DECENTRALIZED LOAD BALANCING IN HETEROGENEOUS COMPUTATIONAL GRIDS

A thesis submitted for review

Kai Lu

November 11, 2007

Abstract

With the rapid development of high-speed wide-area networks and powerful yet low-cost computational resources, grid computing has emerged as an attractive computing paradigm. The space limitations of conventional distributed systems can thus be overcome, to fully exploit the resources of under-utilised computing resources in every region around the world for distributed jobs. Workload and resource management are key grid services at the service level of grid software infrastructure, where issues of load balancing represent a common concern for most grid infrastructure developers. Although these are established research areas in parallel and distributed computing, grid computing environments present a number of new challenges, including large-scale computing resources, heterogeneous computing power, the autonomy of organisations hosting the resources, uneven job-arrival pattern among grid sites, considerable job transfer costs, and considerable communication overhead involved in capturing the load information of sites.

This dissertation focuses on designing solutions for load balancing in computational grids that can cater for the unique characteristics of grid computing environments. To explore the solution space, we conducted a survey for load balancing solutions, which enabled discussion and comparison of existing approaches, and the delimiting and exploration of the apportion of solution space. A system model was developed to study the load-balancing problems in computational grid environments. In particular, we developed three decentralised algorithms for job dispatching and load balancing—using only partial information: the desirability-aware load balancing algorithm (DA), the performance-driven desirability-aware load-balancing algorithm (P-DA), and the performance-driven region-based load-balancing algorithm (P-RB). All three are scalable, dynamic, decentralised and sender-initiated. We conducted extensive simulation studies to analyse the performance of our load-balancing algorithms. Simulation results showed that the algorithms significantly outperform pre-existing decentralised algorithms that are relevant to this research.

Acknowledgements

It is with great pleasure that I wish to acknowledge several people that have helped me tremendously during the difficult, challenging, yet rewarding and exciting path towards a Ph.D. Without their help and support, none of this work could have been possible.

First and foremost, I wish to express my sincere gratitude to my research advisor, Prof. Albert Y. Zomaya for his guidance, encouragement, motivation, and continued support throughout my academic years at the University of Sydney, Australia. Albert has allowed me to pursue my research interests with sufficient freedom, while always being there to guide me. Working with him has been one of the most rewarding experiences of my professional life.

I would like to express my gratitude to Dr. Riky Subrata who took the time to review my work and with whom I was able to co-author some papers. His valuable comments are very helpful. I would like to sincerely thank Dr. Bing Bing Zhou for helpful discussions and advices. Many thanks go also to the rest of the Advanced Networks Research Lab members for my help. My fruitful discussions and interactions with them helped me grow professionally.

I am grateful to Dr. Tony Souter not only for reviewing my dissertation, but more importantly for teaching me how to improve the presentation of the dissertation in many ways. I am very thankful to my dissertation committee members for agreeing to serve on my committee.

Last but not least, I am forever indebted to my wife, my parents, and the rest of my family. My wife, Rong Zhang, has been a great source of inspiration to me. None of

this would have been possible without her love, support, and continuous encouragement. My parents' prayers have always accompanied me. Their love keeps me going. My daughter Sophia has been the greatest source of motivation and inspiration during the last year of my Ph.D. I am very lucky to have been blessed with her. I am grateful to all of them. This work is dedicated to my family.

1

Introduction

This chapter introduces the context of the research. It starts with an introduction to the general area of grid computing, including a layered architecture of the computational grid, and a short overview of core grid middleware. It discusses the motivation and challenges for scheduling and load balancing in these environments, and summarises the primary contributions of the project and the evaluative methodology.

1.1 Cluster and grid environments

The emergence of computer and information technology has changed society dramatically. At the same time, the advances in high-speed networking have enabled computers to collaborate. This has created a tremendous source of processing power by the use of distributed systems, opening up many possibilities for running advanced computation-intensive jobs in a reasonable time frame.

There are several options for establishing distributed systems; cluster systems [5] and grid systems [1] are the most common for distributed jobs. Cluster systems combine several personal computers or workstations to conduct distributed applications through a

high-speed local network. The disadvantage is that the use of cluster systems must be limited to a fixed area (e.g., [73, 110, 131]), making the job inflexible in terms of its performance.

We refer to a cluster system as a "site". A computational grid uses the network and combines computing resources from geographically disperse sites for distributed jobs. Because grid computing uses Internet connections, compared with conventional distributed systems, it provides better large-scale resource sharing, improved resource utilisation and the broad-field Internet access environment [3]. Space limitations of conventional distributed systems can thus be overcome to fully exploit these under-utilised computing resources [119]. A computational grid can form more powerful computing capabilities to assist in the computing of large amounts of more complicated scientific jobs than can a cluster system; thus, processing power is more efficiently used and the quality of service is improved, for instance, in shorter response times. Computational grids are emerging as next-generation computing platforms for large-scale computation problems in industry, academic and government organisations.

1.2 Resource management models

There are two kinds of resource management models and corresponding metrics.

- ***System-centric*.** Most jobs in an open grid system are independent. They are submitted to the grid at different times, and require different resources and durations for their execution. When a single job arrives at a grid within a unit-scheduling time-slot, the scheduling system will analyse the load situation of every node and select one node to run the job. At this stage, the scheduling policy is to optimise the total performance of the whole system. If the grid system is heavily loaded, the scheduling system must realise the load balancing and increase the system throughput and resource utilisation under restricted conditions. In this dissertation, this kind of scheduling is classified as "system-centric scheduling", for which the objective is to optimise system performance, such as

[6], [8–9], [16], and [32]. The system-level resource scheduling was our main focus.

