailed a set of load balancing algorithms covering different information and control policies. The load level at a node represents the number of jobs waiting plus the currently executing task. The term *load.i* is used at the node which activates the load balancing process while *load.j* is used at the node being polled.

### 2.5.1. Representative Load Balancing Algorithms

*1) Random Algorithm*

This is the simplest algorithm. When a node load level crosses the threshold $T_{si}$ (load.i > $T_{si}$), it sends the newly arrived job to a randomly selected node. Only the local information is used. A variant to this algorithm is to consider a *transfer_limit* greater than one by allowing a transferred job to be transferred again if its destination is found overloaded too.

*2) Sender Algorithm*

This algorithm is based on the Sender policy [Eager85] and THRHLD policy [Zhou87]. When a node becomes overloaded (load.i > $T_{si}$ used for load balancing initiation), it sequentially polls a set of (Lp) random nodes looking for one whose load is below the *threshold* (load.j < $T_{sa}$ used for remote job acceptance). If so an ACCEPT message is sent back, otherwise it replies with a REJECT message. Then if the

requesting node is still overloaded when the ACCEPT reply arrives, the newly arrived job is transferred, otherwise the job is processed locally. The job is also processed locally when the *probing limit* is reached or if the node is no longer overloaded before the probing is exhausted or when a polling session is already in progress when the job arrives. The probing is sequential, no simultaneous negotiations are allowed.

To avoid the situation where a node is the sending and receiving states simultaneously the choice of $T_{si}$ and $T_{sa}$ must be such that $T_{sa} \leq T_{si}$.

A variant of this algorithm called LOWEST in [Zhou87] transfers the job to the host with the lowest load among those randomly polled. Probing stops when the first empty host is found. However, no significant improvement is reported. Another potential variant to this algorithm (Sender_Df) is to search for a partner whose *load difference* is less than that of the requesting node by a constant (e.g. load.i - load.j > Df).

*3) Receiver Algorithm*

This algorithm is based on the Receiver policy [Eager85]. If the completion of a job brings the load of a node below the *threshold* (load.i < $T_{ri}$), this node polls a random set of nodes up to a *probe limit* looking for an overloaded node (load.j > $T_{ra}$), in which case a non-preemptive "migration" of a job from the ready queue of the overloaded node is done. The transfers are receiver-initiated. The triggering of load balancing is done when the completion of a job bring the load level of the node below a threshold value. A special case is the idle node state (load.i = 0).

A variant of this receiver-initiated algorithm is to use a timer-driven load balancing activation instead of using departing jobs. Periodically (i.e. *timer_period*), the load of a node is checked to identify if the node is idle (load.i= 0) or if its load is below the *threshold* (load.i < $T_{ri}$). If so a polling session is initiated for up to the *probe limit.*

This strategy has an advantage over the Receiver algorithm in that the situation where a polling session has failed and no new job arrives leaving the node idle forever, will not occur because the algorithm is periodically awakened.

To avoid the situation where a node is the sending and receiving states simultaneously the choice of $T_{ri}$ and $T_{ra}$ must be such that $T_{ra} \geq T_{ri}$.

*4) Shortest Algorithm*

This algorithm is based on the DISTED algorithm in [Zhou87]. It allocates a new job that brings load.i above $T_{si}$, to the node with the shortest queue (node.j = min ($L_1$, $L_2$,..., $L_n$)). It maintains a load vector at each node. This vector is periodically (i.e. *exchange_period*) updated using a broadcast mechanism. To reduce the number of information exchanges, the nodes broadcast their state only when the load changes (a new job arrives, a job is transferred in, a job is transferred out, or a process departs).

*5) Symetric Algorithm*

This algorithm is a combination of the Sender and Receiver algorithms. It involves a symmetric initiation of load balancing [Krueger88a], depending on the value of the load relative to the *thresholds* ($T_{si}$, and $T_{ri}$), with $T_{ri}$=1 for whom the idle node initiates the load balancing negotiation. The load balancing strategy is dynamically adjusted based on the node load level by allowing the algorithm to switch automatically between a sender-initiated (SI) or a receiver-initiated (RI) policy.

To avoid the situation where a node is the sending and receiving states simultaneously, which corresponds to a node sending its new local jobs to remote nodes while accepting remote jobs to be executed locally, or having both sender-initiated and receiver-initiated negotiations engaged at the same time, the choice of the thresholds must be such that $T_{ri} <= T_{si}$ and $T_{ra} \geq T_{si}$.

It is to be noted that all the algorithms described so far are $T_{si}$ or $T_{ri}$ threshold driven or both. Random and Shortest algorithms have no $T_{sa}$ acceptance condition at all (i.e. overriding the remote node autonomy), while Sender and Symetric can have $D_f$ the inter-node load difference as an acceptance condition instead of $T_{sa}$ or $T_{ra}$ threshold.

### 2.5.4. Other Load Balancing Algorithms

The load balancing algorithms commonly reported in comparative studies include: centralised, distributed, preemptive, non-preemptive, adaptive and non-adaptive examples. A sample of the algorithms is described below. They all work on a broadcast communication model, however most of them could be implemented on a point-to-point model. Except for PollGen which dynamically adjusts its degree of redistribution, all these algorithms have load balancing as a global objective. The next three algorithms have been evaluated by Zhou [Zhou87a].

GLOBAL (centralised control)

One host, designated as the LIC (load information centre), assembles the load of all the hosts in a LV (load vector) and broadcasts the LV to all the hosts every P seconds. The placement policy is as follows: send new job to the host with lowest load (i.e. load ≤ local load - δ, where δ is a constant), if there is more than one host with the lowest value, select one arbitrarily.

CENTRAL (centralised control)

The LIC acts both as the load information centre and the central scheduler for all the hosts (e.g. Process Server [Hagmann86]). Such a distinguished agent requires less overheads making the algorithm more scalable.

RESERVE

It is a receiver-initiated algorithm based on job reservation. If the load gets below $T_l$, the host probes the other hosts to register R reservations at R hosts with load

above $T_l$. At the overloaded host, outstanding reservations are stored in a stack. When a job arrives, it is sent to the node that made the most recent reservation. If the load falls below $T_l$, all reservations are cancelled. An improvement of this algorithm is made if before sending the job the host makes sure the server host is still lightly loaded. This is the only non-preemptive receiver-initiated algorithm evaluated in Zhou's work [Zhou87a]. Most receiver-initiated algorithms are preemptive.

GLOBAL_AVG

This is a preemptive algorithm developed by Krueger [Krueger84]. Each node maintains a value for the network average load (Av) and strives to keep its own load to within a pre-defined acceptable range (A) from it. If the load is not within the acceptable range then it attempts to find a transfer partner by broadcasting its conditions and waiting for a reply within a reasonable time (Tr). If no complementary partner can be found, it updates the global average load by (U) amount and broadcasts the new average value to the other nodes otherwise it migrates an advantageous process to a complementary partner. A symmetrically-initiated version of the algorithm has been developed by Johnson [Johnson88].

DRAFTING

This is a receiver-initiated preemptive algorithm based on a drafting strategy [Ni85]. Each node maintains a load table of candidate processors from among its neighbours, but instead of using numerical values to describe the load of a node, three states are used: *Low state* when the node can accept remote processes, *Normal state* when no transfer in either direction is desirable, and *Heavy state* when the node needs help from other nodes. The negotiation engaged is as follows. A message (draft request) is sent by L-load node to those H-load identified from local table. A response message from H-load sent indicating how

much help they need (draft age). This value is zero if the node is no longer in the H-load state. After a timeout period all the draft ages must be received. At this point a draft-standard is calculated based on all the received draft-ages. The node with the highest draft age (i.e. the one that needs help most) is selected. If all draft ages are zero then suspend the drafting process. The drafted node sends a new task or responds with a "too late" message.

Bidding

This is a sender-initiated non-preemptive algorithm [Stankovic84]. The loaded node (based on threshold crossing) requests bids from neighbours or all nodes (i.e. through broadcast). The bids (i.e. current load) are sent by underloaded nodes. The node with the winning bid (i.e. shortest load) is selected and will receive a transferred job. If no appropriate bids arrive within a time window, then extend the request for further bids in the network or process locally.

PollGen

This is a preemptive algorithm with an adaptive feature [Krueger88]. It is based on the PID algorithm [Livny84] and Threshold received-initiated version [Eager86]. It has also a sender-initiated aspect and can be symmetrically-initiated. Several parameters can be manipulated to tune the algorithm to the changing system conditions.

$T_{Rmax}$: The maximum load of a suitable receiver which indicates appropriate degree of sharing or load redistribution objective. Three objectives are possible: LS(0) for load sharing, LS(1) for load sharing with an anticipatory migration, and LB($\infty$) for load balancing.

$T_{Sneg}$ : The negotiation is initiated when the load is above this threshold.

$T_{Tdiff}$: The minimum load difference between transfer partners when the load difference is used in the negotiation policy (load balancing objective).

*SendProb:* A sending node initiates negotiation with a probability SendProb when arrival of a process causes the load to be at least $T_{Sneg}$.

*RecvProb:* A receiving node initiates negotiation with a probability RecvProb when the completion of a process causes a node to become idle.

*PollLimit:* The maximum number of nodes polled before giving up.

## 2.6. Load Balancing Studies and Implementations

In this section, the comparative studies of different algorithms, taking into account the model assumptions made, are analysed. The approach used (i.e. analytical, simulation, measurement) is also indicated. Last the implementation of a few load balancers is reviewed.

### 2.6.1. Load Balancing Comparative Studies

The performance study of load balancing algorithms can be carried out along two dimensions: system characteristics and algorithm nature. The first involves the model assumptions made and experimental factors while the second is concerned with the load balancing strategy used.

Eager *et al.* [Eager86] investigated the trade-offs in the level of complexity of the load sharing policies and the level of performance gain. Three types of decentralised, threshold-based algorithms (Random, Threshold, Shortest) with various amount of information are evaluated (no information, threshold-type information, complete information). The load balancing overhead is added to the CPU and corresponds to an increased load. Other assumptions made are no delay in transferring jobs and perfect global state information. The main conclusion of their work is that collecting little

information is more advantageous in terms of performance improvement and communication cost trade-offs. Parametric tuning investigations were also carried out on threshold, probe limit, and transfer limit. In [Eager85] sender-initiated and receiver-initiated policies (Sender, Receiver, Reservation) were compared. It was shown that sender-initiated are preferable to receiver-initiated at light to moderate load levels while receiver-initiated policies perform better at higher load levels. In both references simulation results are used to validate the use of simple analytic models.

Zhou has carried out a thorough comparative performance study of seven non-preemptive dynamic load balancing algorithms among the most commonly described in the literature [Zhou87a]. A homogeneous distributed system based on the broadcast model and a trace-driven simulation of independent sequential jobs are assumed. This implementation is aimed at minimum changes to the system kernel. A foreground/background round-robin local scheduling discipline with 100 milliseconds time slice for the CPU is used.

These algorithms are non-preemptive and, except for the RESERVE algorithm, are all sender-initiated. Most algorithms are decentralised, except for the GLOBAL and CENTRAL, which include some centrally controlled components. The other algorithms are DISTED, RANDOM, THRHLD and LOWEST. All these algorithms use the same load index (i.e. CPU queue length) and the same transfer policy (i.e. based on the command name of the job and local load threshold) but they differ in their information and corresponding negotiation (called placement for non-preemptive algorithms) policies. Within the assumed constraints the most promising algorithms are GLOBAL, CENTRAL, THRHLD, and LOWEST. One conclusion that might be drawn is that centralised algorithms perform well and that a small amount of state information used is sufficient to gain most improvements for decentralised algorithms. These schemes, which use current system load in determining job placements, have been shown to

improve significantly the average response time of jobs, especially under heavy and/or unbalanced workload and make response time more predictable, even with the transfer of a small number of jobs. The trace-driven simulation results have been confirmed through measurement studies. The simulation work has been repeated on data from three computing environments (Berkeley, Bell Labs, and Lawrence Labs).

In a similar effort Johnson [Johnson88] has compared the performance of fewer algorithms but included both preemptive (actually jobs are migrated from ready queue, not while executing or blocked), and non-preemptive dynamic algorithms. The algorithms called RANDOM, THRHLD are non-preemptive algorithms while GLOBAL_AVG and P_THRHLD are preemptive algorithms. P_THRHLD is the same as THRHLD except that it is triggered periodically to allow anomalies that occur before a new process arrival to be corrected through process migration. He used probability distribution generated artificial workloads to drive a simulated distributed system composed of a Manhattan connection of virtual processors (i.e. a point-to-point communication structure). The local scheduling is based on round-robin discipline with 50 milliseconds time slice and the CPU queue length used (i.e. no. of resident processes) as a load index. The performance of the algorithms was tested using both independent processes and cooperating process groups. He also modified the GLOBAL_AVG based preemptive algorithm to make it adapt its policy to changes in the system load for a simple case of a group of cooperating processes. Based on the current system load the algorithm switches between the sender-initiated policy which performs best for light system load, and the receiver-initiated policy which performs best under heavy system load. The experiment on this Sender_Receiver version of the GLOBAL_AVG showed that, using the instantaneous value of the global system load as the indicator to switch between sender and receiver initiated negotiation, the best results are obtained under both light and heavy system load.

The load balancing cost is equated to the communication cost and is evaluated only in terms of the number of the messages exchanged. A modification is made to the GLOBAL_AVG algorithm to limit the simulated broadcast to immediate neighbours only, with the aim of reducing the number of messages exchanged.

However, a more elaborate work on adaptive load balancing was done by Krueger. In [Krueger84] the GLOBAL_AVG algorithm is described. This algorithm is adaptive in the sense that each node attempts to keep its load within a close range of a dynamically updated global average load. It also adapts to communication medium utilisation (i.e. a broadcast token ring) by allowing only the most advantageous transfers to occur. Using a system wide negotiation for transfer partner and the update of the global average, this algorithm has performance limitations due to indiscriminate broadcast overhead costs. In [Krueger87, Krueger87a, Krueger88a], analytical studies of load balancing strategies were carried out. An analytical justification for adaptive scheduling is given. The PollGen algorithm was designed and using simulation, it was shown that good performance and stability can be maintained over a broad range of system environment changes for independent processes, through adaptability.

In [Concepcion88] a testbed, based on the Simscript II.5 simulation language, for the comparative performance study of dynamic load balancing algorithms is described. It addresses particularly the effect of the network topology (i.e. ring, bus, and mesh) on the performance of three algorithms (drafting [Ni85], bidding [Stankovic84], and probabilistic [Hsu86]) which are not adaptive according to the definition in Section 2.2.2 despite the title of the paper. Various algorithmic parameters are experimented with to identify heuristic values for the best performance under fixed workload conditions. A variety of performance criteria (CPU queue length, CPU utilisation, mean response time, balance factor, and communication overhead) are used.

In [Mirchandan89] the authors provide an analysis of the effects of jobs and messages transfer delays on the performance of three load balancing algorithms (Forward, Reverse, and Symmetric). The model is based on the disk-based structure and consists of homogeneous nodes with Poisson job arrivals and exponentially distributed service times and job transfer times. The delays incurred by the probes are assumed negligible. Simulation is used to validate the analytic results. The performance of the three algorithms have been evaluated at 0.1S and 2S delay levels where S is the mean service time. The performance difference is significant at low network delays with the best results obtained by the Symmetric algorithm. At high delays the performance of the algorithms are identical except at high load levels ($\rho \geq 0.9$) where the performance of the algorithms is more spread out. The relative performance order of the algorithms is: Symmetric, Reverse, and Forward. Forward performs better than Reverse at low to moderate load with the break-even obtained at a system load $\rho=0.75$. The network delays have no effect on the relative performance order of the algorithms.

### 2.6.2. Load Balancers Implementations

Although most work on load balancing in distributed systems has been based on analytic or simulation techniques there have been some measurement studies on prototype systems usually with a small number of processors [Dikshit89], and a simplified workload model [Barak85].

Most implementations of load balancing in distributed systems have been done in an ad hoc manner [Bershad86, Hagmann86, Ezzat86] and have been added on the top of already existing operating systems. This involves a special syntax for command submission and a modification of the operating system to provide for remote execution mechanisms. Zhou and Ferrari [Zhou87] implemented an automatic load balancing

scheme with minor modifications to the operating system. Through an evaluation of a several load balancing algorithms they showed that load balancing can have beneficial effects on the system performance. If prototype measurement results based on the insights of the simulation results increase the confidence in the performance through load balancing, it is still a step away from a real product. Commonly missing are remote process management and control, and the user interface facilities.

Over the last decade many analytical, simulation, and prototyping studies of load balancing on distributed systems have been carried out. Despite the beneficial effect of load balancing shown through experimental systems, no commercial products are reported. However, in the case of parallel system there are some implementations such as Helios [1] which do effect load balancing on transputers systems. The potential reasons are:

*Technical*

Few distributed system built from scratch are successful. Even their developers do not use such systems because they are too slow [Renesse88]. This is due to the inherent complexity of distributed software. There is also a lack of distributed applications which justify the load balancing approach.

*Economical*

The workstations and communication hardware keep getting faster and cheaper. In most computing environments there is no real incentive to use resources efficiently. However, when the physical limit of single processor speed is reached, there would be a drive for more efficient use of the existing resources. Most of the software engineering experience is in a centralised environment where a vast amount of software packages exist. These products are not compatible to a

[1] Perihelion Software Limited

distributed system environment [Beck90]. The problem is therefore with the distributed systems rather than with the load balancing scheme.

We conjecture that the lack of commercial products that include load balancing schemes has more to do with the need for further maturing of distributed systems rather than the viability of load balancing schemes. More general investigations are needed for a better understanding of the behaviour of distributed systems subject to load balancing strategies, together with a clearer picture on the level of performance improvement achievable to justify the implementation costs. This could make load balancing services a reality in future distributed systems, thus achieving a near optimal utilisation of global computing resources without an adverse effect on the users' expectations.

## 2.7. Summary

In this chapter, we have surveyed research on load balancing algorithms according to the algorithm components and attributes, and the modelling of distributed systems for performance studies. An algorithm taxonomy was developed with an extented review of the adaptability attribute. A methodology for the design of adaptive load balancing algorithms was outlined. Based on this approach the design of some adaptive load balancing algorithms is considered in Chapter 3. A review of previous performance studies of load balancing algorithms revealed some over-simplifications in the system model assumptions made in both analytical and simulation work. This point is expanded in next chapter.

# CHAPTER 3

## Performance Study of Load Balancing Algorithms

### 3.1. Introduction

After many years of research into load balancing for distributed systems, there still remains many open questions that require further research. One of the most important is to understand the performance of load balancing algorithms on realistic systems and under more realistic operating conditions. Earlier studies have used very simple models of distributed systems and it is difficult to assess these load balancing algorithms on real distributed systems. The common approach followed is to propose new ideas on a load balancing algorithm component, and using simulation or mathematical techniques for a simple system model, it is shown that the proposed strategy performs better than the no load balancing case or some other algorithms.

In this research a much more complex system model is simulated and a thorough empirical investigation is carried out. Based on this model we evaluate a selected set of load balancing algorithms. We also propose an adaptive algorithm called *Diffuse* and modify the Random, Sender, Receiver, Symetric algorithms described in Section 2.5.1, as well as for the Diffuse algorithm to take into account the case of heterogeneous hosts processing speed in a distributed system.

### 3.2. System Modelling Issues

Below are described the system modelling issues that have a potential effect on the performance of a distributed system and the assumptions commonly made in related work. In a distributed system the essential performance factors are: file system structure, hosts speed configuration, communication bandwidth and protocols, load

balancing overheads, and the workload model.

*File System Structure*

In most reported studies a single file system structure is assumed (e.g. diskless with a shared file server, disk-based, or diskless with shared file server and a small local disk attached to each node). The disk-based file system structure is commonly assumed in analytical models. An exception is the work by Krueger [Krueger88a] where non-preemptive and preemptive transfers are compared under both diskless and disk-based structures. A *Place factor* is used to indicate the size of the task transferred which characterises each file structure. It is shown that non-preemptive transfers are preferred on diskless systems because they rely on a shared file server and only involve the transfer of the job command name. This contrast with a preemptive transfer where the complete process file and current state are transferred. This leads to improvement under disk-based model because a preemptive transfer is not more expensive than a non-preemptive one. The process state added to the transfer does not increase its size or complexity. Comparative studies of the effect of the file system structure on the performance of load balancing strategies are needed.

For the case of diskless workstation based distributed systems, Lazowska *et al.* [Lazowska86] point out that the file server's CPU tends to be the first resource in the system that gets saturated. Zhou[Zhou87] reached the same conclusion in the context of load balancing and reduced the number of clients from six to five to cope with a slow file server. What is the effect of the file server speed on the performance of different load balancing strategies?

*Hosts Speed Configuration*

In previous studies all the processors are assumed to have the same computing power as well as functionality (homogeneous processors). With the proliferation of

personal computer/workstations and the constant increase of their processing speed, it is very common to have a computer network with nodes of different computing speeds but which are compatible at the operating system and binary code levels. The case of heterogeneous processor speeds have been mainly considered in the context of centralised control systems [Tantawi85, Bonomi88]. Optimal probabilistic schemes are used where a weighting factor is given to the processor speed. In the simulation work of distributed systems by Stankovic [Stankovic84] heterogeneous processor speeds (i.e. different average service time for each host) were considered but no attempt has been made to adapt the algorithms to the heterogeneous environment.

Castagnoli states that heterogeneous environments are where many load balancing algorithms break down [Castagnoli86]. He suggests that a weighting factor be assigned for the particular CPU in the formula used to identify the node with the shortest queue of jobs (i.e. the best destination B).

$B = \min (w_1*(l_1+d_1), ....., w_n*(l_n+d_n))$

where:

$w_i$ : CPU speed weighting factor

$l_i$ : CPU load average

$d_i$ : total no. of jobs queued on that machine

Banawan [Banawan87] developed a heuristic algorithm based on the idea of scaled load index for an algorithm similar to Shortest algorithm described in Section 2.5, with scaled arrival rates. He concludes that the adapted version does improve the performance over the standard version. When only one fast node is used, the standard version degrades the performance at low utilisation levels. At heavy load level for all the speeds configurations the standard and adapted algorithms performance converges. The scaled load index is applicable only to algorithms with load vector based information policies. There is a need to develop adaptation mechanisms for random

polling negotiation based algorithms to take into account the heterogeneity of nodes speeds and job arrival rates in a distributed system.

*Communication Network*

In most reported work it is assumed that the communication device has a bandwidth large enough for there to be no contention or significant communication delays. The communication protocols commonly assumed in simulation work are "first come first serve", and "CSMA/CD" or "Token Passing" for prototyping based studies, but no comparative analysis has been undertaken in the context of load balancing. The work in [Mirchandan89] addresses the effect of job transfer delays on load sharing in a disk-based distributed system. It is concluded that the delays have no effect on the relative order of algorithms (Forward, Reverse, Symmetric). However, under long delays the algorithms have an identical performance except for heavy load levels. There is also a global degradation of the level of response time for all the algorithms under short delays with a spreading out of the curves. Further investigations of the interdependence of the communication device attributes (communication bandwidth and protocols) and the load balancing activities under both file system structures are needed.

*Workload Environment*

In previous studies it is commonly assumed that the workload consists of homogeneous users and jobs. For this type of workload non-selective job transfers are acceptable. When the workload involves two or more classes of jobs which reflects more accurately actual computing environments [Cabrera86], selective transfers where only long jobs are transferred to overcome the overhead of a remote execution, seem more appropriate. However, selective transfers involve a non-negligible job separation cost that must be taken into account. Heterogeneous jobs have been used in

[Krueger88a] for non-selective transfers, but no comparison to homogeneous jobs is reported.

*Load Balancing Overheads*

The overhead of a load balancing algorithm includes a communication cost and an execution cost. It affects the sending node, the receiving node as well as the transfer device. The communication cost is due to the exchange of status messages and the transfer of jobs across the network (CPU cost and communication delay). For a practical system the execution of communication protocols for packing of messages far outweighs the communication delay [Lazowska86] , and must be taken into account. The other costs associated with the algorithm are due to the execution of the information, transfer, and the negotiation policies of the algorithm. This is referred to as the execution cost and its level depends on the complexity of the load balancing algorithm.

The cost of handling the load balancing messages (probing/information, job transfer) and the increased traffic on the transfer device are usually assumed negligible. In [Zhou87a] the effect of non-negligible message overheads (5 to 40 msecs) and job transfer cost (50 to 400 msecs) for a diskless model are evaluated. It is concluded that under this wide range of overhead assumptions load balancing does still reduce the job mean response time. Further experiments are needed to evaluate the impact of non-negligible message overheads on both diskless and disk-based models.

## 3.3. Design of Load Balancing Algorithms

The following algorithms have been designed to address some disadvantages of the Symetric algorithm described in Section 2.5.1, and to provide load balancing algorithms adapted to a distributed system with heterogeneous hosts speed.

*1) Diffuse Algorithm*

This algorithm is inspired from the information exchange policy in [Barak85]. It emulates a "gas diffusion" process in its negotiation policy as opposed to the information policy in Barak's algorithm. It is symmetrically initiated and uses periodic polling of a single remote node. The load level at a node represents the number of jobs waiting plus the currently executing task. The term *load.i* is used at the node which activates the load balancing process while *load.j* is used at the node being polled (see Section 2.5). For every *timer_period* (using different start times to make the initiation of load balancing globally asynchronous), the node load is checked against the threshold:

1) if exceeding the *threshold* (load.i > $T_{si}$) , a request is sent to a random node ($L_p$= 1), this node replies with an ACCEPT message if it is underloaded (load.j < $T_{sa}$), otherwise it ignores the request. The requesting node transfers a job from its transferable jobs queue as a response to an ACCEPT message, or ignores the request if it is no longer overloaded (or overloaded but with an empty transferable jobs queue in the case of selective transfers).

2) if below the *threshold* (load.i < $T_{ri}$), a request to receive a job is made to a random node, the chosen node will respond by sending a job from its transferable jobs queue, or just ignores the message if it is also underloaded (load.j < $T_{ra}$). However, in the case of selective transfers, if the node is overloaded but the transferable jobs queue is empty, the node is considered as if underloaded and the message is also ignored.

3) if the load is normal (load.i = $T_{si}$), no load balancing is attempted.

To avoid the situation where a node is the sending and receiving states simultaneously the choice of the thresholds must be such that $T_{ri} \leq T_{si}$ and ($T_{ra} \geq T_{si}$) The node load regions and load balancing states for the Diffuse algorithm are depicted in Figure 3.1. This algorithm is adaptive in the sense that based on the current load level, it activates either is sender-initiated (SI) component or its receiver-initiated (RI) component. It is to be noted that this algorithm can be used for homogeneous as well as heterogeneous types of workload.

*2) Strategies Adapted to Heterogeneous Hosts*

When the processing speeds of the nodes in a distributed system are different, the instantaneous CPU queue length is not a good load metric. The load index, among other system and algorithm parameters, needs to be adjusted to maintain the performance of the system through load balancing. As has been shown in previous studies, the most influential parameters are: the threshold level above or below which the load balancing is triggered, the remote location selection (e.g. a random destination, one with the shortest queue or the first one whose load is below a given threshold), and the timer period for periodic algorithms.

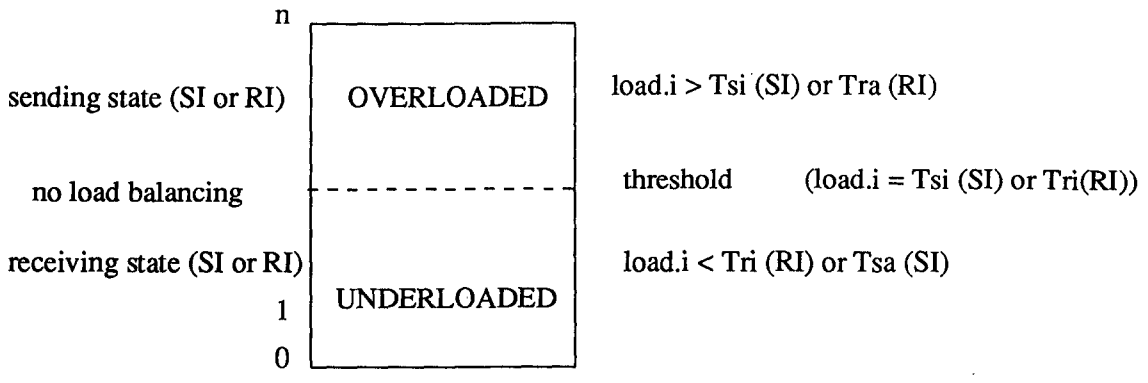


*Figure 3.1 Node Load Regions and Load Balancing States*

The scaled load index has been suggested as a way to deal with the heterogeneity of processing speeds [Castagnoli86]. The Shortest algorithm, modified to have a load vector where load values are scaled by a factor proportional to the node speed, lends itself to the scaled index mechanism. The newly arrived job is transferred to the node with the smallest value of *d*.

$d = (l+1)/s = K*l + K$ for $K = 1/s$ where

| | |
|---|---|
| $d$ | destination node scaled queue length |
| $l+1$ | CPU queue length including the new job |
| $s$ | node processing speed factor |

However, Zhou [Zhou87] has shown that this algorithm called DISTED in his work, when evaluated on homogeneous nodes, performs better than the Random algorithm only. This is due to the out of date global state information collection and the overriding of the nodes autonomy it involves. This approach is not considered further.

There are two ways to specify the workload for a heterogeneous system. These are known as *scaled arrival rates* and *identical arrival rates*:

- Scaled arrival rates

  The objective of scaled arrival rates is to maintain the same CPU utilisation level at the different nodes. The inter-arrival time used to generate the jobs is chosen to get the same utilisation level on all the nodes regardless of the processing speed.

- Identical arrival rates

  The above assumption about the jobs arrival rates is not justified in a workstation-based computing environment with similar users. It is the job arrival rates (not the CPU utilisation level) that are to be kept the same (i.e. identical inter-arrival times) for all the nodes regardless of their processing speed.

Two adaptation mechanisms have been developed to make some random polling based load balancing algorithms take into account the processing speed of the nodes in

the network: weighted destination and scaled timer period. The random destination is not chosen based on a uniform probability function as in the Random algorithm, but each node destination is given a weight proportional to the processor speed (i.e. *weighted destination*). This also applies to the choice of node to be polled for the algorithms based sender-initiated as well as receiver-initiated probing. For the Diffuse algorithm, the timer period is scaled to the node service rate (*i.e. scaled timer*).

Based on a the weighted destination and the scaled timer mechanisms, the following adapted algorithm versions have been developed: Random_a, Sender_a, Receiver_a, Symetric_a, and Diffuse_a. The algorithms versions without these mechanisms are referred to as standard algorithms.

### 3.4. Summary

This review of performance studies of load balancing algorithms revealed that some over-simplifying assumptions are made in the modeling of distributed systems, and that there is a need for load balancing strategies that adapt their policies to heterogeneous and rapidly changing workload, or include mechanisms to take into account the heterogeneity of nodes processing speeds. We set out to evaluate the load balancing algorithms on different models of distributed systems with more realistic assumptions and system design alternatives. Then it would become possible to identify the most appropriate algorithm for a distributed system knowing its attributes and workload environment. The research questions to be addressed are:

i) What is the effect of system attributes on the performance of load balancing and how does the Diffuse algorithm compare to others in the literature (i.e. Sender, Receiver, Symetric)?

ii) What is the effect of the workload model on the performance of load balancing algorithms?

iii) What is the effect of the heterogeneity of nodes speed on the performance on the the standard algorithms and what is the level of performance improvement when adapted versions of these algorithms are used?

In Chapter 4, the design as well as the implementation of the system built to examine the research questions identified above, are described.

# CHAPTER 4

## A System to Measure the Performance of Load Balancing Algorithms

### 4.1. Introduction

In this Chapter, the experimental system design and implementation, how to use the simulation package developed, and the simulated system validation are described. This description is divided into four sections:

(i) The design of the experimental system is outlined. This involves a description of experimental method, the distributed system models considered, the experimental objectives and factors, and the nature of the investigations to be carried out.

(ii) The essential components of the system under investigation are modelled. the default parameters values indicated have been arrived at through modelling decisions or experimental tuning for optimal performance, and correspond to the baseline system.

(iii) The simulated system implementation is described. This includes the simulation environment, the distributed system components, and an overview of the simulation package.

(iv) The calibration and validation of the simulated system is described.

Section 4.6 concludes this Chapter by providing a summary of the system features and its validation.

## 4.2. Experimental System Design

The goal of this study is to evaluate the performance of load balancing algorithms against particular distributed system attributes and workload models. For this purpose a system is to be built to allow experimentation with distributed system attributes, load balancing algorithms, and workload models.

Three methods have been used to study the performance of load balancing algorithms: analytical modelling (using queuing theory), simulation modelling, and prototyping. The first approach is often based on simplified model assumptions (e.g. instantaneous job transfers and at no cost), leading to results useful only to set performance bounds [Krueger87]. Also as shown in a survey by Wang *et al.* [Wang85] even simple load balancing schemes can lead to unsolved problems in queuing theory. This is particularly true for systems with nonhomogeneous process initiation rates, hyperexponentially distributed service demands, and a variable number of nodes participating in the system [Krueger88a]. We also reject the prototype based measurement method for the specific equipment required, the excessive development time needed, and the restricted control of the system parameters. We chose the simulation method of analysis for the advantages it provides: much less time to set up a model with realistic assumptions, makes it possible to have a complete control over all parameters and events of the system under study, and experimentation in virtual time [Jard88].

The objective of this work is to use modelling for load balancing not to present a comprehensive measurement study of a prototype system. This experimental methodology is justifiable since no specific real environment is targeted, and the aim of the experimentation being a demonstration of the effects different system characteristics on load balancing strategies, and an exhibition of the system behaviour to its full extent.

### 4.2.1. Experimental Models and Factors

Since the approach used for this research is to study the interdependence of various distributed system attributes and workload parameters, and the load balancing algorithms performance based on simulation experiments, the simulation model input involves many factors. Some have different quantitative levels and some have different qualitative nature. In order to have a manageable experimental environment, the maximum number of parameters are to be kept fixed based on modelling decisions or through experimental identification of optimal values. Given the large number of factor and level combinations that can be manipulated in the simulated distributed system, three categories of parameters can be identified. The structural assumptions which represent the system components that are fixed across a set of experiments (e.g. file system structure, workload model), while a second category of parameters are changed one at a time and constitute the experiment options or decision variables (e.g. communication bandwidth). The third category which includes the system load level and the load balancing algorithm option are used as experiment variables in the evaluation of the distributed system performance.

The description of an experiment involves the specification of the experiment attributes: the objectives sought, the input factors, and the performance metrics along with the format of the results presentation (tables, graphs, etc:). As the analysis of the results progresses, more model factors will be discovered as not having an impact on the performance and therefore their values should be fixed or the number of their levels reduced (e.g. load balancing strategies). It is also to be noted that changing the level of one factor might require the adjustment of other parameters to get an optimal operation.

As shown in Chapter two, a load balancing system modelling involves a representation of the load index, the load balancing algorithm, the workload, and the

distributed system attributes. Based on the distributed system attributes assumed four distributed system models are identified:

- system with disk-based homogeneous nodes.
- system with diskless homogeneous nodes.
- system with diskless heterogeneous nodes.
- system with disk-based heterogeneous nodes.

The baseline version for each model corresponds to the default parameter values and serves as a reference to the models with more realistic assumptions. The default system components have been arrived at through a combination of modelling decisions and experimental parametric tuning for optimal performance.

The essential system attributes considered are: the file system structure, the system nodes configuration and host modelling, and the communication device. The fixed parameters and default values of the simulated system have been presented under the default tables in the following sections. After many preliminary tests only the parameters for which the system response time is potentially sensitive are considered. The load index is defined as the node CPU queue length. Homogeneous as well as heterogeneous workload models are considered. Table 4.1 depicts the experimental factors to be investigated for each system component along with their options and levels.

The algorithms implemented are random polling based around the following strategies: Sender, Receiver, Symetric, Random, and Diffuse. A textual form description is given in Sections 2.5.1 and 3.3. They cover different information and control policies. This includes the following features: dynamic, adaptive, sender-initiated, receiver-initiated, symmetrically-initiated, periodic and aperiodic activation, and versions adapted to heterogeneous nodes.

Periodic algorithm (a timer-driven version of the Receiver algorithm), Shortest (system-wide load information vector based algorithm), Sender_Df variant of Sender, Symetric_Df variant of Symetric algorithm have also been investigated. These variants use the inter-node load difference in the negotiation policy instead of the threshold level. In order to keep the number of algorithms under evaluation small, they have been eliminated from further investigations either because they do not improve the performance significantly or because they behave with no significant difference to other selected algorithms or do not have the autonomy or scalability attributes.

### 4.2.2. Performance Studies

Previous studies of the performance of load balancing strategies were based on simplified distributed system models and with no consideration of the effect of some essential system design options such as communication model, heterogeneity of node speed, and file system structure. To address the research questions stated in Chapter 3, a series of simulation runs are carried out. The experimental factors are varied one at a time and their influence studied.

#### 1) System Calibration and Validation Experiments

The aim of this first experimental phase is the calibration and validation of the simulation model of our system. Based on the model used in Eager *et al.* [Eager86] and Michandaney *et al.* [Mirchandan89] the following issues are addressed.

- calibration of workload model, local and global schedulers, load balancing algorithms, and transfer device model.
- reproduction of literature results for Sender, Receiver, and Symetric algorithms.
- validity of results checking.

A. Distributed System Attributes

1) System size and hosts configuration
  a) Network size: 5, 10, 20 nodes
  b) Speeds configuration
    -Homogeneous nodes with single speed configuration: 1 job/time unit
    -Heterogeneous nodes with two classes of nodes speed: 1 and 2 jobs/time unit
  c) Local scheduling discipline: FCFS, Round Robin, preemptive priority FCFS
2) File system structure
  -Diskless nodes with shared file server
  -Disk-based nodes with no shared file server
3) Communication device attributes
  -topology: bus, ring
  -protocol type: FCFS, CSMA/CD, TOKEN PASSING
  -data transfer rate: 5 to 100 Mbits/sec

B. Workload Models

  a) nature of jobs and service demands
    -homogeneous jobs (single class of jobs)
    -heterogeneous jobs (two-classes jobs with short/long proportions: 95/5, 70/30)
      .non-selective transfers
      .selective transfers
  b) job initiation rates or load levels
    -homogeneous users load levels:
    (0.1), S (0.2), (0.3), L (0.4), (0.5), M (0.6), (0.7), H (0.8), V (0.9)
    -combination of heterogeneous users: 4S, 2M, 4V for homogeneous nodes
    -arrival rates for heterogeneous nodes: scaled, identical

C. Load Balancing Algorithms

  a) homogeneous nodes
    -Algorithm: Sender, Receiver, Symetric, Diffuse, Random, Shortest
    -Algorithm adjustable parameters:
      threshold (T), number of probes (Lp), timer period (Pt)
    -Load balancing overheads:
      .Message packaging/unpackaging cost (Msend/Mrecv)
      .Transferable jobs separation cost for selective transfers (Job_sep)
  b) heterogeneous nodes
    -Standard algorithms: Sender, Receiver, Symetric, Diffuse, Random
    -Algorithm adjustable parameters:
      threshold (T), number of probes (Lp), timer period (Pt)
    -Adapted algorithms: Sender_a, Receiver_a, Symetric_a, Diffuse_a, Random_a

*Table 4.1 Experimental Factors and their Levels*

**2) Experiments on Homogeneous Systems**

The objective of the experiments below is to measure the effect of system attributes on the performance of load balancing and in particular to classify the performance of the *Diffuse* algorithm with regard to others in the literature: Sender, Receiver, Symetric, and NOLB case. The effect of the following system attributes and workload models is to be evaluated.

- File System Structure

  How do the algorithms implemented behave under the shared file structure and local file system structure and which algorithm is most appropriate for each structure? Are the diskless and disk-based systems affected differently by the other experimental factors?

- Communication Attributes

  In most reported work, the broadcast device is assumed to have a large bandwidth (i.e. network subsystem not heavily loaded), there is no contention for communication device and therefore no communication delays. *Is this assumption valid for realistic conditions?* In this experiment we investigate the effects of the communication device attributes (device speed, communication protocols) which determine the level of communication delay and its effect on the performance of load balancing strategies.

- Load Balancing Overheads

  The impact of non-negligible load balancing overhead (message and job separation costs) on the distributed system performance is assessed.

- File Server Speed in Diskless Model

Does the file server speed have an effect on the distributed system performance in the presence of a load balancing scheme?

- Workload Model

Three workload models are considered: homogeneous users with homogeneous jobs, heterogeneous users with homogeneous jobs, and homogeneous users with heterogeneous jobs. For heterogeneous jobs the transfers can be non-selective or selective. This involves the identification of the level of performance improvement under different workload models as well as the relative order of the load balancing algorithms. For each workload model, the most appropriate local scheduling discipline is used.

**3) Experiments on Heterogeneous Systems**

The objective of the experiments below is to measure the effect of the heterogeneity of network nodes speed on the performance of standard load balancing algorithms, and to assess the performance improvements made when adapted versions of these algorithms are used. The following algorithms are evaluated: Random, Sender, Receiver, Symetric, and Diffuse. This evaluation is done on a ten nodes network: five nodes with service rate $\mu_1$ and five nodes with service rate $\mu_2$, where $\mu_1$ = jobs/time unit and $\mu_2 = 1$ job/time unit. Two types of workload are considered: identical arrival rates for all nodes, and identical node utilisation level on all nodes or scaled arrival rates.

## 4.3. Distributed System Model

The model of the system under investigation is divided into the following parts: file system structure, hosts configuration, communication network, workload model, and performance metrics. Each aspect is detailed below with a table of default and variable parameters given where appropriate. The notation used to describe the parameters of the workload model is depicted in Table 4.2.

| | |
|---|---|
| n | Number of nodes |
| E[T] | Mean job inter-arrival time to a node |
| $\lambda_i$ | Arrival rate at node i ($\lambda_i$ = 1/E[T]) |
| M | Exponential distribution describing jobs arrival process |
| $\mu_i$ | Processor service rate at node i |
| $\rho_i$ | Utilisation of node i = $\lambda_i / \mu_i$ |
| $\rho$ | System load = $\sum_{i=0}^{i=n} \rho_i / n$ |
| H | Hyper-exponential distribution describing jobs service demands |
| p, 1-p | Probability mix for the hyper-exponential distribution |
| E[S] | CPU expected service time for a job |
| $\sigma_S$ | Standard deviation of job service time |
| $C_S$ | Coefficient of variation of job service time = $\sigma_S$ / E[S] |

*Table 4.2 Workload Model Notation*

### 4.3.1. Overview of the Model

The loosely-coupled distributed systems modelled in this study consists of a set of autonomous computers connected by a local area network, exchanging information through a message passing mechanism [Chandras90], and operating in a cooperative fashion. In this environment the resulting pool of processors can be shared to improve the system performance by relieving overloaded nodes through remote execution of part of their load on less loaded nodes. The load balancing strategies to be investigated apply to a general-purpose distributed computing system composed of a cluster of workstations/compute servers [Ezzat86]. The nodes are assumed to be publically

owned, therefore there is no priority for local jobs over remote jobs of the same category.

Figure 4.1 shows a distributed-queue representation of the system inspired from the models in [Livny84, Ezzat86]. It consists of n identical nodes subject to external as well as transferred jobs arrivals. No prior knowledge of the jobs arrival time and service demands is assumed. This system can best be approximated by the n*(M/H/1) queuing theory model [Krueger88a]. The communication network is based on a broadcast bus device.

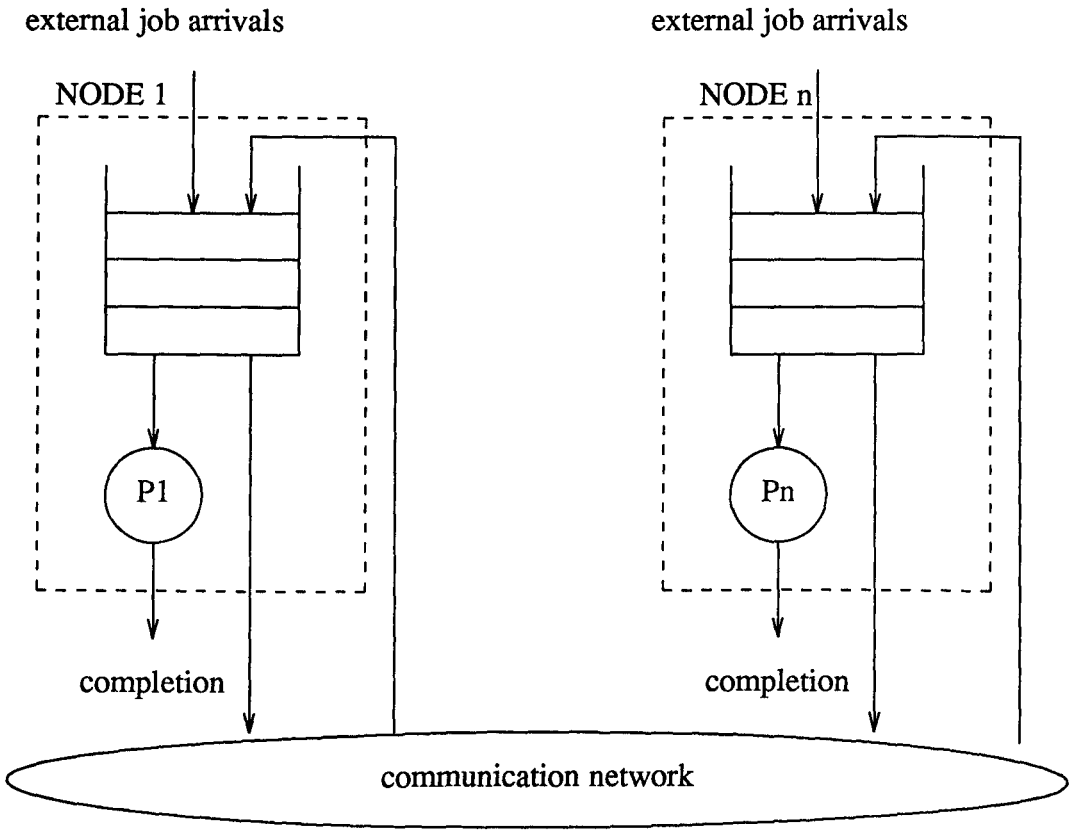


*Figure 4.1 n*(M/H/1) distributed system queuing network*

The file system structure of a distributed system is a major component whose impact on the performance of a load balancing scheme needs to be assessed. At one end of the spectrum are systems with no local secondary storage relying solely on a shared disk server (i.e. diskless nodes); at the other end are systems with local disk

storage at each node and no files replication (i.e. disk-based nodes). Between the two are hybrid architectures for example a common structure is one with shared file server and small local disk attached to each node used for swapping or holding of temporary files. We are interested in upper and lower bound performance, therefore we deal only with the two extreme cases: diskless and disk-based system structures.

In a diskless system, the communication device is used for remote file access, and other shared servers access (e.g. printer) as well as for load balancing activities. A remote job placement requires only the sending of a message (e.g. 1 Kbytes of data for program name, input and output files path description), and the saving of the results onto the shared server. A local job execution involves fetching the job image from the shared disk through the transfer device and saving the results back through the same channel onto the shared disk or their display to the screen for interactive jobs. The shared disk I/O operation demands are assumed evenly distributed and requiring 60,000 machine cycles. This structure is represented in Figures 4.2 and 4.3.

For a disk-based system, since each node has its own local disk, no file access is done through the transfer device. In this case a remote job placement entails transferring the full job along with its input to the new node. When the job completes the output is returned back through the communication device to the originating node. A local job execution involves fetching the job image from the local disk and saving back the output or their display to the screen for interactive jobs. The transfer device is used mainly for the load balancing activities. This structure is represented in Figures 4.4 and 4.5 .

The choice of the parameters for shared and local file system structures have been made through realistic system abstraction to achieve a comparable utilisation level and service time when the nodes are subject to the same workload with no load balancing

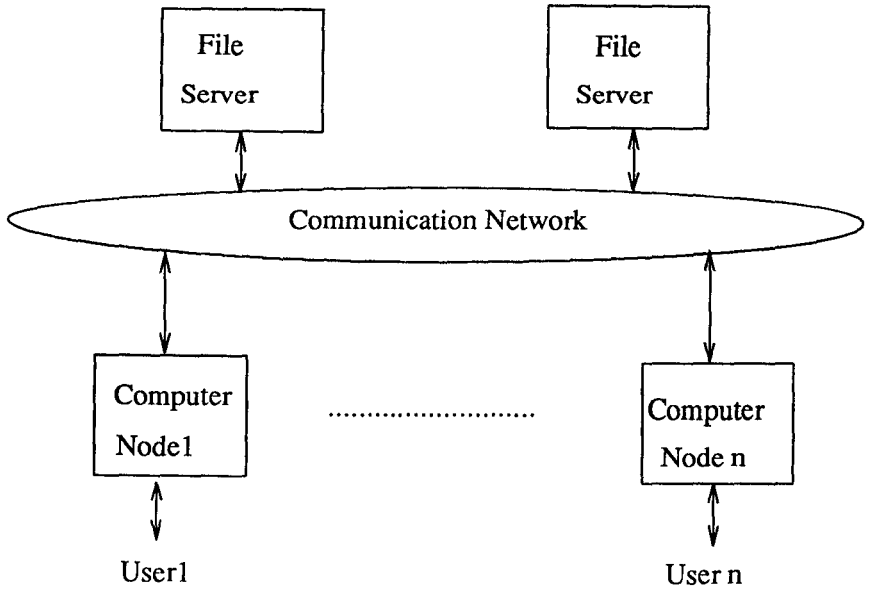


*Figure 4.2 Diskless Distributed System Architecture*

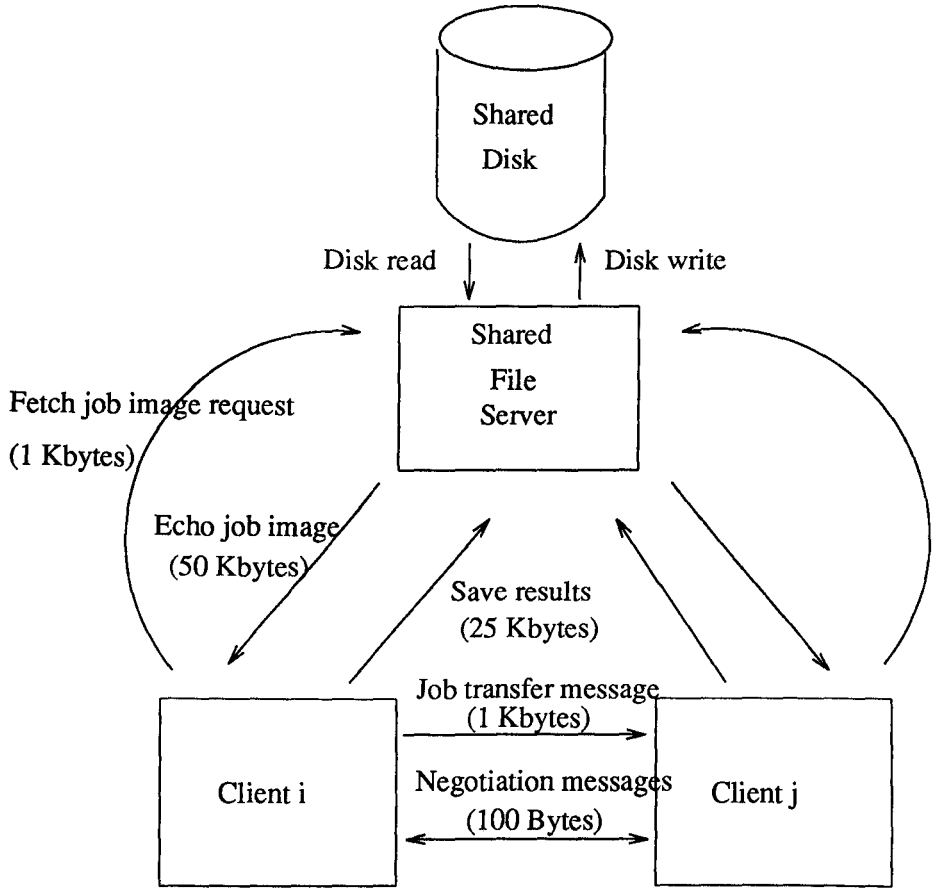


*Figure 4.3 Data Flow in Diskless Model*

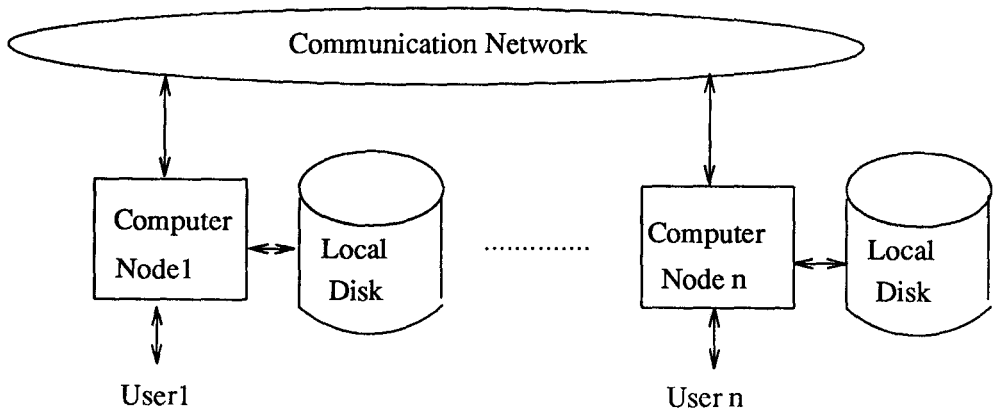


*Figure 4.4 Disk-based Distributed System Architecture*

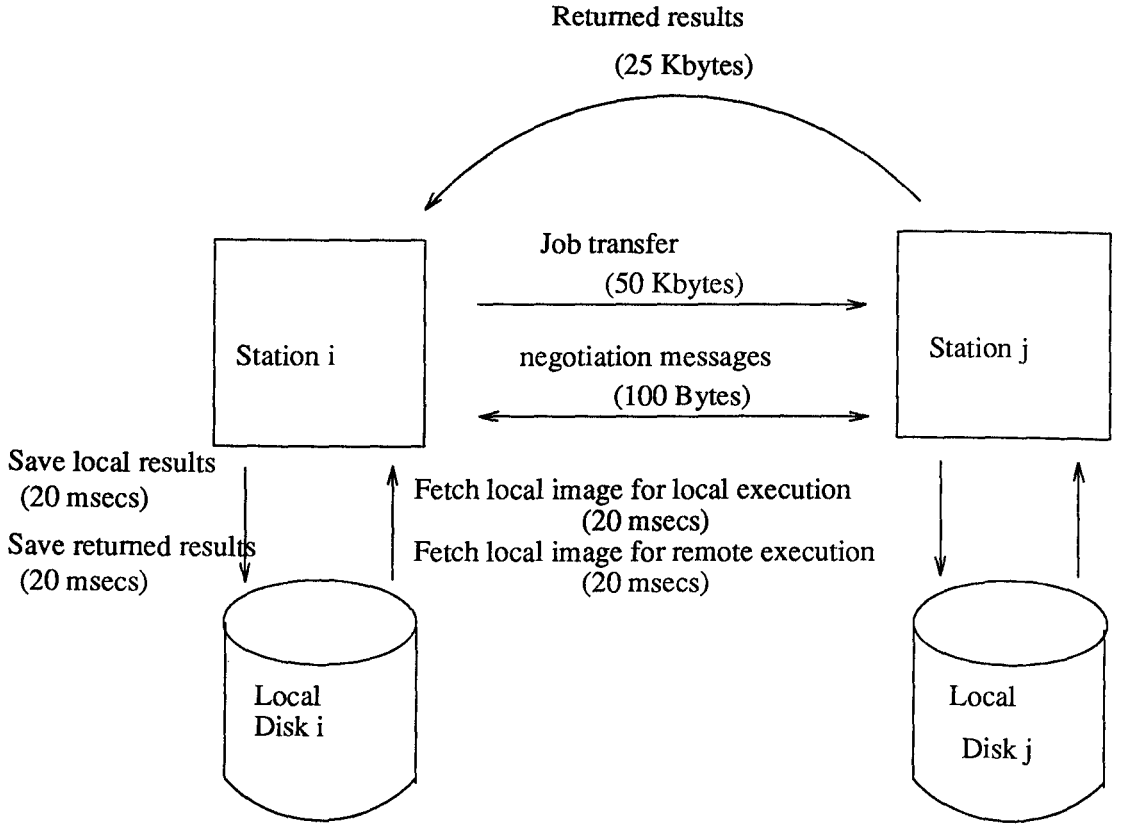


*Figure 4.5 Data Flow in Disk-based Model*

activated.

Each host is managed by a separate copy of kernel [Cheriton88] with associated communication protocols and the load redistributing software referred to in this study as a global scheduler. It is implemented based on the message passing paradigm, and is

replicated at every node to provide an interface to the users jobs, the load balancing strategy, identical global schedulers at other nodes, and to its own local scheduler. It is also in charge of the maintenance of the system state, resource queues, and the handling of other special provisions such as an ageing mechanism to solve the potential "queued forever" situation inherent to the preemptive local scheduling discipline.

### 4.3.2. Host Modelling

Although the network is made up of functionally identical processors, the processing speed of the nodes can be identical (homogeneous hosts) or the nodes come in different classes of processing speeds or service rates (heterogeneous hosts). Both configurations are briefly described below.

The first-come-first-serve (FCFS) local scheduling discipline [Ferrari85] for the execution of jobs is assumed. A first alternative discipline to FCFS is the preemptive priority FCFS (pFCFS) which gives a priority to short jobs [Eager88], taking into account the characteristics of the jobs by filtering out jobs with small demands which do not justify a remote execution [Ezzat86]. This also results in a better ratio of response time to service demands. The second alternative local discipline is the round robin (RR) strategy. Both the FCFS and the RR disciplines schedule the jobs independently of their actual service time.

Zhou points out that the CPU is the main contention resource in a computer system [Zhou86]. The memory is assumed large enough or a local hard disk is provided for swapping. The secondary storage devices are modelled as infinite servers with 40,000 cycles processing delays for each I/O operation as a rough approximation. This corresponds to about 20 milliseconds on common workstation processors with a processing speed (i.e. service rate $\mu_i$) of one job/second for a 0.5 μsecs machine cycle time. Further host parameters are depicted in Table 4.5 (Section 4.4.2).

With the proliferation of personal computer/workstations and the constant increase of their processing speed, it is very common to have a computer network with nodes of different computing speeds but which are compatible at the operating system and binary code levels. The execution time of a job depends on speed of the node where it is executed. There is an intuitive advantage of dynamic load balancing in such an environment. While some substantial related work has been done for centralised system, very few works have been reported on load balancing for distributed heterogeneous systems. Models with homogeneous hosts are evaluated in Sections 5.2 and 5.3, while heterogeneous hosts are considered in Section 5.4.

### 4.3.3. Communication Network Model

The broadcast communication device which is widely used for the interconnection of multicomputer systems and networks of workstations is assumed in this work. The jobs are executed independently at individual computers, with no inter-communication. The communication device is used for remote file access, job transfer, nodes status exchange, negotiation messages, and the interaction with other shared servers. The information delivered across the network can be classified into two categories: messages (i.e. status information, negotiation, job descriptor), and files (i.e. remote job image from shared file server or originating host, returned results, other files). The default communication device assumed is a 10 Mbits/second broadcast device with a FCFS protocol. However, in Sections 5.2.1 and 5.3.1, a simulation study of the performance of other protocols [Mitrani87] (i.e. CSMA/CD, TOKEN PASSING ), as well as the transfer speed, is undertaken. It is to be noted that the practical transmission speed of a communication device is only a fraction (20 to 40 %) of the theoretical speed limit given [Johnson89]. This is due to contention delays and packets overheads. Table 4.3 depicts the communication device default parameters.

| | |
|---|---|
| Network topology | bus, ring |
| Device data transfer rate | 5 to 100 Mbits/sec |
| Word size | 32 bits |
| Packet size | 1 Kbytes |
| Protocols | FCFS *(options: CSMA/CD, TOKEN PASSING)* |
| Packet Overheads | 12.8 μsecs |

*Table 4.3 Communication Device Parameters*

The transmission time of a job depends on device data transfer rate, job size, and file system structure. However, the actual job transfer time is unpredictable and depends on: message packing time (Msend: fixed value), transmission time (based on job size, packet structure), unpredictable network delay (device speed, traffic level, and communication protocols used), message unpacking time (Mrecv: fixed value).

In a typical general purpose computing environment the individual nodes would be presenting a range of input/output to the network because the hosts operate with differing performance characteristics (e.g. file server, line printer, nodes with heterogeneous speeds, heterogeneous users). As a consequence the load on the network would most likely be asymmetrical both in arrival rate and transfers size.

### 4.3.4. Workload Model

To stress the importance of the workload model for load balancing we use a quotation from Zhou "*Load balancing is based on exploiting the dynamics of workload*"[Zhou87]. The users' environment being modelled consists of independent jobs arriving at individual computer nodes based on Poisson distributions. Both homogeneous as well as heterogeneous users are considered.

The workload nature can be assessed along the arrival patterns and intensities (i.e. arrival rate $\lambda_i$ ), and the job characteristics: type, mix, size, service time [Krueger88]. The models used in this study are based on the results in [Cabrera86, Lee86, Zhou86].

The exponential distribution is frequently used to represent job arrivals at each node. This is referred to as the Poisson process [Lavenberg83]. This model fits homogeneous random job arrivals well, but to represent bursty job arrivals to the system each node is subjected to a different load level. These are called heterogeneous initiation rates and correspond to heterogeneous users.

The other aspect of the workload model is the job size and the service demand. Zhou [Zhou88] and Leland *et al.* [Leland86] have shown that exponential distributions approximate poorly to process service demands, instead hyperexponential distributions are to be used [Leland86]. A hyper-exponential distribution H is a mixture of two or more exponentials [Krueger88]. In our model two-classes of jobs are assumed and are simulated by two exponentials distributions one for short/immobile jobs and the other for long/transferable jobs. The combined service time S is given by [Kobayashi78]::

$$P(S \leq t) \equiv H(t) = p(1-e^{-\mu_1 t}) + (1-p)(1-e^{-\mu_2 t}) \tag{4.1}$$

where $p$ is the probability of a job being from the short class such as $0 < p < 1$. The service time $E[S]$ is defined by

$$E[S] \equiv 1/\mu = p/\mu_1 + (1-p)/\mu_2. \tag{4.2}$$

The coefficient of variation $C_S$ for a 2-stage hyper-exponential job service demands is defined by [Lavenberg83]:

$$C_S = [[2(p/S_s^2+(1-p)/S_l^2)/(p/S_s+(1-p)/S_l)^2]-1]^{1/2} \tag{4.3}$$

where $S_s$ and $S_l$ are mean service time for short and long jobs respectively.

In addition to the service time length, jobs can be categorised based on the nature of their service demands: CPU-bound jobs, and I/O bound jobs also called interactive jobs. In this simulation study only CPU-bound jobs are eligible for remote execution, all interactive jobs are processed locally. Also processed locally are immobile jobs which include jobs requiring short service time, and local node dependent jobs whether short or long.

It is difficult to estimate the execution time of a job as opposed to transfer time which can be assumed proportional to the program length. However, the separation of short jobs from long jobs can be based on the job initiation command. The choice is made by the user or by an enhanced command interpreter. For the latter case a configuration file containing the names of jobs eligible for remote execution is provided. If the expected processing time of a job is less than $T_{cpu}$, a threshold value, then it is not worth executing remotely. An empirical value of $T_{cpu}$ is 2*C where C is the minimum wall clock transmission time of a job [Castagnoli86]. The service time of a job depends on its service demands, the file system structure, the load level, and the processor speed.

As further defined in Section 4.4.5, the system workload is generated artificially using probability distributions. Although these functions may not represent any specific real environment, they give a good approximation of the fluctuations of workload under small (S), light (L), moderate (M), heavy (H), and very heavy (V) load levels, and the service demands that the load balancing strategies must handle. To evaluate the performance at different load levels, the system load is varied by shortening or lengthening the mean inter-arrival time for users jobs at each node. Table 4.4 depicts the workload parameters with one job/second service rate hosts assumed.

### 4.3.5. Load and Performance Metrics

For load balancing algorithms the local processor load level is the prime factor used to decide whether to allocate a process locally or to transfer it to a remote node for execution. Many alternatives for its evaluation are outlined in Section 2.3.1. Among these alternatives, the CPU queue length is the most favoured load index [Zhou86], for its correlation to the response time and the instantaneous CPU utilisation, and for its quick and efficient evaluation. The CPU queue length is considered as the main

| | |
|---|---|
| Job size | exponential with mean= 50 Kbytes |
| Jobs arrival | Poisson process with different system load levels: |
| | (0.1), S (0.2), (0.3), L (0.4), (0.5), M (0.6), (0.7), H (0.8), V (0.9) |
| a) Homogeneous jobs service demands | exponential |
| | E[S] = 1.0 secs |
| b) Heterogeneous jobs service demands | hyper-exponential |
| combined job | E[S]= 1.0 secs, ($p$*short + (1-$p$)*long) |
| short job | $S_s$ = 0.8 secs, $p$ = 0.95 |
| long job | $S_l$ = 4.8 secs, 1-$p$ = 0.05 |
| $C_S$ | 1.04 |
| c) Heterogeneous jobs service demands | hyper-exponential |
| combined job | E[S]= 1.0 secs, ($p$*short + (1-$p$)*long) |
| short job | $S_s$ = 0.4 secs, $p$ = 0.70 |
| long job | $S_l$ = 2.4 secs, 1-$p$ = 0.30 |
| $C_S$ | 1.23 |

*Table 4.4 Workload Parameters*

resource of contention and used as the load indicator throughout this work.

The main objectives of the scheduling strategies for an autonomous computer system are to minimise the job (process) response time or the average time spent by a job in the system, to maximise the CPU utilisation, to maximise the system throughput, and to ensure fairness. The latter represents the quality of service from the user's point of view and can be represented by the response ratio of the wait time over the service demands.

When dealing with a distributed system, the notions of system balance and stability are to be introduced. The goal of a load balancing scheme is to 1) minimise the job mean response time with a minimum job movement, 2) to balance the load over the nodes in the network, and 3) to minimise the load balancing costs. The performance of a load balancing algorithm is a trade-off between its benefits (overall system response time, balance factor, node utilisation) and its costs (job movements,

communication and control overhead). The balance factor represents the queue length difference between the least loaded and the most loaded nodes in the network [Livny84].

There are two types of *system instability* [Zhou87a] that appear in a distributed system: the host overloading or flooding, and the job thrashing. *Job thrashing* corresponds to a successive transfer of a job from one node to the other due to bad decisions. Job thrashing is not considered since jobs are allowed a single move in the load balancing strategies under investigation. *Host overloading* occurs when a number of nodes detect that a node is underloaded and each simultaneously transfers a job to it. This can be evaluated by the level of job movement introduced by the load balancing algorithm and the resulting bad decisions rate.

In this study the system performance is evaluated in terms of:

a) Overall job response time ($R$)
   - -mean value
   - -standard deviation to measure the response time variability (i.e. response time predictability)

b) System stability
   - -percentage of jobs moved across the network
   - -percentage of bad decisions which indicate the level of host overloading

c) Load balancing cost
   - -number of negotiation/information messages exchanged per host per second
   - -percentage of CPU utilisation increase due to load balancing activities

d) Transfer device performance indices [Hayter88]
   - -percentage of network utilisation
   - -mean request delay (level of network/host devices interactions)

The relation between the response time $R$ and the system load is given by the Little formula $L = \lambda R$ [Lavenberg83] ; where $L$ is the load, $\lambda$ the arrival rate, and $R$ the response time. This formula states that the average number of customers in the system is equal to the product of the arrival rate and the average system response time. When

applied to a system of n subsystems, it becomes:

$$R = \sum_{i=1}^{i=n} \lambda_i R_i / \lambda \qquad (4.4)$$

## 4.4. Simulated System Implementation

The testbed environment for this study is composed of artificial workloads generated using probability distributions to drive a simulator that implements a number of load balancing algorithms in a loosely-coupled distributed system environment. Four system options are simulated: homogeneous diskless workstations with file servers, homogeneous disk-based workstations with no shared file server, heterogeneous diskless workstations with shared file servers, and heterogeneous disk-based workstations with no shared file server. A graphical representation of the components of the simulated system is shown in Figure 4.6. In this section, the implementation of this simulated system is described.

### 4.4.1. Simulation Environment

Simulation is an important stage in the development of new load balancing algorithms. It is based on an abstract model of the real system, which is usually specified by mathematical or logical relationships. The model is described in terms of its state, entities and their attributes, sets, events, activities and delays [Banks84]. It is by simulating an algorithm that we increase the confidence in its superior performance, demonstrate the existence of errors or gain new insights into its behaviour under more realistic model assumptions. The intended study requires the simulation of the following distributed system components:

1) Distributed system structure
    a) computing node
        -CPU, processor speed
        -user process creation, execution, destruction functions

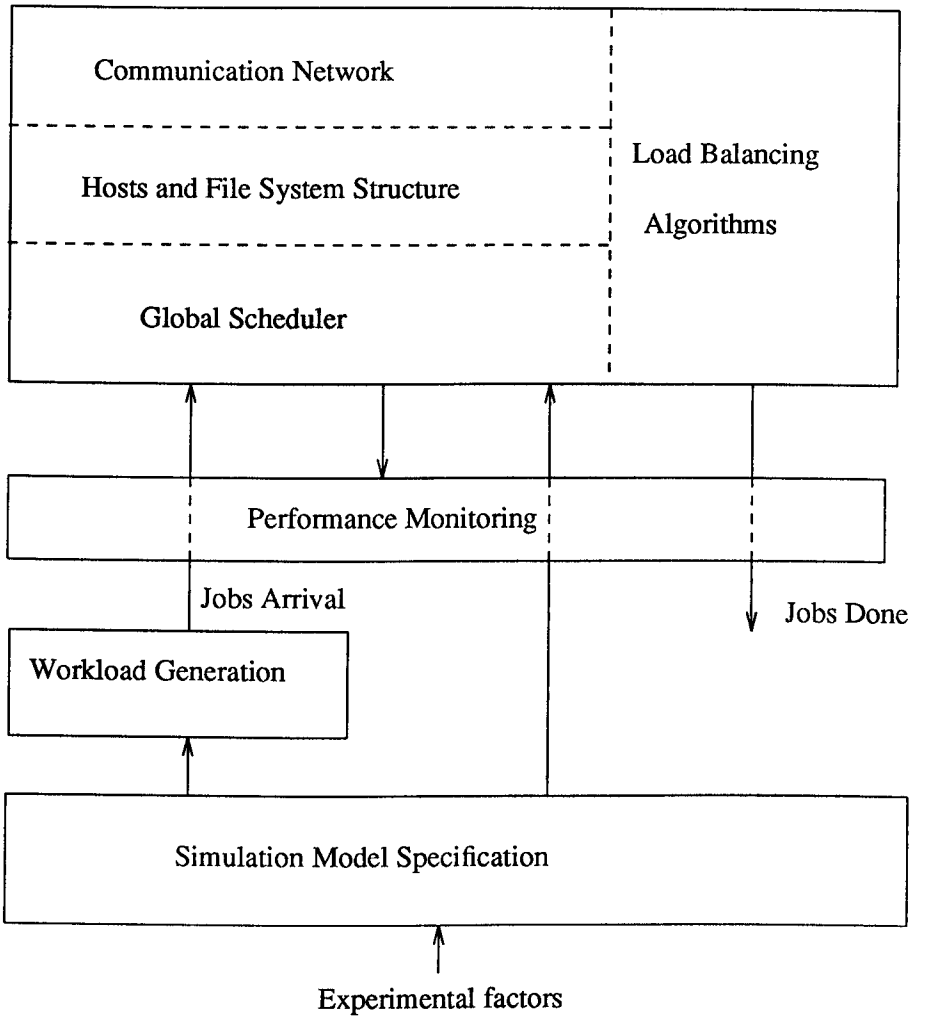


*Figure 4.6 Simulated System Components*

-local process scheduling discipline

b) communication device

-communication protocols, communication bandwidth

-network data packet structure

c) distributed kernel mechanisms for

-interprocess message-passing

-process placement (non-preemptive)

-global state information exchange

d) maintenance of a global time source

2) Decentralised load balancing algorithms

-algorithm structure

-adjustable parameters and policies

3) Workload generation
   -probability distributions type and mean value
      for job arrivals, job size and service demands
   -system load specification
4) Statistics information generation (both at node and network levels)
   -length of virtual time over which the experiment is carried out
   -period at which evaluation is invoked
   -nature and volume of information generated
   -computation/presentation of relevant results in tables, graphs etc:.

This tool must also place the simulation model under the control of the user [Casavant87a]. The performance objectives and metrics specification, the distributed system components parameters, the workload specification, and the load balancing algorithms options and parameters should be accessible to the user.

The simulation of a system is not a goal in itself. It is a means to learn more about the behaviour of the system under study and to make specific decisions based on the simulation results obtained. The main purpose of this simulated system is to implement the essential features needed to study the behaviour of various models of distributed computer systems, and to provide an environment in which experiments on load balancing algorithms can be carried out.

We have chosen Network II.5 [CACI89] as a software design aid for the building blocks it provides for the simulation of computer systems, and the computer communication structure which is included. It is to be noted that Network II.5 provides a more realistic simulation of communication device, and powerful probability distributions to simulate the workload. On top of this structure we designed and implemented our models of distributed computer systems, and the load balancing algorithms to be evaluated as specified in Section 4.2.1. This simulator has been

developed within a Simscript II.5 [1] language environment. It is a general purpose simulation language based on the process interaction simulation strategy and is of a declarative type. The simulation model obtained on such an environment lends itself to a diagrammatic representation (two-dimensional picture) [Evans88], which is preferred to a textural representation (sequential program text or flow chart). The software module bubble chart representation is used to describe the simulator logic. It is based on modules entities, modules precedence, semaphore dependency, and message dependency.

In the following sections, the designed and implemented entities are documented at a high level and their representation justified in accordance with the system model specified in Section 4.3. The choice of the features of the distributed computer system to be implemented is to be tailored to the experimental needs and system model requirements. Some aspects of these entities are assumed negligible, while others are actually simulated using abstract models which capture the essential features, and may be considered as a good approximation of reality. The main system hardware and software entities simulated are: autonomous hosts, communication network, file system structure, global scheduler, local scheduler, load balancing algorithms, workload generation, performance metrics monitoring, and other simulation control considerations. Below is a summarised description of each entity. The scheduling component of the distributed system is split into a local scheduler which manages the access to the processor for the jobs that are to be processed locally, and a global scheduler which redistributes the system workload among the nodes through job transfer. The local scheduler is described under the next section.

[1] CACI Products Company

### 4.4.2. Autonomous Hosts

The processing nodes are simulated as autonomous entities with the instruction repertoire (processing, message, assignment, read/write) needed to implement the system model software components. The relevant parameters are made tunable to test the effect of different values. Since the autonomous nodes communicate only through message passing, the message passing primitives implementation is fairly sophisticated and includes all the needed global inter-process communication mechanisms for buffering, packing, routing/broadcast, and interactions with the transfer device. Messages are sent to other nodes in a non-blocking manner. To reduce the processing capacity taken by the load balancing messages from the main processor, an input controller is provided. This allows the processor to receive input messages while executing other modules. The received messages are put in a received messages list and can be consumed when appropriate provided there are modules to consume them. In the absence of the input controller, the processor must work the entire amount of time it takes to receive the message from the communication device. If the processor is busy, it will block both the sending processor and the connecting transfer device.

The local scheduler or kernel provides the necessary mechanisms for the process creation, execution, interruption, destruction of user as well as supervisory processes. The local discipline options implemented are: FCFS, preemptive priority FCFS, and Round Robin. An optimal time quantum of 50 msecs for the Round Robin discipline was identified experimentally. The access to the local scheduler is regulated through a maximum value of the local queue (LQ) based on the local discipline. Except for the transferred jobs which are fed to the local scheduler upon arrival at the remote node, both local short and long jobs are queued in their respective wait queues before their submission. Since we are not dealing with preemptive load balancing policies, once a job is handed to the local scheduler, it becomes a process out of the control of the

global scheduler, its process image is already fetched or currently being fetched for local execution. A graphical representation of the local scheduler is included in Figure 4.7 (Section 4.4.4). The relationships between the different queues is as follows. The CPU queue length includes the short wait queue, the long wait queue, and the local queue (LQ). The local queue (LQ) represents the local ready queue, the interrupt queue, and the resident process. The main characteristics of the local scheduler are depicted in Table 4.5.

To get an accurate model abstraction, the file system structure is to be represented. Basically the file system structure specifies where the files are held and where the computing results are to be saved. In this study the case where all files reside at a shared file server (i.e. diskless model), and the case where each computing station has its own file system (i.e. disk-based model) are both simulated. The default network size experimented with is ten clients and one server for the diskless model and ten autonomous hosts for the disk-based model. Network sizes of five and twenty nodes have also been considered to measure the sensitivity of the results on the choice of the network size.

### 4.4.3. Communication Network Attributes

The simulated transfer device models the communication layers up to the transport layer, and allows the user specification of the transfer device speed, the data packet structure, and the medium access control protocols with their relevant parameters. The nodes connected to the broadcast bus are: FILE_SERVER, $NODE_1$, ...., $NODE_n$ where n

| | |
|---|---|
| Number of hosts | 10 hosts *(options: 5, 20 nodes)* |
| Host service rate ($\mu_i$) | 1 job/time unit (*option: 2 jobs/time unit for fast hosts)* |
| Local discipline | FCFS *(options: pFCFS, RR100)* |

*Table 4.5 Local Hosts Parameters*

represents the network size. The common characteristics of the transfer device are chosen as specified in Table 4.2 (Section 4.3.3). The protocols options implemented are: FCFS, Collision, and Token Passing. They are detailed below.

*FCFS protocol*

For this protocol a simple rule is used. The request to the transfer device are serviced in the order they are made. The node keeps the device until its transfer instruction is completed, regardless of how long it takes. A central controller is used to arbitrate among contending communication device users.

*COLLISION protocol*

This corresponds to the IEEE 802.3 carrier sense multiple access protocol with collision detection (CSMA/CD). A broadcast transfer device can be in one of the states: *idle*, *unsettled* or *busy*. It is unsettled during the collision window. When still in use after the collision window period is over, its state is busy.

A *collision* occurs if two or more nodes "see" the transfer device as idle and both try to use it (i.e. execute a message instruction or transmit a set of packets). The collision is detected when the packet received during the collision window is different from the packet transmitted. The *collision window* is the period of time during which the transfer device is vulnerable to collision after a new user takes it. This is due to 1) propagation delays due to physical separation of devices, and 2) delays between checking a transfer device status and actually beginning to transmit. It is estimated as the time required by light to travel between the two most widely separated stations [Cheung88, Coulouris88]. For a distance of 20 meters, the collision window is $20/3*10^8 = 0.066$ μsecs. It should be less than 5 μsecs for a one kilometer distance cable.

When a collision occurs, a *jamming signal* is sent to all stations. It ensures that all stations know of the collision and back off when they should. It is a collision consensus

enforcement strategy, though it is not an essential feature of CSMA/CD protocols [Hammond86].

The *contention interval* is the additional amount of time a node has to wait before attempting to access a transfer device, once the requested device becomes idle taking into account the assumed inter-packet interval time. Since CSMA/CD does not include such a feature, it can be assumed with a value zero.

After a collision the period of time to wait before trying again is called (i.e. *retry interval*). It can be chosen as an arbitrary multiple of the collision window. The *IEEE backoff* algorithm based statistic distribution is commonly assumed for the retry interval [Hammond86].

The *jam time* is the time length of the jamming signal sent by both users when a collision is detected, then wait for retry interval. Since the jamming signal is not an essential feature of CSMA/CD, the jam time can be assumed with a value zero.

The main parameters of the collision protocol are depicted in Table 4.6.

| | |
|---|---|
| Collision window | 0.066 μsecs |
| Retry interval | standard backoff distribution |
| Jam time | 0.0 μsecs |
| Contention interval | 0.0 μsecs |

*Table 4.6 Collision Protocol Parameters*

*TOKEN PASSING protocol*

For this protocol the requests to the communication device are ordered in a dynamic manner. The nodes are arranged into a logical ring; with the access granted in a sequential manner. Once holding the device, the node can use it for a continuous series of transfer instructions. The size of the series is specified by the key attribute of the node. A *node key* is used to indicate the number of consecutive transfer device accesses (i.e. message instructions). The *token passing time* is the time it takes to pass

the token from one node to the next (i.e. delay to add realism to the model). When a token passing time value is specified the transfer device will always be 100% busy. For this protocol, a key value is specified for each node connected to the ring communication device. In this study the key value is one for all the nodes including the file server.

### 4.4.4. Global Scheduler

The global scheduler is implemented on top of the local scheduler and has a network-wide scope. It is the scheduling component of the distributed system that is replicated on every node, and provides an interface to user jobs, the load balancing algorithm, identical global schedulers at other nodes, and to its own local scheduler. The functions of the global scheduler include:

- *Job Separation*

  This function involves the identification of long jobs from short jobs. It is an important function, in the case of heterogeneous jobs, because only long jobs are worth executing remotely on lightly loaded nodes despite the communication overheads for a service time E[S] > $T_{cpu}$. There are two ways to implement it, by putting the burden on the user to identify transferable jobs and submitting them with a specific command identifier, or by enhancing the command interpreter for example adding a software routine that uses a database containing the name of possible commands to separate the two categories of jobs. In this simulation model, this function is assumed to be a black box which generates the job category based on the uniform distribution ($p$) which is also used to generate the hyper-exponential service time distribution. However, a job separation parameter is provided to evaluate the effect of this overhead on the load balancing algorithms performance.

- *System Information Gathering and Maintenance*

  This module continuously updates the instantaneous local load value, periodically updates the system load vector for the load balancing requiring full system state information. It also periodically updates the estimated job arrival rate for the adaptive load balancing strategies.

- *Maintenance of Resources Queues*

  Once the decision to place a job in the local node is made, this routine requests a job image from the file system and puts the job into the queue of the local node. The local node queue is regulated by this routine to keep the number of jobs/processes allowed into the local scheduler below a fixed maximum value. This value depends on the local scheduling discipline, and the file system structure. It includes the process currently running, the interrupted processes, and the jobs in the local ready queue. Other resources queues to be maintained are: transferable jobs queue, immobile jobs queue. Jobs are guaranteed execution after one transfer, they are fed to the local scheduler upon their arrival at the remote node.

- *Load Balancing Algorithm Activation*

  Whenever the conditions of a need for load redistribution are met, the load balancing algorithm is activated. The transferring of jobs is given a preemptive priority over the processing of users jobs.

- *Remote Nodes Interface*

  An inter-processor messages handler is implemented to manage the negotiation, information, and jobs messages exchanged between the different nodes of the distributed system. It also transfers jobs for remote execution, and handles the jobs transferred from other nodes.

The implementation of the simulation model involves supervisory software and user processes. The supervisory software includes the global scheduler functions, the jobs generation modules, the load balancing algorithm modules, and is executed at a priority higher than the user job processing modules. Two graphical representations of the global scheduler can be made depending on the nature of the activation of the load balancing algorithm: aperiodic (job arrival or departure initiated) and periodic initiation for the Diffuse algorithm. The distributed scheduler structure which include both local and global schedulers is shown in Figures 4.7 and 4.8 for the case of heterogeneous jobs with selective transfers. In the case of homogeneous jobs and heterogeneous jobs with non-selective transfers, the "Jobs Separation" and "Immobile Jobs" boxes can be removed. This means that the external jobs are fed directly to the "Load.i" box (in the case of an aperiodic scheduler) and to the "Transferable Jobs" queue (in the case of a periodic scheduler).

### 4.4.5. Models Generation and Performance Monitoring

After the description of the system to be simulated and the specification of the structure of the system model, now we discuss the generation of the simulation model. The simulation model generator, developed using a set of Unix shell scripts, takes the experimental factors as input and uses a library of model components to generate the appropriate simulation model. The options for each component of the distributed system, the workload model, the load balancing algorithm including the values for its parameters are selected. Next the simulator is invoked with the proper control parameters. At the end of the run length period the results are dumped into the output file. Not all the data in this file is useful for each experiment. To get only the needed results an output filter has been developed. It collects the essential metrics and generate the results tables and graphs input files. It also computes and produces other

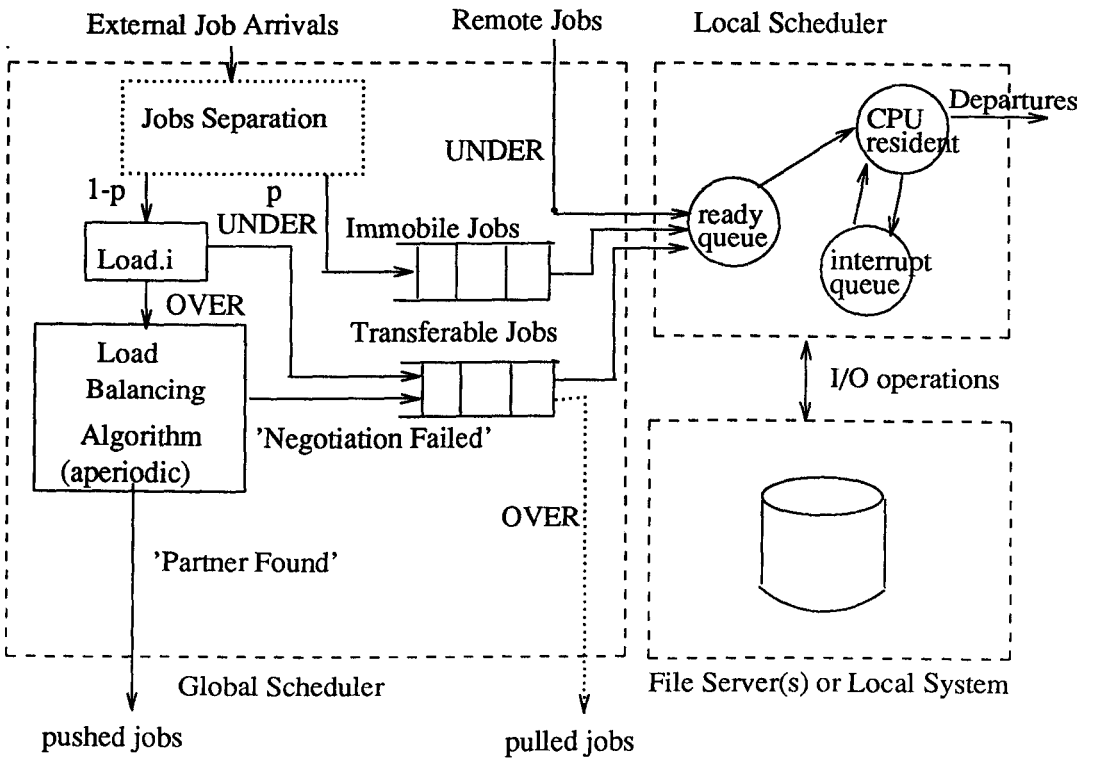


*Figure 4.7 Distributed Scheduler Structure- Aperiodic Algorithms (selective transfers)*

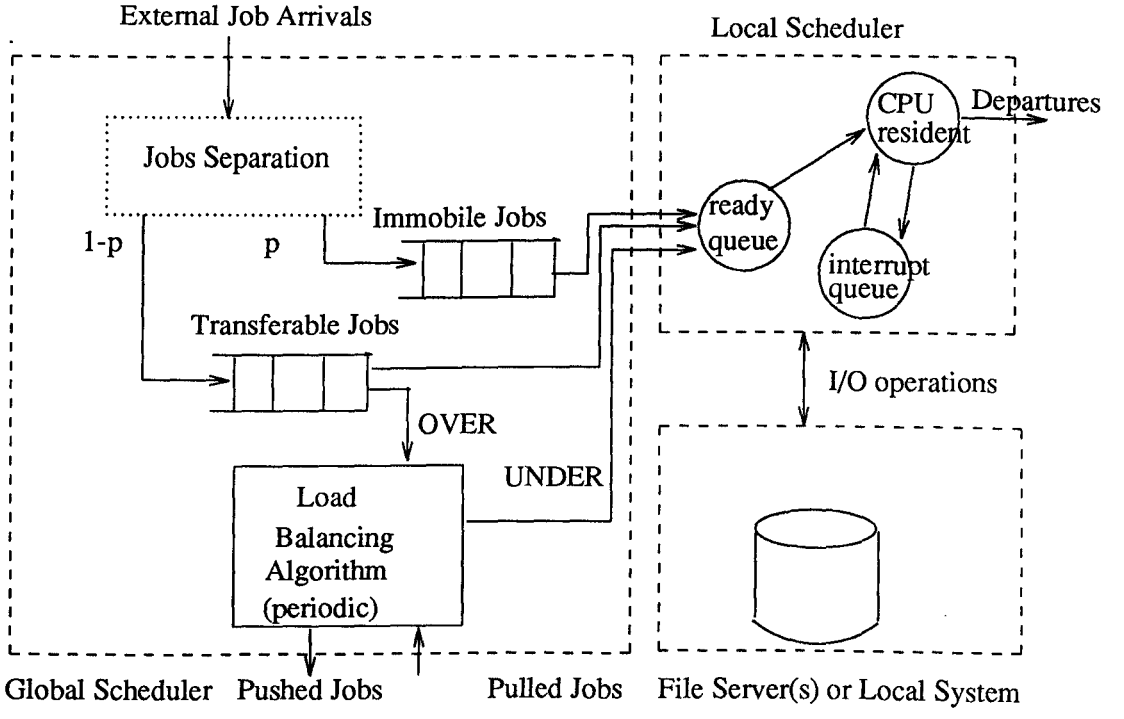


*Figure 4.8 Distributed Scheduler Structure- Diffuse Algorithm (selective transfers)*

miscellaneous results. These simulation stages are sketched in Figure 4.9.