- ***Application-centric.*** If a parallel application with a number of tasks arrives within a unit-scheduling time-slot, the scheduling system will attempt to allocate and finish it in terms of a defined objective. The objective is usually the minimal completion time for the entire application. This scheduling policy is application-oriented, so here, we refer to it as "application-centric scheduling" [2, 4, 13–14]. Application-centric models deal with three kinds of application. The first kind is task farming, also known as "Bag of Tasks", in which numerous independent jobs arrive simultaneously. The second kind is a co-allocation application, in which each task is modelled as performing all-to-all communication patterns throughout its execution. The last kind can be represented as a direct acyclic graph (DAG) application, in which there is data-dependency between a task and other tasks.

1.3 Core grid middleware: Globus

In practical terms, all major grid projects are built on protocols and services provided by the Globus Toolkit [10], a software "work in progress" developed by the Globus Alliance, which involves primarily Professor Ian Foster's team at Argonne National Laboratory and Professor Carl Kesselman's team at the UCLA [1,7]. A brief description for the core grid middleware is given below.

Globus [11] provides software infrastructure that enables applications to handle distributed heterogeneous computing resources as a single virtual machine. The Globus project is a US multi-institutional research effort that is enabling the construction of computational grids. A computational grid, in this context, is hardware and software infrastructure that provides dependable, consistent, and pervasive access to high-end computational capabilities, despite the geographical dispersion of both resources and users. Globus provides basic services and capabilities required to construct a computational grid. Their toolkit consists of a set of components that implement basic services, such as security, resource location, resource management, and communications.

Computational grids need to support a wide variety of applications and programming paradigms. Consequently, rather than providing a uniform programming model, such as the object-oriented model, Globus provides a bag of services that developers of specific tools or applications can use to meet their own particular needs. This methodology is possible only when the services are distinct and have well-defined application programming interfaces (APIs) that can be incorporated into applications or tools incrementally.

Globus is constructed as a layered architecture in which high-level global services are built on essential low-level core local services. The Globus toolkit is modular, and an application can exploit Globus features, such as resource management or information infrastructure, without using the Globus communication libraries. The Globus toolkit supports the following:

- grid security infrastructure;
- gridFTP;
- Globus resource allocation manager;
- metacomputing directory service;
- global access to secondary storage;
- data catalogue and replica management; and
- advanced resource reservation and allocation.

Globus can be viewed as a grid computing framework based on a set of APIs to the underlying services. Globus provides application developers with a practical means of implementing a range of services to provide a wide-area application execution environment.

The major grid tools and application projects using Globus as their low-level middleware include AppLeS [13], Ninf [15], Nimrod-G [14], NASA IPG [21], Condor-G [16], Gridbus Broker [17], UK eScience Project [18], GriPhyN [20], and the EU Data Grid [19].

1.4 Motivation of the research

Workload and resource management are key grid services provided at the service level of grid software infrastructure, where issues of load balancing are a common concern for most developers. Although these are established research areas in parallel and distributed computing, grid computing environments present at least seven new challenges.

- ***Scalability.*** The grid may potentially encompass all high-performance computing resources. A given component of the grid will have its own functions, resources and environment. These are not necessarily geared to work together in the overall grid; they may be physically located in different organisations and may not be aware of each other. Scalability is of particular concern to grid environments that can attain planetary scale in terms of both the number of resources and the number of jobs for those resources. This raises the problem of potential performance degradation as the size of grids increases.

- ***Autonomy.*** Grid resources typically span multiple organisations each having one or more administrative domains. Each administrative domain typically has autonomous policies that govern the sharing of its resources with the rest of the grid. Resource management must preserve the autonomy of organisations hosting the resources. Each computing resource is independent. The computing resource itself determines whether or not to run a new job, how to schedule it and how to optimise resource utilisation. The strength of this autonomy varies at different levels of the grid architecture. Generally speaking, the higher a level, the more autonomous.

- ***Adaptability.*** A grid is a dynamic environment where the location, type and performance of the components are constantly changing. For example, a component resource may be added to or removed from the grid at any time. In addition, a component resource may fail within a certain time. How to handle these unexpected system activities in as close to local resource as possible, such as within a local organisation.

- ***Heterogeneity.*** A grid involves a multiplicity of resources that are heterogeneous and encompass a vast range of technologies. A grid exhibits heterogeneity of many types—including hardware, operating systems, file systems and network heterogeneity. Heterogeneity poses a challenge because it must be managed to enable the parts of the grids to work together; however, it also presents an opportunity, since the variety of different resources suggests that it may be possible to select the best resources for a particular problem. The variety and amount of computing resources in the grids offers significant potential for high-performance computing. Zhou et al. [22] define three forms of heterogeneity that may exist in distributed systems: (1) architectural heterogeneity (the hardware design differs from node to node, so nodes require different executable code); (2) operating system heterogeneity (facilities provided at each node may differ and may be incompatible); and (3) configuration heterogeneity, also known as performance heterogeneity (nodes are differently resourced, e.g., using different processing speeds and communication capacities). Performance heterogeneity is the most common form of heterogeneity, found in almost every loosely coupled system. Due to its prevalence, this work is concerned with performance heterogeneity. Other forms of heterogeneity are beyond the scope of this study.