An important aspect of the simulation model is the workload generation. Each autonomous host has an exponential statistical distribution source of user jobs which has the inter-arrival time E[T], and the start time of the arrival as variables. The service time of the jobs are taken from a hyper-exponential distribution source with $p$, E[$S_s$], E[$S_l$], as variables, for which the combined E[S] is fixed to one second for a one job/second host service rate. The service demands pattern are kept fixed except in the experiment on long jobs proportion where the probability ($p$) is changed. To get different levels of CPU utilisation, different levels of arrival rate are used.

In addition to the statistics automatically generated by Network II.5 which include node utilisation, node queues, transfer device utilisation, transfer device queues, module execution times, others statistics generation mechanisms have been added to account for system-wide performance considerations as well as at individual nodes such as job response time, jobs movement, bad decisions, load balancing costs, wait time in queues before access to the processor, and job throughput.

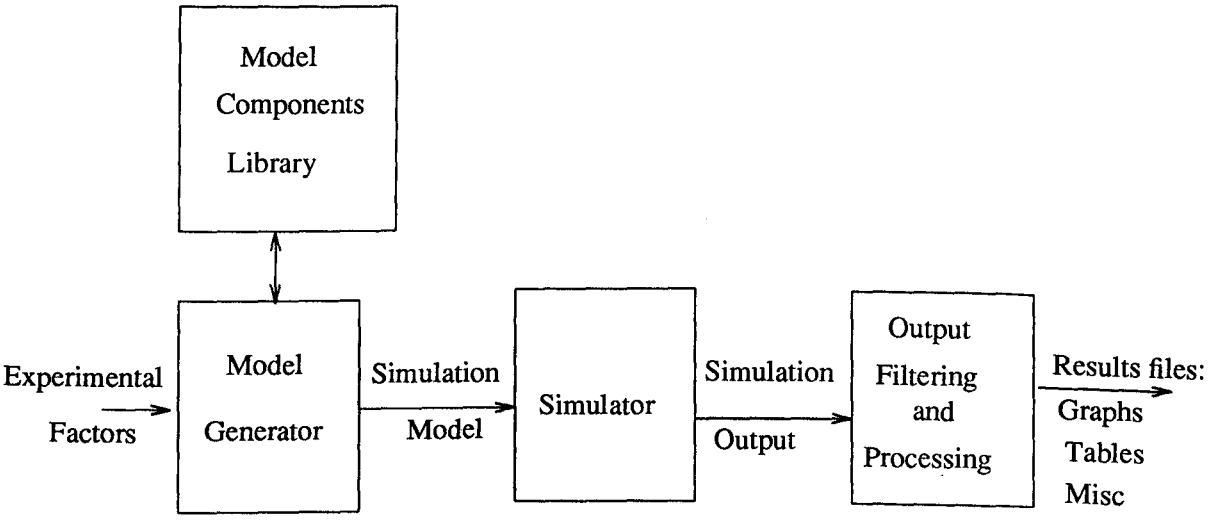


*Figure 4.9 Simulation Stages*

## 4.5. Simulated System Calibration and Validation

The first experimental phase is devoted to the validation of the simulated system. To this end a simulation model based on the work by Eager *et. al* [Eager86] and Mirchandaney *et. al* [Mirchandan89] was built. This corresponds to the following characteristics:

- network size: ten homogeneous nodes with one job/sec service rate
- disk-based file structure
- local discipline: FCFS
- homogeneous jobs with mean service time S= 1.0 sec
- average job transfer delay:
  0.1S for short communication delay
  2S for long communication delay
- transfer device access protocol: FCFS
- algorithms: Sender, Receiver, Symetric

On this baseline model the experiments on calibration of the simulation model, the reproduction of literature results for Sender, Receiver, and Symetric algorithms (see Figures 4.11 and 4.12), and the checking of the validity of results , were carried out. As a result of these tuning experiments the optimal parameters values identified are in accordance with the reported results. Based on the definition of the threshold parameters introduced in Section 2.5, the following convention on the threshold level is used: $T_{si} = T_{sa} = T_{ri} = T_{ra}$ = T+1, where T+1 represents the waiting jobs at a node plus the executing job. In Table 4.7 these default experimental values of the tunable parameters of the load balancing algorithms are given. The default values for other system components are depicted in the tables in Sections 4.3 and 4.4.

These results confirm that sender-initiated algorithms perform best for light to moderate load levels while receiver-initiated algorithms do better at moderate to heavy load levels. The symmetrically-initiated version has the best performance for the whole range of load levels. Under long transfer delays the performance of all three algorithms

| | |
|---|---|
| Threshold (T+1) | 1 job (for short communication delay) |
| | 3 jobs (for long communication delay) |
| Poll_limit ($L_p$) | 2 nodes |
| Fixed message overhead | 5.0 msecs |
| Jobs separation overhead | 0.0 msecs for homogeneous jobs |

*Table 4.7 Load Balancing Algorithms Default Parameters*

is nearly identical. The optimal threshold value varies with the level of job transfer delay or communication bandwidth.

For a simulation run length of 4000 seconds, the percentage of error on the job mean response time is less 3% for a load level $\rho \leq 0.8$, and less than 5% for a load level $\rho=0.9$. Further details on the simulation run length required for a steady state simulation output and the confidence levels for the numerical results obtained can be found in Section 5.5.2.

## 4.6. Summary

A simulated system was built to allow the evaluation of different load balancing algorithms taking into account the effect of various system attributes and workload models. Through a reproduction of the literature results, the validation of the simulation model was undertaken. In Chapter five, the simulation results obtained on this system for the experiments designed in Section 4.2, are presented and analyzed.

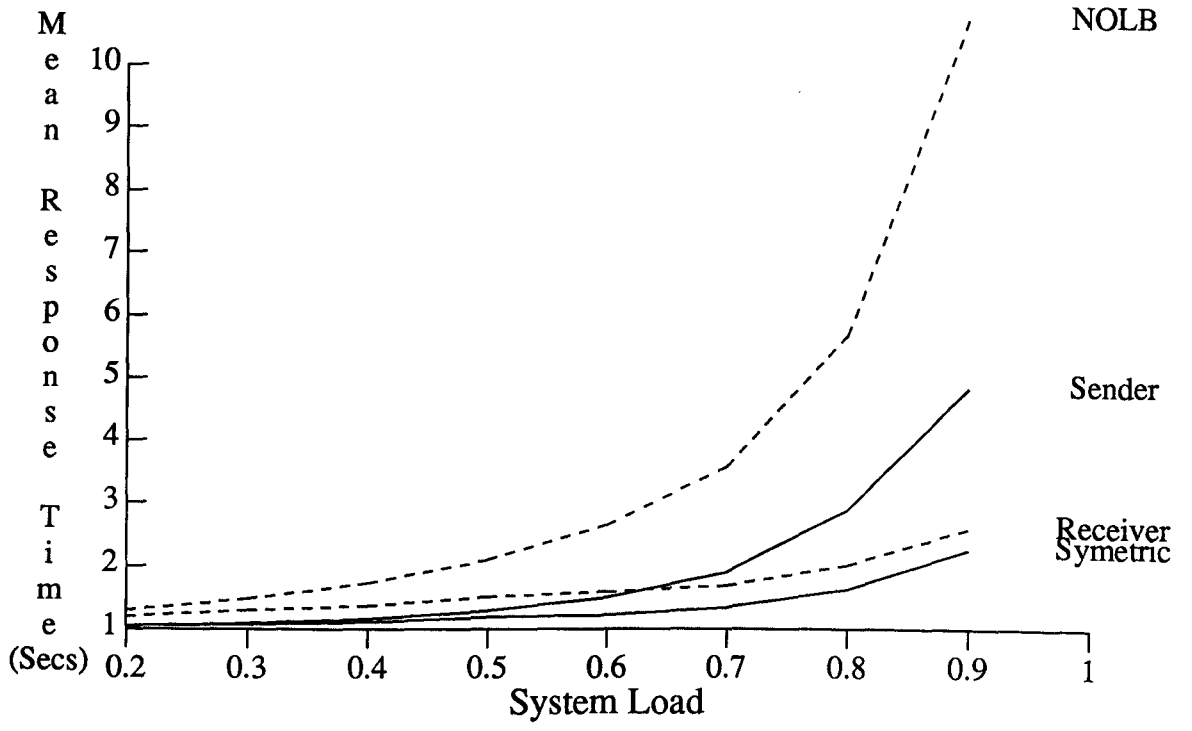


*Figure 4.11 Algorithms Performance under Short Communication Delays*

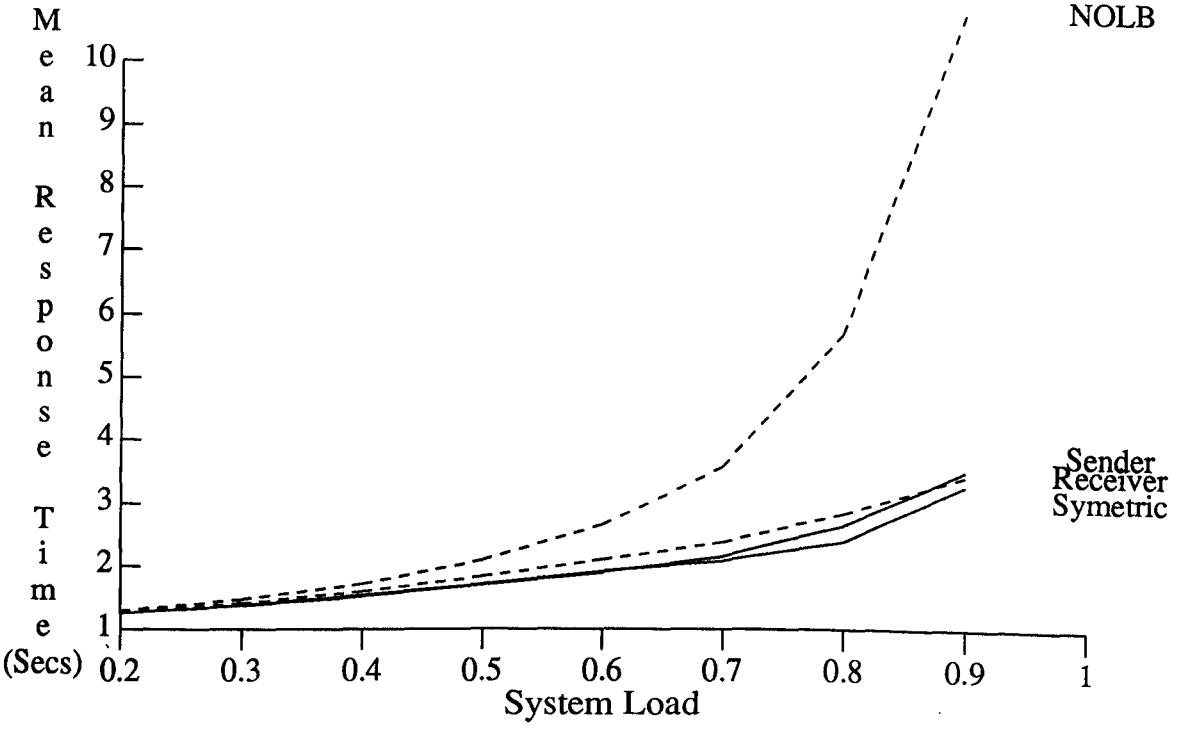


*Figure 4.12 Algorithms Performance under Long Communication Delays*

# CHAPTER 5

## Simulation Results

### 5.1. Overview

In this chapter the results of a performance study of several load balancing algorithms on four models of distributed systems are reported. As explained in Section 4.2, these models are broadly distinguished by the *file system structure* and the *homogeneity* of the processing speed of the nodes. For each model the performance of the load balancing algorithms is compared for a range of the distributed system attributes, including: the *communication bandwidth and protocols*, the *load balancing overheads*, the *file server speed* (for the diskless model). The *workload models* considered are homogeneous users with homogeneous jobs, heterogeneous users with homogeneous jobs, and homogeneous users with heterogeneous jobs. This work will contribute to answering the research questions posed in Section 3.4.

The purpose of load balancing is to reduce the job response time of a distributed system by increasing the utilisations of the processors. Care is needed to minimise the overheads of moving jobs around the system. The main measure of the performance of the load balancing algorithms is the metric: *job mean response time*. This measures the average time a job spends in the system. Also, to gain further insights into the performance of the load balancing algorithms, the following metrics are obtained: the *response time predictability* (standard deviation of job response time), the mean CPU queue length, the system *instability* (level of job movement), the *quality of remote allocation* (bad decisions rate), the level of negotiation *message traffic* per node per second on the network, the total load balancing overhead added to the computing nodes (i.e. increased *CPU utilisation*), and the final metric is the *communication device*

*utilisation* (e.g. percent of busy time, mean request delay).

A pictorial representation of the experimental investigations undertaken in this chapter is depicted in Figure 5.1. The details of the simulation and system validation have been described in Chapter 4. The experiments are grouped as follows:

i) Experiments on Homogeneous Diskless Systems

ii) Experiments on Homogeneous Disk-based Systems

iii) Experiments on Heterogeneous Systems

In Section 5.2, the performance of the load balancing algorithms is compared on a simulated system using homogeneous diskless nodes and a shared file structure. The system comprises a fixed set of system attributes and workload parameters. It is referred to as the baseline system. Following this, the performance characteristics of the more promising algorithms are investigated using different system attributes and workload parameters. The trade-offs involved between creating a stable, balanced system and the overheads incurred in bringing this about, are also discussed. With the exception of the file server related experiment, an almost identical set of investigations are carried out in Section 5.3, for a system comprising disk-based homogeneous nodes.

Load balancing in heterogeneous systems with various processors speed is investigated in Section 5.4. Strategies adapted for such configurations are developed and evaluated using arrival rates scaled to the nodes speed to have the same utilisation level on all the nodes, and identical job arrival rates on all the nodes regardless of their speed.

Finally the scalability and confidence levels issues are addressed in Section 5.5. The tables containing the detailed simulation results are presented in the appendix.

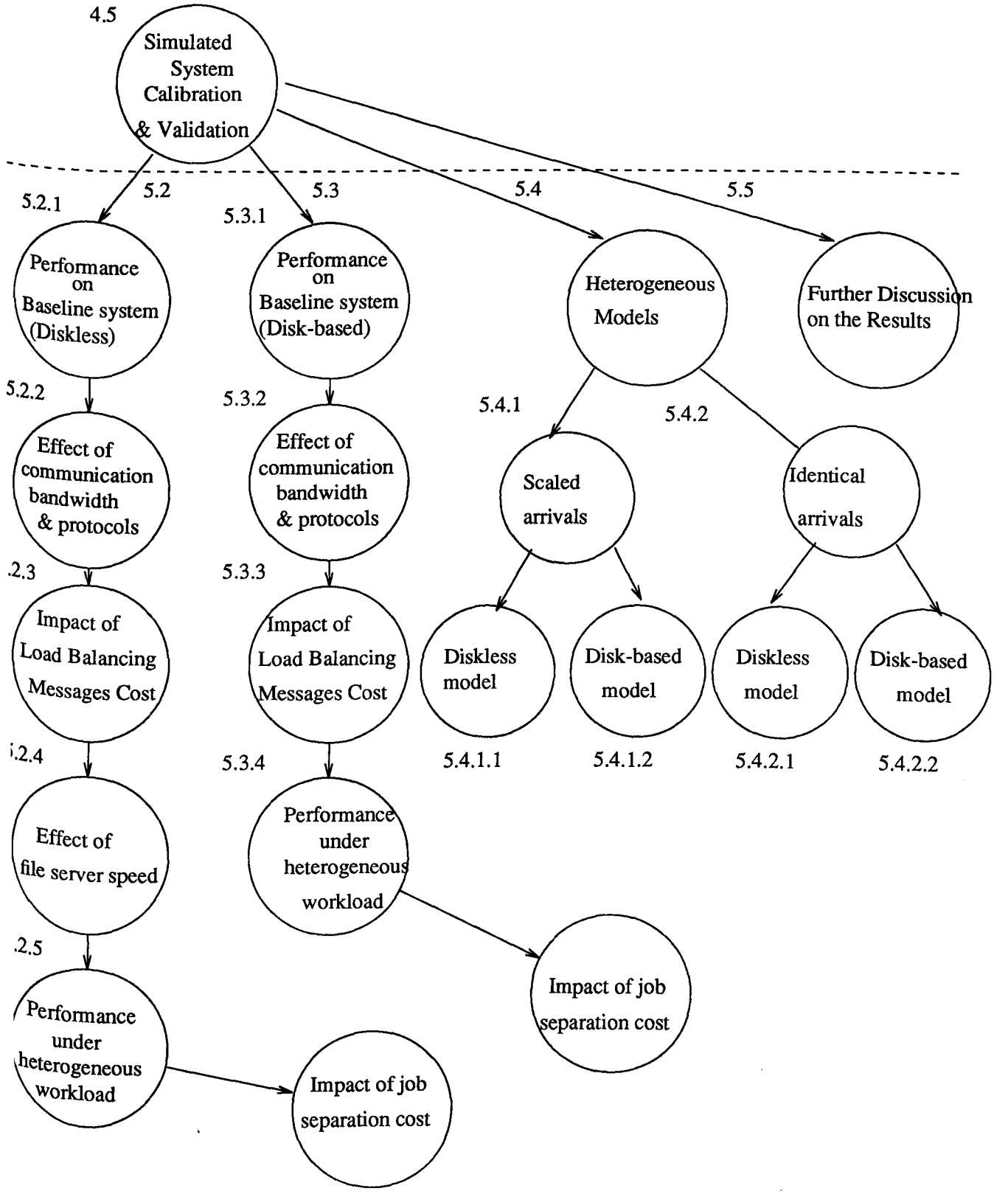


*Figure 5.1 Investigations Structure*

## 5.2. Load Balancing in Systems with Diskless Homogeneous Nodes

The objectives of this performance study are:

- To evaluate the load balancing algorithms on a baseline system using the job mean response time and other performance metrics. This also involves an assessment of the interdependence of the algorithms properties and the performance obtained.
- To determine the relative performance ordering for the algorithms and the sensitivity of this ordering to changes in the system attributes and workload parameters.
- To identify the system attributes which have a significant effect on the algorithms performance.
- To determine which algorithms perform well under a wide range of system behaviours.

The simulated system comprises identical diskless workstations connected through a broadcast communication device. A shared file server to support a distributed file system is connected to the same communication device and used to hold all the files and other information needed by the diskless nodes. Further details on the system design assumptions can be found in Section 4.3. The essential characteristics of the baseline system which are based on the assumptions commonly made in the literature, are summarised below. The communication bandwidth is expressed relative to the system job service rate. A ratio $R$= *compute rate /communicate rate* is used. The nodes service rate is kept fixed while the communication data transfer rate is varied.

The baseline system attributes are:

- File system structure: diskless
- System size: 10 homogeneous hosts

- System service rate: 10 jobs/sec (Host service rate: 1 job/sec)
- Compute/Communicate ratio: R= 0.13
- Communication protocol: token passing
- File server I/O overhead time: fixed 3.75 msecs + communication delay related to file size
- Workload model: homogeneous users, homogeneous jobs with E[S]= 1.0 secs
- Local scheduling discipline: FCFS
- Fixed load balancing message overhead: 5.0 msecs

The essential algorithm parameters to be tuned are the *probe limit (Lp)*, the *threshold (T)*, and the *timer period (Pt)*. Experimentally the following values have been found optimal for the baseline system: Lp = 2, T= 1 (for R= 0.13), Pt = 0.4 msecs. Based on this baseline model, the relative performance of Sender, Receiver, Symetric, Diffuse, and Random algorithms is computed. These algorithms use different information and control policies. The effect of the various system factors on the performance of these algorithms is assessed. The system factors considered are communication bandwidth and protocols; the load balancing messages cost; the file server speed; and the workload model.

### 5.2.1. Algorithms Performance on the Baseline System

The performance factors considered in this section are the load balancing algorithm, the load level, and the load pattern. The selected set of algorithms is evaluated for homogeneous users at different load levels, and for heterogeneous users combination (4S, 2M, 4V) (see details in Table 4.7).

The results for the baseline system are shown in Figure 5.2, and Table 5.1. It is to be noted that the tables are in the appendix. The following remarks can be made:

- While Sender performs better than Receiver at low to moderate load levels, it is outperformed by the latter at heavy load levels. As all the nodes become heavily loaded, it gets more difficult to find an idle or underloaded node through a sender-initiated load balancing.
- The Symetric algorithm, which is a combination of Sender and Receiver, does well over all the range of load levels. However it involves a higher number of load balancing messages and job movements. This tends to increase the percentage of CPU utilisation significantly.
- The Diffuse algorithm (i.e. periodic version of Symetric) produces the best mean job response time, though it involves a larger number of wrong job movements. This algorithm results in fewer number of messages and job movements than the Symetric one.
- The Random algorithm which has the lowest overhead, since no system information collection is needed, has the poorest performance due to large job movements and high wrong decision rate.
- A reduction of the job mean response time of up to 80% is possible for the baseline system. For a load level between 0.65 to 0.9, the performance ordering for the algorithms is: Diffuse, Symetric, Receiver, Sender, Random.

### 5.2.2. Effect of Communication Bandwidth and Protocols

In earlier studies on the performance of load balancing algorithms, it has been commonly assumed that a large communication bandwidth is available, so there is no contention on the communication device. However, this is not realistic since in present day technology the processor speed is increasing at a faster rate than the bandwidth of the communication network. Therefore it is worthwhile to investigate the performance of the load balancing algorithms under a large compute/communicate ratio. The experiments carried out on the baseline system, were repeated using a compute/communicate ratio R= 0.4. This makes it possible to compare the effect of the compute/communicate ratio on the algorithms performance.

#### 1) Performance under Large Compute/Communicate Ratio (R= 0.4)

A larger value of the threshold was found more appropriate when a large compute/communicate ratio is used (T= 2 for R= 0.4). The results of using slower communication device are shown in Figure 5.3, and Table 5.2. It is possible to draw the following conclusions.

- The relative performance order of the algorithms is unchanged. The only exception is the Symetric algorithm for which the job mean response time tends to saturate at very heavy load level. This can be explained by the large number of load balancing messages inherent to this algorithm.
- Due to the longer communication delay, the level of improvement of all the algorithms drops by up to 10%. Even under the NOLB case, the job mean response time degrades because it takes longer to access the file server.
- The performance of all the algorithms is nearly identical, except for the Diffuse algorithm which maintains a more significant improvement of the mean response time.

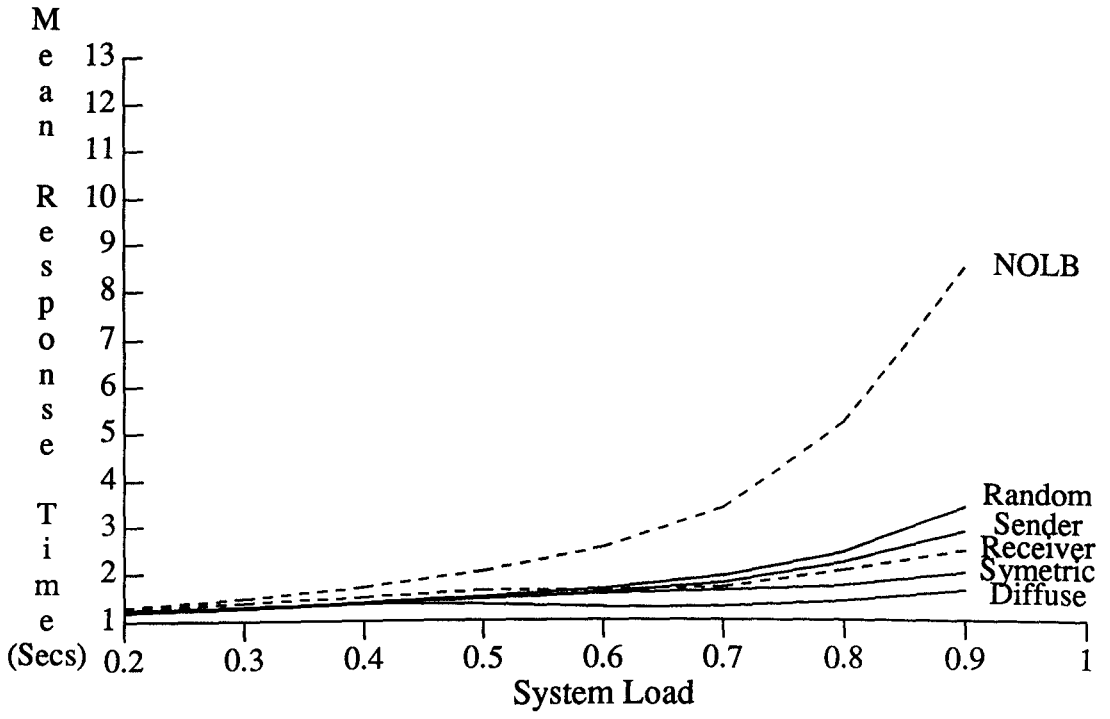


*Figure 5.2 Performance under Small Compute/Communicate Ratio (R= 0.13)*

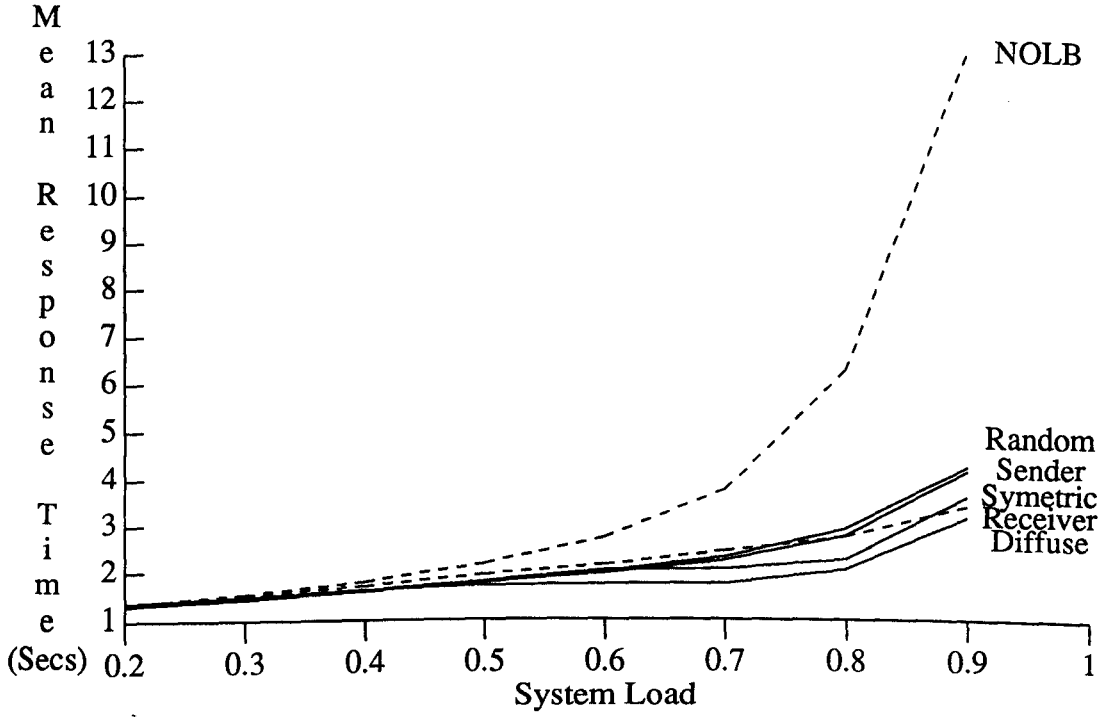


*Figure 5.3 Performance under Large Compute/Communicate Ratio (R= 0.4)*

**2) Performance under Heterogeneous Users**

In previous experiments, the generation of workload was based on identical users on all the nodes. The results shown in Tables 5.3, and 5.4, are for a heterogeneous combination of users (4S, 2M, 4V). The following remarks can be made:

- The Random algorithm does well in terms of reduction of job mean response time, while the Receiver performs poorly due to the low negotiation success rate. It is difficult to find an overloaded node when most nodes are barely used.
- The performance of Symetric, Random, and Sender are very similar, though at a lower cost for the latter. Only the Diffuse algorithm has a significant mean response time reduction.
- The compute/communicate ratio has no effect on the relative performance order of the algorithms. However for long communication delays, the level of improvement is smaller and at a higher cost.

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 5.47 | 0.00 | 2.46 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.63 | 70.18 | 0.56 | 77.43 | 29.77 | 28.40 | 0.00 | 0.31 |
| RECEIVR | 2.07 | 62.10 | 0.76 | 69.08 | 17.25 | 0.63 | 1.42 | 1.20 |
| SENDER | 1.55 | 71.67 | 0.52 | 79.05 | 24.14 | 0.49 | 0.39 | 0.54 |
| SYMTRIC | 1.54 | 71.90 | 0.51 | 79.38 | 25.59 | 0.55 | 2.50 | 1.50 |
| DIFFUSE | 1.30 | 76.16 | 0.54 | 77.86 | 23.83 | 7.25 | 2.22 | 2.22 |

Load Pattern= 4S, 2M, 4V

*Table 5.3 Performance under Small Compute/Communicate Ratio (R= 0.13)*

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 6.40 | 0.00 | 2.87 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 2.01 | 68.54 | 0.62 | 78.32 | 19.44 | 16.79 | 0.00 | 0.42 |
| RECEIVR | 2.48 | 61.32 | 0.81 | 71.69 | 14.01 | 0.72 | 1.60 | 2.14 |
| SENDER | 1.98 | 69.00 | 0.61 | 78.87 | 16.53 | 0.58 | 0.23 | 0.75 |
| SYMTRIC | 2.03 | 68.25 | 0.61 | 78.87 | 18.30 | 0.55 | 2.35 | 2.81 |
| DIFFUSE | 1.69 | 73.53 | 0.66 | 77.09 | 17.18 | 4.26 | 2.32 | 3.73 |

Load Pattern= 4S, 2M, 4V

*Table 5.4 Performance under Large Compute/Communicate Ratio (R= 0.4)*

### 3) Load Balancing Overheads

A load balancing scheme consumes CPU cycles for the execution of its policies and adds message traffic onto the communication device. Three types of overhead can be identified : the eligible job separation for selective transfers; the load balancing messages; and the job transfer overheads. Global load balancing overheads depend on the structure of the algorithm, the level of load balancing activity assumed, and the job arrival rates. To assess the average CPU utilisation due to the load balancing activities, the percentage of processor busy time of individual nodes is monitored for each algorithm and compared with the NOLB case. The percentage average increase is then computed. From the results shown in Figures 5.4 and 5.5, it can be concluded:

- The algorithms that perform best have higher overheads. The level of overhead increases with the load level. However, under long communication delays, the Symetric algorithm cost is more than that of Diffuse at very heavy load level, though its reduction of the mean response time is less. The load balancing overhead is nearly twice as big as for the baseline system.
- As the load balancing overhead is mainly due the handling of load balancing messages and job transfers, the increase in CPU utilisation is affected by the

number of load balancing messages induced by the load balancing algorithm and the level of job movement.

The increase in the percentage of communication device utilisation is less than 2% under both compute/communicate ratios, for all load balancing algorithms and at all load levels. From this we conclude that on a diskless model of distributed systems, the load balancing overhead is mainly on the CPU utilisation.

**4) Effect of Communication Protocols**

In the baseline system a token passing communication protocol was assumed. To assess the effect of the choice of the communication protocol, the performance of the two more promising algorithms (i.e. Diffuse and Symetric) is evaluated under First-Come First-Serve and CSMA/CD communication protocols. This evaluation was carried out under both large (R= 0.4) and small (R= 0.13) compute/communicate ratios. The results for the Symetric algorithm are shown in Figures (Figures 5.6, and 5.7). No significant effect on the mean response time was noticed. This can be explained by the low device utilisation level which was (< 20%) for slow device and (< 50%) for fast device, and the similar load put by all the nodes on communication device.

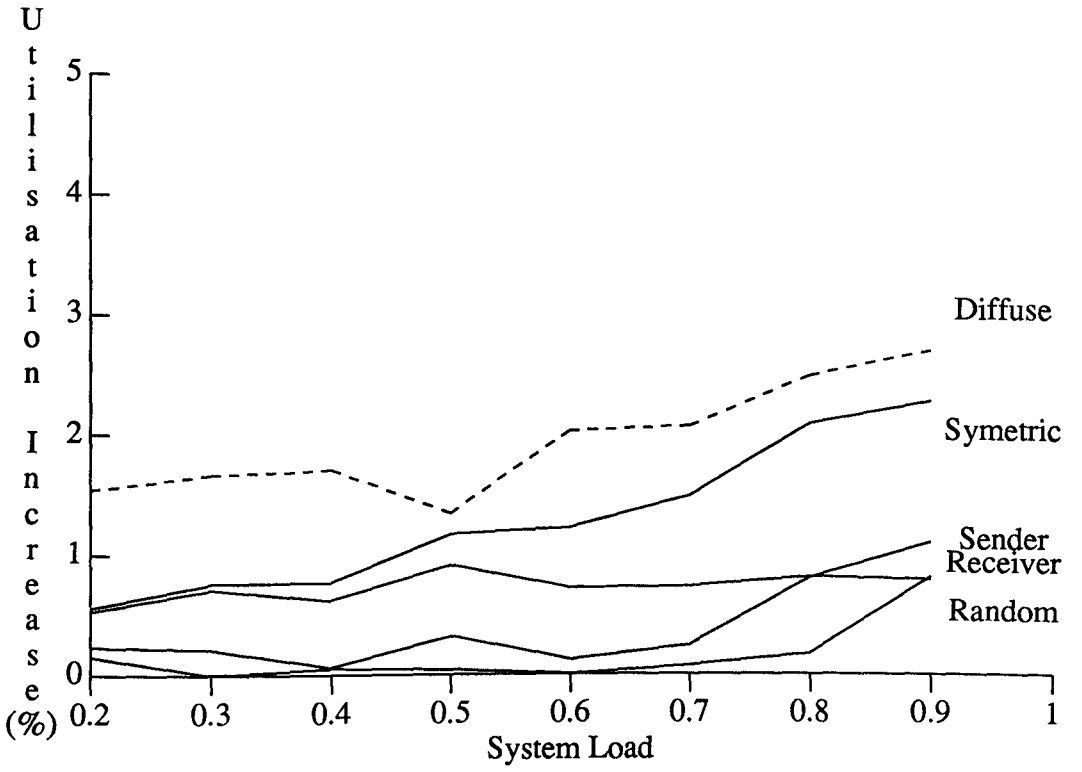


*Figure 5.4 Load Balancing Overhead for the Baseline System (R= 0.13)*

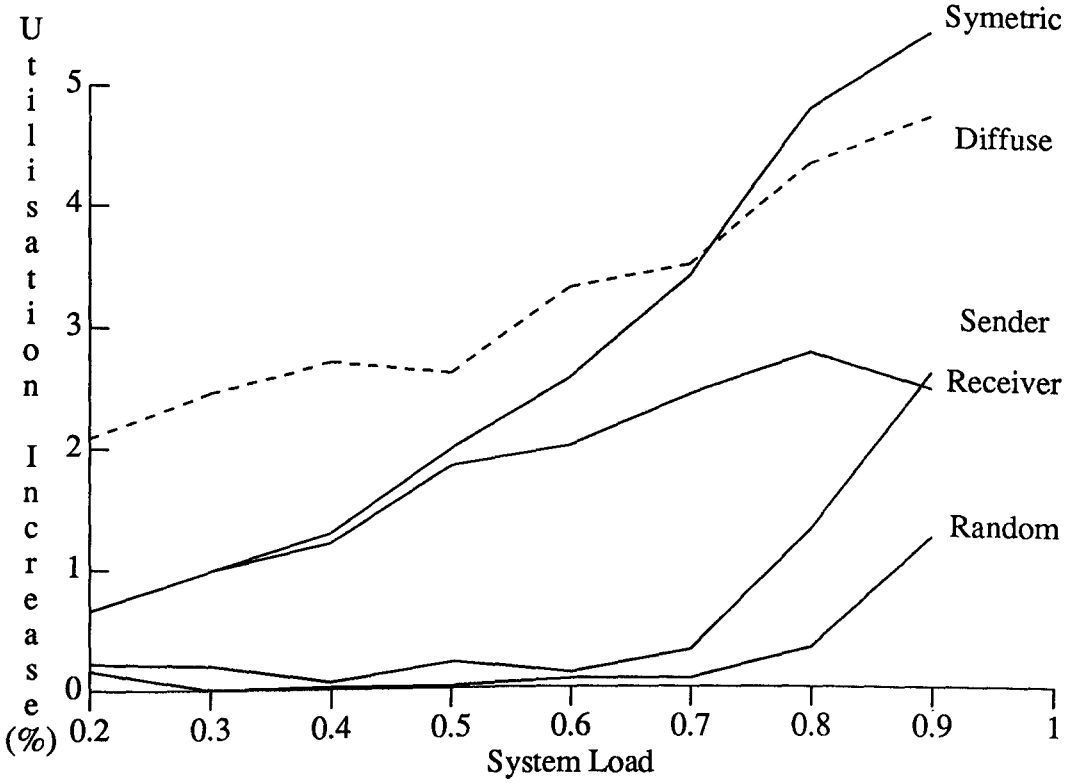


*Figure 5.5 Load Balancing Overhead for R= 0.4*

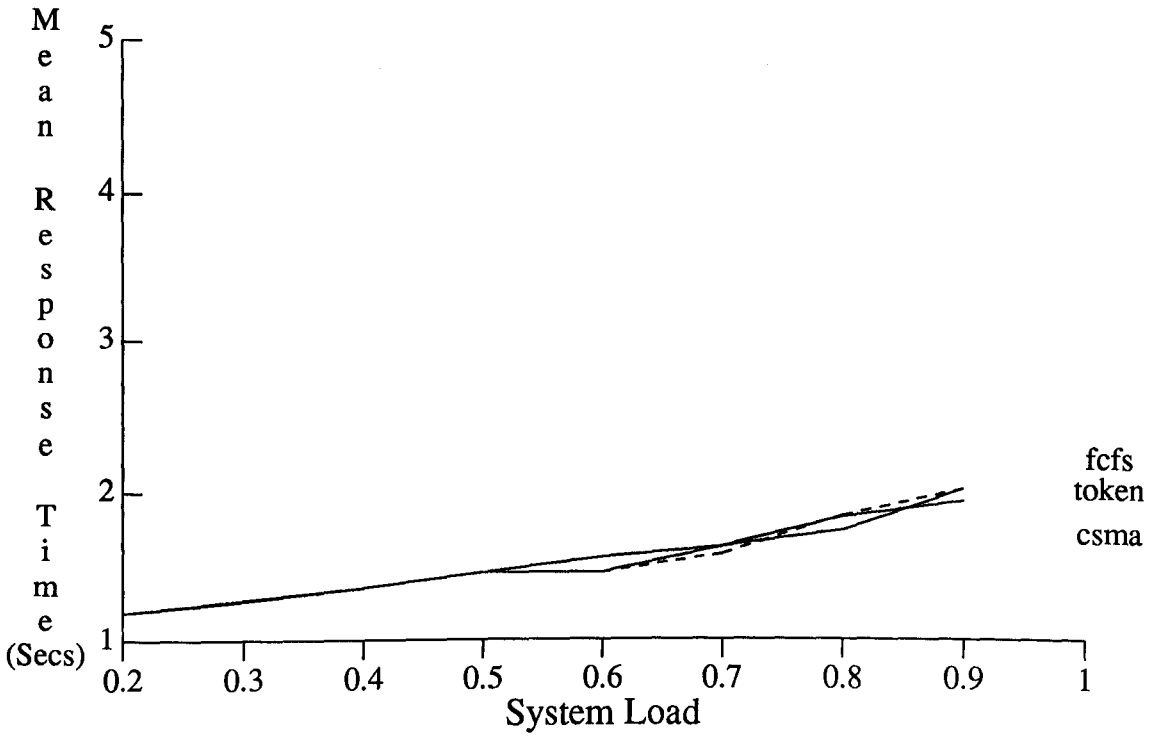


*Figure 5.6 Effect of Communication Protocols on Symetric Algorithm (R= 0.13)*

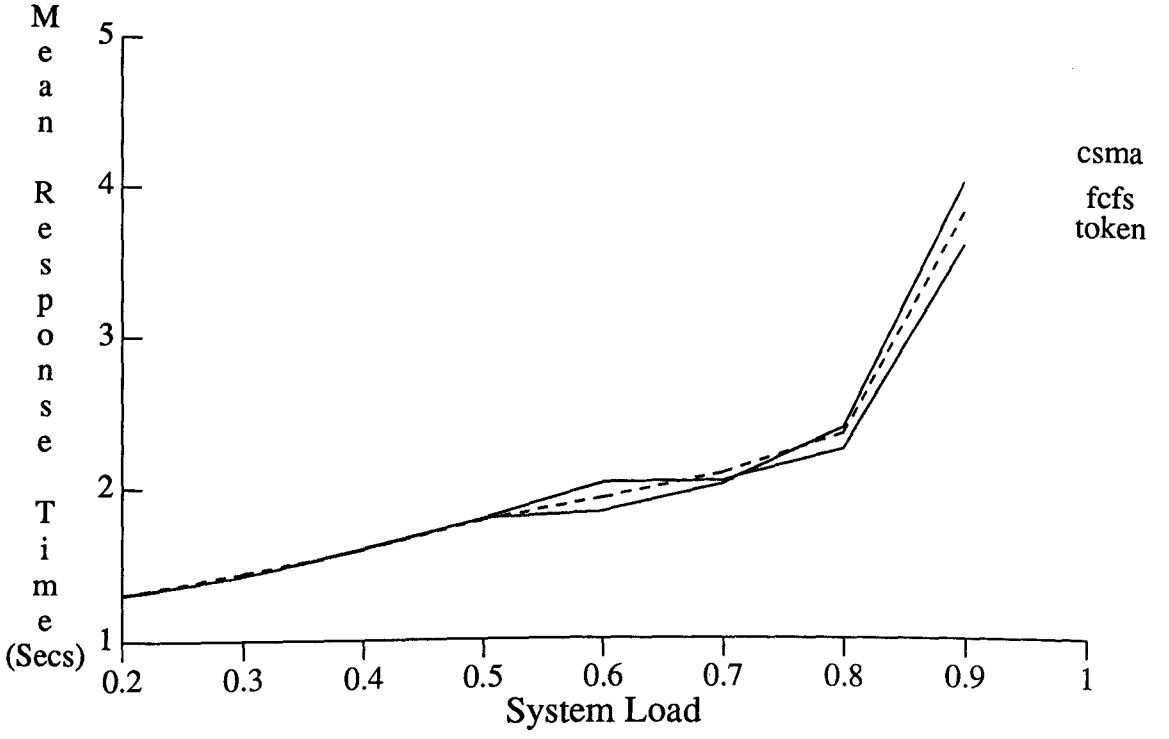


*Figure 5.7 Effect of Communication Protocols on Symetric Algorithm (R= 0.4)*

### 5.2.3. Impact of Load Balancing Messages Cost

To get a better understanding of how well the results obtained for the baseline system apply to environments with important message costs, two experiments were conducted.

The first evaluates the effect of a range of message cost (2 to 30 msecs) on the Symetric and Diffuse algorithms at a heavy load level and short communication delays. From this it is reasonable to conclude that the performance improvement is consistent over a wide range of load balancing messages cost (Figures 5.8, and 5.9), provided they are not too high, as shown in [Zhou88]. The advantage of the Diffuse algorithm over Symetric algorithm is clearly maintained.

The second experiment compares the performance of all algorithms for a message cost of 20 msecs. Diffuse and Receiver algorithms are less sensitive to message cost and maintain a good performance over the whole range of workload. Symetric and Sender algorithms degrade sharply at very heavy load level.

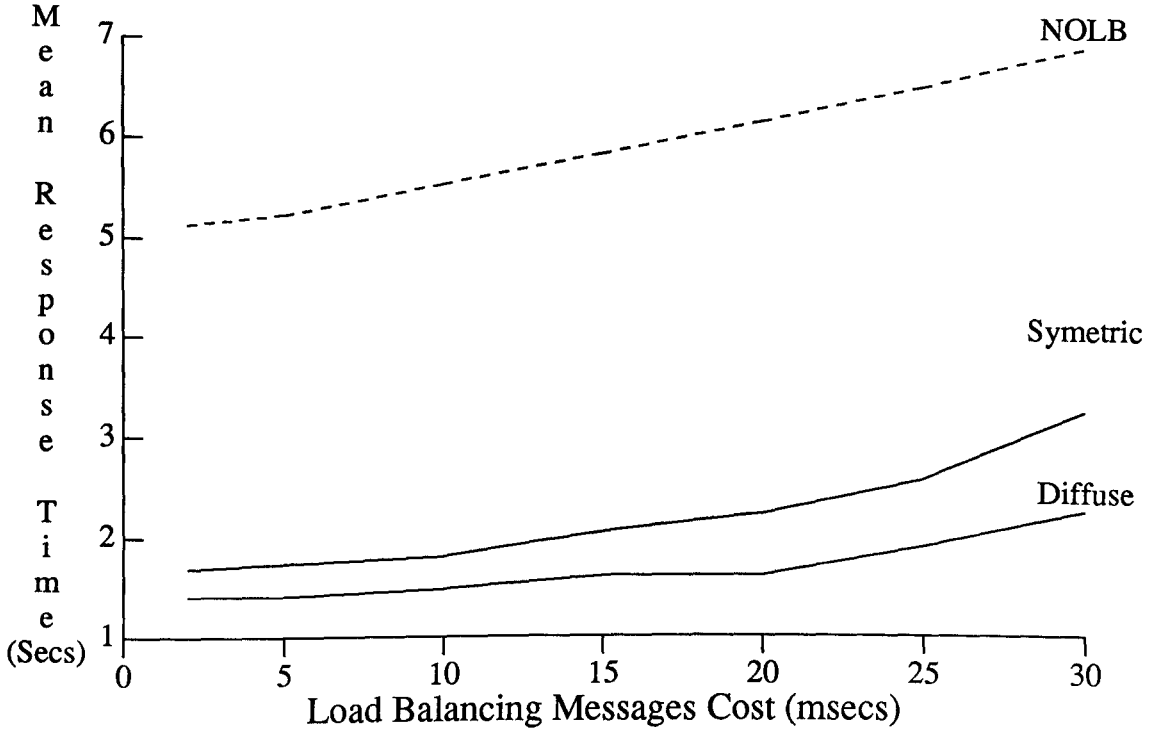


*Figure 5.8 Effect of Load Balancing Messages Cost for Baseline System (R= 0.13)*

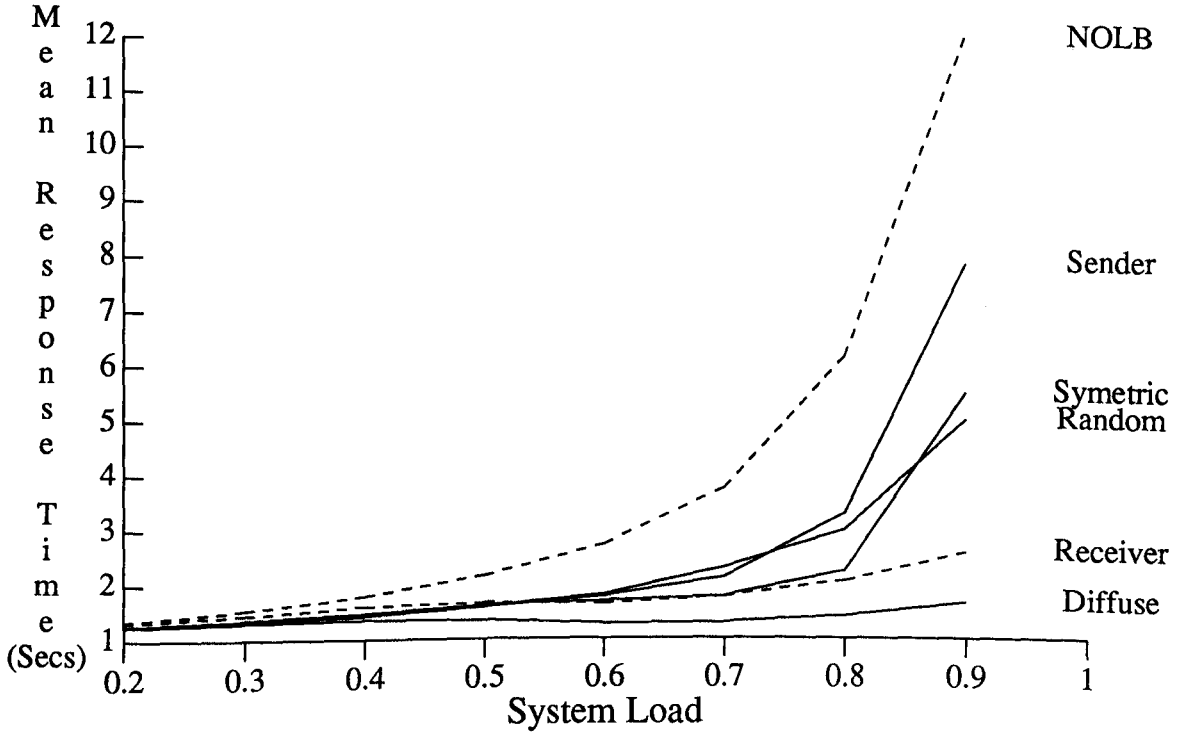


*Figure 5.9 Performance for Message Cost of 20 msecs (R= 0.13)*

### 5.2.4. Effect of File Server Speed

The effect of the file server speed on the mean response time is evaluated for the Diffuse and Symetric algorithms(see Figure 5.10). The file server speed attribute used is the fixed overhead time it takes to service an I/O operation. To remove any side effects due to the contention on the communication device, short communication delays are assumed. As would be expected, there is a range of server speeds which has little effect on the the job mean response time. However, when the file server takes over 30 msecs to service an I/O operation, a sharp increase of the mean response time occurs. The same behaviour can be observed when a large compute/communicate ratio is used. This supports the conclusion drawn by Zhou [Zhou87], that the file server is the first resource to saturate. The performance order of the algorithms is not affected.

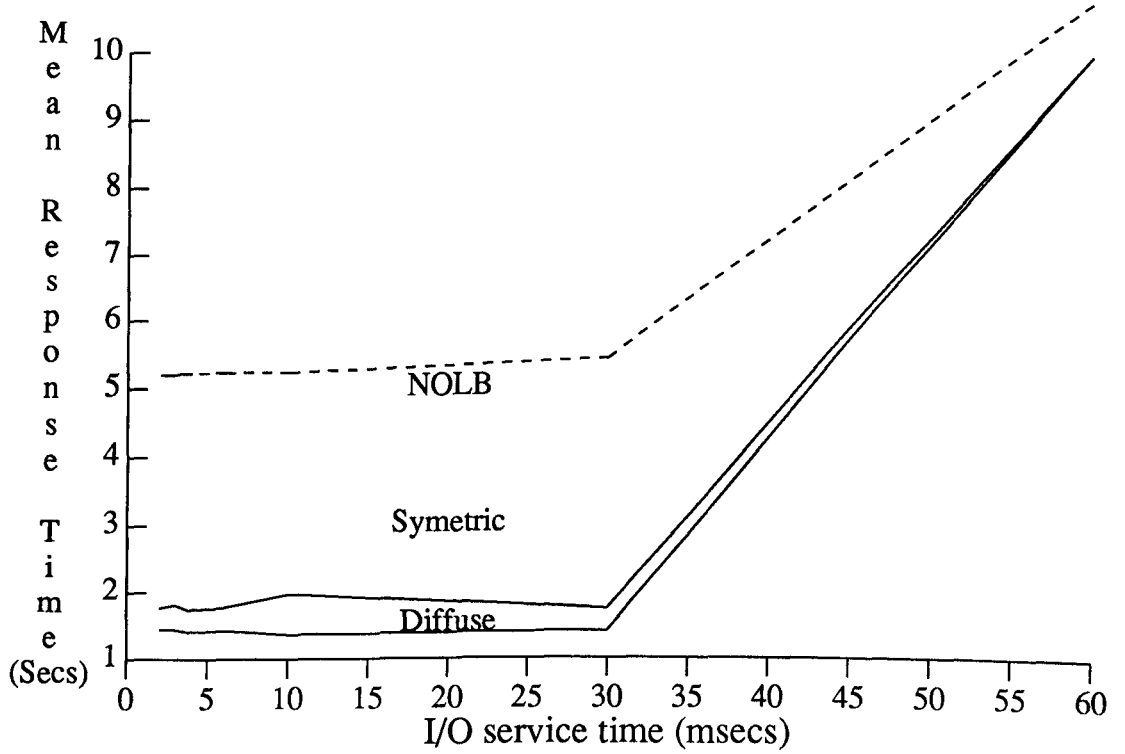


*Figure 5.10 Effect of File Server Speed*

### 5.2.5. Performance under Heterogeneous Workload

The Poisson arrival- exponential service demands workload is commonly assumed as the default model. In this experiment a workload model with hyper-exponential service demands is investigated. Furthermore two types of job transfer for load balancing purposes are considered: (i) non-selective transfers where all jobs are eligible for transfer whether it is a long or a short job, and (ii) a selective transfer which is restricted to long jobs. To accommodate heterogeneous jobs the following changes have been made to the system model. The FCFS local scheduling discipline is replaced by a Round Robin local scheduling discipline because the FCFS is not suitable for heterogeneous jobs [Mitrani87]. A parameter to assess the job separation cost is introduced. Its default value is fixed to 10 msecs. The tuning of the timer period for Diffuse algorithm had also to be repeated. A new value of 1.6 msecs was found optimal. The effect of three issues is investigated:

i) Proportion of short/long jobs: 95/05, 70/30
ii) Type of job transfer
    -non-selective transfers of jobs
    -selective transfers of jobs
iii) job separation cost: 10 to 200 msecs

The results for a 95/05 proportion of short/long jobs with non-selective transfers are shown in Figure 5.12, and Table 5.5. It can be concluded that the Receiver algorithm performance degrades, even at very heavy load level it does not catch up with the Sender algorithm. Due to the long running jobs the state of near idleness takes place less often. This causes the Receiver algorithm to be activated less often. In fact the latter algorithm has the lowest job movement level (i.e. < 50%) than the other algorithms. The other algorithms performance ordering is similar to the ordering under homogeneous jobs.

The results for selective transfers are shown in Figure 5.13, and Table 5.6. It can be seen that the level of improvement is reduced significantly (i.e. over 20%) because at most 5% of the jobs can be transferred, while the relative performance order of the algorithms is similar to previous experiment.

When the proportion of long jobs is changed to 70/30 (see Figure 5.14, and Table 5.7), the following conclusions can be drawn :

- The performance order remains the same as under 95/05 proportions. A similar level of performance improvement is obtained for non-selective transfers.
- For selective transfers (Figure 5.15, and Table 5.8), the level mean response time improvement is higher because a larger number of jobs are eligible for transfer. Due to a significant number of wrong movement of jobs at very heavy load level, the performance of Diffuse degrades. This can be overcome by slowing down the algorithm at very heavy load levels, a situation which should be rare, otherwise a significant upgrade of the system is necessary.

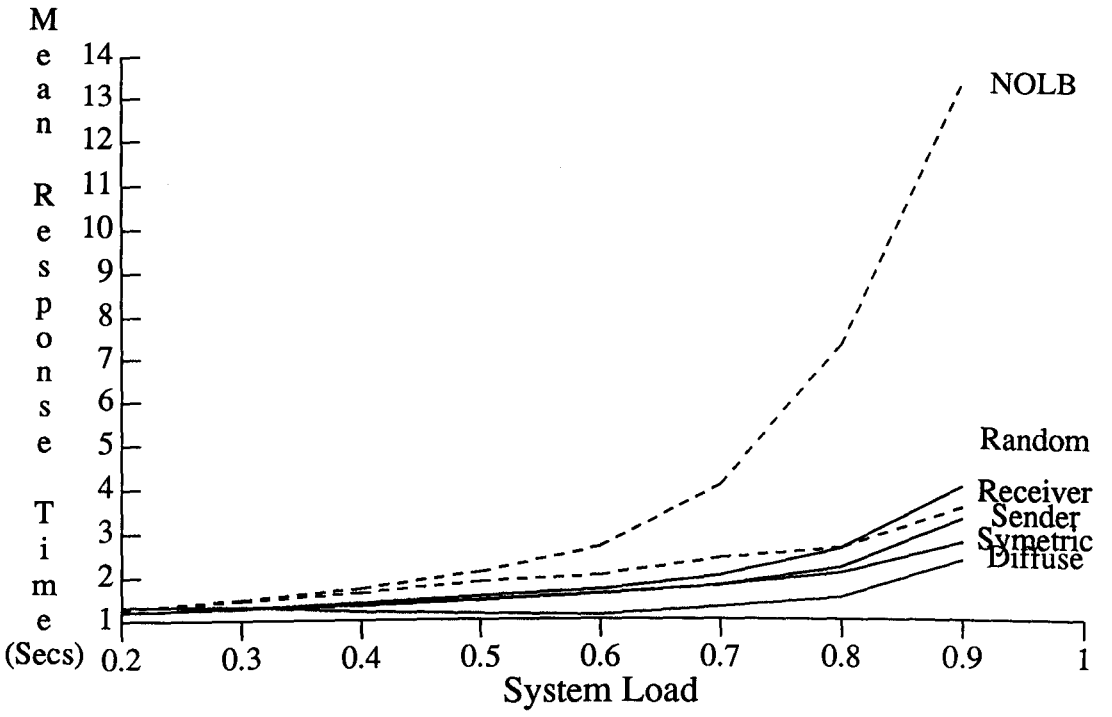


*Figure 5.12 95/05 Jobs Proportion with Non-selective Transfer*

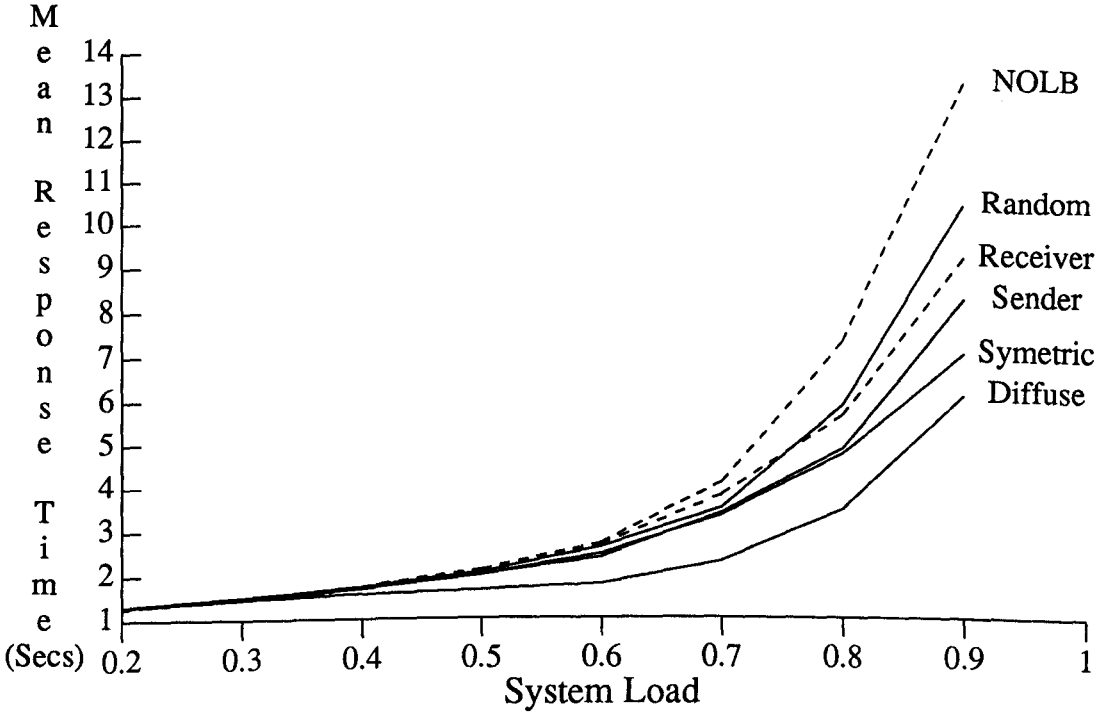


*Figure 5.13 95/05 Jobs Proportion with Selective Transfer*

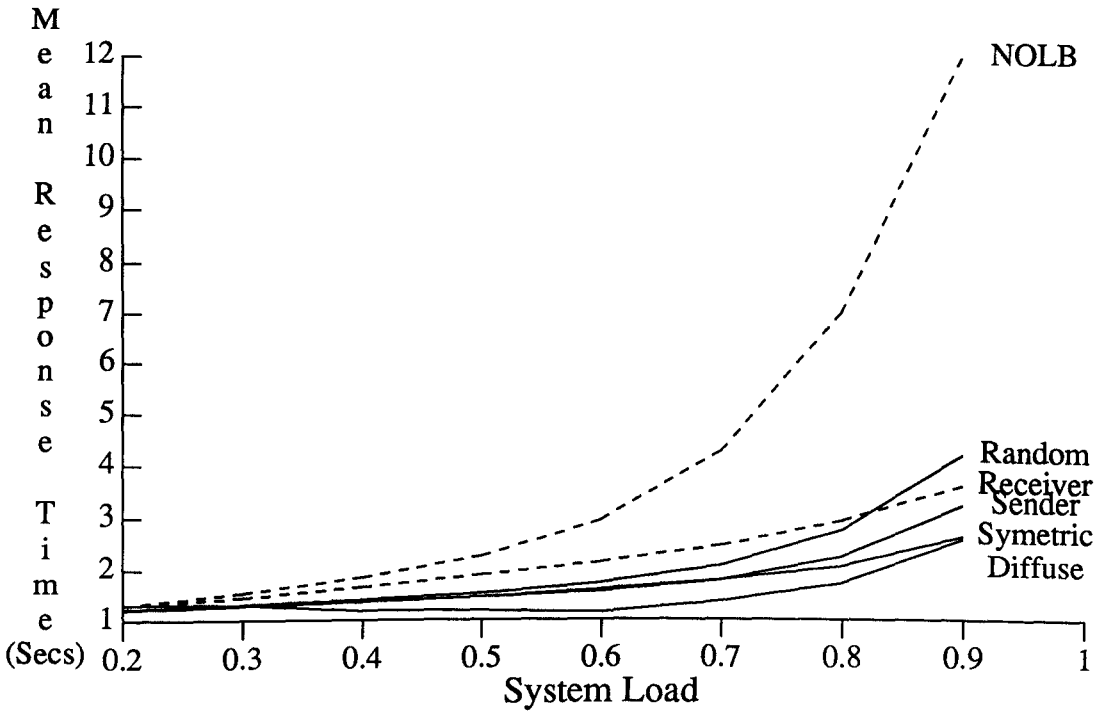


*Figure 5.14 70/30 Jobs Proportion with Non-selective Transfer*

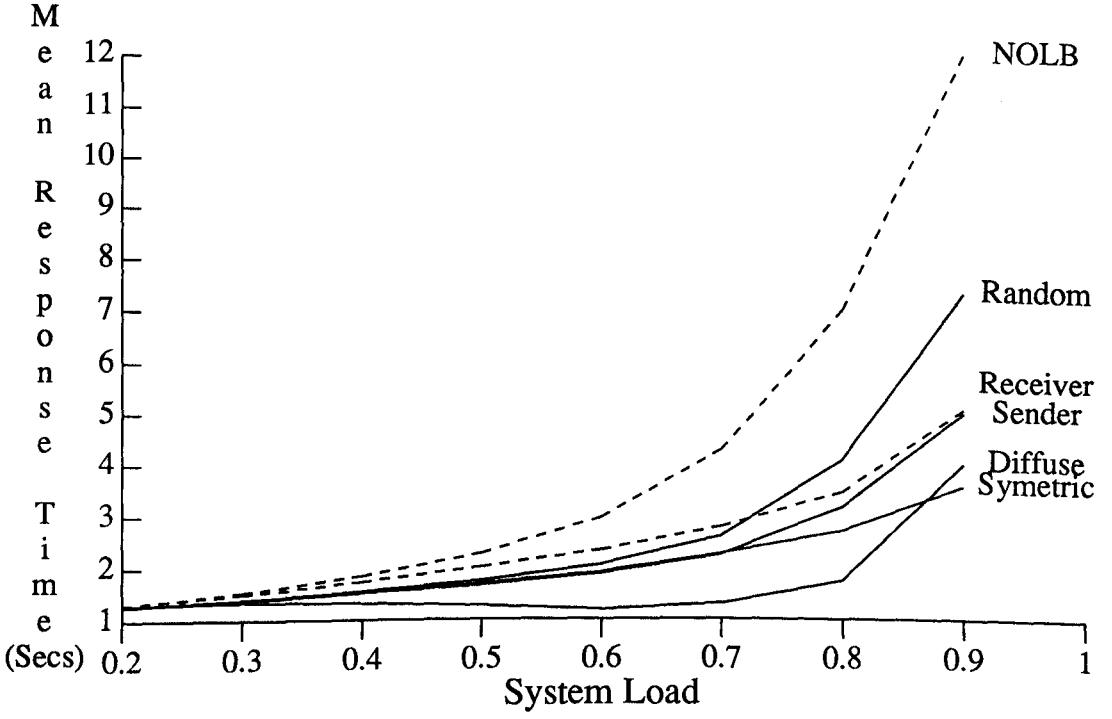


*Figure 5.15 70/30 Jobs Proportion with Non-selective Transfer*

For a job separation cost of up to 200 msecs (see Figure 5.16), selective transfer based load balancing improve the job mean response time. However, for more than 100 msecs the level of improvement is not worthwhile. The degradation of the job mean response time is sharper for the Diffuse algorithm.

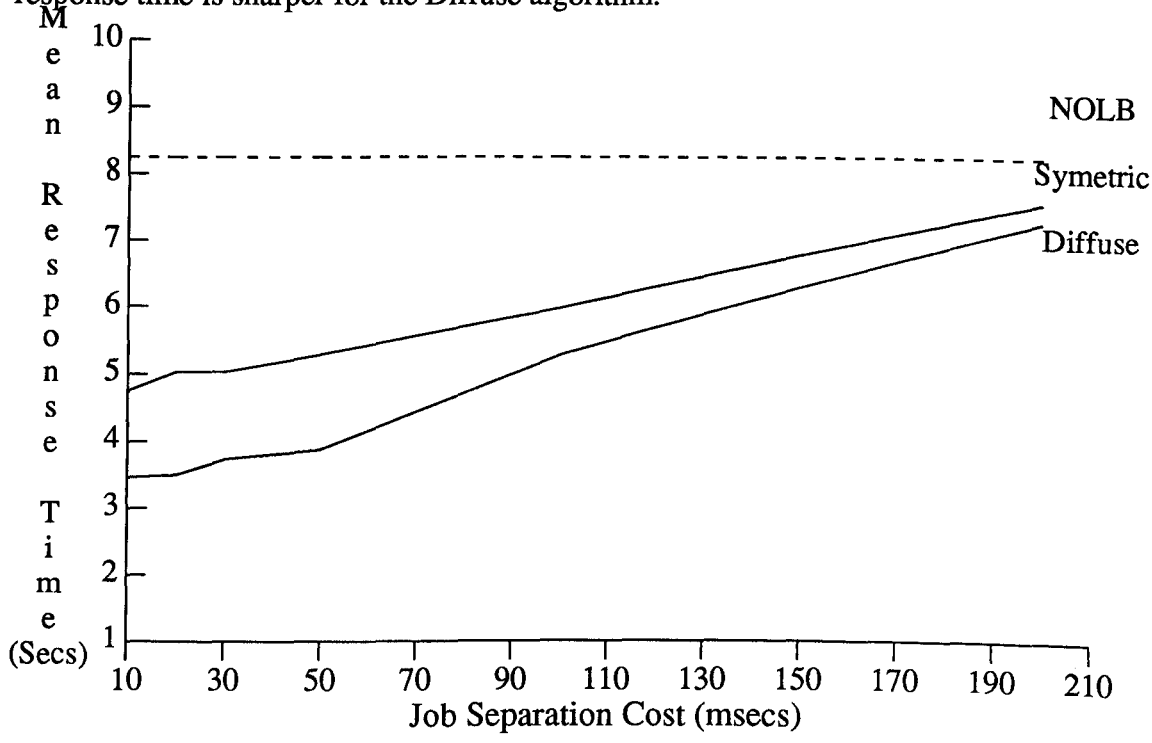


*Figure 5.16 Effect of Job Separation Cost under 95/05 Proportion*

### 5.2.6. Conclusion

The main conclusions to be drawn on relative performance of different load balancing algorithms evaluated on the diskless system model are as follows:

- All the algorithms evaluated improve the job mean response time of the system particularly for a load level greater than 0.4. The amount of improvement increases with the load level. Improvements of up to 80% were found at very heavy system loads.
- The Diffuse algorithm produces the lowest mean response time. This is due to its symmetric and periodic structure. Although it has a high level of wrong job movements.

- Random algorithm has the lowest communication overhead since no system state information is gathered but it produces a large number of bad decisions and the highest system instability, leading to a poor performance. These results confirm the findings of Eager *et al.* [Eager85]. That is sender-initiated policies are good at light load while receiver-initiated policies perform better only at heavy load. The symmetrically-initiated version performs well over the whole range of load levels.

Below are summarised the effects of the distributed system attributes and workload. The communication device speed has an impact on the performance of the load balancing algorithms. When the device is slow the mean system response time degrades even for the NOLB case. Another significant effect is a reduced level of job movement and a slight increase of wrong decisions rate. The performance of all the algorithms is nearly identical making the choice of the algorithm less relevant. For heterogeneous users an increased predictability of the mean response time is observed. For a communication device utilisation level of up to 50%, the load balancing algorithms perform similarly with all the three *communication protocols* evaluated.

The algorithms performance is robust over a wide range of load balancing cost. Symetric algorithm is the most sensitive to this cost at very high load levels. This suggests that this algorithm should be used when there is a large communication bandwidth is available. As far as the global load balancing cost (i.e. message cost, job transfer cost) is concerned a general pattern emerges. The algorithms with the best improvement tend to have the worst overheads associated with them. This cost is increased when the communication delay and load level are higher. However, a general conclusion is that the load balancing is still effective for a wide range of *load balancing overheads*. Provided a minimum file server speed is available, load balancing performance is not affected.

When heterogeneous workloads are used, the relative performance order of the algorithms is unchanged. Selective transfers are worthwhile only when a large number of long jobs are generated and the job separation cost is not too high.

The results show clearly that the Diffuse algorithm is the most promising one in reducing the job mean response time. It is robust in the sense that it performs well over a wider range of system attributes and workload. However, some care is needed in interpreting the results because the algorithm parameters have been tuned for optimal performance on the systems considered.

## 5.3. Load Balancing in Systems with Disk-based Homogeneous Nodes

The distributed system considered in this section consists of a set of identical autonomous nodes. Each node has its own local file system and is connected to a broadcast communication device. In this environment load balancing involves the actual transfer of the complete job information (i.e. programs, files) to the remote host, and the return of the results data and files to the job originating host. Except for the file server related experiment, which does not apply to the disk-based system, all the experiments of the previous section were repeated on this disk-based system model. A structure similar to that of Section 5.2 has been adopted for this section.

### 5.3.1. Algorithms Performance on the Baseline System

Except for the file system structure which is changed to a disk-based model, all the characteristics of the system under study are the same as those used in the baseline system described in Section 5.2. Since no file server is used, all I/O operations are handled by a local disk. The time to service an I/O operation is assumed evenly distributed and fixed to 20 msecs.

The results obtained under the baseline system conditions are shown in Figure 5.17, and

Table 5.9. The following conclusions can be drawn:

- The algorithms performance order for moderate to heavy load level is: Diffuse, Symetric, Receiver, Sender, Random. The cross-over of Sender/Receiver algorithm takes place at 0.75 load level. For light to moderate load level, all the algorithms perform similarly with a slight degradation for Receiver. For such load levels the probability of finding an overloaded node is small.
- The level of performance improvement of up to 80% is possible. The standard deviation of the response time obtained is similar for all the algorithms.
- The poor performance of Random algorithm is due to its high level of job movement (nearly twice that of other algorithms at heavy load level) and wrong job movements.
- The most promising algorithms are Diffuse and Symetric, which are both symmetrically-initiated. Diffuse algorithm produces the best mean response time, but involves a higher level of wrong job movement.

It is interesting to observe that there is no significant difference between the performance ordering of the algorithms for the diskless and disk-based models of the baseline system.

### 5.3.2. Effect of Communication Bandwidth and Protocols

The performance of the load balancing algorithms is compared under a large compute/communicate ratio, heterogeneous users, and different communication protocols. The level of load balancing overhead is also assessed.

**1) Performance under Large Compute/Communicate Ratio (R= 0.4)**

The results obtained when a large Compute/Communicate ratio is used are shown in Figure 5.18, and Table 5.10. The following remarks can be made on the performance

of the load balancing algorithms:

- The relative performance ordering of the algorithms remains unchanged. Surprisingly the level of performance improvement is significant (i.e. up to 75%) even for long communication delays, though the algorithms curves tend to cluster making the choice of the load balancing algorithm less relevant.
- The Receiver/Sender cross-over takes place at a much higher load level. This indicate that the probability of finding an overloaded node is reduced (i.e. the threshold level was raised to 2 to make the probing of remote nodes cost-effective under long communication delays).
- The only significant performance difference observed between diskless and disk-based models, is the poorer performance of Random at heavy load levels on disk-based model. This can be explained by the nature of transfers on the disk-based model where the actual job is moved and the very level of job movement inherent to the Random algorithm.

The main conclusions to be drawn from this experiment is that the compute/communicate ratio does not affect the relative performance order of the algorithms. For a large ratio a lower level of improvement is obtained and performance of the algorithms is almost identical. However, the Diffuse algorithm still produces the highest reduction of the job mean response time.

**2) Performance under Heterogeneous Users**

In previous experiments, the generation of workload was based on identical users on all the nodes. The results shown in Tables 5.11 and 5.12 are obtained for a heterogeneous combination of users (4S, 2M, 4V). The following remarks can be made. Diffuse, Symetric, and Random algorithms perform similarly for both short and long communication delays. This unexpected performance of Random can be

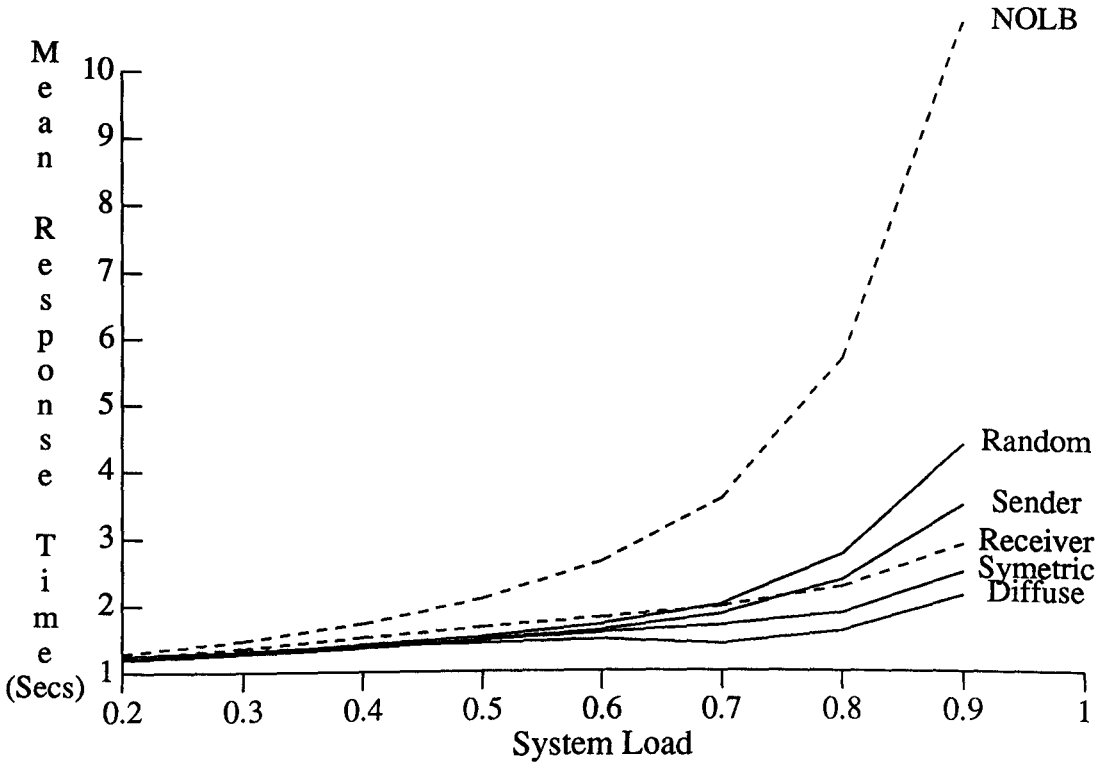


*Figure 5.17 Performance under Small Compute/Communicate Ratio (R= 0.13)*

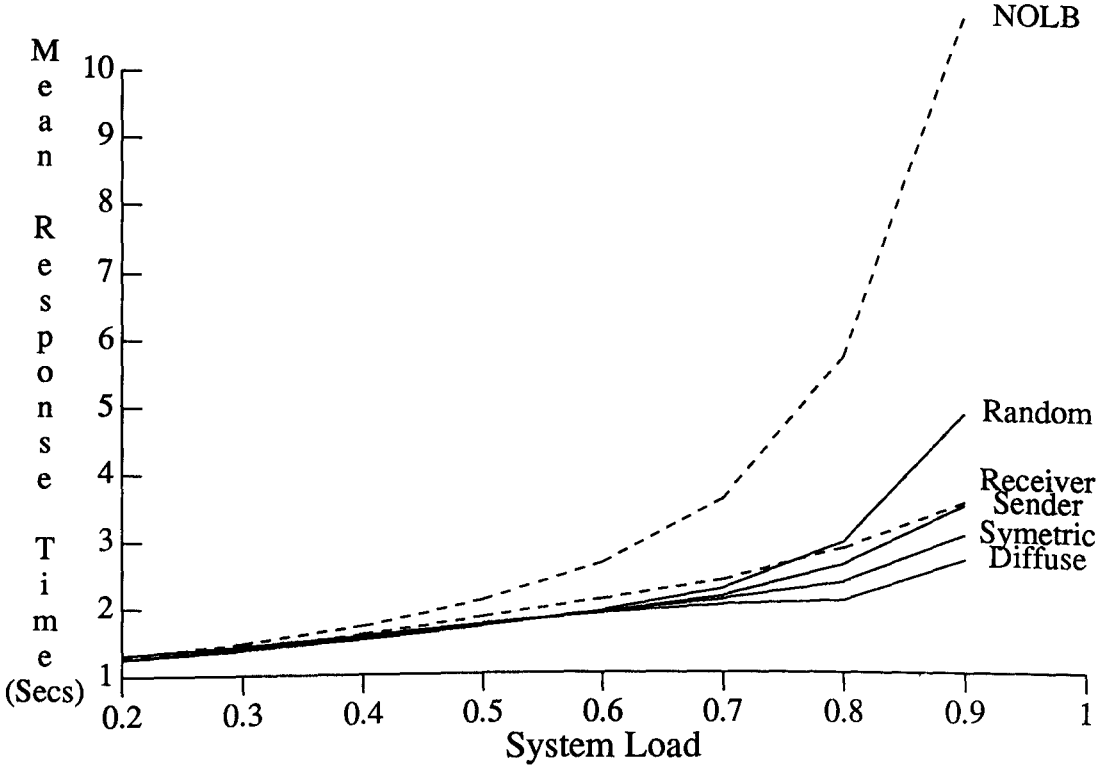


*Figure 5.18 Performance under Large Compute/Communicate Ratio (R= 0.4)*

explained by the large number of lightly loaded nodes. For the same reason Receiver algorithm does rather poorly (i.e. reduced probability of finding an overloaded node). No significant difference in the performance ordering of the algorithms with the diskless model is observed.

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 6.10 | 0.00 | 2.66 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.69 | 72.33 | 0.61 | 77.07 | 32.49 | 29.56 | 0.00 | 0.45 |
| RECEIVR | 2.11 | 65.39 | 0.78 | 70.77 | 18.38 | 1.75 | 1.40 | 1.28 |
| SENDER | 1.59 | 73.93 | 0.52 | 80.33 | 25.24 | 1.72 | 0.41 | 0.84 |
| SYMTRIC | 1.52 | 75.15 | 0.51 | 80.66 | 27.45 | 2.08 | 2.51 | 1.66 |
| DIFFUSE | 1.48 | 75.82 | 0.53 | 79.93 | 24.51 | 8.77 | 2.22 | 1.87 |

Load Pattern= 4S, 2M, 4V

*Table 5.11 Performance under Small Compute/Communicate Ratio (R= 0.13)*

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 6.10 | 0.00 | 2.66 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 2.01 | 67.03 | 0.66 | 75.32 | 20.20 | 17.53 | 0.00 | 0.64 |
| RECEIVR | 2.45 | 59.87 | 0.82 | 69.32 | 14.62 | 1.55 | 1.64 | 1.62 |
| SENDER | 1.95 | 68.11 | 0.62 | 76.71 | 17.68 | 1.26 | 0.24 | 0.97 |
| SYMTRIC | 1.95 | 68.00 | 0.62 | 76.82 | 18.43 | 1.45 | 2.39 | 2.00 |
| DIFFUSE | 2.00 | 67.28 | 0.64 | 75.96 | 17.16 | 4.16 | 2.35 | 2.15 |

Load Pattern= 4S, 2M, 4V

*Table 5.12 Performance under Large Compute/Communicate Ratio (R= 0.4)*

### 3) Load Balancing Overheads

The results shown in Figures 5.19, and 5.20, represent the load balancing overhead on a disk-based model. It can be concluded that:

- The load balancing overheads increase with the increase of the performance gain. However, under long communication delays, the overheads for the Random algorithm increases sharply due the large number of jobs transferred (up to 22% utilisation of a slow communication device).
- The overhead level is slightly affected by the communication delay for low to moderate load levels, because the utilisation of the communication device is relatively low (< 14% for large ratio and < 5% for small ratio). This is the main difference with the diskless model where the communication device utilisation is higher because it is used for both load balancing activities and shared file server accesses.
- The communication device utilisation for the disk-based model, being mainly due to the load balancing activities, the increase in CPU utilisation gives an indication on the load balancing overhead put on the communication device.

**4) Effect of Communication Protocols**

The increase in the traffic imposed on the communication device by the load balancing algorithm also depends on the file system structure. On a disk-based system a transfer of a job requires the transfer of the full program and associated files as well as the return of results. On this basis one would expect a more important overhead imposed on the communication device for the disk-based model. This is not the case, it is on the diskless model that, the accesses to the files on the shared device puts a much bigger burden on the communication device.

The effect of the communication protocol was evaluated for both Diffuse and Symetric algorithms. The results for the Symetric algorithm are shown in Figure 5.21. All the communication protocols perform similarly. This can be explained by the low device utilisation level which was (< 5%) for slow device and (< 14%) for fast device,

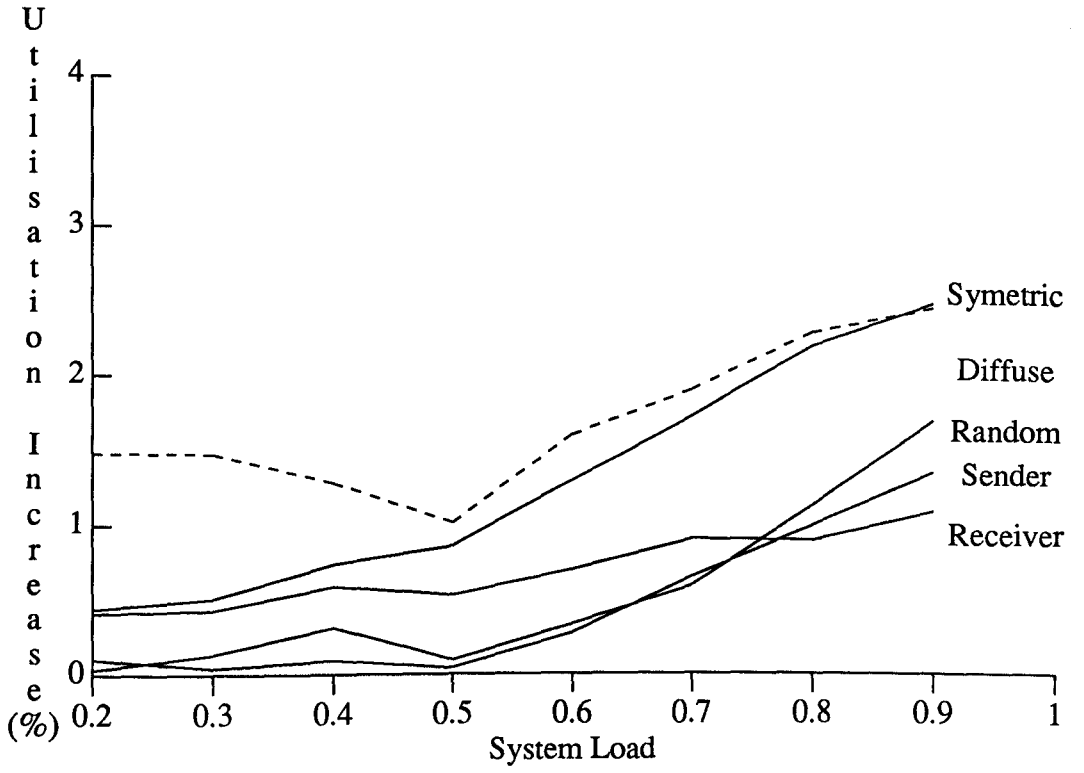


*Figure 5.19 Load Balancing Overhead for the Baseline System (R= 0.13)*

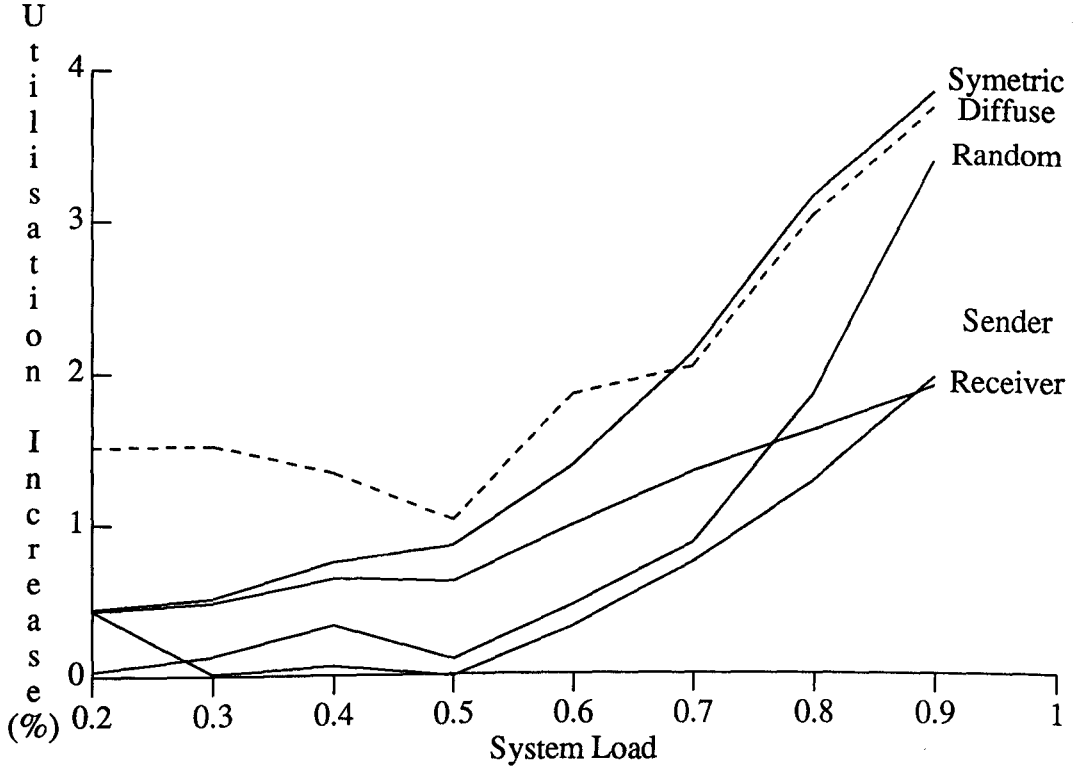


*Figure 5.20 Load Balancing Overhead for R= 0.4*

and the uniform network traffic.