- ***Information freshness.*** A widely used solution for resource management in a grid is the Globus MDS [23–26]. It is based on a hierarchical scheme that devises a set of nodes, each one hosting an Index Service. Each Index Service maintains a database on current resource availability for a group of grid nodes. Index Services can be also aggregated in higher-level Index Services, using a hierarchical approach typical of information repositories for widely distributed systems (such as the Internet domain name system). Even though it has been proved that hierarchical approaches are well suited to efficiently manage huge quantities of information in large distributed systems [27], the MDS approach suffers from a key problem: since resource information is advertised from grid nodes to index services, and then forwarded up in the hierarchy, inconsistences due to propagation latencies are possible, between the real information present in a grid node and the relevant copy stored in the repositories. As a result, matchmaking is

performed using data that cannot be real. The communication overhead involved in capturing the load information of sites before making a dispatching decision can be a major issue negating the advantages of job migration.

- ***Considerable transfer cost.*** Since computers within a site are typically connected by a high speed network, the network cost of remote job execution can be ignored when a job is scheduled to run on a different computer in the same site. However, in the grid environment, the related files of a job need to be transferred through much slower Internet links if the job is scheduled to run in a remote site. Therefore, the cost of file transfers must be taken into consideration in the scheduling algorithm.

- ***Uneven job arrival pattern.*** In a grid, some computers may be overloaded while others may be under-utilised. An analysis [28] of the resource usage pattern at several supercomputer centers (San Diego Supercomputer Center, National Center for Supercomputer Applications, Cornell Theory Center, KTH Royal Institute of Technology), shows an interesting "sine wave" pattern. During evenings, the resource requested reaches and sometimes exceeds the maximum capacity of the system, while usage dips to a minimum in the early hours of the morning. A computational grid, in addition to providing more computation power than any single site can provide, the time-dependent and bursty nature of resource requests can be better averaged by distributing the requests to different computing sites. Thus, effective load balancing is important in optimising resource usage, but the task of load balancing is more complex in a grid environment.

The main motivation of the study is to propose decentralised dynamic load balancing solutions that can cater for these unique characteristics of grid computing environments.

1.5 Major contributions

We present our research on decentralised dynamic load balancing in heterogeneous computational grids with the aim of improving the overall utilisation and performance, by developing effective load-balancing strategies. Before we start to explore the solution

space, we designed a survey for load-balancing solutions. This survey was useful for discussing and comparing pre-existing approaches. Using the survey, we delimited and explored apportion of solution space. We also developed a system model to study load-balancing problems in computational grid environments. This research has made three main contributions.

1.5.1 Desirability-aware load-balancing algorithm

In Chapter 4, we present an efficient desirability-aware load-balancing algorithm for heterogeneous computational grids. We give two definitions of desirability of sites based on how site characteristics will affect the performance of future load balancing: processing power and transfer delay. Using the desirability of sites, a set of partners and neighbours are formed for each site. Partners are sites with comparable or greater processing powers; neighbours are nearby sites with low transfer delays. We have designed an approach for constructing the partner sites for each site when a site joins the grid. We determined an approach to enable the set of partners for a site to be updated dynamically at runtime based on feedback information, and a relatively simple approach to form neighbouring sites for each grid site. The algorithm consists of two specific policies for load distribution: Instantaneous Distribution Policy (IDP) and Load Adjustment Policy (LAP). When a new job arrives at a site, it either remains at that site or is immediately allocated by IDP to one of its partner sites. Due to the likely fluctuating behaviour of grid resources, continuous load adjustment is employed among neighbour sites under the guidance of LAP to better exploit the grid environment. To reduce or minimise the communication overhead involved in information collection, state information exchange between sites is performed via mutual information feedback (MIF).

1.5.2 Performance-driven desirability-aware load-balancing algorithm

We developed this algorithm for heterogeneous computational grids in Chapter 5, an extended study of the algorithm introduced in Chapter 4, by considering the performance benefit that jobs can gain in the load distribution triggered by IDP and LAP. We developed another load adjustment policy, the Augmented Load Adjustment Policy

(ALAP). This policy determines whether there is a better placement beyond the neighbourhood of a site, where IDP has failed to show that a relatively not powerful site in that neighbourhood is a performance bottleneck.

1.5.3 Performance-driven region-based load balancing algorithm

We developed this algorithm for heterogeneous computational grids. The grid sites are clustered into regions around a set of well-known broker sites in terms of network transfer delay, and the regional brokers are organised in a fully decentralised fashion. For each regional grid, the algorithm integrates static IDP and dynamic LAP to make load distribution and redistribution driven by the performance benefit that jobs can gain. The LAP also considers load redistribution across regional grids. The intra-region communication is minimised by MIF. To control inter-region communication, the random polling of a remote regional broker site is performed by each regional broker site at a set time interval. The algorithm achieves a balance between the inherent efficiency of a centralised approach, and the autonomy, load balancing and fault tolerant features offered by this distributed approach.

1.6 Evaluation methodology

We have used discrete-event simulation to evaluate the performance of load-balancing algorithms in heterogeneous computational grids. The simulation programs were developed using Java object-oriented programming language.