### 5.3.3. Impact of Load Balancing Messages Cost

The performance of the two most promising algorithms Diffuse and Symetric at heavy load level is slightly affected by load balancing messages fixed overhead. The mean response time is shown for a range of 2 to 30 msecs (see Figure 5.22).

The same conclusion is reached when all the algorithms are evaluated with a load balancing message overhead fixed at 20 msecs (see Figure 5.23). As shown is Section 5.2, Sender and Symetric algorithms degrades sharply at very heavy load level under a diskless model. This is not observed for a disk-based model and can be explained by the higher communication device mean request delay which was up to five times higher than for a disk-based model.

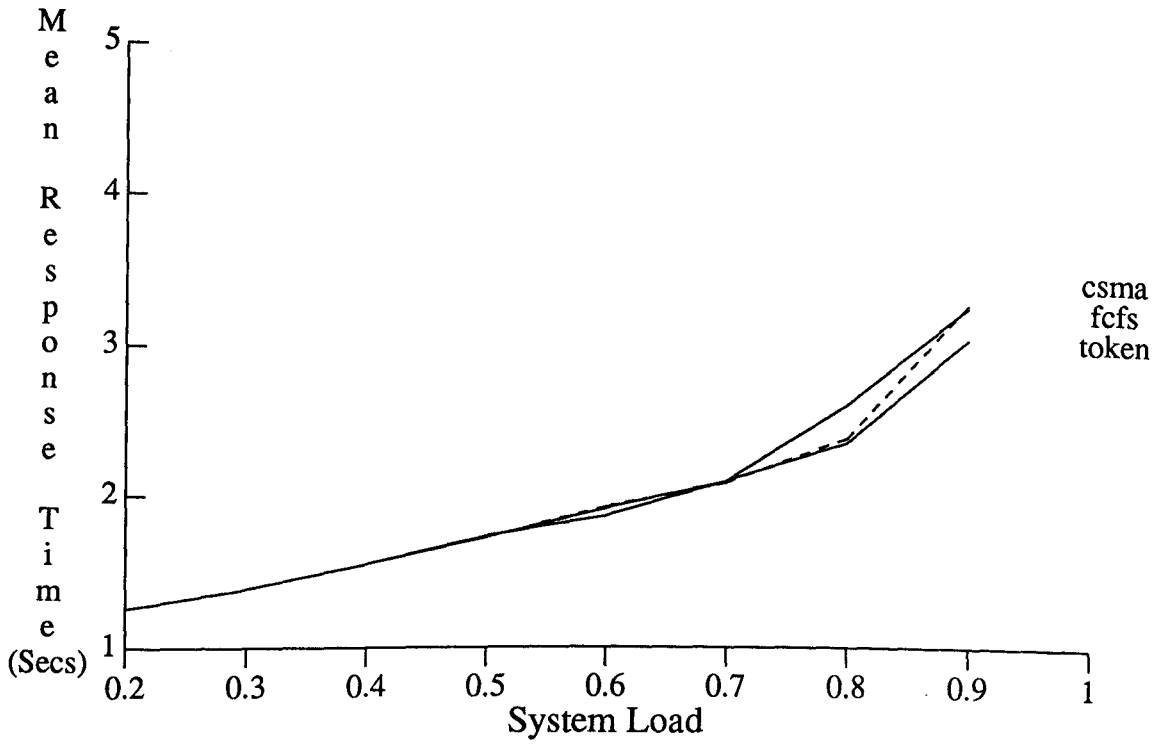


*Figure 5.21 Effect of Communication Protocol on Symetric (R= 0.4)*

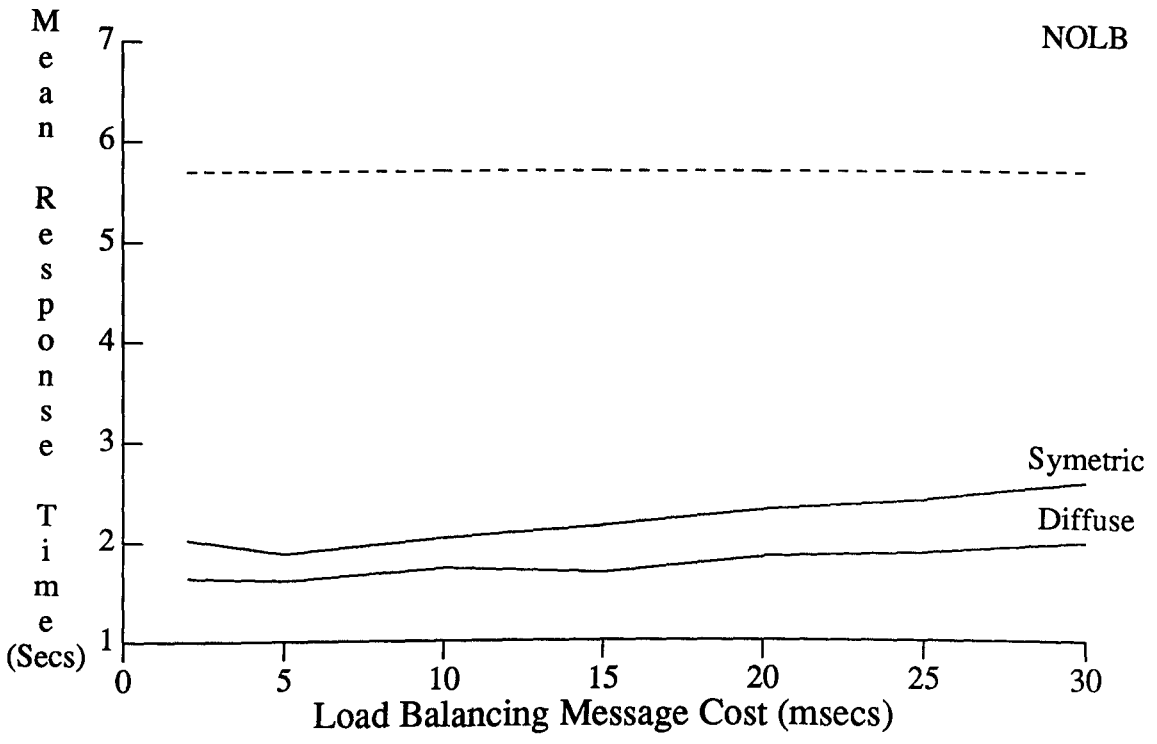


*Figure 5.22 Effect of Load Balancing Message Cost under R= 0.13*

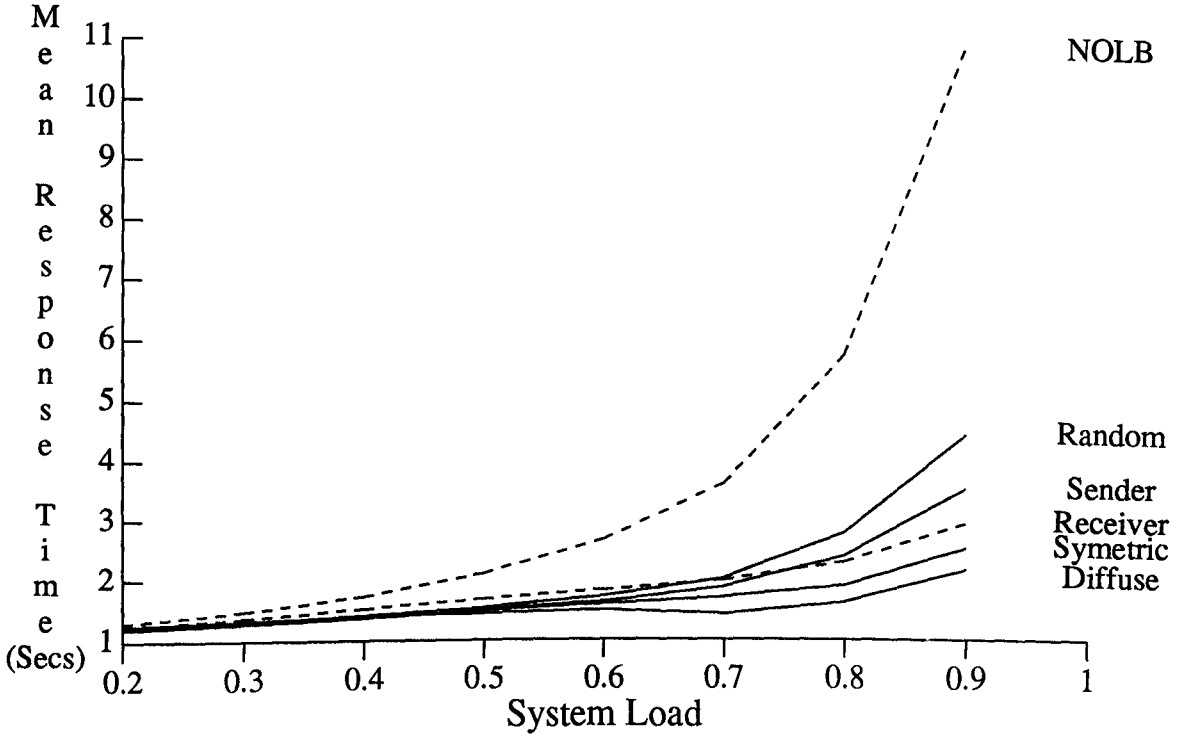


*Figure 5.23 Performance for a Message of 20 msecs (R= 0.13)*

### 5.3.4. Performance under Heterogeneous Workload

In this experiment the performance of the load balancing algorithms using a workload model based on Poisson arrival but with hyper-exponential service demands is investigated. The service demands used have the following characteristics:

a) 95/05 proportion of short/long jobs
   -short jobs mean service time: 0.80 secs
   -long jobs mean service time: 4.80 secs
   -coefficient of variation: 1.04
b) 70/30 proportion of short/long jobs
   -short jobs mean service time: 0.40 secs
   -long jobs mean service time: 2.40 secs
   -coefficient of variation: 1.23

To accommodate this type of workload, some adjustments to the system model were necessary. These adjustments involve the local scheduling discipline and the tuning of the Diffuse algorithm (see details in Section 5.2.5).

When a 95/05 proportion with non-selective transfers is used (see Figure 5.24, and Table 5.13), the same relative performance order of the algorithms as for the diskless model is observed, though the level of reduction of the mean response time is slightly higher for the disk-based model.

For selective transfers (see Figure 5.25, and Table 5.14), the level of performance improvement is much less. The relative performance order of the algorithms is basically unchanged. This is due to the very low number of transferable jobs.

The effect of changing the proportion of long jobs to 70/30 is shown in Figure 5.26, and Table 5.15. Although the performance order of the algorithms is similar to that under 95/05 proportion, the level of improvement is higher. For selective transfers (see Figure 5.27, and Table 5.16), a degradation in the performance of Diffuse algorithm is observed. It is marginally outperformed by the Symetric algorithm.

A general conclusion to be drawn is the similarity of the performance under both diskless and disk-based models. Also for non-selective transfers the level of mean response time reduction is higher than that under homogeneous jobs. For a job separation cost higher than 100 msecs, load balancing with selective transfers is not worthwhile.

### 5.3.5. Conclusion

The main conclusions that can be drawn from the results in this section on the relative performance of different load balancing algorithms are as follows:

- All the algorithms evaluated improve the mean response time of the system at all levels of utilisation. The level of improvement increases with the load level. It reaches 80% at very heavy system loads for the Diffuse algorithm.
- The performance of the algorithms obtained supports the results by Mirchandaney *et. al* [Mirchandan89], in terms of both the relative ordering of the algorithms and the level of performance improvement obtained.
- The main conclusion is that Diffuse is the most promising algorithm. The Symetric algorithm does well but generates more load balancing messages which put more load on the communication device.

Below are summarised the effects of the system attributes for a distributed system based on the disk-based model. As under the diskless model the *compute/communicate ratio* does affect the level of performance improvement but not the relative order of the algorithms. Even under large compute/communicate ratio the utilisation of the communication device due to load balancing activities is less than 14%. For this level it can be justified to assume that there is no contention on the communication device and that the *communication protocols* perform similarly.

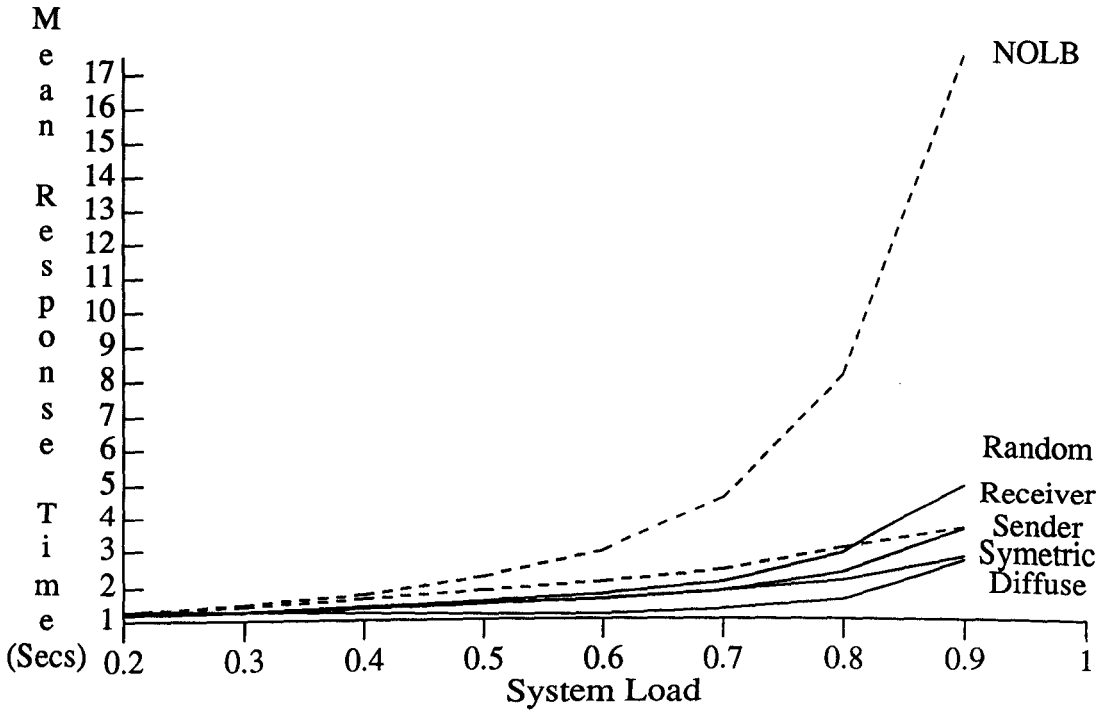


*Figure 5.24 95/05 Jobs Proportion with Non-selective Transfer*

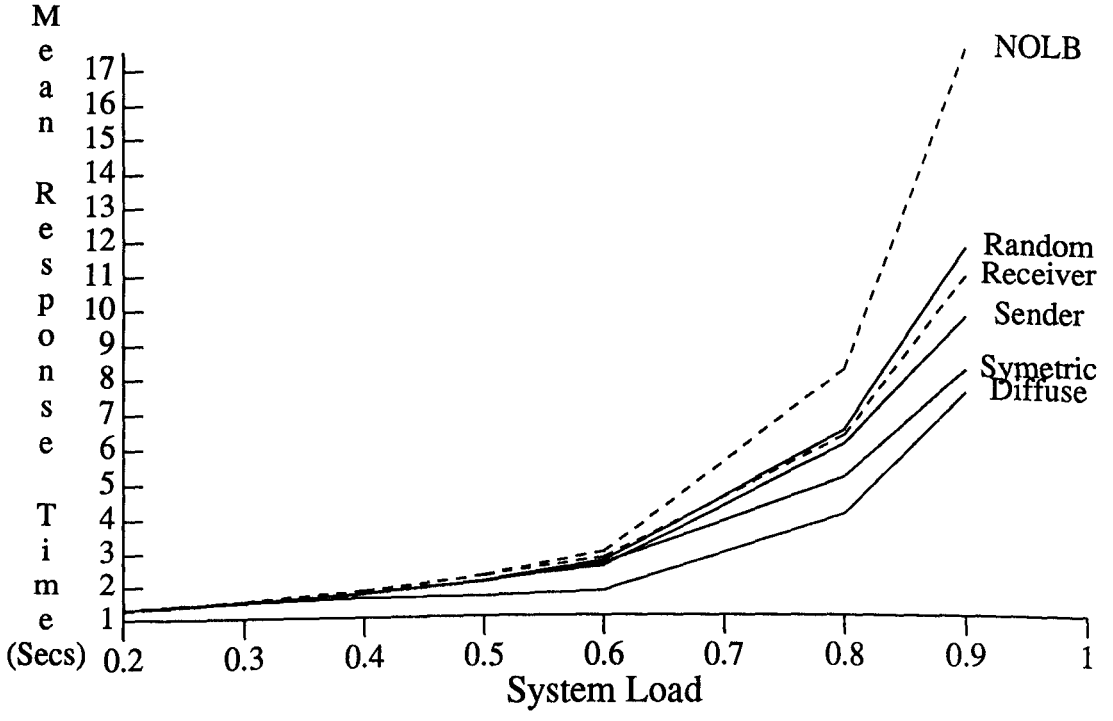


*Figure 5.25 95/05 Proportion of Jobs with selective Transfer*

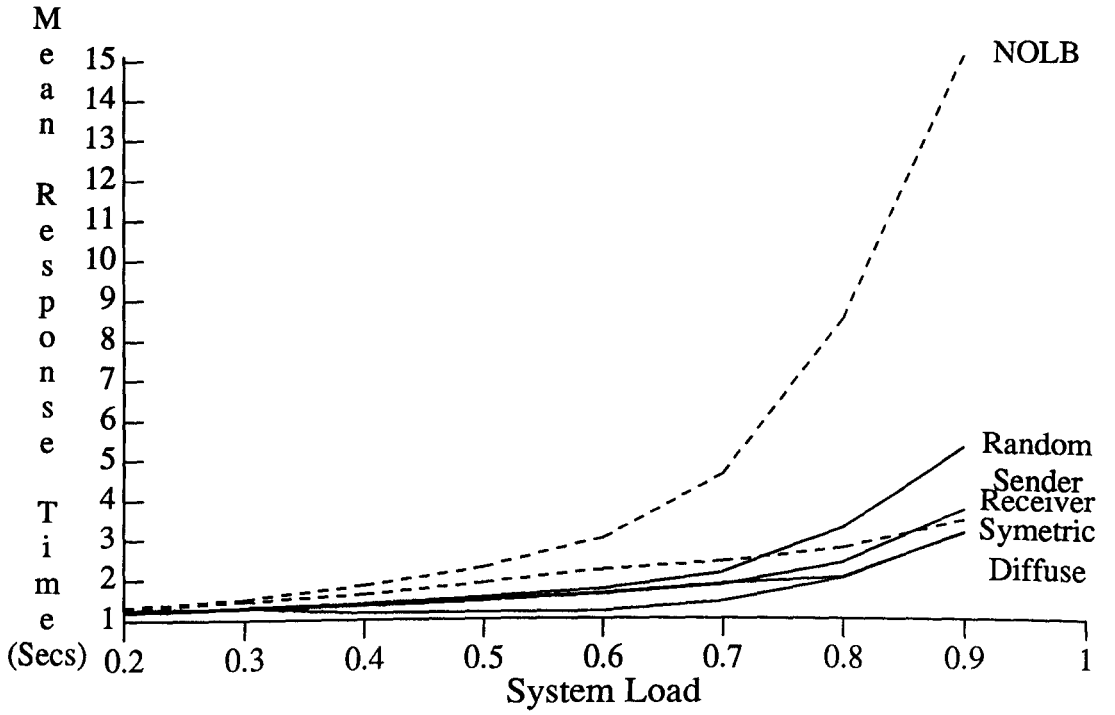


*Figure 5.26 70/30 Proportion of Jobs with Non-selective Transfer*

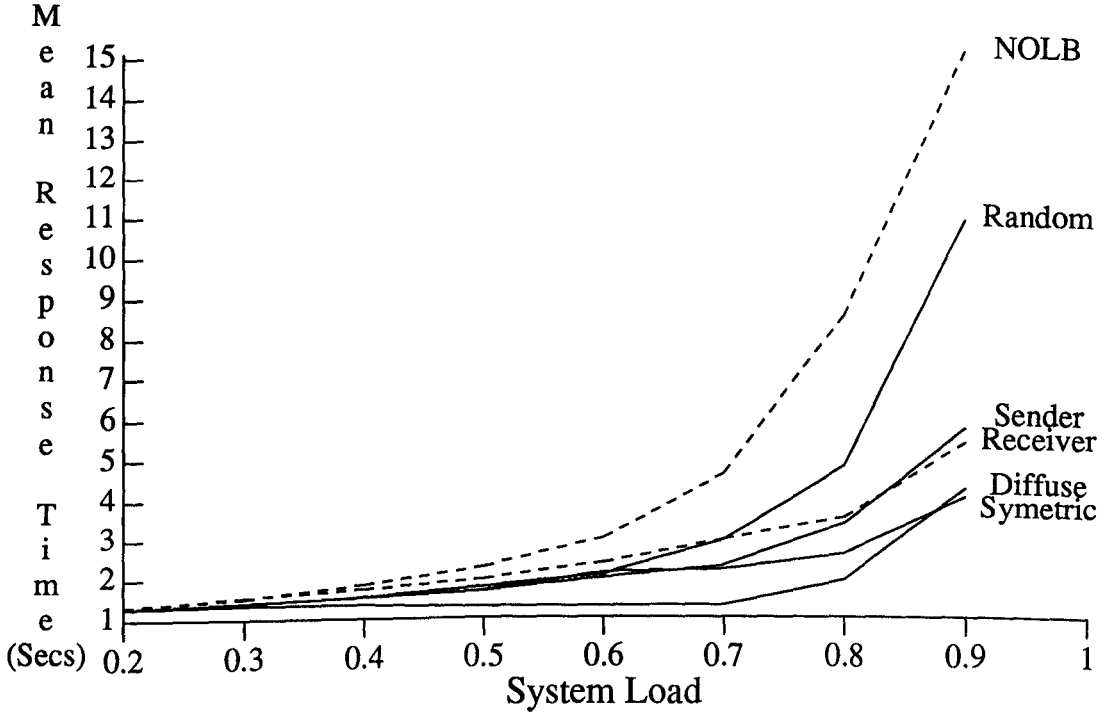


*Figure 5.27 70/30 Proportion of Jobs with selective Transfer*

Load balancing is still effective for a wide range of *load balancing overheads*. For a load balancing message fixed overhead of up to 30 msecs there is no significant degradation of the mean response time. Even Sender and Symetric algorithms are not affected when a 20 msecs fixed message overhead is used, which was not the case under the diskless model. The total load balancing overhead is less than 4%.

When a *heterogeneous workload* is used, the same performance ordering of the algorithms is maintained.

## 5.4. Load Balancing in Systems with Heterogeneous Nodes

In previous experiments the nodes were assumed homogeneous. In the system studied in this section the nodes are assumed to have the same functionality and are subjected to the same job arrival rate or to the same utilisation level but have different computing speeds. A job can run on any node, but its service time depends on the speed of the node where it is executed. Consequently, the load index (i.e. CPU queue length), and remote node selection weight when polling is used, will not have the same system wide weight. To take this into account, adapted versions of the algorithms with information about the nodes computing speed built-into are considered.

There are two ways to specify the workload for a heterogeneous system, namely: scaled arrival rates (in Section 5.4.1) and identical arrival rates (in Section 5.4.2) . The parameters that need to be adjusted to the node speed are the polling probability weight and the timer period for the Diffuse algorithm. In the following sections the adapted algorithms are evaluated against the NOLB case as well as the standard versions which ignore the nodes speed. Further details on standard/adapted algorithms, and scaled/identical arrival rates can be found in Section 3.3.

The system model used is based on the baseline system described in previous Sections with the following modifications made to accommodate heterogeneous hosts:

- System size: 10 heterogeneous hosts with 15 jobs/sec total service rate

  -5 fast hosts: $\mu_1 = 2$ jobs/sec
  -5 slow hosts: $\mu_2 = 1$ job/sec

- Compute/Communicate ratio: R= 0.6

- File server I/O time: 3.75 msecs + communication delay (diskless model)

- Local I/O time: fast hosts: 10 msecs, slow hosts: 20 msecs (disk-based model)

- Workload model: homogeneous users, homogeneous jobs with E[S]= 0.75 secs

  -service time on fast hosts: $S_1$= 0.5 sec
  -service time on slow hosts: $S_2$= 1.0 sec

The experimental factors in this section are three-fold: diskless and disk-based file system structures, standard and adapted algorithm versions, and scaled as well as identical job arrival rates.

### 5.4.1. Evaluation of Algorithms under Scaled Arrival Rates

In this section the system load is specified by scaled arrival rates (jobs/sec). This corresponds to a same level of processor utilisation on all the nodes.

#### 5.4.1.1. Diskless Model

Based on the results represented in Figure 5.28 and Table 5.18, the following assessment can be made:

- At very heavy load level all the algorithms, except the Receiver, a sharp increase of the mean response time takes place. The Receiver maintains its level of response time as the load level increases. This is due to an activation of the algorithm mainly on fast nodes which clear their queue of jobs more often.

- Given the large number of fast nodes, standard algorithms are good enough so that no further improvement can be made by the adapted versions.
- When the Diffuse algorithm is used with a scaled timer, the slow nodes get saturated leading to a sharp degradation of the mean response time .
- A degradation of Sender and Symetric algorithms performance is observed at very heavy load levels. This is due to a higher failure of the sender-initiated negotiations of these algorithms.
- Performance order for standard algorithms at heavy load level is: Receiver, Diffuse, Symetric, Sender, Random.
- Performance order for adapted algorithms at heavy load level is: Receiver, Diffuse, Random, Sender, Symetric.

The advantage of weighted destination is to focus receiver-initiated transfers from slow to fast nodes and sender-initiated transfers from slow to fast nodes. For scaled and identical arrival rates used, this property of adapted algorithms is is barely used. A third arrival pattern where adapted algorithms would be more advantageous is when fast nodes are lightly loaded and while slow nodes are over-used. This case needs to be investigated.

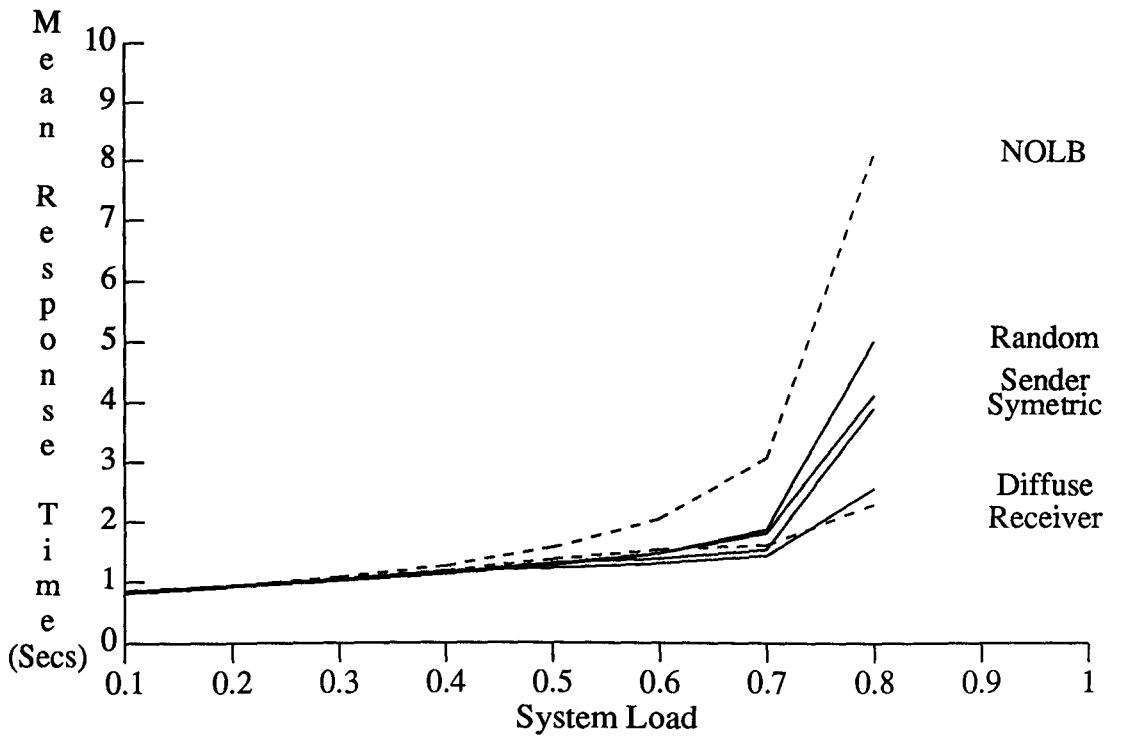


*Figure 5.28 Performance of Standard Algorithms under Scaled Arrivals*

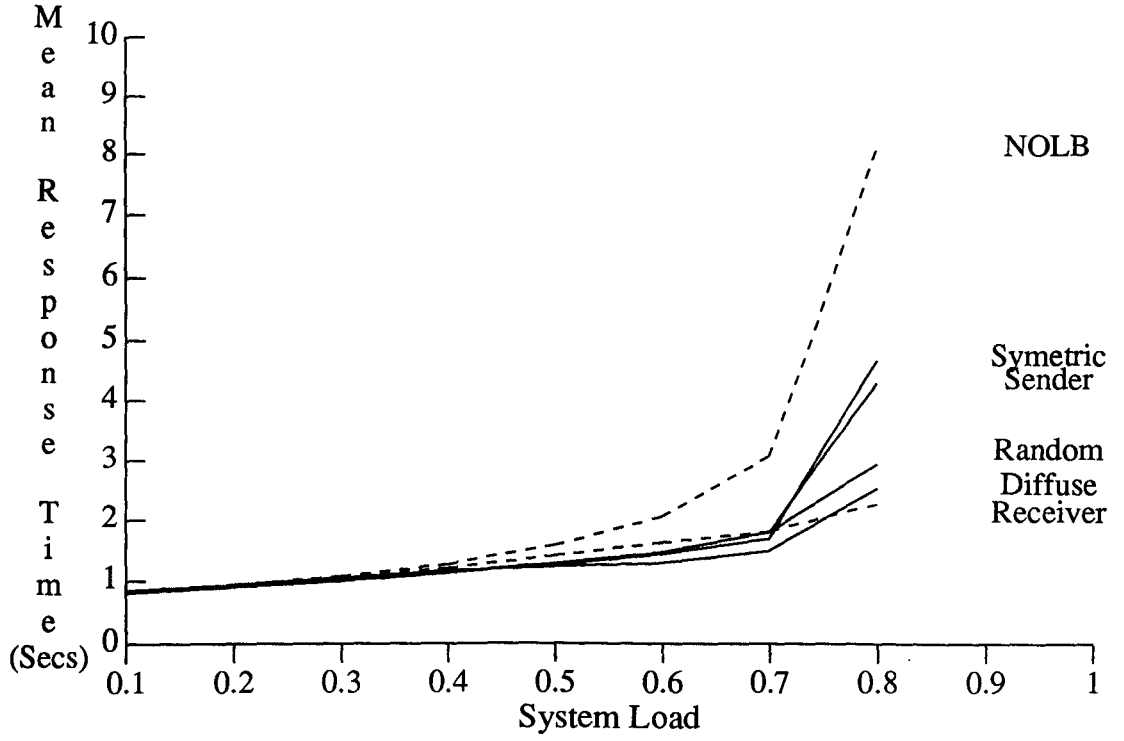


*Figure 5.29 Performance of Adapted Algorithms under Scaled Arrivals*

### 5.4.1.2. Disk-based Model

Based on the results represented in Figure 5.30 and Table 5.20, the following assessment can be made:

- There is a slight improvement of adapted version of the algorithms over standard ones.
- All the algorithms perform similarly. When an important number of nodes in the network are fast, any random polling based algorithm will do.
- The Random algorithm results in the saturation of the slow nodes, particularly in its standard version, leading to a sharp degradation of the mean response time at heavy load levels.
- Due to the higher number of jobs generated at the fast nodes, a significant difference between diskless and disk-based is the saturation of the nodes for diskless model even at 0.8 load level, which means there is a need for a faster file server or a more appropriate local scheduling discipline.
- No clear superiority of Diffuse is observed under either of the two file system structures.

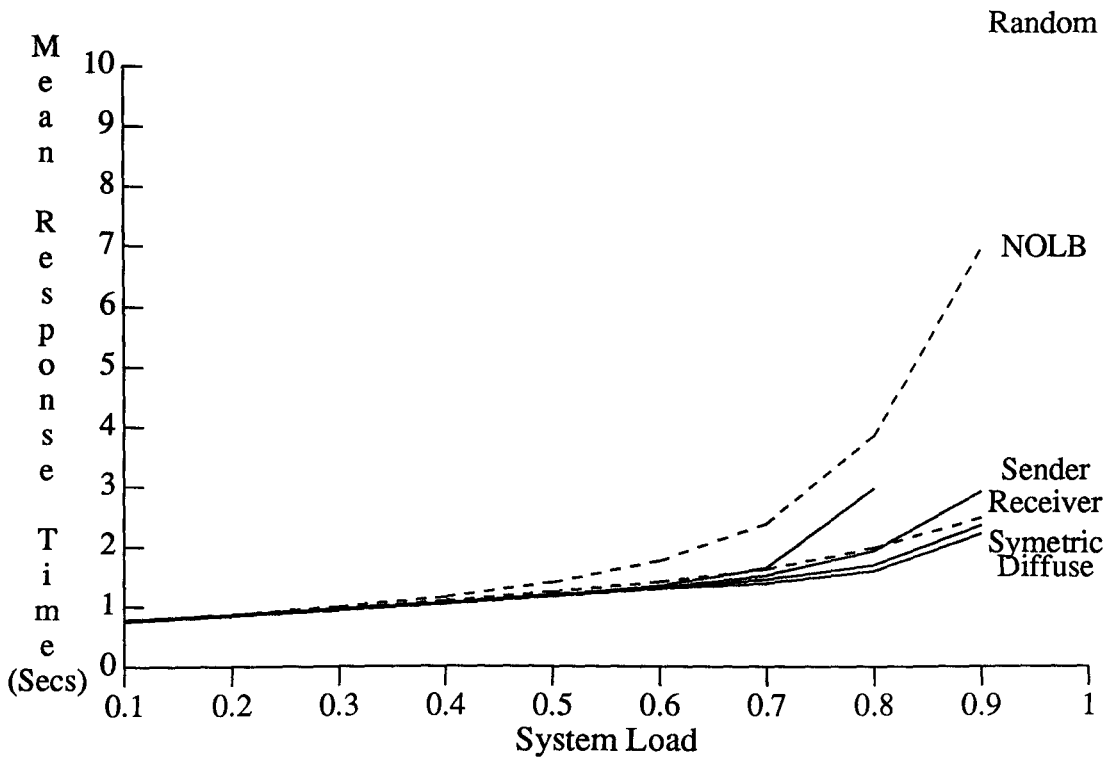


*Figure 5.30 Performance of Standard Algorithms under Scaled Arrivals*

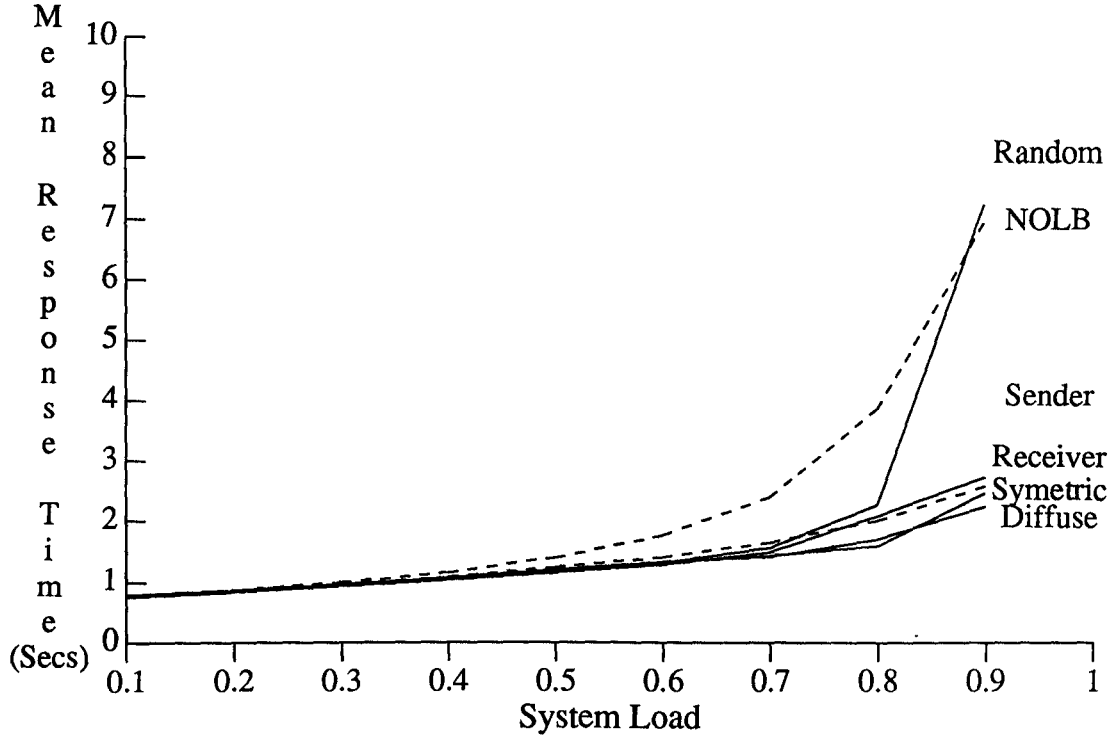


*Figure 5.31 Performance of Adapted Algorithms under Scaled Arrivals*

### 5.4.2. Evaluation of Algorithms under Identical Arrival Rates

In this section, the system load is specified by identical arrival rates (jobs/sec). This corresponds to a normal load on slow nodes and a light load on fast nodes.

#### 5.4.2.1. Diskless Model

Several observations can be made on the results represented in Figures 5.32 and 5.33:

- Both standard and adapted versions of all the algorithms improve the mean response time and its standard deviation by up to 80%, when compared to the NOLB case.
- The level of mean response time is kept nearly constant as the load level is increased. Receiver has the poorest performance. Keeping all the nodes busy may not be appropriate in the context of heterogeneous speeds because keeping a slow node busy while a fast node has only few jobs in its queue can be counter-productive.
- The relative performance order is the same for standard and adapted versions: Diffuse, Symetric, Sender, Random, Receiver. One advantage of adapted version is a lower overhead.
- When the load level is increased to 0.95, the mean response time becomes lower than that under moderate load levels. As the load increase more jobs are transferred from slow nodes to fast nodes where it takes them less time to execute, the mean system response time is reduced.
- For a heterogeneous system the average system percent utilisation can be lower than the NOLB case (e.g. Sender and Random). If a job is generated at a slow node but remotely executed at a fast node, its service time is shorter and the

utilisation level of the slow node is smaller.

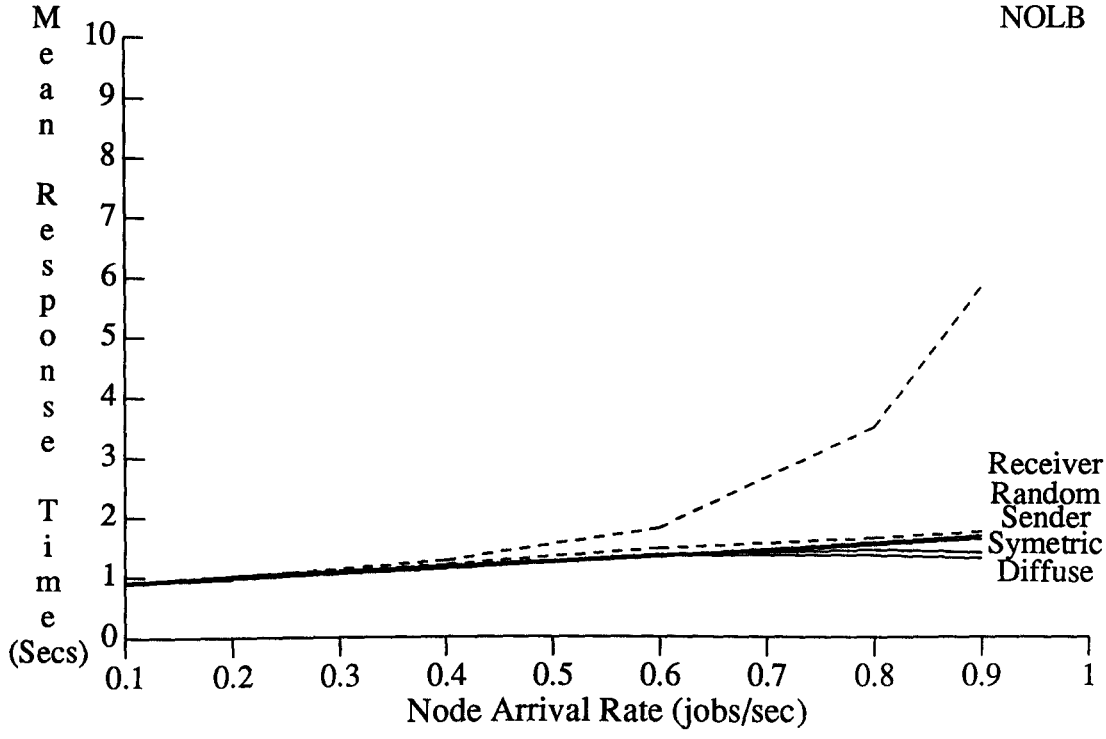


*Figure 5.32 Performance of Standard Algorithms under Identical Arrivals*

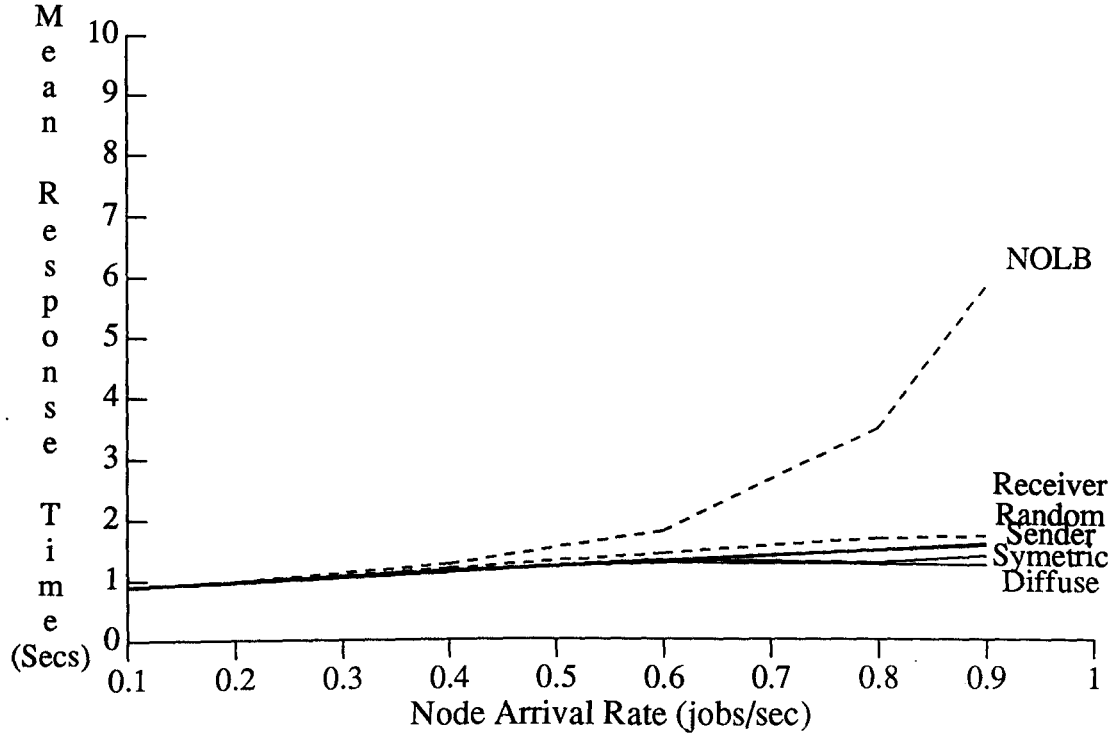


*Figure 5.33 Performance of Adapted Algorithms under Identical Arrivals*

#### 5.4.2.2. Disk-based Model

Several observations can be made on the results represented in Figures 5.34 and 5.35:

- The performance order observed for both standard and adapted versions is: Symetric, Diffuse, Sender, Receiver, Random. However, the mean response time they produce is nearly identical.
- Under identical arrival rates even a 0.95 load level does not does not degrade the mean response time.
- When compared to homogeneous systems the level of wrong movement of jobs is reduced for Diffuse algorithm.
- The performance of Receiver under identical arrival rates is rather poor.