1.7 Structure of the dissertation

In Chapter 2, related work in the literature is briefly reviewed. In Chapter 3, we describe a heterogeneous computational grid system model that we subsequently consider. In Chapter 4, we develop a decentralised and desirability-aware load-balancing algorithm for heterogeneous computational grids, which utilises the desirability of sites for load distribution. In Chapter 5, we develop a performance-driven load-balancing algorithm for heterogeneous computational grids. In Chapter 6, we develop a performance-driven and

region-based load balancing algorithm for heterogeneous computational grids. Chapter 7 summarises the main contributions of this dissertation and comments on future directions for this research.

2

Survey of load balancing

Recent years have been witness to the increasing use of distributed computing systems. This may be attributed to two main factors: the growth of the Internet, and the emergence of low-cost solutions for end-user computing devices. Distributed systems are collections of autonomous processing nodes connected by a communication network. Through the communication network, the resources of the system can be shared by users at different locations. However, a fundamental problem arises in making effective use of the total computing power of a distributed computing system. It is often the case that a certain node has very few tasks to handle at a given time, while another node has many. It is desirable to spread the total workload of the distributed system over all of its nodes. This avoids under-utilisation of power, and decreases response times for work introduced at more heavily loaded sites. This form of computing power sharing for improving the performance of a distributed system by redistributing the workload among the available nodes is commonly called “load balancing”. The purpose of load balancing is to improve the performance of a system by redistributing the workload among nodes, thus increasing the processing capacity of the system.

The chapter is structured as follows: in Section 2.1, simple classifications are introduced to organise the different techniques and methods that have appeared in the load balancing in distributed computers and Grid systems. Section 2.2 is an overview of the policies of load-balancing algorithms. Section 2.3 gives a brief overview of decentralised approaches, which focus on the load-balancing algorithms that utilise partial information to make decisions.

2.1 Load-balancing algorithms: a simple classification

Many different load balancing algorithms are described in the literature. However, most of these descriptions are presented in a mixture of text, drawings and pseudo-code, using inconsistent terminology details. Readers' ability to evaluate and compare the various algorithms is severely impaired by the absence of a common reference framework. The concepts used to classify the algorithms are also useful for the methodical design and analysis of new load-balancing algorithms. This section presents a simple classification of load balancing algorithms most relevant to this research.

2.1.1 Static versus dynamic

Load balancing could be done statically at compile-time or, dynamically, at run-time. Static load-balancing algorithms assume that *a priori* information about all of the characteristics of the jobs, the computing nodes and the communication network are known. Load-balancing decisions are made deterministically or probabilistically at compile time, and remain constant during run-time. The static approach is attractive because of its simplicity and the minimised run-time overhead. However, the static approach cannot respond to a dynamic run-time environment, and may lead to load imbalance on some nodes and significantly increase the job response time. The majority of loosely coupled distributed systems exhibit significant dynamic behaviour, having load varied with time. For these systems, dynamic scheduling, in which policy decisions are based on the load-state of nodes, is required. As a result, there are fewer studies on static approaches compared with those on dynamic approaches [43–51].

In contrast, dynamic load-balancing algorithms attempt to use the run-time state

information to make more informative decisions in sharing the system load. Recent studies have focused on schemes that base decisions directly on the current system state. Dynamic load-balancing policies [52–135] attempt to dynamically balance the workload reflecting the current system state, and are therefore thought to be able to further improve system performance. Thus, compared with static ones, dynamic load-balancing policies are thought to be better able to respond to system changes and to avoid states that result in poor performance. The clear disadvantages of dynamic load-balancing policies are that these policies are more complex than their static counterparts, in the sense that they require information on the run-time load and activities of state collection. Due to the communication costs of load information collection and distribution, the communication cost of job transfer and processing cost of making scheduling decisions, dynamic load-balancing algorithms definitely incur non-zero run-time overhead. A good dynamic load-balancing algorithm always makes these costs minimised. Thus, it is now commonly agreed that, despite the higher run-time complexity, dynamic algorithms potentially provide better performance than do static algorithms.

Hybrid algorithms [64, 127] combine the advantages of both static and dynamic strategies. In hybrid algorithms, the static algorithm is considered a “coarse” adjustment, and the dynamic algorithm a “fine” adjustment. When the static algorithm is used, load imbalance may result. Once this happens, the dynamic algorithm starts to work and guarantees that the jobs in the queues are balanced in the entire system. Our algorithms belong to this category.

2.1.2 Non-preemptive versus preemptive

Dynamic load-balancing policies may be either non-preemptive or preemptive. A non-preemptive load-balancing policy [53, 90, 110, 117, 119] assigns a newly arriving job to what appears at that moment to be the best node. Once the job execution begins, it is not moved, even if its run-time characteristics, or the run-time characteristics of any other jobs, are changed after assigning the job in such a way as to cause the nodes to become much unbalanced. An improvement in the spread of load is desirable, but it is accepted that this does not have to be optimal and that the load at each node need not be fully

equalised. This relaxation allows schemes to be devised that deal with a large-grain division of the workload, such as at the task level, and that use load transfers sparingly and thus do not require such high-speed communication between nodes. Non-preemptive load-balancing policies can be applied to any distributed system; however, they are particularly suited to loosely coupled systems, which have relatively low-speed inter-node communication and tend to consist of performance heterogeneous nodes. An example of such loosely coupled system is computational grids.

By contrast, a preemptive load-balancing policy [80, 89] allows load-balancing whenever the imbalance appears in the workloads among nodes. If a job that should be transferred to a new node is in the course of execution, it will continue at the new node. Since, in most systems, an initial distribution of jobs across nodes makes those systems appear balanced, they will become unbalanced as shorter jobs complete and leave behind an uneven distribution of longer jobs. Migration allows these imbalances to be corrected. However, to migrate a job in execution is much more complex and requires considerable overheads (caused by gathering and transferring the state of the job, resulting in performance degradation). If the preemptive policies were attempted in a loosely coupled large-scale system, the system performance would probably suffer significantly more, since there would be a large number of messages generated, which would congest the communication system. The preemptive policies are suitable only for tightly coupled distributed systems, in which the processing nodes are homogeneous and are connected by a high-speed low-latency interconnect. Many studies (e.g., [56, 81, 82]) have also shown that: (1) job migration is often difficult in practice, (2) the operation is generally expensive in most systems, and (3) there are no significant benefits of such a mechanism over those offered by non-migratory counterparts. Hence, we consider only non-preemptive load-balancing strategies.

2.1.3 Site-level versus grid-level

When a job arrives at a site, the load-balancing system of the site will analyse the load situation of every node in the site and will select a node to run the job. Even though the site is heavily loaded, each job must queue in the site and wait to be processed. We

classify this kind of load-balancing as site-level load-balancing, for which the objective is to optimise the system performance in a single site. Many traditional load-balancing algorithms fall in the category of site-level (e.g., [71, 110, 131]).

On the contrary, if a site lacks sufficient resources to complete the newly arriving tasks, or the site is heavily loaded, the load-balancing system of the site will transfer some tasks to other sites, and will increase the system throughput and resource utilisation in multiple sites. We call this load-balancing as grid-level load-balancing [53, 79, 97, 117, 119, 127, 132]. The focus of this dissertation is on grid-level load-balancing.

2.1.4 Centralised versus distributed

Load-balancing policies can be classified as centralised or distributed. Centralised policies (e.g., [53, 60, 87]) may be considered as a system with only one load-balancing decision maker. Arriving jobs to the system are sent to this load-balancing decision maker, which distributes jobs to different processing nodes. The centralised policies have the advantages of easy information collection about job arrivals and departures, and natural implementation that employs the server-client model of distributed processing. It appears that this policy is closely related to the overall optimal policy, in that there is only one load-balancing decision maker, which makes all of the load-balancing decisions. The major disadvantages of centralised policies are the possible performance and reliability bottleneck due to the possible heavy load on the centralised job load-balancing decision maker [111]. For this reason, centralised approaches are inappropriate for large-scale systems; furthermore, failure of the load-balancing decision maker will make the load-balancing inoperable.

On the other hand, distributed policies delegate job distribution decisions to individual nodes. Usually, each node accepts the local job arrivals and makes decisions to send them to other nodes on the basis of its own partial or global information on the system load distribution. It appears that this policy is closely related to the individually optimal policy, in that each job (or its user) optimises its own cost (e.g., its own expected average response time) independently of the others. The distributed load-balancing is widely used to handle imperfect system load information [64, 111].

There are two kinds of hybrid models. One is a combination of fully centralised and distributed algorithms [131]. The other is a hierarchical model, which combines partially centralised and distributed algorithms to overcome some of the limits of fully centralised algorithms [75, 130, 132]. The first model is applicable only for small-scale distributed systems; the latter still has fault-tolerance problems, due to single point of failure in a set of manager nodes of clusters. The system is logically divided into clusters, and each cluster of nodes will have a single node that maintains the state information on the nodes within the cluster. The state information on the whole system is maintained in the form of a tree, where each tree-node maintains the state information on the set of processing nodes in the sub-tree, rooted by the tree-node. The hierarchical model can be simplified as two-level if the set of manager nodes are organised in a fully distributed style [83, 129].

2.1.5 Partial versus global information

How much load information on the system should be collected for load-balancing in the distributed policies is a major issue. Any dynamic load-balancing algorithms include a decision part, which may use load information from a subset of the whole system (e.g., [69, 119–120]) or information from the whole system (e.g., [90, 118, 127]). The former is called “partial decision base” and the latter “global decision base”. For an initiating node, a subset of the whole system may be its nearest neighbours or nodes that are polled at random or formed by specific criteria. In all cases, the degree of the knowledge of the system load status and the accuracy of the redistribution decisions conflict. On the one hand, more load information implies that there is a better chance of reaching a higher quality of load redistribution decisions. On the other hand, more load information also means more overhead to collect, and thus more chance for the load information to be out of date, unpredictably leading to an even worse load imbalance. Therefore, using detailed load information does not always significantly aid system performance, and a tradeoff must be made. Xu et al. [105] has shown that nearest-neighbour algorithms using only local load information work very efficiently.

2.1.6 Sender-initiated versus receiver-initiated

Distributed load-balancing policies can be broadly characterised as sender-initiated and receiver-initiated. Sender-initiated algorithms [54, 55, 57, 75, 79, 86, 120] let the heavily loaded sites take the initiative to request the lightly loaded sites to receive the jobs; receiver-initiated algorithms [54, 55, 75, 79, 86, 120] let the lightly loaded sites invite heavily loaded sites to send their jobs. Sender-initiated load-balancing algorithms perform better than receiver-initiated load-balancing algorithms at low or moderate system loads. At these loads, it is reasoned, the probability of finding a lightly loaded node is higher than that of finding a heavily loaded node; similarly, at high system loads, the receiver-initiated policy performs better since it is much easier to find a heavily loaded node [54].