Under identical arrival rates only minor performance differences are observed between diskless and disk-based file system structures.

### 5.4.3. Conclusions

The main conclusions that can be drawn from the results in this section are:

- For the workload models used (i.e. scaled and identical arrival rates) and five fast five slow nodes configuration, there was no significant advantage in the adapted version of algorithms.
- Under scaled arrival rates all the algorithms perform similarly, with the exception of the Sender and Symetric algorithms which degrade the response time for the diskless model at heavy load levels. Also the Random algorithm performs rather poorly.
- Under identical arrival rates, there is a marginal difference in the response time for all the algorithms. However, the performance of Receiver algorithm is slightly

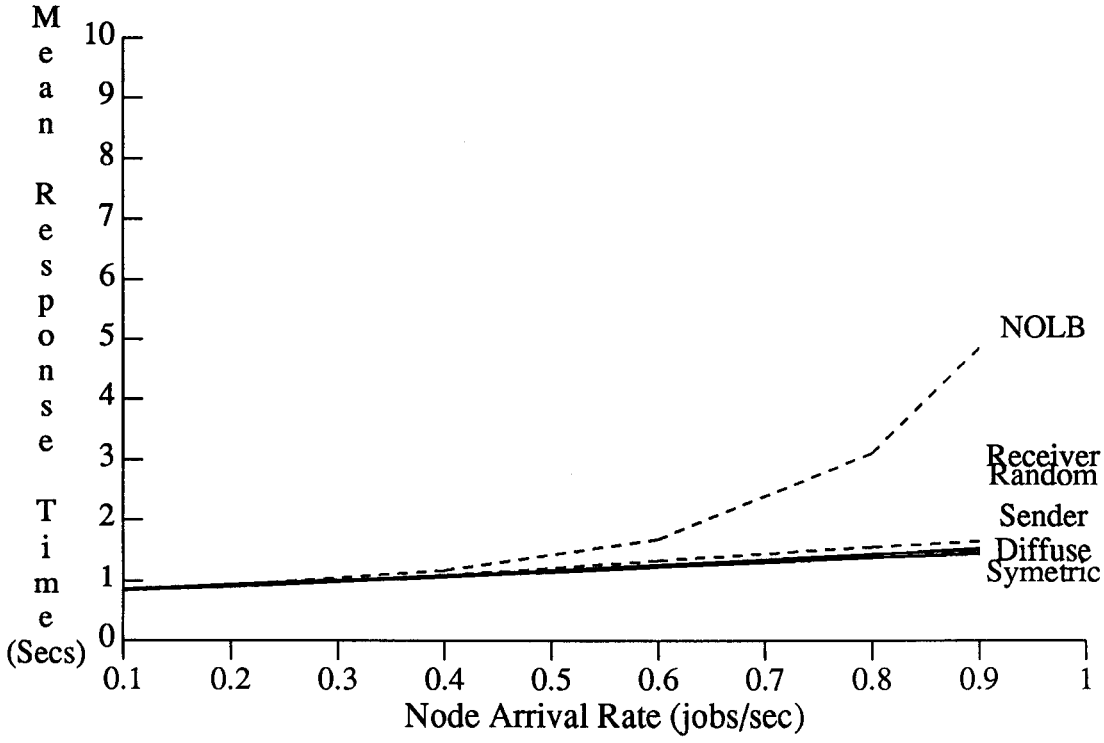


*Figure 5.34 Performance of Standard Algorithms under Identical Arrivals*

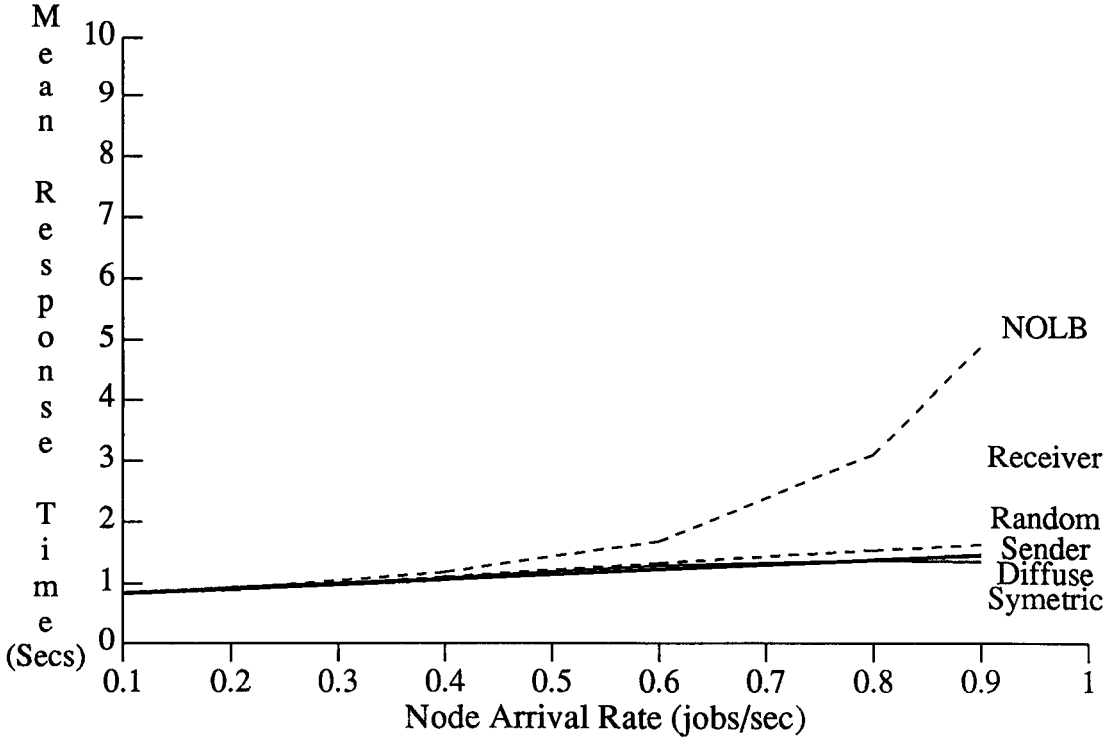


*Figure 5.35 Performance of Adapted Algorithms under Identical Arrivals*

worse than that of Random algorithm. This is because it is harder to find overloaded nodes as the fast nodes complete their jobs in less time. The mean response time is kept nearly steady over the range of load levels.

- The average system busy time can be lower under load balancing than under NOLB. As in the case where jobs generated at a slow nodes are remotely executed at a fast node where they take less time to run.

These conclusions basically hold for both the diskless and disk-based file structures. One exception is the saturation of nodes observed at a load level higher than 0.8, and the sharp degradation of Sender and Symetric algorithms at a load level of 0.8 and higher for the diskless model. The Receiver algorithm maintains its level of improvement over the range of load levels. The advantage of adapted version of the algorithms could be more important when a workload model with high arrival rate on the slow nodes and light arrival rate on the fast nodes is used. Finally this study has shown that heterogeneous systems can be accommodated after minor modifications to the random polling based class of load balancing algorithms.

## 5.5. Further Discussion on the Results

The study given in this chapter has shown that the Diffuse algorithm leads to the smallest job mean response time of the load balancing algorithms studied for a range of system attributes and workload models. Two other issues are explored in this section:

- Scalability
- Confidence Levels for the Results

### 5.5.1. Scalability

An important feature of any load balancing algorithm is that performance improvements are maintained as the number of processors in the system increases. This

is referred to as scalability and some scalability principles have been reviewed in Section 2.3.2. In this thesis we have looked at three broad types of algorithms based on their information policy:

i) no system information
ii) system wide information
iii) information about subset of nodes

For scalability it is important to avoid algorithms that use system wide information. Instead it is better to use algorithms that make their decisions based on a small subset of the nodes.

A study by Zhou [Zhou88] on the effect of varying the system size on the mean job response time, for systems comprising up to 49 nodes connected through an Ethernet network, has shown that for THRHLD (an algorithm from type iii) the best that can be achieved is the performance improvements obtained for systems with 28 nodes. For larger number of nodes no further improvement is obtained. This result can be explained by the fact that the potential gain from having a larger system is consumed by the processing of a larger number of messages and a high number of wrong job transfers. In the case of DISTED (an algorithm from type ii), it is shown that the best results are obtained for a 14 nodes system while a performance deterioration was observed for larger systems. The latter is due to the periodic broadcast nature of the algorithm information policy, which leads to an excessive exchange of information, heavy contention on the communication device, and load balancing decisions based on out of date information. The overhead is higher for each node and grows linearly with the system size.

All the algorithms investigated in this study adhere to the scalability principles established in [Barak87] (see Section 2.3.2). For example the load balancing decision is based on information from a subset of the other nodes (on demand information

gathering policy). A medium size distributed system comprising ten hosts was assumed in this work. To assess the scalability of the results obtained, some further experiments were conducted. The performance of the load balancing algorithms under a heavy load level for 5, 10, 20 nodes (with 0.06, 0.13, and 0.26 compute/communicate ratio respectively) is shown in Figures 5.36 and 5.37. From these results it can be seen that the relative performance ordering of the algorithms holds for different system sizes and that the level of performance improvement increases with the system size. This agrees with the conclusions drawn by Zhou [Zhou88]. In Figures 5.38 and 5.39, the effect of the system size is shown for the Diffuse and Symetric algorithms over the range of load levels. The best results are obtained for a 20 nodes system. We conjecture that these results remain valid for system size of few tens of nodes larger, but as shown in the work in [Livny84] and [Zhou88], even for scalable algorithms, the performance becomes insensitive to the number of nodes as the number of nodes increases. When a larger number of nodes is available in an organization, the way forward is clustering. The nodes can be divided into clusters of few tens to hundred nodes (based on present day communication technology), and should reflect the physical proximity, the administrative boundaries, or other groupings. In a study of load balancing for two-level hierarchical distributed systems Banawan [Banawan87] suggests that most expected gains can be obtained through intra-cluster load balancing, and that no significant further performance improvement can be achieved through inter-cluster load balancing particularly if the job transfer cost between clusters is higher.

### 5.5.2. Confidence Levels for the Results

The need for statistical output analysis is based on the observation that the output data from a simulation exhibits random variability when random numbers generators are used to produce the values of the input variables [Banks84]. Consequently two

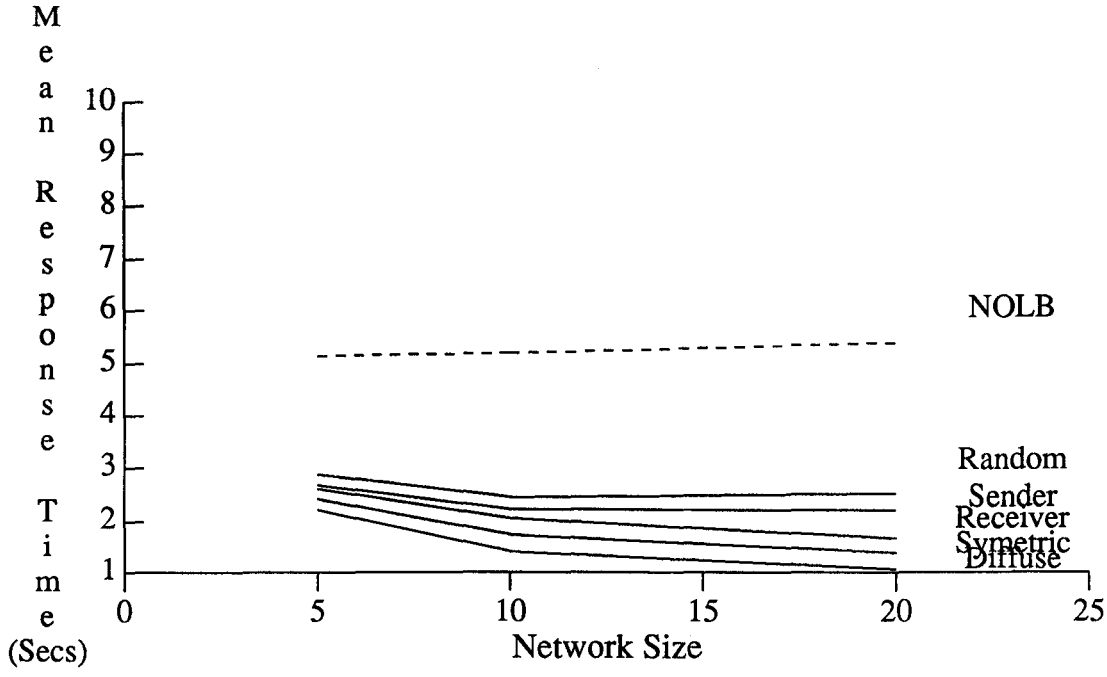


*Figure 5.36 Scalability of Algorithms (Diskless Model)*

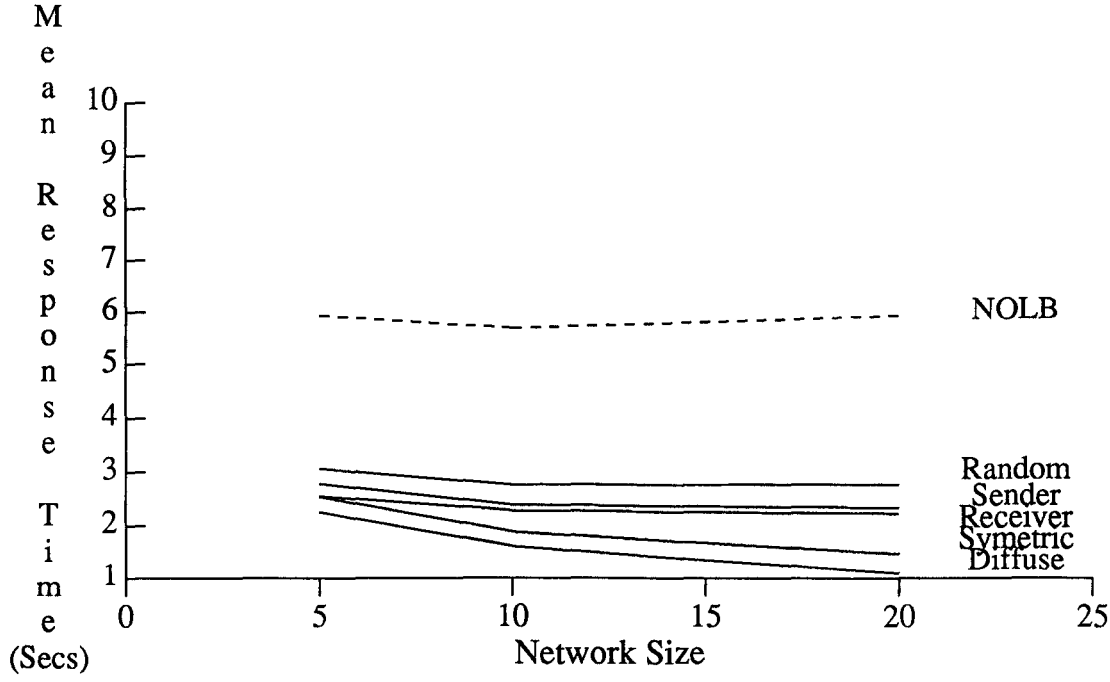


*Figure 5.37 Scalability of Algorithms (Disk-based Model)*

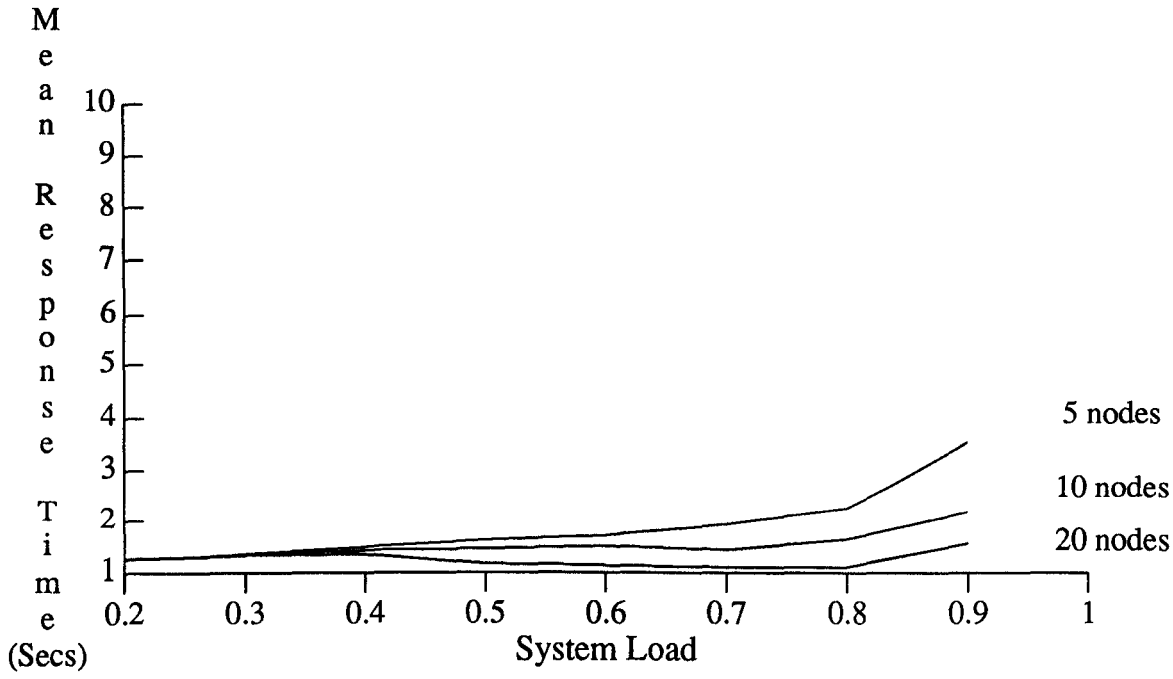


*Figure 5.38 Effect of Network Size on Diffuse Algorithm (Disk-based Model)*

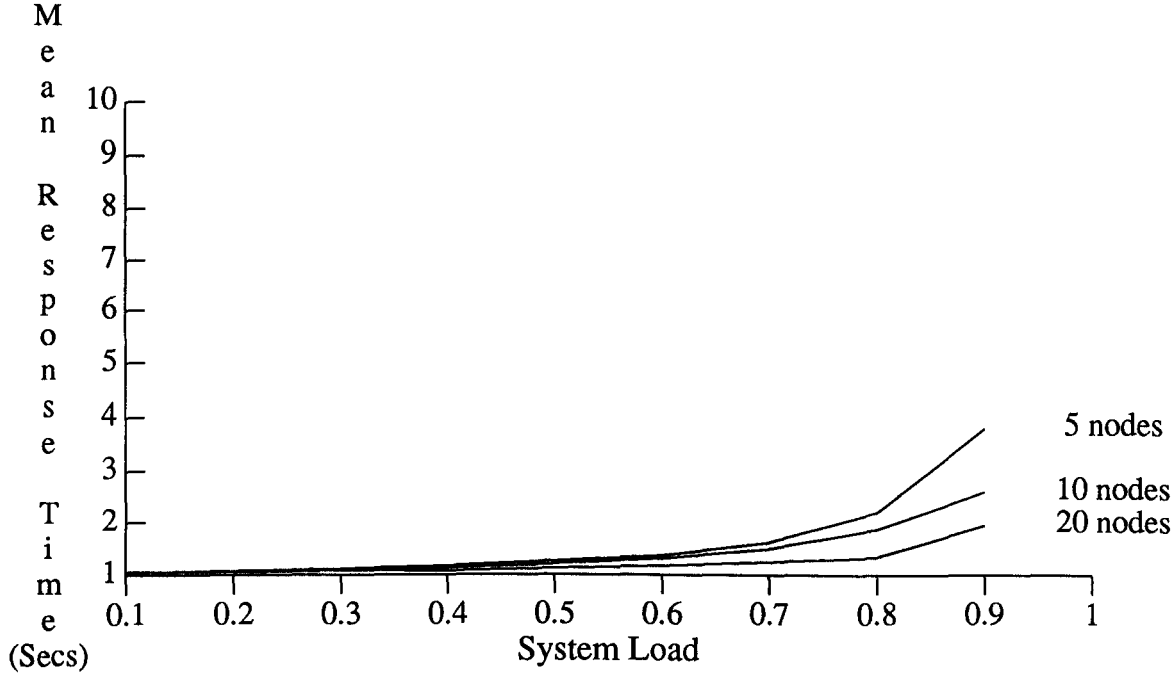


*Figure 5.39 Effect of Network Size on Symetric Algorithm (Disk-based Model)*

different streams of random numbers will produce two sets of output which would highly likely be different. Two other sources of error are the arbitrary or artificial nature of the initial conditions (e.g. starting with an "empty or idle" system) and the lack of accounting of the jobs "left over" when the simulation is terminated. A solution

to the effect of initial conditions is to reset the statistics after some initial phase. However, a common way of dealing with these sources of error is to increase the experiment run length for a long enough interval to make the effect of any error introduced negligible. The choice of the simulation run length is crucial to the validity of the results. There are two ways to deal with this issue: (i) the batch means method and (ii) the replication method. The objective of the batch means method (or one long run) is to monitor the performance measure for a large time interval until the steady state is reached (i.e. the successive performance metric values remain within an acceptable range). In the replication method the same experiment is repeated for a given run length but with different seeds and making sure the results fall within an acceptable confidence interval. Both methods have been used in the analysis of our results.

The minimum simulation run length has been experimentally determined for one seed. This is achieved by recording the mean response time as a function of the run length for a large time interval at different load levels. From the results for a disk-based baseline system shown in Figures 5.40, 5.41, 5.42, and 5.43, it can be seen that a 4000 secs run length is long enough to ensure the effect of initial conditions and "left over" jobs can be ignored, and a reliable ranking of the algorithms is obtained. Also from the results it is apparent that as the load level increases so does the variability of the output. Similar results were obtained for a diskless baseline system. The minimum run length of 4000 seconds corresponds to a generation of about 3,600 jobs population on each host at a very heavy arrival rate for our workload model. To increase the confidence in the results further, the same experiments are repeated at $\rho$=0.8 and $\rho$=0.9 for a 4000 secs simulation run length but for 9 different random number seeds. This is to smooth out the perturbations caused by the statistical nature of the random number generator. On

these replications the 95% confidence interval has been evaluated using the Minitab [1] data analysis software, from which the percentage of error was computed. It has been found that on the average for this run length the mean response time stabilises and the percentage of error is less than *3%* for a system load $\rho \leq 0.8$, and less than *5%* for a system load $\rho=0.9$. For subsequent experiments one seed is chosen. The confidence levels in the job response time numerical results are shown below:

*Diskless baseline system*

| Load level | Percentage of error | 95% Confidence Interval |
|---|---|---|
| $\rho \leq 0.8$ | 3% | ±0.077 (algorithms average) |
| | 1.69% | ±0.024 (Diffuse) |
| $\rho=0.9$ | 5% | ±0.176 (algorithms average) |
| | 3.65% | ±0.065 (Diffuse) |

*Disk-based baseline system*

| Load level | Percentage of error | 95% Confidence Interval |
|---|---|---|
| $\rho \leq 0.8$ | 3% | ±0.085 (algorithms average) |
| | 2.18% | ±0.052 (Diffuse) |
| $\rho=0.9$ | 5% | ±0.222 (algorithms average) |
| | 4.12% | ±0.096 (Diffuse) |

The curves representing the load balancing algorithms ranking on the diskless baseline system and including the confidence intervals are shown in Figure 5.44. Although at the lower load levels the confidence interval is smaller no significant ranking is obtainable. For load levels $\rho<0.5$ the performance of the algorithms is nearly identical because the need for load balancing activation is reduced.

This statistical analysis of the simulation results shows that the estimates of the performance are sufficiently accurate to make the use of the simulation model and the conclusions drawn on the performance ranking of the Diffuse and other algorithms, reliable.

[1] Minitab Inc.

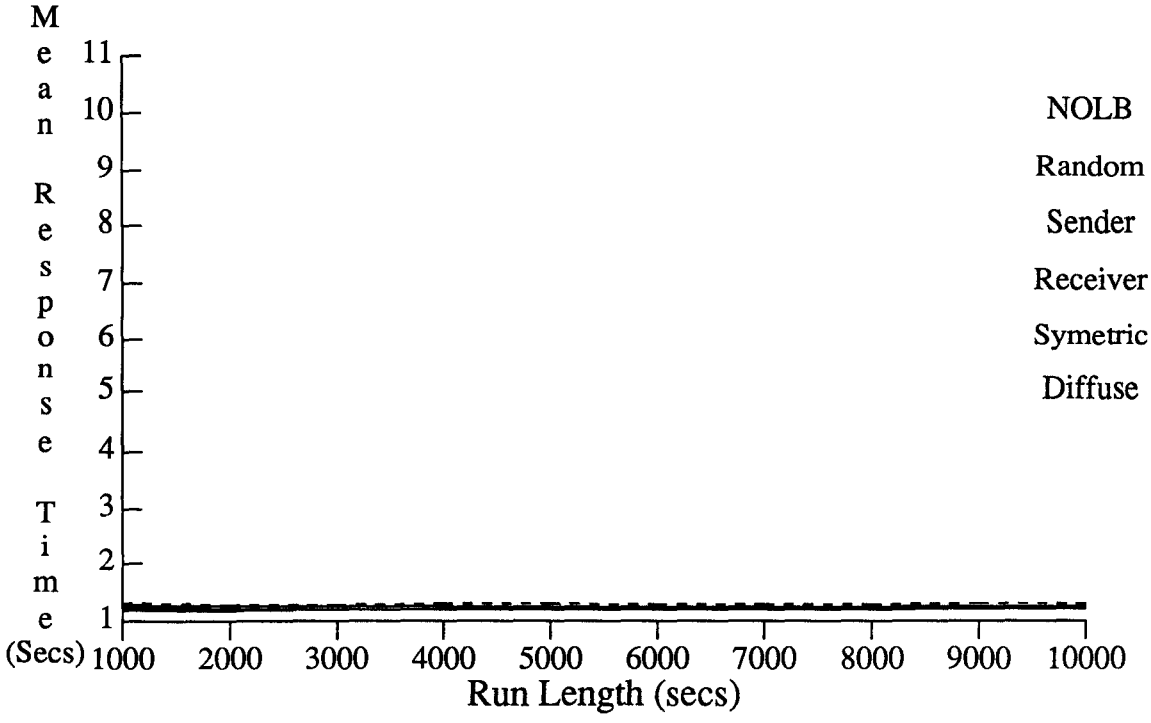


*Figure 5.40 Steady State Performance for* ρ=0.2 *(Disk-based Model)*

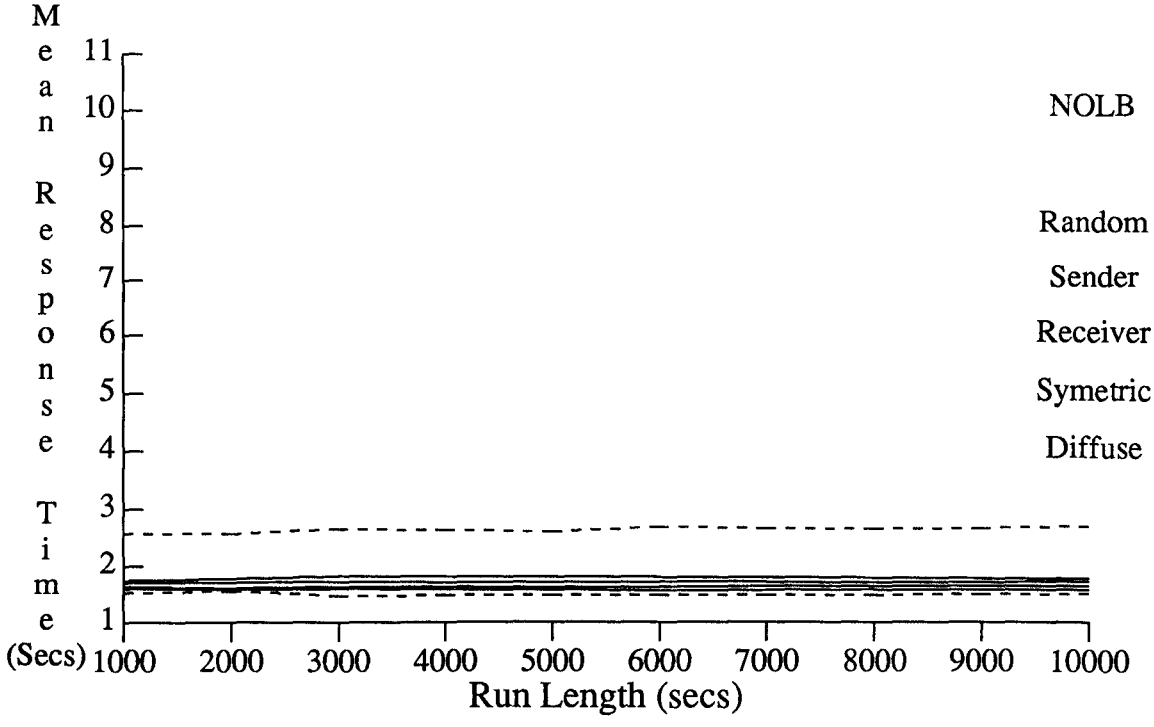


*Figure 5.41 Steady State Performance for* ρ=0.6 *(Disk-based Model)*

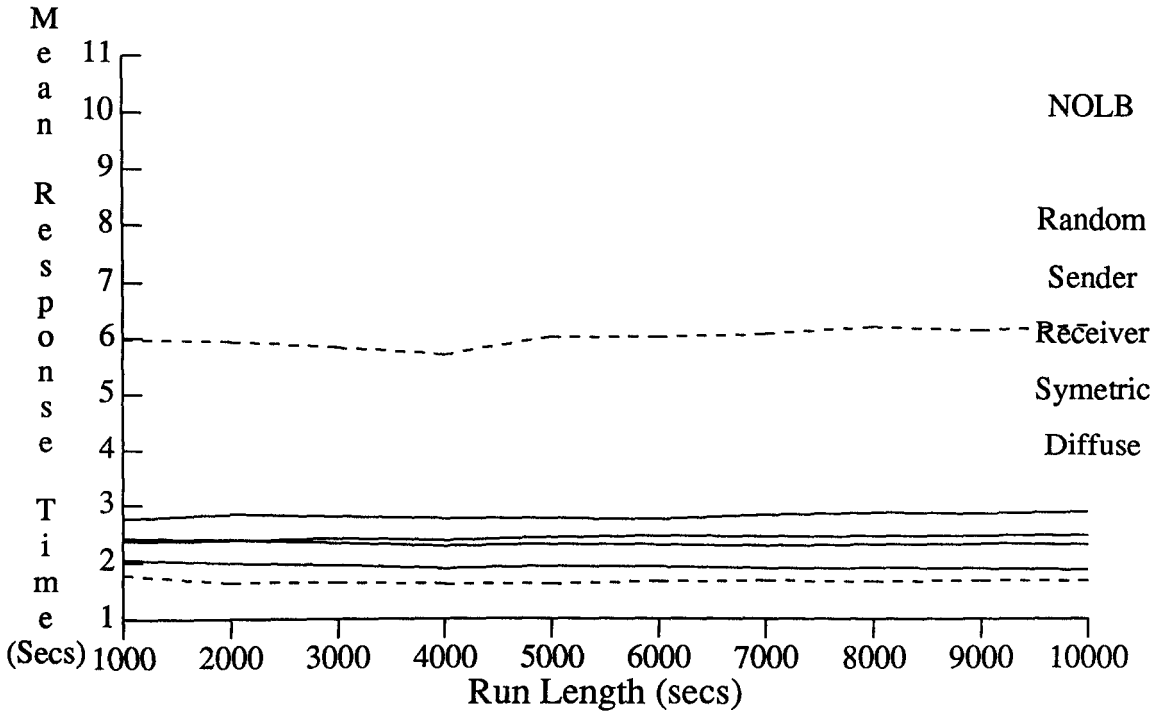


*Figure 5.42 Steady State Performance for* ρ=0.8 *(Disk-based Model)*

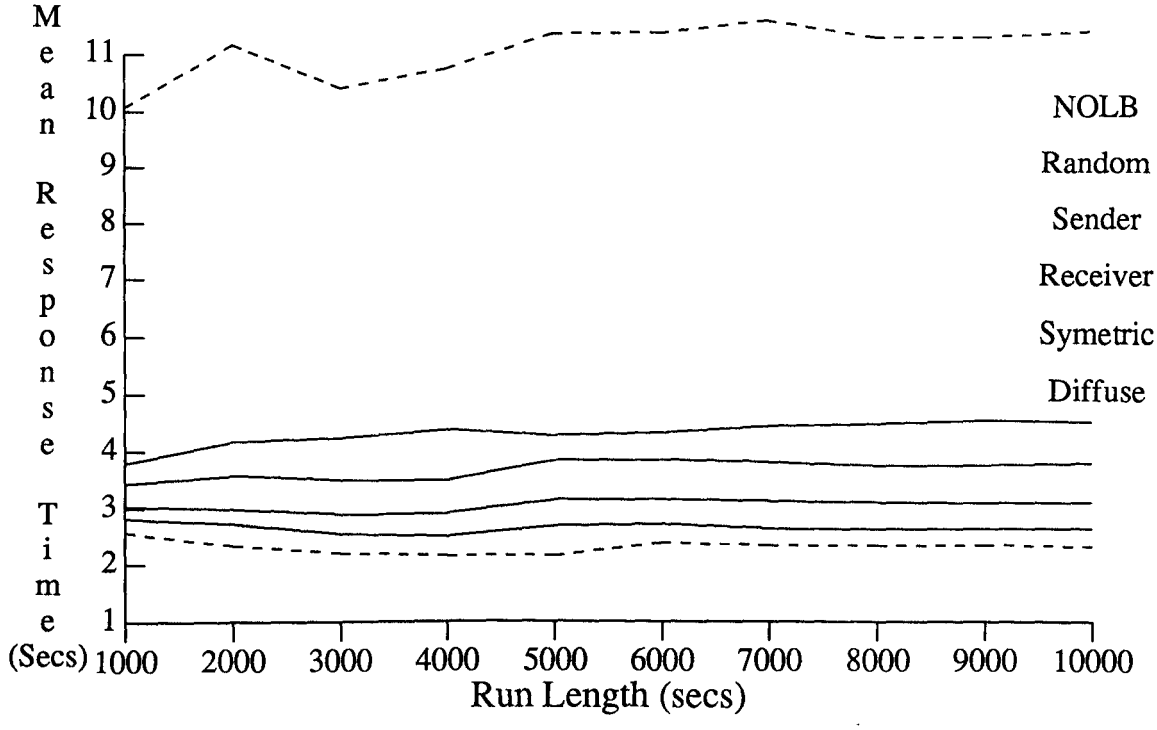


*Figure 5.43 Steady State Performance for* ρ=0.9 *(Disk-based Model)*

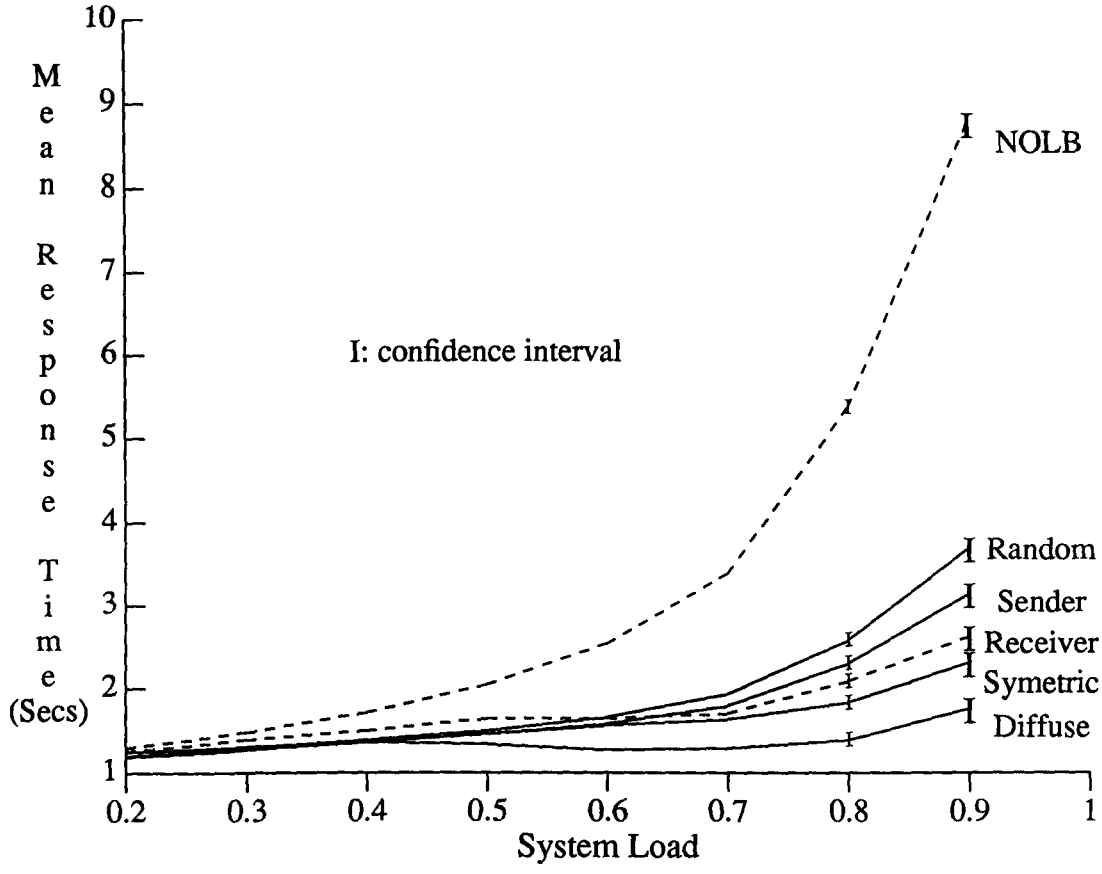


*Figure 5.44 Performance under Compute/Communicate Ratio (R= 0.13, Diskless)*

# CHAPTER 6

## Summary and Future Work

In this chapter, we review the research work described in previous chapters, summarise the main results and draw some conclusions. We then discuss future work related to this research.

### 6.1. Survey of Load Balancing Algorithms

As a result of a survey of the literature on load balancing in distributed systems a taxonomy on load balancing algorithms has been developed. This extends previous taxonomies by considering the algorithm attributes as well as the algorithm components (e.g. information policy, transfer policy, and location/negotiation policy). The attributes include: the load redistribution objective, the decision-making structure, the transparency, the autonomy, the scalability, and the adaptability of the load balancing scheme. For the case of a rapidly changing distributed system environment, the motivation for adaptive scheduling is given and the concept of tolerance of an algorithm is used in the review of the different adaptability approaches. The adaptive load balancing strategies are then structured according to adaptability issues and the dimensions involved. Based on this framework for adaptive scheduling, a design methodology for adaptive load balancing algorithms has been outlined.

A new algorithm called *Diffuse* has been proposed. It is symmetrically initiated and uses periodic polling of a single random node. It produces the best mean response time among the algorithms evaluated.

## 6.2. Performance of Load Balancing Algorithms

The assessment of this performance study of load balancing algorithms is made at three levels: the relative ordering of the algorithms within the simulated systems, the effect of the system attributes and workload on the performance of the algorithms, and the wider implications of these results.

### 6.2.1. Algorithms Performance within Simulated Systems

A simulated system was used to evaluate a set of load balancing algorithms. To assess the effect of distributed system attributes three system versions were built. The relative performance ordering of these algorithms across all three versions is deduced from the results of the simulation given in the last chapter.

In the case of heterogeneous systems modifications to the Sender, Random, Receiver, Symetric, and Diffuse algorithms have been made to take into account the processing speed of the hosts. When choosing a random node to transfer a job to, or a node to engage a polling negotiation with, a weighting factor proportional to the node speed is used in the probability distribution (i.e. *weighted destination*). For the Diffuse algorithm a *scaled timer* mechanism is used.

The summary of the performance of the algorithms is:

**Random**

This simple algorithm does not require the knowledge of the system state information. Hence no load balancing overheads will be incurred with the exception of the overhead associated with transferring a job. Although it has the lowest overhead, its performance is the poorest due to the high level of wrong job movement. Also it does not preserve the autonomy of the nodes. It is not recommended.

**Receiver**

Under this algorithm load balancing is initiated when the load at a node drops below a pre-specified threshold. It has a good performance for moderate to heavy load levels. In the case of heterogeneous systems with scaled arrival rates, this is the only algorithm that maintains the reduction of the response time even for heavy to very heavy load levels.

**Sender**

In this algorithm the load balancing is initiated by the overloaded node. It performs well for light to moderate load levels. Its advantage over Receiver algorithm is maintained even at heavy load levels under a heterogeneous workload. However, a sharp degradation of the mean response time is observed under scaled arrival rates on diskless heterogeneous systems at heavy load levels.

**Symetric**

This algorithm is a combination of Sender and Receiver algorithms. It has a good performance over the whole range of load levels. However, this algorithm tends to generate more load balancing messages which results in more overheads on the nodes. This makes it more sensitive to the communication bandwidth.

**Diffuse**

This is a novel periodic version of the Symetric algorithm. To reduce the number of load balancing messages only single probes are allowed and periodically the load at a node is checked. The algorithm is a hybrid of the above ones and selects between one of three possible cases: i) the receiver-initiated component of the algorithm is activated if the load is below the threshold, ii) the sender-initiated component of the algorithm is activated if the load is above the threshold, and iii) no load balancing i s activated if the load equals the threshold value. The Diffuse

algorithm is the most promising one and on the simulated system produces the lowest job mean response time. This advantage is maintained practically under all three system versions, and over the range of system attributes and workload. However, this algorithm does involve a high level of wrong job movements. Also some care is needed in interpreting these results because the timer period of this algorithm has been tuned for optimal performance for the simulated system.