As a result, adaptive policies have been proposed, which combine the desired features of both sender and receiver-initiated techniques, and are called symmetrically-initiated [63, 75, 79, 88]. They seek to find suitable receivers when senders wish to send jobs, and to find suitable senders when receivers wish to acquire jobs. Efficient symmetrical policies (e.g., [68]) behave as sender-initiated under low and medium load conditions, and as receiver-initiated under heavy load conditions, following the result of Eager, Lazowska, and Zahorjan [55].

2.2 Policies for dynamic load-balancing algorithms

Many issues involved in dynamic load-balancing have already been addressed in load-balancing algorithms, such as how to measure the load of a processing node, how much load information we should collect and where they should reside. However, the real activities happening for different algorithms on differently designed systems may differ significantly. These issues are usually grouped into several policies (or components) at a higher level. For example, Xu et al. [112] considers that a dynamic load-balancing algorithm consists of four components: a load measurement rule, an information exchange rule, an initiation rule, and a load-balancing operation (defined by location rule, distribution rule and selection rule); Niranjan et al. [59] groups the issues into a transfer

policy, a selection policy, a location policy, and an information policy. Although the grouping of the issues and the naming of the policies may differ significantly among studies, they tend to discuss in common a set of key issues. In this section, we regroup the issues, name the policies, and discuss their possible choices. The policy names may or may not mean the same as in other studies.

- ***Information policy:*** this decides what, when and where information about states of other nodes is collected.
- ***Transfer policy:*** this determines whether a node is in a suitable state to participate in a task transfer.
- ***Selection policy:*** this decides which task should be transferred, if the node is a sender.
- ***Location policy:*** this locates a suitable transfer partner.

2.2.1 Information policy

Information policy covers most issues related to the load information necessary for making load-balancing decisions. Information policy decides what information is collected, and when information about the states of other nodes is to be collected, and from which nodes. It is also responsible for the dissemination of each node load information.

2.2.1.1 Load measurement rule

Measuring the load of the various nodes in the system accurately is very important for the success of a load-balancing algorithm. Measuring the load of the nodes in a distributed system is an extremely difficult task. Usually, load is measured by a metric, the "load index" [83]. A number of possible metrics have been studied in the past. These can be broadly divided into two main categories: simple and complex.

- ***Simple indices.*** They consider the load on only a single resource. This approach usually focuses on the load on the CPU. A simple load index includes processor queue length, average processor queue length over a given duration, the amount

of memory available, the context switch rate, the system call rate, and CPU utilisation.

- ***Complex load indices.*** They consist of a number of metrics, each relating to a single resource, such as CPU, disk, memory and network. The metrics that make up the load index may be combined to give a single load value or may be represented as a tuple consisting of a number of elements, one per metric. The load index used in [126] and MOSIX [89] comprises the CPU load and the amount of free memory. San Luis [115] uses a load index based on the performance-weighted CPU run-queue length, the amount of free memory, disk traffic level, and network traffic level. The memory requirements and the desired response-time of tasks are taken into account in scheduling decisions. Utopia [22] uses a load index that incorporates: CPU run-queue length, available memory, disk transfer rate, the amount of swap disk-space available, and the number of concurrent users. LSF [30] uses the same metrics as Utopia, with three additions: CPU utilisation, paging rate, and the amount of idle time at processing nodes.

A candidate load index should be easy to compute and correlate well with the parameter (e.g., the job response time) that is to be optimised. It has been found that simple load indices are particularly effective and impose less overhead. One of the most effective load indices is simply the processor queue length, and this choice seems to be unanimous [54, 83].

In a heterogeneous environment, the load indices from different nodes must be adjusted to make them comparable. For example, if two different nodes have different processing power, their CPU utilisation may have to be divided by their processing power to compare their CPU utilisation load index values. A better measurement may be the total job execution time [87]. Although in most cases the execution time of a job cannot be predicted accurately, it can be estimated by parameters such as the size of the program, the type of the job, or based on past statistics and experience.

2.2.1.2 Load information exchange policies

The information exchange policies can be broadly classified into three types, although

hybrid versions of these types may exist.

- ***Demand driven policies*.** Each node collects information when it needs it to make a load sharing decision. A poll-limit is usually used. The main advantage is that load information is exchanged only when it is required. This has the following disadvantages in practice.
  - Repeated polling wastes the processing time of the polling sites and polled sites. This problem becomes significant when the general system load is heavy. When most of the sites are heavily loaded, they continue to poll each other for the sparse lightly loaded site. In the worst case, polling may cause system instability when all the sites are heavily loaded.
  - Repeated polling generates a large amount of network traffic. This problem becomes more significant if the network bandwidth is limited.
  - As the job needs to wait for the polling result, polling will increase the response time of the waiting job. This is a problem if the communication delay is significant.
  - It is difficult to obtain a good value for the probe limit. The probability of a successful poll (the hit ratio) depends on the load level in the system; no predetermined number of polls can guarantee a hit. There is little or no benefit achieved by increasing the poll limit beyond 3 or 4 [91]. Small probe limits, such as 3, are appropriate as they return most of the benefits of larger values, at lower cost [62]. In a medium-to-heavily loaded system, if the probe limit is small, lightly loaded nodes may not be discovered. If the probe limit is large, then (i) most of the heavily loaded nodes may find the same lightly loaded nodes and dump their loads to them; and (ii) the problems caused by repeated polling will multiply.