**Adapted Algorithms Versions** (case of heterogeneous systems)

These versions attempt to focus sender-initiated probes from slow nodes to fast nodes and receiver-initiated probes from fast nodes to slow nodes. For the speeds configuration (i.e. large number of fast nodes) and job arrival patterns considered, no significant advantage of adapted versions was observed when compared the standard versions where no focusing is attempted. Further work is needed on these systems by considering other nodes speed configurations and heterogeneous users.

### 6.2.2. Effect of System Attributes and Workload Models

An important aspect of this study was the assessment of the effect of system attributes and workload model on the load balancing algorithm performance. This effect can be on three performance aspects: the job mean response time, the level of job mean response time reduction relative to the NOLB case, and the relative ordering of the algorithms.

#### File System Structure

The file system structure has no significant effect on the relative ordering of the load balancing algorithms. However, the results might not generalise to all practical systems because only non-preemptive transfers were considered. It is to be noted that a sharp degradation of the job mean response time is observed under diskless model at

heavy load level with scaled arrivals for heterogeneous systems.

### Communication Bandwidth

Although the relative ordering of the algorithms is not affected, the communication bandwidth does affect the level of performance improvement. As the communication delays get longer an increase of the threshold level becomes necessary. Also the curves of the algorithms tend to cluster, making the choice of the load balancing algorithm less relevant.

### Communication Protocols

The performance of algorithms under "First Come First Serve", "CSMA/CD", "Token Passing" protocols is nearly identical. However, some care is needed in interpreting this results because under the operating conditions used (i.e. system size, job size, homogeneous users), the network traffic was such that a less 50% utilisation of the communication device was induced, with similar medium access demands from all nodes.

### Load Balancing Overheads

Provided it is not too high (e.g. < 20 msecs), the fixed message cost has no significant effect on the algorithm performance ordering. The job separation cost for selective transfers can significantly affect the level of algorithms performance. Load balancing is not worthwhile if this cost exceeds a minimum value (e.g. 50 msecs).

### File Server Speed

Provided a minimum service rate is available, the file server speed does not affect the job mean response time substantially. It is to be noted that the file server saturates while the communication device and computing node are still providing normal service.

**Workload Model**

When a heterogeneous workload is used, the relative ordering of the algorithms is basically unchanged. However, the cross-over Sender/Receiver does not take place even at a 0.9 load level. Non-selective transfers are more advantageous than selective transfers.

### 6.2.3. Wider Implications

The wider implications of this study on the development of load balancing schemes for distributed systems are two-fold:

**1) Design of Distributed Systems**

The performance level provided by the Diffuse algorithm and its consistency over a range of system attributes and workload makes it a very promising algorithm. It is recommended that this algorithm be evaluated using a real distributed system.

**2) Simulation**

This thesis has demonstrated the value of simulation in the design of load balancing algorithms. It has enabled a range of algorithms to be evaluated and a new algorithm (Diffuse) has been proposed. It has been shown that it is important to consider more complex systems than that can be studied using theoretical tools. For example this has enabled us to study more realistic file system structure. Our investigations suggest that in future experiments it is not necessary to consider the communication protocols and the heterogeneous workload model. The same conclusions can be drawn from a homogeneous model if only the ordering of the algorithms is sought. However, the modelling of other system attributes can affect the quality of the results significantly. The correct representation of the communication bandwidth and the algorithm parameters tuning (i.e. $T$, $Pt$) is important to get an

accurate ordering of the algorithms. Assuming non-preemptive transfers, any file system structure is sufficient for load balancing algorithms ordering purposes. However, to get a clearer idea on the level of performance improvement the modelling of the specific file system structure is necessary. For the diskless file system structure, to realise the potential benefits of load balancing a minimum file server speed is needed to avoid a major I/O bottleneck (i.e. incorrectly configured system). A reasonable value of the load balancing message overhead is needed to obtain a realistic level of performance improvement.

## 6.3. Future Work

This work has demonstrated the utility of simulation in identifying the most promising load balancing algorithms. However, it is important to test these simulation results, particularly for Diffuse and Symetric algorithms, on a real distributed systems to confirm the conclusions drawn based on simulation. With simulation only load balancing policies can be investigated with confidence. The load balancing mechanisms such as remote process management and user interface facilities are much more difficult to simulate. Load balancing can become a reality only when its performance and cost effectiveness is proven on actual distributed systems.

Another investigation worthwhile to undertake is the issue of adaptive load balancing based on dynamic parameter tuning and multi-options algorithms, which are more suitable to a changing environment (i.e. in this study a fixed size system with mainly homogeneous users were assumed). This aims for the development of load balancing algorithms where explicit adaptability is added to maintain the performance of a distributed system with a rapidly changing environment. This involves an automatic switching of the load balancing algorithm policies and the dynamic adjustment of the algorithm parameters to take into account the fluctuating system

attributes and workload environment. A preliminary work on this approach to adaptive load balancing is described in Section 2.4. This could increase the confidence in the Diffuse algorithm further.

In this study the jobs were assumed independent sequential units. In parallel computation, which is becoming more popular due to advances in the related hardware, a program consists of several modules which need to be dispatched to different hosts for execution. Since these modules are not independent because they need to interact with each other to carry out their tasks, load balancing in this context involves different objectives and requirements. It is worthwhile to investigate the applicability of the load balancing concepts considered in this work to parallel systems.

## REFERENCES

ANSA87.

ANSA, *ANSA Reference Manual Release 00.03,* ANSA Project, Cambridge (1987).

Alonso86.

R. Alonso, "The Design of Load Balancing Strategies for Distributed Systems," *Proc. of Workshop on Future Directions in Computer Architecture & Software,* pp. 202-207 (May 1986).

Alonso88.

R. Alonso and L. L. Cova, "Sharing Jobs Among Independently Owned Processors," *IEEE Proc. 8th International Conference on Distributed Computing Systems,* pp. 282-288 (June 1988).

Banawan87.

S. A. Banawan, "An Evaluation of Load Sharing in Locally Distributed Systems," PhD Thesis, University of Washington (1987).

Banks84.

J. Banks and J. S. Carson, *Discrete-Event System Simulation,* Prentice-Hall International (1984).

Barak85.

A. Barak and A. Shiloh, "A Distributed Load-balancing Policy for a Multicomputer," *Software- Practice and Experience* **15** (9) pp. 901-913 (September 1985).

Barak87.

A. Barak and Y. Kornatzky, "Design Principles of Operating Systems for Large Scale Multicomputers," pp. 104-123 in *International Workshop on Experiences*

*with Distributed Systems*, ed. J. Nehmer ,Springer Verlag (September 1987).

Beck90.

B. Beck, "AAMP: A Multiprocessor Approach for Operating System and Application Migration," *ACM Operating Systems Review* **24** (2) pp. 41-55 (April 1990).

Bershad86.

B. Bershad, "Load Balancing with Maitre d'," *;login:* **11(1)** pp. 32-43 (Jan/Feb. 1986).

Bonomi88.

F. Bonomi and A. Kumar, "Adaptive Optimal Load Balancing in a Hetereogeneous Multiserver System with a Central Job Scheduler," *IEEE Proc. 8th International Conference on Distributed Computing Systems*, pp. 500-508 (June 1988).

Broy87.

M. Broy and T. Streicher, "Views of Distributed Systems," pp. 114-143 in *Lecture Notes in Computer Science on Mathematical Models for the Semantics of Parallelism*, ed. M. V. Zilli,Springer-Verlag (1987).

Bryant81.

R. M. Bryant and R. A. Finkel, "A Stable Distributed Scheduling Algorithm," *IEEE Proc. 2nd Intern. Conf. on Distributed Computer Control Systems*, pp. 314-323 (April 1981).

CACI89.

CACI, *NETWORK II.5 User's Manual Version 5.0*, CACI Products Company (August, 1989).

Cabrera86.

L. F. Cabrera, "The Influence of Workload on Load Balancing Strategies," *Proc. of the 1986 Summer USENIX Conference*, pp. 446-458 (June 1986).

Casavant87.

T. L. Casavant and J. G. Kuhl, "Analysis of Three Dynamic Distributed Load-Balancing Strategies with Varying Global Information Requirements," *IEEE Proc. 7th Internat. Conf. on Distributed Computing Systems*, (September 1987).

Casavant87a.

T. L. Casavant, "DSSAP - An Automated Design Aid for Algorithms and Software Development in Distributed Computing Systems," *2nd International Conf. on Supercomputing*, pp. 123-132 (May 1987).

Casavant88.

T. L. Casavant and J. G. Kuhl, "A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems," *IEEE Transactions on Software Engineering* **14** (**2**) pp. 141-153 (February 1988).

Castagnoli86.

C. Castagnoli, "Load Balancing Computational Servers in a UNIX Environment," *EUUG Autumn 1986*, pp. 267-272 (1986).

Chandras90.

R. G. Chandras, "Distributed Message Passing Operating Systems," *ACM Operating Systems Review* **24** (**1**) pp. 7-17 (January 1990).

Cheriton88.

D. R. Cheriton, "The V Distributed System," *Communications of the ACM* **31**(**3**) pp. 314-333 (March 1988).

Cheung88.

S. Cheung, S. Dimitriadis, and W. J. Karplus, *Introduction to Simulation using NETWORK II.5,* CACI Products Company (September, 1988).

Chow86.

T. C. K. Chow, "Distributed Control of Computer Systems," *IEEE Transactions on Software Engineering* **C-35**(June 1986).

Concepcion88.

A. I. Concepcion and W. M. Eleazar, "A Testbed for Comparative Studies of Adaptive Load Balancing Algorithms," *Proc. of the SCS Multiconference on Distributed Simulation,* pp. 131-135 (February 1988).

Coulouris88.

G. F. Coulouris and J. Dollimore, *Distributed Systems: Concepts and Design,* Addison Wesley (1988).

Dikshit89.

P. Dikshit, S. K. Tripathi, and P. Jalote, "SAHAYOG: A Test Bed for Evaluating Dynamic Load-sharing Policies," *Software- Practice and Experience* **19 (5)** pp. 411-435 (May 1989).

Eager85.

D. L. Eager, E. D. Lazowska, and J. Zahorjan, "A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing," *Proc. of the 1985 ACM SIGMETRICS Conference on Measurement and Modelling of Computer Systems,* pp. 1-3 (August 1985).

Eager86.

D. L. Eager, E. D. Lazowska, and J. Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems," *IEEE Transactions on Software Engineering*

**SE-12 (5)** pp. 662-675 (May 1986).

Eager88.

D. L. Eager, E. D. Lazowska, and J. Zahorjan, "The Limited Performance Benefits of Migrating Active Processes for Load Sharing," *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pp. 63-72 (May 1988).

Efe82.

K. Efe, "Heuristic Models of Task Assignment Scheduling in Distributed Systems," *IEEE Computer*, (June 1982).

Evans88.

J. B. Evans, *Structures of Discrete Event Simulation: An Introduction to the Engagement Strategy*, Ellis Horwood Series in Artificial Intelligence (1988).

Ezzat86.

A. K. Ezzat, "Load Balancing in NEST: A Network of Workstations," *Proc. Fall Joint Computer Conference*, pp. 1138-1149 (November 1986).

Ferrari85.

D. Ferrari, "A Study of Load Indices for Load Balancing Schemes," Report No. UCB/CSD 86/262, Computer Science Division (EECS), University of California, Berkeley, California 94720 (October 1985).

Goscinski90.

A. Goscinski and M. Bearman, "Resource Management in Large Distributed Systems," *ACM Operating Systems Review* **24 (4)** pp. 7-25 (October 1990).

Green88.

J. J. Green, "Load Balancing Algorithms in a Distributed Processing Environment," PhD Thesis , University of California at Los Angeles (1988).

Hac87.

A. Hac and X. Jin, "Dynamic Load Balancing in a Distributed System Using a Decentralized Algorithm," *IEEE Proc. 7th Internat. Conf. on Distributed Computing Systems*, (September 1987).

Hagmann86.

R. Hagmann, "Process Server: Sharing Processing Power in a Workstation Environment," *IEEE Proc. 6th Inter. Conf. on Distributed Computer Systems*, pp. 260-267 (May 1986).

Hammond86.

J. L. Hammond and J. P. O'Reilly, *Performance Analysis of Local Computer Networks*, Addison-Wesley (1986).

Hayter88.

T. Hayter and G. R. Brookes, "Simulation of some Local Area Network Topologies," Report no. 88/5, Department of Computer Science - University of Hull (1988).

Hoare85.

C.A.R. Hoare, *Communicating Sequential Processes*, Prentice Hall (1985).

Hong88.

J. Hong, X. Tan, and M. Chen, "From Local to Global: An Analysis of Nearest Neighbor Balancing on Hypercube," *Proc. ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pp. 73-82 (May 1988).

Hsu86.

C. Y. H. Hsu and J. W. S. Liu, "Dynamic Load Balancing Algorithms in Homogeneous Distributed Systems," *IEEE Proc. 6th International Conference on Distributed Computing Systems*, pp. 216-223 (May 1986).

Jard88.

C. Jard, J. F. Monin, and R. Groz, ''Development of Veda, a Prototyping Tool for Distributed Algorithms,'' *IEEE Transactions on Software Engineering* **14(3)**(March 1988).

Jesty88.

P. H. Jesty and K. Benmohammed-Mahieddine, ''Modelling Distributed Systems Services,'' Report 88.16, School of Computer Studies, University of Leeds (July 1988).

Johnson88.

I. D. Johnson, ''A Study of Adaptive Load Balancing Algorithms for Distributed Systems,'' PhD Thesis, Aston University, U.K. (January 1988).

Johnson89.

M. Johnson, ''Network Protocol Performance,'' *Proc. ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, (May 1989).

Kara89.

M. Kara, P. H. Jesty, and T. G. Gough, ''A Distributed Scheduling Algorithm Based on a Decentralised Global Plan Strategy,'' Report 89.29, School of Computer Studies, University of Leeds (December 1989).

Kleinrock85.

L. Kleinrock, ''Distributed Systems,'' *Communications of the ACM* **28(11)**(November 1985).

Kobayashi78.

H. Kobayashi, *Modelling and Analysis: An Introduction to System Performance Evaluation Methodology,* Addison-Wesley (1978).

Krueger84.

P. Krueger and R. Finkel, "An Adaptive Load Balancing Algorithm for a Multicomputer," Computer Sciences Technical Report #539, University of Wisconsin- Madison (April 1984).

Krueger87.

P. Krueger and M. Livny, "The Diverse Objectives of Distributed Scheduling Policies," *IEEE Proc. 7th Internat. Conf. on Distributed Computing Systems*, (September 1987).

Krueger87a.

P. Krueger and M. Livny, "When is the Best Load Sharing Algorithm a Load Balancing Algorithm?," Computer Sciences Technical Report #694, University of Wisconsin - Madison (April 1987).

Krueger88.

P. Krueger, "Distributed Scheduling for a Changing Environment," Computer Sciences Technical Report #780, University of Wisconsin- Madison (June 1988).

Krueger88a.

P. Krueger and M. Livny, "A Comparison of Preemptive and Non-Preemptive Load Distributing," *IEEE Proc. 8th International Conference on Distributed Computing Systems*, pp. 123-130 (June 1988).

Kunz91.

T. Kunz, "The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme," *IEEE Transactions on Software Engineering* **17** (7) pp. 725-730 (July 1991).

Lavenberg83.

S. S. Lavenberg, *Computer Performance Modeling Handbook,* Academic Press

(1983).

Lazowska86.

E. D. Lazowska, J. Zahorjan, D. R. Cheriton, and W. Zwaenepoel, "File Access Performance of Diskless Workstations," *ACM Transactions on Computer Systems* **4(3)**(August 1986).

Lee86.

K. J. Lee and D. Towsley, "A Comparison of Priority-Based Decentralized Load Balancing Policies," *Proc. of Performance '86 and ACM SIGMETRICS 1986*, pp. 70-77 (May 1986).

Leland86.

W. E. Leland and T. J. Ott, "Load-balancing Heuristics and Process Behavior," *Proc. of the ACM SIGMETRICS Conference*, pp. 54-69 (May 1986).

Lin87.

F. C. H. Lin and R. M. Keller, "The Gradient Model Load Balancing Method," *IEEE Transactions on Software Engineering* **13(1)**(January 1987).

Livny84.

M. Livny, "The Study of Load Balancing Algorithms for Decentralized Distributed Processing Systems," Computer Sciences Technical Report #570, University of Wisconson- Madison (December 1984).

Lo84.

V. M. Lo, "Heuristic Algorithms for Task Assignment in Distributed Systems," *IEEE Proc. 4th Inter. Conf. on Distributed Computing Systems*, (1984).

Mirchandan89.

R. Mirchandaney, D. Towsley, and J. A. Stankovic, "Analysis of the Effects of Delays on Load Sharing," *IEEE Transactions on Computers* **38 (11)** pp.

1513-1525 (November 1989).

Mitrani87.

I. Mitrani, *Modelling of Computer and Communication Systems,* Cambridge University Press (1987).

Mullender86.

S. J. Mullender and A. S. Tanenbaum, "The Design of a Capability-based Distributed Operating System," *The Computer Journal* **29 (4)** pp. 289-299 (1986).

Mutka87.

M. W. Mutka and M. Livny, "Profiling Workstations' Available Capacity for Remote Execution," *Computer Sciences Technical Report #697*, University of Wisconsin- Madison, (April 1987).

Needham82.

R. M. Needham and A. J. Herbert, *The Cambridge Distributed System,* Addison Wesley (1982).

Ni85.

L. M. Ni, C. W. Xu, and T. B. Gendreau, "A Distributed Drafting Algorithm for Load Balancing," *IEEE Transactions on Software Engineering* **11 (10)** pp. 1153-1161 (1985).

Ousterhout82.

J. K. Ousterhout, "Scheduling Techniques for Concurrent Systems," *IEEE Proc. 3rd Internat. Conf. on Distributed Computer Systems*, pp. 22-30 (1982).

Power89.

J. Power, "Distributed System Evolution - Some Observations," *ACM Operating Systems Review* **23 (2)** pp. 31-32 (1989).

Pulidas88.

S. Pulidas, D. Towsley, and J. A. Stankovic, "Imbedding Gradient Estimators in Load Balancing Algorithms," *IEEE Proc. 8th International Conference on Distributed Computing Systems*, (June 1988).

Ramamritha87.

K. Ramamritham and W. Zhao, "Meta-Level Control In Distributed Real-Time Systems," *IEEE Proc. 7th Internat. Conf. on Distributed Computing Systems*, (September 1987).

Renesse88.

R. Renesse, H. Staveren, and A. S. Tanenbaum, "Performance of the World's Fastest Distributed Operating System," *ACM Operating Systems Review* **22** (4) pp. 25-34 (October 1988).

Shamir87.

E. Shamir and E. Upfal, "A Probabilistic Approach to the Load-Sharing Problem in Distributed Systems," *Journal of Parallel and Distributed Computing* **4** pp. 521-530 Academic Press, (1987).

Siewiorek82.

D. P. Siewiorek, C. G. Bell, and A. Newell, *Computer Structures: Principles and Examples*, McGraw Hill (1982).

Smith88.

J. M. Smith, "A Survey of Process Migration Mechanisms," *ACM Operating Systems Review* **22(3)** pp. 28-40 (July 1988).

Stankovic82.

J. A. Stankovic, N. Chowdhury, R. Mirchandaney, and I. Sidhu, "An Evaluation of the Applicability of Different Mathematical Approaches to the Analysis of

Decentralized Control Algorithms,'' *Proc. IEEE COMPSAC*, pp. 62-69 (November 1982).

Stankovic84a.

J. A. Stankovic, ''Simulations of three Adaptive, Decentralized Controlled, Job Scheduling Algorithms,'' *Computer Networks* **8** pp. 199-217 Elsevier Science, (1984).

Stankovic85.

J. A. Stankovic, ''An Application of Bayesian Decision Theory to Decentralized Control of Job Scheduling,'' *IEEE Transactions on Computers* **C34(2)** pp. 117-130 (February 1985).

Stankovic84.

J. H. Stankovic and I. S. Sidhu, ''An Adaptive Bidding Algorithm for Processes, Clusters and Distributed Groups,'' *IEEE Proc. 4th Internat. Conf. on Distributed Computing Systems*, pp. 49-58 (1984).

Stumm88.

M. Stumm, ''The Design and Implementation of a Decentralized Scheduling Facility for a Workstation Cluster,'' *IEEE Proc. 2nd Conference on Computer Workstations*, pp. 12-21 (March 1988).

Tanenbaum85.

A. S. Tanenbaum and R. Van Renesse, ''Distributed Operating Systems,'' *ACM Computing Surveys* **17** pp. 419-470 (December 1985).

Tantawi85.

A. Tantawi and D. Towsley, ''Optimal Static Load Balancing in Distributed Computer Systems,'' *Journal ACM* **32(2)**(April 1985).

Theimer85.

M. M. Theimer, K. A. Lantz, and D. R. Cheriton, "Preemptable Remote Execution Facilities for the V-System," *ACM 10th Symposium on Operating Systems Principles*, (1985).

Theimer88.

M. M. Theimer and K. A. Lantz, "Finding Idle Machines in a Workstation-based Distributed System," *IEEE Proc. 8th International Conference on Distributed Computing Systems*, pp. 112-122 (June 1988).

Tilborg84.

A. M. Van Tilborg and L. D. Wittie, "Wave Scheduling- Decentralized Scheduling of Task Forces in Multicomputers," *IEEE Transactions on Computers* **C-33 (9)** pp. 835-844 (September 1984).

Walker83.

B. J. Walker, G. J. Popek, R. Kline, and G. Thiel, "The LOCUS Distributed Operating System," *Proc. 9th ACM Symposium in Operating System Principles*, pp. 49-70 (Oct 1983).

Wang85.

Y. T. Wang and R. J. T. Morris, "Load Sharing in Distributed Systems," *IEEE Transactions on Computers* **C-34 (3)** pp. 204-217 (March 1985).

Zhou86.

S. Zhou, "An Experimental Assessement of Resource Queue Lengths as Load Indices," Report No. UCB/CSD 86/298, Computer Science Division (EECS), University of California, Berkeley, California 94720 (June 1986).

Zhou87.

S. Zhou, "Performance Studies of Dynamic Load Balancing in Distributed

Systems," Report No. UCB/CSD 87/376, Computer Science Division (EECS), University of California, Berkeley, California 94720 (October 1987).

Zhou87a.

S. Zhou and D. Ferrari, "A Measurement Study of Load Balancing Performance," *IEEE Proc. 7th Internat. Conf. on Distributed Computing Systems*, (September 1987).

Zhou88.

S. Zhou, "A Trace-Driven Simulation Study of Dynamic Load Balancing," *IEEE Transactions on Software Engineering* **14 (9)** pp. 1327-1341 (September 1988).

# Appendix

## Detailed Simulation Results

The detailed simulation results included in this appendix are divided according to the three groups of experiments identified in Chapter 5:

i) Homogeneous Diskless Systems (Tables 1-9)

ii) Homogeneous Disk-based Systems (Tables 10-18)

iii) Heterogeneous Systems (Tables 19-26)

These tables indicate the benefit brought by the load balancing algorithms and the cost incurred. The benefit is expressed in terms of job mean response time and its predictability shown by the standard deviation. The cost is expressed in terms of the percentage of job movement (out of the total number of jobs processed), the percentage of wrong job movements, the number of load balancing messages per node per second, and the average percentage increase in processor utilisation. The load pattern indicates the average external load level present at each node in the system (LLLLL: $\rho_i$= 0.4, MMMMM: $\rho_i$= 0.6, HHHHH: $\rho_i$= 0.8, and VVVVV: $\rho_i$= 0.9).

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 1.71 | 0.00 | 0.71 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.39 | 18.64 | 0.54 | 24.45 | 13.74 | 16.68 | 0.00 | 0.05 |
| RECEIVR | 1.50 | 12.07 | 0.60 | 15.99 | 6.15 | 0.00 | 1.41 | 0.62 |
| SENDER | 1.36 | 20.65 | 0.54 | 24.50 | 12.50 | 0.25 | 0.12 | 0.06 |
| SYMTRIC | 1.36 | 20.31 | 0.53 | 25.66 | 13.00 | 0.29 | 1.71 | 0.77 |
| DIFFUSE | 1.36 | 20.38 | 0.54 | 24.02 | 10.47 | 3.80 | 2.31 | 1.70 |

Load Pattern= LLLLL

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 2.53 | 0.00 | 0.86 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.66 | 34.30 | 0.56 | 35.23 | 30.46 | 33.83 | 0.00 | 0.00 |
| RECEIVR | 1.64 | 35.19 | 0.58 | 33.08 | 15.18 | 0.39 | 1.79 | 0.72 |
| SENDER | 1.58 | 37.35 | 0.52 | 39.46 | 24.89 | 0.72 | 0.47 | 0.12 |
| SYMTRIC | 1.56 | 38.31 | 0.49 | 42.56 | 27.00 | 0.80 | 2.75 | 1.22 |
| DIFFUSE | 1.27 | 49.93 | 0.56 | 35.28 | 24.50 | 9.69 | 2.20 | 2.02 |

Load Pattern= MMMMM

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 5.22 | 0.00 | 1.64 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 2.45 | 52.98 | 0.88 | 46.47 | 57.56 | 60.45 | 0.00 | 0.17 |
| RECEIVR | 2.06 | 60.50 | 0.78 | 52.69 | 26.32 | 0.82 | 1.70 | 0.81 |
| SENDER | 2.23 | 57.22 | 0.84 | 48.80 | 31.58 | 1.59 | 1.44 | 0.80 |
| SYMTRIC | 1.74 | 66.70 | 0.67 | 59.10 | 43.00 | 1.95 | 3.92 | 2.08 |
| DIFFUSE | 1.41 | 72.97 | 0.74 | 54.64 | 37.93 | 19.05 | 2.10 | 2.48 |

Load Pattern= HHHHH

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 8.48 | 0.00 | 2.36 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 3.42 | 59.66 | 1.31 | 44.54 | 71.07 | 73.14 | 0.00 | 0.81 |
| RECEIVR | 2.49 | 70.58 | 1.07 | 54.76 | 28.87 | 1.05 | 1.43 | 0.79 |
| SENDER | 2.89 | 65.94 | 1.18 | 50.12 | 29.06 | 1.81 | 2.06 | 1.10 |
| SYMTRIC | 2.02 | 76.12 | 0.95 | 59.82 | 44.72 | 3.03 | 4.22 | 2.27 |
| DIFFUSE | 1.65 | 80.49 | 1.01 | 57.31 | 41.54 | 26.44 | 2.10 | 2.69 |

Load Pattern= VVVVV

*Table 5.1 Performance under Small Compute/Communicate Ratio (R= 0.13)*

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 1.79 | 0.00 | 0.72 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.58 | 11.46 | 0.61 | 15.79 | 4.66 | 5.23 | 0.00 | 0.02 |
| RECEIVR | 1.70 | 4.94 | 0.66 | 8.83 | 2.58 | 0.24 | 1.51 | 1.22 |
| SENDER | 1.58 | 11.71 | 0.61 | 15.91 | 4.57 | 0.27 | 0.04 | 0.06 |
| SYMTRIC | 1.61 | 10.11 | 0.60 | 16.91 | 5.26 | 0.24 | 1.63 | 1.30 |
| DIFFUSE | 1.61 | 9.90 | 0.62 | 13.77 | 4.26 | 1.90 | 2.38 | 2.71 |

Load Pattern= LLLLL

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 2.71 | 0.00 | 0.89 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.98 | 27.06 | 0.63 | 29.48 | 16.19 | 19.27 | 0.00 | 0.07 |
| RECEIVR | 2.13 | 21.23 | 0.67 | 24.53 | 9.73 | 0.43 | 2.02 | 2.00 |
| SENDER | 1.95 | 27.98 | 0.61 | 31.93 | 14.25 | 0.44 | 0.23 | 0.12 |
| SYMTRIC | 2.02 | 25.61 | 0.61 | 31.60 | 16.06 | 0.96 | 2.54 | 2.56 |
| DIFFUSE | 1.72 | 36.43 | 0.64 | 28.54 | 14.41 | 4.94 | 2.30 | 3.30 |

Load Pattern= MMMMM

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 6.25 | 0.00 | 1.92 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 2.90 | 53.54 | 0.93 | 51.66 | 44.32 | 50.30 | 0.00 | 0.33 |
| RECEIVR | 2.73 | 56.26 | 0.89 | 53.44 | 23.18 | 1.71 | 1.93 | 2.76 |
| SENDER | 2.75 | 55.99 | 0.90 | 52.97 | 28.42 | 2.12 | 1.01 | 1.30 |
| SYMTRIC | 2.24 | 64.11 | 0.95 | 50.55 | 37.80 | 3.98 | 3.67 | 4.77 |
| DIFFUSE | 2.03 | 67.46 | 1.08 | 43.92 | 34.20 | 16.58 | 2.19 | 4.31 |

Load Pattern= HHHHH

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 12.92 | 0.00 | 3.52 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 4.25 | 67.09 | 1.58 | 55.01 | 64.84 | 68.99 | 0.00 | 1.24 |
| RECEIVR | 3.38 | 73.83 | 1.25 | 64.37 | 27.56 | 2.63 | 1.49 | 2.47 |
| SENDER | 4.16 | 67.78 | 1.71 | 51.45 | 26.69 | 3.40 | 1.92 | 2.60 |
| SYMTRIC | 3.59 | 72.20 | 1.85 | 47.35 | 37.07 | 8.37 | 3.83 | 5.40 |
| DIFFUSE | 3.14 | 75.72 | 1.98 | 43.89 | 36.18 | 27.37 | 2.17 | 4.71 |

Load Pattern= VVVVV

*Table 5.2 Performance under Large Compute/Communicate Ratio (R= 0.4)*

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 5.47 | 0.00 | 2.46 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.63 | 70.18 | 0.56 | 77.43 | 29.77 | 28.40 | 0.00 | 0.31 |
| RECEIVR | 2.07 | 62.10 | 0.76 | 69.08 | 17.25 | 0.63 | 1.42 | 1.20 |
| SENDER | 1.55 | 71.67 | 0.52 | 79.05 | 24.14 | 0.49 | 0.39 | 0.54 |
| SYMTRIC | 1.54 | 71.90 | 0.51 | 79.38 | 25.59 | 0.55 | 2.50 | 1.50 |
| DIFFUSE | 1.30 | 76.16 | 0.54 | 77.86 | 23.83 | 7.25 | 2.22 | 2.22 |

Load Pattern= 4S, 2M, 4V

*Table 5.3 Performance under Small Compute/Communicate Ratio (R= 0.13)*

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 6.40 | 0.00 | 2.87 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 2.01 | 68.54 | 0.62 | 78.32 | 19.44 | 16.79 | 0.00 | 0.42 |
| RECEIVR | 2.48 | 61.32 | 0.81 | 71.69 | 14.01 | 0.72 | 1.60 | 2.14 |
| SENDER | 1.98 | 69.00 | 0.61 | 78.87 | 16.53 | 0.58 | 0.23 | 0.75 |
| SYMTRIC | 2.03 | 68.25 | 0.61 | 78.87 | 18.30 | 0.55 | 2.35 | 2.81 |
| DIFFUSE | 1.69 | 73.53 | 0.66 | 77.09 | 17.18 | 4.26 | 2.32 | 3.73 |

Load Pattern= 4S, 2M, 4V

*Table 5.4 Performance under Large Compute/Communicate Ratio (R= 0.4)*

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 1.74 | 0.00 | 0.77 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.41 | 18.75 | 0.60 | 22.34 | 13.95 | 16.84 | 0.00 | 0.33 |
| RECEIVR | 1.64 | 5.49 | 0.71 | 7.57 | 0.77 | 0.00 | 1.34 | 0.12 |
| SENDER | 1.34 | 22.83 | 0.55 | 28.03 | 12.18 | 0.36 | 0.12 | -0.40 |
| SYMTRIC | 1.36 | 21.75 | 0.57 | 26.46 | 12.74 | 0.20 | 1.70 | 0.32 |
| DIFFUSE | 1.20 | 31.01 | 0.76 | 1.53 | 7.78 | 12.69 | 0.64 | 2.36 |

Load Pattern= LLLLL

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 2.69 | 0.00 | 0.94 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.70 | 36.71 | 0.59 | 36.88 | 31.35 | 34.47 | 0.00 | 0.45 |
| RECEIVR | 2.03 | 24.50 | 0.73 | 22.10 | 4.25 | 0.39 | 1.56 | -0.21 |
| SENDER | 1.59 | 41.06 | 0.56 | 40.63 | 24.18 | 0.73 | 0.47 | -0.62 |
| SYMTRIC | 1.61 | 40.31 | 0.57 | 39.08 | 24.80 | 0.64 | 2.72 | 0.40 |
| DIFFUSE | 1.10 | 58.99 | 0.72 | 23.49 | 15.18 | 18.03 | 0.68 | 3.49 |

Load Pattern= MMMMM

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 7.32 | 0.00 | 2.74 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 2.66 | 63.66 | 1.09 | 60.27 | 59.27 | 62.91 | 0.00 | 0.14 |
| RECEIVR | 2.67 | 63.55 | 0.92 | 66.49 | 10.36 | 0.51 | 1.36 | -0.38 |
| SENDER | 2.22 | 69.73 | 0.83 | 69.61 | 31.67 | 1.66 | 1.43 | -0.28 |
| SYMTRIC | 2.09 | 71.44 | 0.78 | 71.71 | 33.16 | 1.63 | 3.79 | 0.86 |
| DIFFUSE | 1.52 | 79.28 | 1.32 | 51.70 | 19.71 | 26.10 | 0.70 | 2.93 |

Load Pattern= HHHHH

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 13.27 | 0.00 | 4.08 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 4.07 | 69.31 | 2.00 | 51.06 | 73.29 | 75.79 | 0.00 | 0.40 |
| RECEIVR | 3.60 | 72.90 | 1.40 | 65.71 | 12.65 | 0.93 | 1.07 | -0.03 |
| SENDER | 3.34 | 74.82 | 1.82 | 55.42 | 27.55 | 2.12 | 2.10 | 0.52 |
| SYMTRIC | 2.81 | 78.79 | 1.48 | 63.67 | 30.73 | 2.15 | 4.04 | 1.52 |
| DIFFUSE | 2.40 | 81.93 | 2.22 | 45.70 | 18.15 | 29.73 | 0.69 | 1.99 |

Load Pattern= VVVVV

*Table 5.5 95/05 Jobs Proportion with Non-selective Transfer*

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 1.74 | 0.00 | 0.77 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.74 | 0.16 | 0.78 | -1.80 | 0.83 | 20.30 | 0.00 | 0.83 |
| RECEIVR | 1.74 | 0.10 | 0.76 | 0.94 | 0.08 | 0.00 | 1.33 | 0.62 |
| SENDER | 1.70 | 2.41 | 0.74 | 4.42 | 0.77 | 0.81 | 0.01 | -0.02 |
| SYMTRIC | 1.69 | 2.79 | 0.72 | 6.82 | 0.80 | 0.00 | 1.35 | 0.57 |
| DIFFUSE | 1.57 | 10.00 | 0.74 | 3.25 | 0.70 | 19.47 | 0.50 | 1.43 |

Load Pattern= LLLLL

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 2.69 | 0.00 | 0.94 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 2.60 | 3.22 | 0.98 | -4.54 | 1.89 | 36.78 | 0.00 | 0.78 |
| RECEIVR | 2.65 | 1.33 | 0.97 | -3.45 | 0.34 | 1.23 | 1.48 | 0.74 |
| SENDER | 2.37 | 12.02 | 0.83 | 11.82 | 1.48 | 0.00 | 0.03 | 0.07 |
| SYMTRIC | 2.45 | 9.06 | 0.89 | 5.49 | 1.61 | 1.04 | 1.55 | 0.88 |
| DIFFUSE | 1.77 | 34.15 | 1.17 | -24.37 | 3.34 | 42.64 | 0.38 | 7.71 |

Load Pattern= MMMMM

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 7.32 | 0.00 | 2.74 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 5.89 | 19.58 | 2.73 | 0.43 | 3.27 | 66.16 | 0.00 | 0.85 |
| RECEIVR | 5.65 | 22.82 | 1.99 | 27.50 | 0.76 | 0.41 | 1.03 | 1.60 |
| SENDER | 4.87 | 33.44 | 1.90 | 30.52 | 1.78 | 0.70 | 0.08 | 0.78 |
| SYMTRIC | 4.75 | 35.14 | 1.67 | 38.99 | 1.95 | 0.64 | 1.21 | 1.62 |
| DIFFUSE | 3.46 | 52.78 | 2.79 | -1.76 | 4.70 | 70.41 | 0.16 | 13.05 |

Load Pattern= HHHHH

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 13.27 | 0.00 | 4.08 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 10.42 | 21.50 | 5.09 | -24.70 | 3.93 | 80.28 | 0.00 | 0.87 |
| RECEIVR | 9.22 | 30.51 | 3.70 | 9.34 | 0.81 | 0.71 | 0.70 | 1.72 |
| SENDER | 8.29 | 37.50 | 3.55 | 12.91 | 1.43 | 0.60 | 0.12 | 1.20 |
| SYMTRIC | 7.07 | 46.72 | 2.45 | 40.01 | 1.88 | 0.92 | 0.90 | 1.65 |
| DIFFUSE | 6.13 | 53.80 | 4.85 | -18.95 | 4.36 | 75.36 | 0.12 | 9.34 |

Load Pattern= VVVVV

*Table 5.6 95/05 Proportion of Jobs with Selective Transfer*

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 1.84 | 0.00 | 0.93 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.41 | 23.20 | 0.63 | 31.75 | 14.24 | 17.37 | 0.00 | -0.15 |
| RECEIVR | 1.65 | 10.14 | 0.73 | 22.03 | 0.71 | 0.00 | 1.34 | -0.11 |
| SENDER | 1.36 | 26.31 | 0.59 | 36.84 | 12.29 | 0.15 | 0.12 | -0.59 |
| SYMTRIC | 1.37 | 25.32 | 0.60 | 35.09 | 12.36 | 0.45 | 1.70 | 0.16 |
| DIFFUSE | 1.18 | 36.13 | 0.78 | 15.84 | 9.92 | 11.27 | 0.63 | 1.99 |

Load Pattern= LLLLL

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 2.95 | 0.00 | 1.30 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.73 | 41.36 | 0.67 | 48.45 | 32.00 | 35.98 | 0.00 | -0.04 |
| RECEIVR | 2.14 | 27.38 | 0.76 | 41.47 | 4.10 | 0.20 | 1.57 | -0.50 |
| SENDER | 1.56 | 47.03 | 0.57 | 56.41 | 24.37 | 0.67 | 0.46 | -0.91 |
| SYMTRIC | 1.60 | 45.82 | 0.58 | 55.41 | 25.47 | 0.80 | 2.73 | 0.20 |
| DIFFUSE | 1.14 | 61.47 | 0.83 | 36.41 | 17.50 | 18.97 | 0.67 | 3.19 |

Load Pattern= MMMMM

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 6.98 | 0.00 | 2.72 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 2.77 | 60.31 | 1.14 | 58.04 | 60.85 | 62.66 | 0.00 | 0.38 |
| RECEIVR | 2.95 | 57.73 | 0.99 | 63.56 | 11.03 | 0.57 | 1.36 | -0.42 |
| SENDER | 2.25 | 67.73 | 0.91 | 66.61 | 32.09 | 1.51 | 1.44 | -0.29 |
| SYMTRIC | 2.06 | 70.43 | 0.75 | 72.49 | 34.20 | 1.68 | 3.82 | 0.87 |
| DIFFUSE | 1.72 | 75.38 | 1.29 | 52.46 | 21.52 | 26.60 | 0.69 | 2.23 |

Load Pattern= HHHHH

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 11.94 | 0.00 | 4.18 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 4.19 | 64.88 | 1.87 | 55.26 | 74.14 | 75.50 | 0.00 | 0.51 |
| RECEIVR | 3.60 | 69.83 | 1.28 | 69.49 | 13.65 | 0.78 | 1.09 | -0.32 |
| SENDER | 3.24 | 72.84 | 1.59 | 61.93 | 28.65 | 1.74 | 2.11 | 0.30 |
| SYMTRIC | 2.65 | 77.80 | 1.21 | 70.96 | 31.79 | 2.41 | 4.07 | 1.26 |
| DIFFUSE | 2.59 | 78.34 | 1.96 | 53.01 | 18.77 | 29.16 | 0.68 | 1.33 |

Load Pattern= VVVVV

*Table 5.7 70/30 Proportion of Jobs with Non-selective Transfer*

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 1.84 | 0.00 | 0.93 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.55 | 15.91 | 0.70 | 24.74 | 4.71 | 18.15 | 0.00 | 0.47 |
| RECEIVR | 1.73 | 5.82 | 0.81 | 13.26 | 0.32 | 0.00 | 1.34 | 1.00 |
| SENDER | 1.52 | 17.26 | 0.69 | 26.34 | 4.48 | 0.28 | 0.04 | 0.42 |
| SYMTRIC | 1.52 | 17.40 | 0.66 | 28.55 | 4.31 | 0.14 | 1.47 | 1.04 |
| DIFFUSE | 1.32 | 28.06 | 0.75 | 19.73 | 2.22 | 16.85 | 0.58 | 1.83 |

Load Pattern= LLLLL

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 2.95 | 0.00 | 1.30 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 2.05 | 30.41 | 0.79 | 39.12 | 10.67 | 37.44 | 0.00 | 0.83 |
| RECEIVR | 2.32 | 21.46 | 0.90 | 30.85 | 1.38 | 0.00 | 1.53 | 0.20 |
| SENDER | 1.87 | 36.61 | 0.68 | 47.62 | 8.45 | 0.30 | 0.16 | 0.25 |
| SYMTRIC | 1.90 | 35.54 | 0.70 | 45.81 | 8.49 | 0.49 | 1.95 | 1.13 |
| DIFFUSE | 1.18 | 59.87 | 0.80 | 38.81 | 6.89 | 27.70 | 0.53 | 6.39 |

Load Pattern= MMMMM

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 6.98 | 0.00 | 2.72 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 4.08 | 41.57 | 1.81 | 33.27 | 19.56 | 65.12 | 0.00 | 1.09 |
| RECEIVR | 3.45 | 50.59 | 1.40 | 48.68 | 3.83 | 0.33 | 1.20 | 0.04 |
| SENDER | 3.16 | 54.78 | 1.47 | 45.79 | 9.91 | 0.95 | 0.47 | 0.01 |
| SYMTRIC | 2.70 | 61.32 | 1.06 | 60.91 | 10.86 | 1.04 | 2.04 | 0.88 |
| DIFFUSE | 1.74 | 75.01 | 1.69 | 37.74 | 11.47 | 42.06 | 0.45 | 7.88 |

Load Pattern= HHHHH

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 11.94 | 0.00 | 4.18 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 7.29 | 38.96 | 3.31 | 20.71 | 23.60 | 77.80 | 0.00 | 1.40 |
| RECEIVR | 5.05 | 57.74 | 2.28 | 45.56 | 4.76 | 0.60 | 0.92 | -0.20 |
| SENDER | 4.98 | 58.30 | 3.09 | 26.07 | 8.75 | 1.22 | 0.67 | 0.21 |
| SYMTRIC | 3.57 | 70.14 | 1.98 | 52.74 | 10.27 | 1.43 | 1.89 | 0.84 |
| DIFFUSE | 4.00 | 66.53 | 3.92 | 6.27 | 10.13 | 47.11 | 0.42 | 6.03 |

Load Pattern= VVVVV

*Table 5.8 70/30 Proportion of Jobs with selective Transfer*

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 1.73 | 0.00 | 0.73 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.42 | 18.12 | 0.56 | 23.19 | 14.09 | 16.91 | 0.00 | 0.31 |
| RECEIVR | 1.52 | 12.42 | 0.60 | 17.66 | 6.31 | 0.79 | 1.42 | 0.59 |
| SENDER | 1.37 | 20.83 | 0.53 | 27.27 | 12.32 | 0.81 | 0.12 | 0.09 |
| SYMTRIC | 1.37 | 20.64 | 0.54 | 26.69 | 13.04 | 0.96 | 1.71 | 0.74 |
| DIFFUSE | 1.41 | 18.56 | 0.54 | 26.09 | 11.14 | 4.20 | 2.31 | 1.28 |

Load Pattern= LLLLL

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 2.64 | 0.00 | 0.92 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.71 | 35.26 | 0.60 | 35.23 | 31.09 | 35.32 | 0.00 | 0.33 |
| RECEIVR | 1.81 | 31.41 | 0.61 | 34.05 | 16.20 | 1.47 | 1.77 | 0.70 |
| SENDER | 1.62 | 38.65 | 0.53 | 42.59 | 24.51 | 1.92 | 0.48 | 0.27 |
| SYMTRIC | 1.58 | 40.28 | 0.51 | 44.45 | 28.32 | 2.10 | 2.75 | 1.29 |
| DIFFUSE | 1.48 | 43.88 | 0.55 | 40.00 | 24.56 | 10.63 | 2.20 | 1.60 |

Load Pattern= MMMMM

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 5.69 | 0.00 | 1.75 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 2.75 | 51.68 | 1.02 | 41.75 | 60.96 | 64.28 | 0.00 | 1.12 |
| RECEIVR | 2.27 | 60.12 | 0.77 | 55.87 | 27.60 | 2.77 | 1.62 | 0.89 |
| SENDER | 2.37 | 58.34 | 0.96 | 45.19 | 31.18 | 4.08 | 1.50 | 0.99 |
| SYMTRIC | 1.88 | 66.97 | 0.82 | 53.21 | 44.18 | 5.18 | 3.85 | 2.18 |
| DIFFUSE | 1.61 | 71.70 | 0.84 | 51.79 | 40.22 | 21.89 | 2.14 | 2.27 |

Load Pattern= HHHHH

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 10.72 | 0.00 | 3.26 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 4.39 | 59.09 | 1.85 | 43.35 | 76.67 | 78.55 | 0.00 | 1.69 |
| RECEIVR | 2.91 | 72.88 | 1.14 | 64.91 | 28.98 | 3.81 | 1.30 | 1.08 |
| SENDER | 3.49 | 67.43 | 1.55 | 52.58 | 26.01 | 5.12 | 2.23 | 1.34 |
| SYMTRIC | 2.50 | 76.72 | 1.23 | 62.19 | 42.82 | 7.42 | 4.08 | 2.46 |
| DIFFUSE | 2.15 | 79.95 | 1.21 | 62.82 | 41.03 | 29.43 | 2.16 | 2.43 |

Load Pattern= VVVVV

*Table 5.9 Performance under Small Compute/Communicate Ratio (R= 0.13)*

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 1.73 | 0.00 | 0.73 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.57 | 9.38 | 0.61 | 16.01 | 4.99 | 5.12 | 0.00 | 0.33 |
| RECEIVR | 1.61 | 6.98 | 0.62 | 14.63 | 2.51 | 0.25 | 1.53 | 0.64 |
| SENDER | 1.53 | 11.45 | 0.60 | 17.80 | 4.57 | 0.55 | 0.04 | 0.06 |
| SYMTRIC | 1.55 | 10.58 | 0.61 | 16.98 | 4.86 | 0.90 | 1.64 | 0.75 |
| DIFFUSE | 1.59 | 7.89 | 0.60 | 17.93 | 3.97 | 0.94 | 2.41 | 1.34 |

Load Pattern= LLLLL

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 2.64 | 0.00 | 0.92 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.93 | 26.71 | 0.63 | 31.76 | 15.47 | 19.85 | 0.00 | 0.45 |
| RECEIVR | 2.10 | 20.52 | 0.68 | 25.61 | 9.59 | 1.17 | 2.05 | 0.98 |
| SENDER | 1.90 | 28.11 | 0.62 | 32.88 | 14.31 | 1.43 | 0.22 | 0.31 |
| SYMTRIC | 1.91 | 27.49 | 0.62 | 32.72 | 15.53 | 2.07 | 2.57 | 1.37 |
| DIFFUSE | 1.89 | 28.49 | 0.64 | 30.70 | 14.17 | 5.73 | 2.34 | 1.84 |

Load Pattern= MMMMM

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 5.69 | 0.00 | 1.75 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 2.94 | 48.37 | 1.06 | 39.65 | 44.37 | 50.79 | 0.00 | 1.84 |
| RECEIVR | 2.85 | 49.96 | 0.88 | 49.93 | 22.91 | 2.72 | 2.03 | 1.60 |
| SENDER | 2.61 | 54.15 | 0.89 | 48.99 | 27.28 | 3.76 | 0.94 | 1.27 |
| SYMTRIC | 2.35 | 58.78 | 0.93 | 46.72 | 36.55 | 6.00 | 3.55 | 3.14 |
| DIFFUSE | 2.07 | 63.56 | 0.89 | 49.20 | 32.93 | 16.72 | 2.24 | 3.01 |

Load Pattern= HHHHH

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 10.72 | 0.00 | 3.26 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 4.84 | 54.82 | 2.09 | 36.01 | 68.26 | 72.13 | 0.00 | 3.38 |
| RECEIVR | 3.52 | 67.16 | 1.27 | 60.90 | 27.27 | 3.88 | 1.66 | 1.90 |
| SENDER | 3.48 | 67.58 | 1.43 | 56.23 | 29.07 | 5.03 | 1.67 | 1.96 |
| SYMTRIC | 3.04 | 71.67 | 1.46 | 55.25 | 41.08 | 8.82 | 3.99 | 3.84 |
| DIFFUSE | 2.68 | 74.95 | 1.41 | 56.74 | 39.28 | 26.82 | 2.25 | 3.74 |

Load Pattern= VVVVV

*Table 5.10 Performance under Large Compute/Communicate Ratio (R= 0.4)*

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 6.10 | 0.00 | 2.66 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.69 | 72.33 | 0.61 | 77.07 | 32.49 | 29.56 | 0.00 | 0.45 |
| RECEIVR | 2.11 | 65.39 | 0.78 | 70.77 | 18.38 | 1.75 | 1.40 | 1.28 |
| SENDER | 1.59 | 73.93 | 0.52 | 80.33 | 25.24 | 1.72 | 0.41 | 0.84 |
| SYMTRIC | 1.52 | 75.15 | 0.51 | 80.66 | 27.45 | 2.08 | 2.51 | 1.66 |
| DIFFUSE | 1.48 | 75.82 | 0.53 | 79.93 | 24.51 | 8.77 | 2.22 | 1.87 |

Load Pattern= 4S, 2M, 4V

*Table 5.11 Performance under Small Compute/Communicate Ratio (R= 0.13)*

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 6.10 | 0.00 | 2.66 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 2.01 | 67.03 | 0.66 | 75.32 | 20.20 | 17.53 | 0.00 | 0.64 |
| RECEIVR | 2.45 | 59.87 | 0.82 | 69.32 | 14.62 | 1.55 | 1.64 | 1.62 |
| SENDER | 1.95 | 68.11 | 0.62 | 76.71 | 17.68 | 1.26 | 0.24 | 0.97 |
| SYMTRIC | 1.95 | 68.00 | 0.62 | 76.82 | 18.43 | 1.45 | 2.39 | 2.00 |
| DIFFUSE | 2.00 | 67.28 | 0.64 | 75.96 | 17.16 | 4.16 | 2.35 | 2.15 |

Load Pattern= 4S, 2M, 4V

*Table 5.12 Performance under Large Compute/Communicate Ratio (R= 0.4)*

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 1.77 | 0.00 | 0.80 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.42 | 19.66 | 0.58 | 27.11 | 14.04 | 16.39 | 0.00 | 0.28 |
| RECEIVR | 1.65 | 6.93 | 0.75 | 6.64 | 0.90 | 0.69 | 1.35 | -0.16 |
| SENDER | 1.36 | 23.43 | 0.55 | 30.97 | 12.27 | 1.07 | 0.12 | -0.63 |
| SYMTRIC | 1.37 | 22.61 | 0.56 | 29.83 | 12.43 | 1.15 | 1.70 | 0.00 |
| DIFFUSE | 1.22 | 31.32 | 0.77 | 4.19 | 7.83 | 13.23 | 0.64 | 1.43 |

Load Pattern= LLLLL

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 2.95 | 0.00 | 1.21 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.76 | 40.39 | 0.65 | 46.63 | 32.86 | 36.38 | 0.00 | 0.29 |
| RECEIVR | 2.12 | 27.99 | 0.74 | 38.59 | 4.32 | 1.25 | 1.58 | -0.79 |
| SENDER | 1.60 | 45.62 | 0.56 | 53.57 | 25.15 | 2.12 | 0.48 | -0.84 |
| SYMTRIC | 1.62 | 45.12 | 0.57 | 53.23 | 24.89 | 1.84 | 2.73 | 0.08 |
| DIFFUSE | 1.15 | 61.06 | 0.77 | 36.53 | 14.21 | 18.74 | 0.68 | 2.28 |

Load Pattern= MMMMM

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 8.18 | 0.00 | 2.65 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 2.92 | 64.34 | 1.24 | 53.09 | 61.88 | 65.60 | 0.00 | 1.01 |
| RECEIVR | 3.05 | 62.75 | 1.21 | 54.52 | 10.76 | 2.09 | 1.32 | -0.62 |
| SENDER | 2.39 | 70.79 | 1.03 | 61.01 | 31.40 | 4.05 | 1.48 | -0.05 |
| SYMTRIC | 2.16 | 73.63 | 1.00 | 62.34 | 32.86 | 4.07 | 3.77 | 0.93 |
| DIFFUSE | 1.59 | 80.56 | 1.22 | 53.88 | 19.78 | 27.62 | 0.70 | 2.17 |

Load Pattern= HHHHH

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 17.52 | 0.00 | 4.54 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 4.98 | 71.57 | 2.49 | 45.12 | 77.63 | 79.56 | 0.00 | 2.09 |
| RECEIVR | 3.64 | 79.23 | 1.45 | 68.04 | 13.43 | 2.82 | 1.04 | 0.21 |
| SENDER | 3.60 | 79.42 | 1.88 | 58.68 | 26.85 | 4.95 | 2.21 | 1.04 |
| SYMTRIC | 2.83 | 83.83 | 1.25 | 72.54 | 31.48 | 5.69 | 4.04 | 1.94 |
| DIFFUSE | 2.74 | 84.38 | 2.12 | 53.36 | 18.38 | 31.29 | 0.69 | 2.04 |

Load Pattern= VVVVV

*Table 5.13 95/05 Jobs Proportion with Non-selective Transfer*

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 1.77 | 0.00 | 0.80 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.77 | 0.17 | 0.77 | 3.22 | 0.98 | 18.47 | 0.00 | 0.63 |
| RECEIVR | 1.84 | -3.88 | 0.78 | 2.93 | 0.04 | 0.00 | 1.31 | 1.54 |
| SENDER | 1.73 | 2.23 | 0.73 | 8.57 | 0.70 | 0.88 | 0.01 | 0.84 |
| SYMTRIC | 1.74 | 1.60 | 0.75 | 6.57 | 0.72 | 0.86 | 1.34 | 1.44 |
| DIFFUSE | 1.62 | 8.35 | 0.77 | 4.16 | 0.54 | 17.24 | 0.50 | 1.62 |

Load Pattern= LLLLL

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 2.95 | 0.00 | 1.21 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 2.68 | 9.02 | 1.19 | 1.35 | 1.97 | 38.56 | 0.00 | 0.59 |
| RECEIVR | 2.78 | 5.85 | 1.08 | 10.54 | 0.29 | 1.45 | 1.44 | 0.99 |
| SENDER | 2.53 | 14.20 | 0.96 | 20.72 | 1.47 | 0.85 | 0.03 | 0.49 |
| SYMTRIC | 2.60 | 11.99 | 1.03 | 14.69 | 1.55 | 0.00 | 1.53 | 1.10 |
| DIFFUSE | 1.77 | 39.91 | 1.24 | -2.11 | 4.21 | 49.56 | 0.36 | 9.53 |

Load Pattern= MMMMM

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 8.18 | 0.00 | 2.65 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 6.46 | 21.06 | 2.73 | -3.11 | 3.44 | 71.03 | 0.00 | 0.89 |
| RECEIVR | 6.30 | 22.97 | 2.50 | 5.83 | 0.67 | 1.87 | 0.99 | 1.19 |
| SENDER | 6.06 | 25.86 | 3.40 | -28.15 | 1.67 | 1.69 | 0.09 | 0.72 |
| SYMTRIC | 5.12 | 37.39 | 2.35 | 11.32 | 1.96 | 2.23 | 1.16 | 1.11 |
| DIFFUSE | 4.07 | 50.21 | 3.45 | -30.14 | 4.67 | 72.72 | 0.14 | 13.39 |

Load Pattern= HHHHH

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 17.52 | 0.00 | 4.54 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 11.76 | 32.86 | 4.90 | -7.83 | 4.07 | 81.94 | 0.00 | 1.47 |
| RECEIVR | 10.93 | 37.60 | 5.31 | -16.92 | 0.75 | 2.68 | 0.64 | 1.69 |
| SENDER | 9.74 | 44.40 | 5.07 | -11.68 | 1.29 | 1.56 | 0.12 | 1.15 |
| SYMTRIC | 8.19 | 53.24 | 4.14 | 8.82 | 1.72 | 1.84 | 0.87 | 1.61 |
| DIFFUSE | 7.54 | 56.98 | 6.50 | -43.16 | 3.98 | 75.61 | 0.11 | 9.09 |

Load Pattern= VVVVV

*Table 5.14 95/05 Proportion of Jobs with Selective Transfer*

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 1.86 | 0.00 | 0.90 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.41 | 24.32 | 0.62 | 31.15 | 14.75 | 14.25 | 0.00 | -0.46 |
| RECEIVR | 1.64 | 12.07 | 0.75 | 16.33 | 1.15 | 1.08 | 1.34 | -0.65 |
| SENDER | 1.36 | 26.89 | 0.61 | 32.69 | 13.05 | 0.76 | 0.13 | -1.00 |
| SYMTRIC | 1.37 | 26.38 | 0.62 | 31.15 | 12.68 | 0.88 | 1.70 | -0.36 |
| DIFFUSE | 1.17 | 37.20 | 0.79 | 12.70 | 10.31 | 11.19 | 0.63 | 1.98 |

Load Pattern= LLLLL

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 3.00 | 0.00 | 1.22 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.75 | 41.71 | 0.66 | 45.82 | 32.30 | 35.37 | 0.00 | 0.61 |
| RECEIVR | 2.22 | 25.98 | 0.80 | 34.61 | 4.58 | 1.36 | 1.57 | -0.10 |
| SENDER | 1.62 | 46.15 | 0.62 | 49.02 | 24.86 | 1.88 | 0.49 | -0.20 |
| SYMTRIC | 1.64 | 45.45 | 0.61 | 50.35 | 24.89 | 1.96 | 2.72 | 0.75 |
| DIFFUSE | 1.19 | 60.42 | 0.84 | 31.16 | 18.19 | 19.95 | 0.67 | 3.56 |

Load Pattern= MMMMM

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 8.53 | 0.00 | 3.06 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 3.27 | 61.72 | 1.47 | 51.97 | 64.63 | 67.65 | 0.00 | 1.21 |
| RECEIVR | 2.76 | 67.60 | 1.10 | 64.06 | 11.35 | 2.09 | 1.31 | -0.43 |
| SENDER | 2.40 | 71.87 | 0.99 | 67.49 | 31.72 | 3.93 | 1.51 | 0.12 |
| SYMTRIC | 2.05 | 75.92 | 0.84 | 72.45 | 33.24 | 4.36 | 3.78 | 1.06 |
| DIFFUSE | 2.03 | 76.18 | 1.48 | 51.58 | 21.69 | 26.60 | 0.69 | 1.99 |

Load Pattern= HHHHH

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 15.13 | 0.00 | 5.30 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 5.33 | 64.74 | 2.67 | 49.58 | 78.56 | 80.32 | 0.00 | 1.55 |
| RECEIVR | 3.47 | 77.09 | 1.42 | 73.18 | 13.83 | 2.59 | 1.01 | -0.02 |
| SENDER | 3.74 | 75.30 | 1.90 | 64.10 | 26.48 | 5.10 | 2.24 | 0.68 |
| SYMTRIC | 3.14 | 79.25 | 1.45 | 72.63 | 31.03 | 5.87 | 4.03 | 1.66 |
| DIFFUSE | 3.16 | 79.10 | 2.22 | 58.11 | 18.16 | 29.46 | 0.68 | 1.17 |

Load Pattern= VVVVV

*Table 5.15 70/30 Proportion of Jobs with Non-selective Transfer*

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 1.86 | 0.00 | 0.90 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.55 | 16.59 | 0.69 | 23.53 | 4.65 | 16.38 | 0.00 | 0.25 |
| RECEIVR | 1.75 | 5.73 | 0.78 | 13.22 | 0.35 | 0.00 | 1.33 | 0.56 |
| SENDER | 1.55 | 16.87 | 0.71 | 21.56 | 4.53 | 0.96 | 0.04 | 0.13 |
| SYMTRIC | 1.54 | 17.46 | 0.68 | 24.67 | 4.46 | 0.56 | 1.47 | 0.69 |
| DIFFUSE | 1.35 | 27.45 | 0.74 | 18.09 | 2.19 | 12.22 | 0.53 | 1.95 |

Load Pattern= LLLLL

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 3.00 | 0.00 | 1.22 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 2.08 | 30.77 | 0.80 | 34.56 | 10.76 | 36.61 | 0.00 | 1.21 |
| RECEIVR | 2.37 | 21.15 | 0.95 | 22.19 | 1.90 | 1.10 | 1.51 | 0.77 |
| SENDER | 1.99 | 33.74 | 0.77 | 36.92 | 8.60 | 1.31 | 0.17 | 0.76 |
| SYMTRIC | 2.13 | 29.12 | 0.76 | 37.99 | 9.10 | 1.70 | 1.89 | 1.46 |
| DIFFUSE | 1.29 | 56.99 | 0.82 | 33.09 | 6.09 | 26.57 | 0.49 | 5.85 |

Load Pattern= MMMMM

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 8.53 | 0.00 | 3.06 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 4.83 | 43.33 | 2.57 | 16.07 | 20.32 | 68.46 | 0.00 | 1.46 |
| RECEIVR | 3.53 | 58.60 | 1.56 | 49.01 | 4.26 | 1.91 | 1.15 | 0.27 |
| SENDER | 3.37 | 60.52 | 1.44 | 53.04 | 10.14 | 2.87 | 0.49 | 0.37 |
| SYMTRIC | 2.60 | 69.47 | 1.29 | 57.89 | 10.94 | 2.72 | 1.99 | 0.87 |
| DIFFUSE | 1.94 | 77.24 | 1.53 | 49.97 | 11.16 | 41.91 | 0.41 | 7.83 |

Load Pattern= HHHHH

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 15.13 | 0.00 | 5.30 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 10.92 | 27.83 | 5.37 | -1.33 | 24.70 | 82.08 | 0.00 | 1.88 |
| RECEIVR | 5.42 | 64.18 | 2.98 | 43.80 | 4.86 | 2.96 | 0.81 | 0.42 |
| SENDER | 5.78 | 61.77 | 2.84 | 46.32 | 8.03 | 3.12 | 0.71 | 0.99 |
| SYMTRIC | 4.07 | 73.07 | 1.95 | 63.15 | 10.23 | 3.43 | 1.80 | 1.57 |
| DIFFUSE | 4.29 | 71.65 | 3.41 | 35.72 | 9.44 | 49.29 | 0.38 | 5.88 |

Load Pattern= VVVVV

*Table 5.16 70/30 Proportion of Jobs with selective Transfer*

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 1.26 | 0.00 | 0.49 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.14 | 9.91 | 0.42 | 13.76 | 5.40 | 6.47 | 0.00 | 0.39 |
| RECEIVR | 1.18 | 6.46 | 0.43 | 11.35 | 2.51 | 0.33 | 2.23 | 1.66 |
| SENDER | 1.12 | 11.09 | 0.41 | 17.13 | 4.99 | 0.17 | 0.07 | -0.08 |
| SYMTRIC | 1.15 | 8.34 | 0.42 | 14.29 | 5.37 | 0.47 | 2.40 | 2.17 |
| DIFFUSE | 1.17 | 6.81 | 0.43 | 11.81 | 4.80 | 2.95 | 2.39 | 3.22 |

Load Pattern= LLLLL

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 2.04 | 0.00 | 0.68 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.47 | 27.74 | 0.43 | 36.30 | 18.15 | 23.24 | 0.00 | 1.14 |
| RECEIVR | 1.53 | 25.15 | 0.48 | 28.84 | 11.07 | 1.63 | 2.50 | 4.07 |
| SENDER | 1.47 | 27.89 | 0.45 | 33.94 | 16.04 | 1.31 | 0.41 | 1.47 |
| SYMTRIC | 1.38 | 32.43 | 0.48 | 29.18 | 19.49 | 2.24 | 2.95 | 6.41 |
| DIFFUSE | 1.30 | 36.51 | 0.50 | 27.08 | 15.61 | 8.72 | 2.34 | 5.18 |

Load Pattern= MMMMM

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 8.06 | 0.00 | 2.63 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 5.00 | 38.01 | 2.79 | -6.18 | 62.27 | 69.54 | 0.00 | 3.55 |
| RECEIVR | 2.28 | 71.69 | 0.88 | 66.64 | 25.49 | 4.62 | 1.88 | 5.15 |
| SENDER | 4.10 | 49.14 | 1.89 | 28.08 | 21.84 | 7.32 | 2.50 | 7.21 |
| SYMTRIC | 3.88 | 51.91 | 2.14 | 18.80 | 29.02 | 16.69 | 3.59 | 10.48 |
| DIFFUSE | 2.54 | 68.43 | 1.63 | 37.87 | 31.48 | 30.99 | 2.28 | 7.58 |

Load Pattern= HHHHH

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 42.85 | 0.00 | 16.81 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 99.78 | -132.85 | 60.27 | -258.55 | 82.47 | 83.94 | 0.00 | 0.35 |
| RECEIVR | 11.66 | 72.79 | 6.54 | 61.10 | 10.96 | 7.49 | 0.28 | 3.43 |
| SENDER | 131.50 | -206.88 | 77.77 | -362.67 | 1.25 | 12.36 | 2.50 | 4.04 |
| SYMTRIC | 132.30 | -208.75 | 77.74 | -362.44 | 1.36 | 22.04 | 2.52 | 4.09 |
| DIFFUSE | 83.43 | -94.71 | 43.73 | -160.12 | 2.30 | 49.65 | 2.35 | 3.81 |

Load Pattern= VVVVV

*Table 5.17 Performance of Standard Algorithms under Scaled Arrivals*

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 1.26 | 0.00 | 0.49 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.13 | 9.96 | 0.42 | 14.11 | 5.39 | 5.79 | 0.00 | 0.22 |
| RECEIVR | 1.20 | 5.15 | 0.45 | 8.86 | 2.28 | 0.37 | 2.21 | 1.56 |
| SENDER | 1.12 | 11.34 | 0.41 | 15.84 | 4.94 | 0.17 | 0.07 | -0.36 |
| SYMTRIC | 1.14 | 9.33 | 0.41 | 15.46 | 5.37 | 0.39 | 2.41 | 1.97 |
| DIFFUSE | 1.16 | 7.78 | 0.44 | 10.17 | 4.64 | 2.15 | 2.39 | 3.18 |

Load Pattern= LLLLL

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 2.04 | 0.00 | 0.68 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.45 | 29.11 | 0.43 | 37.07 | 18.58 | 22.32 | 0.00 | 0.29 |
| RECEIVR | 1.61 | 21.18 | 0.48 | 29.25 | 9.67 | 1.38 | 2.50 | 3.93 |
| SENDER | 1.45 | 29.15 | 0.43 | 36.60 | 15.74 | 1.46 | 0.41 | 0.90 |
| SYMTRIC | 1.42 | 30.41 | 0.47 | 31.37 | 18.92 | 2.11 | 2.95 | 5.56 |
| DIFFUSE | 1.28 | 37.30 | 0.48 | 29.46 | 15.70 | 8.24 | 2.35 | 5.03 |

Load Pattern= MMMMM

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 8.06 | 0.00 | 2.63 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 2.92 | 63.77 | 1.14 | 56.79 | 62.18 | 66.93 | 0.00 | 2.29 |
| RECEIVR | 2.26 | 72.00 | 0.85 | 67.57 | 23.24 | 4.77 | 1.91 | 4.87 |
| SENDER | 4.29 | 46.82 | 1.93 | 26.52 | 20.89 | 7.33 | 2.50 | 7.03 |
| SYMTRIC | 4.68 | 41.95 | 2.43 | 7.73 | 26.49 | 14.69 | 3.52 | 10.40 |
| DIFFUSE | 2.52 | 68.68 | 1.66 | 36.77 | 30.38 | 29.45 | 2.27 | 7.49 |

Load Pattern= HHHHH

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 42.85 | 0.00 | 16.81 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 29.29 | 31.64 | 14.42 | 14.22 | 98.00 | 97.03 | 0.00 | 3.57 |
| RECEIVR | 12.32 | 71.26 | 6.52 | 61.22 | 11.05 | 6.04 | 0.32 | 3.34 |
| SENDER | 138.60 | -223.45 | 81.55 | -385.11 | 1.60 | 8.69 | 2.50 | 4.00 |
| SYMTRIC | 138.60 | -223.45 | 80.51 | -378.96 | 2.12 | 17.37 | 2.53 | 4.11 |
| DIFFUSE | 82.85 | -93.35 | 44.65 | -165.63 | 2.65 | 47.40 | 2.34 | 3.84 |

Load Pattern= VVVVV

*Table 5.18 Performance of Adapted Algorithms under Scaled Arrivals*

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 1.15 | 0.00 | 0.48 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.05 | 8.46 | 0.42 | 11.84 | 5.12 | 6.97 | 0.00 | 0.49 |
| RECEIVR | 1.09 | 5.24 | 0.45 | 6.93 | 2.41 | 0.69 | 2.27 | 0.56 |
| SENDER | 1.03 | 10.01 | 0.43 | 11.02 | 4.60 | 0.55 | 0.06 | 0.05 |
| SYMTRIC | 1.05 | 9.06 | 0.41 | 13.92 | 4.88 | 1.20 | 2.43 | 1.10 |
| DIFFUSE | 1.06 | 7.52 | 0.41 | 13.81 | 3.75 | 2.11 | 2.44 | 1.22 |

Load Pattern= LLLLL

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 1.76 | 0.00 | 0.63 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.34 | 23.88 | 0.43 | 31.18 | 16.11 | 21.37 | 0.00 | 1.49 |
| RECEIVR | 1.41 | 20.05 | 0.45 | 28.85 | 9.39 | 1.80 | 2.50 | 1.72 |
| SENDER | 1.31 | 25.48 | 0.42 | 32.70 | 14.48 | 2.08 | 0.34 | 1.58 |
| SYMTRIC | 1.29 | 26.47 | 0.44 | 30.87 | 16.06 | 2.35 | 2.85 | 3.24 |
| DIFFUSE | 1.29 | 26.78 | 0.46 | 27.50 | 13.00 | 6.80 | 2.42 | 2.64 |

Load Pattern= MMMMM

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 3.84 | 0.00 | 1.28 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 2.96 | 23.04 | 1.56 | -21.57 | 50.49 | 60.27 | 0.00 | 5.07 |
| RECEIVR | 1.98 | 48.40 | 0.60 | 52.80 | 22.55 | 4.14 | 2.50 | 2.81 |
| SENDER | 1.93 | 49.85 | 0.65 | 49.48 | 28.00 | 5.27 | 1.56 | 3.55 |
| SYMTRIC | 1.69 | 56.11 | 0.72 | 43.45 | 38.65 | 8.68 | 4.36 | 6.56 |
| DIFFUSE | 1.59 | 58.71 | 0.72 | 43.76 | 30.33 | 20.59 | 2.42 | 4.57 |

Load Pattern= HHHHH

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 6.91 | 0.00 | 2.21 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 35.63 | -415.63 | 20.71 | -837.31 | 73.24 | 79.55 | 0.00 | 5.23 |
| RECEIVR | 2.47 | 64.29 | 0.79 | 64.31 | 27.02 | 5.99 | 2.18 | 3.00 |
| SENDER | 2.91 | 57.95 | 1.22 | 44.76 | 26.23 | 7.68 | 2.50 | 4.18 |
| SYMTRIC | 2.35 | 66.02 | 1.04 | 52.73 | 40.67 | 13.13 | 4.75 | 6.77 |
| DIFFUSE | 2.21 | 68.00 | 1.19 | 46.14 | 32.98 | 28.99 | 2.42 | 4.65 |

Load Pattern= VVVVV

*Table 5.19 Performance of Standard Algorithms under Scaled Arrivals*

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 1.15 | 0.00 | 0.48 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.05 | 8.86 | 0.41 | 13.72 | 5.17 | 6.35 | 0.00 | 0.25 |
| RECEIVR | 1.08 | 5.91 | 0.44 | 8.83 | 2.10 | 0.20 | 2.27 | 0.55 |
| SENDER | 1.03 | 10.44 | 0.42 | 12.90 | 4.40 | 0.48 | 0.06 | -0.21 |
| SYMTRIC | 1.04 | 9.90 | 0.41 | 13.89 | 4.93 | 1.02 | 2.43 | 0.85 |
| DIFFUSE | 1.07 | 6.94 | 0.42 | 12.17 | 3.68 | 0.90 | 2.44 | 1.13 |

Load Pattern= LLLLL

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 1.76 | 0.00 | 0.63 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.31 | 25.80 | 0.43 | 32.12 | 16.15 | 19.45 | 0.00 | 0.91 |
| RECEIVR | 1.41 | 20.13 | 0.46 | 26.90 | 8.27 | 1.37 | 2.50 | 1.58 |
| SENDER | 1.28 | 27.10 | 0.42 | 33.70 | 14.34 | 1.82 | 0.34 | 1.03 |
| SYMTRIC | 1.32 | 25.25 | 0.42 | 32.65 | 16.19 | 2.34 | 2.85 | 2.70 |
| DIFFUSE | 1.33 | 24.56 | 0.44 | 29.45 | 12.55 | 5.66 | 2.42 | 2.40 |

Load Pattern= MMMMM

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 3.84 | 0.00 | 1.28 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 2.25 | 41.48 | 0.92 | 27.88 | 49.35 | 55.59 | 0.00 | 3.63 |
| RECEIVR | 2.00 | 47.93 | 0.60 | 52.77 | 20.06 | 3.58 | 2.50 | 2.42 |
| SENDER | 2.07 | 46.09 | 0.67 | 47.33 | 29.49 | 6.25 | 1.77 | 3.24 |
| SYMTRIC | 1.59 | 58.54 | 0.65 | 48.88 | 37.34 | 7.87 | 4.23 | 5.76 |
| DIFFUSE | 1.70 | 55.76 | 0.72 | 43.45 | 28.64 | 18.86 | 2.42 | 3.89 |

Load Pattern= HHHHH

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 6.91 | 0.00 | 2.21 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 7.22 | -4.54 | 4.33 | -96.02 | 82.74 | 85.56 | 0.00 | 6.69 |
| RECEIVR | 2.55 | 63.11 | 0.85 | 61.61 | 24.15 | 5.44 | 2.19 | 2.52 |
| SENDER | 2.70 | 60.96 | 1.04 | 53.00 | 27.59 | 7.52 | 2.50 | 3.63 |
| SYMTRIC | 2.44 | 64.67 | 0.99 | 55.11 | 39.31 | 12.73 | 4.90 | 6.33 |
| DIFFUSE | 2.22 | 67.86 | 1.18 | 46.58 | 31.31 | 26.91 | 2.42 | 4.20 |

Load Pattern= VVVVV

*Table 5.20 Performance of Adapted Algorithms under Scaled Arrivals*

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 1.26 | 0.00 | 0.59 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.16 | 7.96 | 0.51 | 13.59 | 2.89 | 3.90 | 0.00 | -0.08 |
| RECEIVR | 1.19 | 5.42 | 0.53 | 9.79 | 1.35 | 0.00 | 1.55 | 0.85 |
| SENDER | 1.12 | 11.01 | 0.50 | 15.69 | 2.57 | 0.24 | 0.02 | -0.42 |
| SYMTRIC | 1.14 | 9.23 | 0.49 | 17.23 | 2.46 | 0.00 | 1.61 | 0.81 |
| DIFFUSE | 1.17 | 7.01 | 0.51 | 14.06 | 2.14 | 0.58 | 2.41 | 2.23 |

Node Arrival Rate: 0.4 jobs/sec

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 1.79 | 0.00 | 0.71 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.32 | 26.40 | 0.46 | 35.55 | 8.12 | 8.52 | 0.00 | -0.92 |
| RECEIVR | 1.45 | 19.18 | 0.52 | 26.36 | 5.02 | 0.25 | 2.17 | 1.27 |
| SENDER | 1.31 | 26.76 | 0.46 | 34.89 | 7.67 | 0.27 | 0.11 | -1.00 |
| SYMTRIC | 1.35 | 24.65 | 0.47 | 33.45 | 7.98 | 0.37 | 2.44 | 1.31 |
| DIFFUSE | 1.33 | 25.68 | 0.49 | 30.37 | 7.07 | 1.88 | 2.36 | 2.10 |

Node Arrival Rate: 0.6 jobs/sec

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 3.48 | 0.00 | 1.40 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.55 | 55.57 | 0.46 | 67.13 | 17.04 | 20.80 | 0.00 | -1.67 |
| RECEIVR | 1.63 | 53.26 | 0.52 | 62.83 | 10.86 | 0.84 | 2.50 | 1.37 |
| SENDER | 1.51 | 56.47 | 0.45 | 67.85 | 14.70 | 0.85 | 0.32 | -1.74 |
| SYMTRIC | 1.43 | 58.89 | 0.46 | 67.24 | 16.35 | 1.28 | 2.82 | 1.92 |
| DIFFUSE | 1.34 | 61.40 | 0.48 | 65.91 | 14.94 | 6.02 | 2.32 | 1.95 |

Node Arrival Rate: 0.8 jobs/sec

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 5.79 | 0.00 | 2.35 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.66 | 71.35 | 0.46 | 80.31 | 22.05 | 26.99 | 0.00 | -2.05 |
| RECEIVR | 1.73 | 70.06 | 0.50 | 78.92 | 13.80 | 1.50 | 2.50 | 1.25 |
| SENDER | 1.61 | 72.22 | 0.45 | 80.70 | 17.59 | 1.47 | 0.44 | -2.02 |
| SYMTRIC | 1.38 | 76.25 | 0.46 | 80.42 | 20.22 | 1.92 | 2.96 | 2.14 |
| DIFFUSE | 1.28 | 77.85 | 0.47 | 80.05 | 17.39 | 7.42 | 2.30 | 1.67 |

Node Arrival Rate: 0.9 jobs/sec

*Table 5.21 Performance of Standard Algorithms under Identical Arrivals*

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 1.26 | 0.00 | 0.59 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.14 | 9.64 | 0.48 | 17.95 | 2.62 | 0.72 | 0.00 | -0.15 |
| RECEIVR | 1.19 | 5.60 | 0.53 | 10.50 | 1.47 | 0.00 | 1.55 | 0.84 |
| SENDER | 1.12 | 11.38 | 0.49 | 16.48 | 2.29 | 0.00 | 0.02 | -0.44 |
| SYMTRIC | 1.13 | 10.22 | 0.50 | 15.77 | 2.51 | 0.00 | 1.61 | 0.73 |
| DIFFUSE | 1.16 | 7.59 | 0.50 | 14.59 | 2.26 | 0.55 | 2.41 | 2.30 |

Node Arrival Rate: 0.4 jobs/sec

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 1.79 | 0.00 | 0.71 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.31 | 27.06 | 0.45 | 36.76 | 7.93 | 8.04 | 0.00 | -1.17 |
| RECEIVR | 1.43 | 20.04 | 0.51 | 28.44 | 5.16 | 0.40 | 2.18 | 1.29 |
| SENDER | 1.29 | 27.82 | 0.46 | 35.83 | 7.04 | 0.18 | 0.10 | -1.27 |
| SYMTRIC | 1.32 | 26.21 | 0.46 | 34.95 | 7.70 | 0.27 | 2.44 | 1.00 |
| DIFFUSE | 1.27 | 29.23 | 0.47 | 33.90 | 7.36 | 2.48 | 2.37 | 2.04 |

Node Arrival Rate: 0.6 jobs/sec

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 3.48 | 0.00 | 1.40 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.49 | 57.07 | 0.43 | 69.13 | 15.83 | 15.51 | 0.00 | -2.11 |
| RECEIVR | 1.68 | 51.61 | 0.47 | 66.26 | 11.22 | 1.03 | 2.50 | 1.26 |
| SENDER | 1.48 | 57.53 | 0.43 | 69.36 | 13.63 | 0.50 | 0.27 | -2.16 |
| SYMTRIC | 1.28 | 63.09 | 0.44 | 68.32 | 15.52 | 1.05 | 2.79 | 1.55 |
| DIFFUSE | 1.25 | 64.20 | 0.46 | 66.90 | 14.22 | 7.15 | 2.32 | 1.80 |

Node Arrival Rate: 0.8 jobs/sec

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 5.79 | 0.00 | 2.35 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.59 | 72.52 | 0.45 | 81.03 | 19.84 | 22.92 | 0.00 | -2.75 |
| RECEIVR | 1.72 | 70.29 | 0.50 | 78.58 | 14.09 | 1.37 | 2.50 | 1.25 |
| SENDER | 1.56 | 73.00 | 0.43 | 81.70 | 16.39 | 1.23 | 0.39 | -2.53 |
| SYMTRIC | 1.39 | 76.03 | 0.46 | 80.45 | 21.33 | 1.67 | 2.96 | 1.39 |
| DIFFUSE | 1.23 | 78.78 | 0.48 | 79.51 | 17.28 | 8.56 | 2.30 | 1.58 |

Node Arrival Rate: 0.9 jobs/sec

*Table 5.22 Performance of Adapted Algorithms under Identical Arrivals*

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 1.19 | 0.00 | 0.59 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.09 | 8.58 | 0.51 | 13.72 | 2.53 | 2.96 | 0.00 | -0.03 |
| RECEIVR | 1.11 | 7.06 | 0.53 | 10.68 | 1.18 | 0.00 | 1.57 | 0.44 |
| SENDER | 1.07 | 10.49 | 0.50 | 15.55 | 2.19 | 0.00 | 0.02 | -0.24 |
| SYMTRIC | 1.07 | 10.13 | 0.50 | 15.56 | 2.34 | 1.06 | 1.62 | 0.39 |
| DIFFUSE | 1.10 | 7.63 | 0.50 | 15.81 | 2.00 | 0.93 | 2.44 | 0.86 |

Node Arrival Rate: 0.4 jobs/sec

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 1.68 | 0.00 | 0.73 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.24 | 25.92 | 0.46 | 37.19 | 7.58 | 8.66 | 0.00 | -0.57 |
| RECEIVR | 1.35 | 19.78 | 0.52 | 29.31 | 4.65 | 0.63 | 2.22 | 0.30 |
| SENDER | 1.24 | 26.44 | 0.47 | 35.01 | 7.38 | 0.96 | 0.10 | -0.81 |
| SYMTRIC | 1.24 | 26.00 | 0.47 | 35.88 | 7.65 | 0.65 | 2.48 | 0.18 |
| DIFFUSE | 1.28 | 23.60 | 0.49 | 32.89 | 6.45 | 2.19 | 2.41 | 0.53 |

Node Arrival Rate: 0.6 jobs/sec

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 3.11 | 0.00 | 1.28 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.44 | 53.61 | 0.45 | 65.19 | 15.94 | 18.01 | 0.00 | -1.28 |
| RECEIVR | 1.56 | 49.68 | 0.51 | 59.87 | 9.97 | 1.41 | 2.50 | -0.48 |
| SENDER | 1.40 | 54.90 | 0.44 | 65.33 | 13.92 | 1.57 | 0.28 | -1.57 |
| SYMTRIC | 1.39 | 55.17 | 0.44 | 65.62 | 15.02 | 2.06 | 2.77 | -0.21 |
| DIFFUSE | 1.45 | 53.30 | 0.46 | 63.69 | 13.06 | 5.69 | 2.38 | -0.33 |

Node Arrival Rate: 0.8 jobs/sec

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 4.86 | 0.00 | 2.11 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.54 | 68.40 | 0.48 | 77.47 | 20.39 | 23.28 | 0.00 | -1.83 |
| RECEIVR | 1.65 | 65.95 | 0.50 | 76.49 | 12.93 | 1.73 | 2.50 | -0.79 |
| SENDER | 1.47 | 69.66 | 0.44 | 79.15 | 16.81 | 2.01 | 0.40 | -1.95 |
| SYMTRIC | 1.45 | 70.12 | 0.44 | 79.22 | 18.70 | 2.51 | 2.89 | -0.38 |
| DIFFUSE | 1.51 | 68.95 | 0.46 | 78.21 | 15.77 | 8.12 | 2.37 | -0.76 |

Node Arrival Rate: 0.9 jobs/sec

*Table 5.23 Performance of Standard Algorithms under Identical Arrivals*

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 1.19 | 0.00 | 0.59 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.07 | 10.36 | 0.49 | 16.70 | 2.45 | 3.05 | 0.00 | -0.11 |
| RECEIVR | 1.12 | 6.24 | 0.52 | 11.49 | 1.32 | 0.00 | 1.57 | 0.42 |
| SENDER | 1.06 | 10.77 | 0.50 | 16.05 | 2.26 | 0.00 | 0.02 | -0.32 |
| SYMTRIC | 1.06 | 10.77 | 0.49 | 17.04 | 2.28 | 0.00 | 1.62 | 0.31 |
| DIFFUSE | 1.10 | 7.74 | 0.51 | 13.39 | 2.23 | 0.84 | 2.44 | 0.87 |

Node Arrival Rate: 0.4 jobs/sec

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 1.68 | 0.00 | 0.73 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.23 | 26.83 | 0.45 | 38.04 | 7.49 | 6.77 | 0.00 | -0.90 |
| RECEIVR | 1.33 | 20.70 | 0.51 | 30.23 | 4.88 | 0.68 | 2.23 | 0.30 |
| SENDER | 1.22 | 27.28 | 0.46 | 36.71 | 7.05 | 0.41 | 0.09 | -0.99 |
| SYMTRIC | 1.24 | 26.13 | 0.45 | 38.16 | 7.47 | 0.56 | 2.48 | 0.02 |
| DIFFUSE | 1.29 | 23.50 | 0.48 | 34.64 | 6.90 | 2.59 | 2.41 | 0.42 |

Node Arrival Rate: 0.6 jobs/sec

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 3.11 | 0.00 | 1.28 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.39 | 55.16 | 0.44 | 65.86 | 14.45 | 15.37 | 0.00 | -1.74 |
| RECEIVR | 1.54 | 50.51 | 0.49 | 61.99 | 10.74 | 1.34 | 2.50 | -0.52 |
| SENDER | 1.37 | 56.00 | 0.43 | 66.60 | 13.14 | 1.69 | 0.25 | -2.06 |
| SYMTRIC | 1.37 | 55.83 | 0.42 | 67.49 | 14.36 | 1.70 | 2.75 | -0.73 |
| DIFFUSE | 1.38 | 55.64 | 0.46 | 63.77 | 12.86 | 5.78 | 2.38 | -0.41 |

Node Arrival Rate: 0.8 jobs/sec

| Algorithm | Benefit | | | | Cost | | | |
|---|---|---|---|---|---|---|---|---|
| | response(AVG) | | response(STD) | | job mov. | | messages | %LB_exec |
| | time(s) | Impr(%) | time(s) | Impr(%) | %job mov | %bad dcs | | |
| NOLB | 4.86 | 0.00 | 2.11 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| RANDOM | 1.46 | 69.94 | 0.43 | 79.43 | 17.19 | 17.97 | 0.00 | -2.52 |
| RECEIVR | 1.62 | 66.66 | 0.49 | 76.83 | 13.08 | 1.82 | 2.50 | -0.84 |
| SENDER | 1.44 | 70.46 | 0.43 | 79.84 | 15.76 | 1.62 | 0.35 | -2.41 |
| SYMTRIC | 1.34 | 72.42 | 0.42 | 80.18 | 17.38 | 2.24 | 2.84 | -0.83 |
| DIFFUSE | 1.43 | 70.63 | 0.45 | 78.46 | 15.27 | 6.98 | 2.37 | -1.02 |

Node Arrival Rate: 0.9 jobs/sec

*Table 5.24 Performance of Adapted Algorithms under Identical Arrivals*