- ***Periodic policies.*** Information is disseminated or collected at regular intervals. This is simple to implement. However, it is important to determine the most appropriate dissemination period as overheads due to periodic communication increase system load and reduce scalability. Here, a fixed amount of state-collection overhead will be induced in the system because each node collects and

maintains state-information of other nodes, regardless whether this information will be used. However, there is no polling delay when a task must be transferred. Mosix [89] used a simple probabilistic model to choose a random subset of hosts to send information about its available resources at regular intervals and cut down communication. In one study, the design extended the polling period but maintain the entire set of hosts to contact [90]. To ensure the system state can still be reasonably accurate when there are fewer updates, a predictive algorithm based on L2E predictive filtering model was employed. The image each node has of the system state (or domain state) may not correspond to the actual system state, due to delays in the communication network and to the periodic nature of information collection. In addition, the image a node has about the state may be different from node to node.

- ***State-change driven policies.*** Nodes issue information about their load state only when it changes by a certain amount [61]. Determining the threshold value is problematic, because the policy must be sensitive to significant changes but not to minor fluctuations. State-change policies generally have lower communication rates than periodic policies. However, if the state at a particular node does not change for a long period of time, the information held about that node will become stale. Aged load-state information is unreliable, since there is no way of telling if the node has crashed or has just not sent a message due to a steady state. A newly joining node will not receive information concerning steady-state nodes, even if those nodes are suitable transfer partners. One way to improve the basic state-change policy is to introduce additional dissemination messages, which are sent if the load-state does not change for a long period of time. These rules differ from demand-driven rules in that each node takes the initiative for disseminating its own state instead of collecting other nodes information.

Various combinations of these types of information exchange policies are possible. An information exchange policy might be periodic, but a node willing to participate in a task transfer might poll its best candidate to confirm that its actual state still corresponds to its local image. A combination of state-change driven and slow periodic update

dissemination was suggested in [58].

2.2.1.3 Where should the load information be maintained?

A central repository can be used to hold load-state information. This is collected from all of the nodes in the system and made available when a load-sharing decision must be made. Some centralised implementations are simply responsible for the collection and dissemination of information, while others additionally act as matchmakers between sender and receiver nodes. Centralised components can work well in small or moderately sized systems, but can become communication bottlenecks when the system is scaled up. Where centralised components are used in the entire system, that system is vulnerable to the failure of the single component unless some form of backup or replication is provided; this increases complexity.

Distributed approaches are more difficult to build than their centralised counterparts. The semantics involved can be complex. Each node collects information concerning the load state at other nodes in the system. Nodes autonomously base load sharing decisions on the information they hold. One advantage of distributed implementations is that the system is not vulnerable to the failure of any single node. There are also disadvantages: there is no consistent system-wide view of state, and each node holds different information depending on which other nodes it has communicated with, how recently that communication took place, and the delay experienced in that communication. This can lead to instability if there are significant differences in the views held.

2.2.1.4 How much load information of the system should be collected for load-balancing?

One extreme option is to collect load information over the global scope, i.e., all of the processing nodes in the system; another extreme is to use no load information at all of nodes, other than the node in question. The choices in between these two extremes use local load information collected from a certain domain of processing nodes in which size may be either fixed or variable. The global knowledge of the system's attributes (like the total work load) is prohibitive, due to the communication overhead produced. This is

especially true for large-scale distributed systems. Thus, the technique of demanding global information is rejected, and partial information is used instead, such as information of the neighbourhood of a node.

2.2.2 Transfer policy

A transfer policy determines whether a node is in a suitable state to participate in a task transfer, either as a sender or a receiver. Many proposed transfer policies are threshold policies, which may be either based on fixed or adaptive thresholds. One way is to set one threshold value for the load imbalance (the difference between the largest and smallest loads on the nodes). If the detected load imbalance is bigger that a preset threshold value, the transfer is initiated. An equivalent method to this is to set two threshold values, T_h and T_l, by which the nodes are classified into three types, i.e., heavily loaded or sender (if loads higher than T_h), lightly loaded or receiver (if loads lower than T_l), and normally loaded otherwise [59]. Depending on the algorithms, T_h and T_l may or may not have the same value. The choice of these thresholds is fundamental for the performance of the algorithm. Clearly, the best threshold values depend on the system load and the task transfer cost. At low loads and/or low transfer costs thresholds should favour task transfers, while at high loads and/or high transfer costs remote execution should be avoided. Although [62] states that the optimal threshold is not very sensitive to system load, [87] and [93] present techniques that efficiently and in run-time adapt the threshold to the system load.

Fixed threshold policies mean that the threshold values are not changed when system loads are changed [119]. There are disadvantages with the fixed threshold policy. If the fixed threshold value is too small, this still causes “useless” job transfers. If the fixed threshold value is too large, the effect of using a load-balancing mechanism may be reduced. Other than using fixed threshold values, thresholds can be set in an adaptive (relative) fashion, by adjusting them when the global system load is changed. In [61], if the load of an individual node is above or below the average load over a certain domain (either the global or some local range) by a preset percentage, then load-balancing actions are initiated and load is balanced either locally or globally. In another adaptive approach

to determining proper thresholds [87, 93], the average load L_{avg} is determined first. Two constant multipliers, H and L, are used in computing the heavy threshold, T_h, and light threshold, T_l. H is greater than one and L is less than one. These two values determine the flexibility and the effectiveness of a load-balancing mechanism. The heavy threshold, T_h, is computed as the product of H and L_{avg}. Similarly, the light threshold T_l is computed as the product of L and L_{avg}.

The transfer policy may be either periodic or event-triggered. The algorithm may periodically check whether the node's state qualifies itself as a candidate for a task transfer. However, the great majority of the policies proposed in the literature are event-triggered. If the state of a node changes, a task transfer may be possible. The state of the node may change because either a task has ended or a new task has arrived. The transfer policy can also be triggered because another node is polling the node, either to receive or to send a task.

For a given policy, a load-balancing policy may be sender-, receiver- or symmetrically-initiated. Sender-initiated algorithms may be ineffective at high system loads, because most of the nodes are senders and thus it is unlikely that the majority of system will ever find a suitable receiver. Even worse, they might overflow some of the potential receivers with too many tasks. Even if the potential receivers are allowed to reject additional work sent to them, more control messages will be introduced and useless work is performed in a system already highly loaded. Under a sender-initiated policy, the burden of initiating the activity is taken by an already-overloaded node.

Under receiver-initiated policies, this overhead is placed on the underloaded nodes, which may be adequate. However, if the system is lightly loaded, these policies will fail to find a suitable sender. How many times, or for so long, should a receiver try to find this sender? It can suspend its activity after a threshold (or timeout), but then it will not detect future overloaded nodes unless its activity is periodically reinitiated: a disadvantage of receiver-initiated algorithms is that the receiver is unaware that the other nodes became potential senders, because neither these senders notified them.

Symmetrically-initiated transfer policies support load transfers initiated by both busy and low-loaded nodes [88]. Symmetrically-initiated algorithms are more complex, but allow the advantages of both sender-initiated and receiver-initiated algorithms to be exploited. Symmetrically-initiated schemes are potentially unstable: there must be a zone between the activation thresholds for the sender and receiver parts of the algorithm so that a node cannot rapidly move between sender and receiver states. Symmetrically-initiated policies have been found to outperform sender-initiated and receiver-initiated policies in the presence of small task-transfer delays [92]. However, when the task transfer delays were increased, the policies were found to perform almost identically.

2.2.3 Selection policy

The role of selection policy is to select tasks for transfer. In sender-initiated schemes, busy nodes choose tasks to transfer to another node, whereas in receiver-initiated schemes, lightly loaded nodes inform potential senders of the types of task they are willing to accept. The policy determines how much load, or how many tasks, to transfer.

A task transfer may be preemptive (e.g., [80, 89]) or non-preemptive (e.g., [52, 119]). Preemptive transfers involve transferring a partially executed task. This is generally expensive, as it involves collecting all of the task's state. Non-preemptive-task transfers involve only tasks that have not begun execution and hence do not require a transfer of the task state. A node may be overloaded and have no tasks available for non-preemptive transfer if it is polled by a receiver. A selection policy should consider at least three factors.

- The overhead incurred in transferring the task should be minimised. Non-preemptive transfers and small tasks (small amounts of information) carry less overhead.
- The execution time of the transferred task should be sufficient to justify the cost of the transfer. Even if task execution is unknown, it should be possible to classify the tasks as short or long tasks, and to consider only the long tasks for migration. Some classification errors might be tolerated as load-balancing algorithms are quite robust with regard to this parameter [56].

- The number of location-dependent resources needed by the selected task should be minimal.

2.2.4 Location policy

The responsibility of location policy is to find a suitable transfer partner. Location policies can be distributed, each node selecting a transfer partner on the basis of locally held information. Location policy, corresponding to information policy, specifies the balancing domain for load-balancing actions; this could be global, nearest-neighbours, a group of random polled nodes, or a set or cluster of nodes based on specified criteria. Alternatively, policies can be devised using a central information source. Busy nodes attempt to locate transfer partners that have low load levels in sender-initiated schemes. In receiver-initiated schemes, low-loaded nodes attempt to locate a busy node from which to transfer work. Five typical policies are listed below.

- ***Random policies***. A transfer partner is selected at random, and its load-state is ignored. This can result in useless task transfers when an already-busy node receives extra work, but has been shown to provide performance improvements over no-load-distribution [58]. The performance improvements stem from the fact that only busy nodes transmit load, while all nodes are potential receivers. Random location policies work best when there are few heavily loaded nodes and many relatively idle nodes. Azar et al. studied an "*n*-ball *n*-bin" placement problem, where *n* balls are randomly and sequentially placed into *n* bins. It was proved that, in the sense of balanced placement, choosing $d \geq 2$ bins independently and uniformly at random and then placing the *i*th ball in the least-loaded one of the *d* bins would improve the result exponentially compared with that of choosing one bin randomly each time [123]. Mitzenmacher's works [122] generalised Azar et al.'s finding, and provided an analytical model to those randomised schemes used in Zhou's work [56]. An improved algorithm was proposed by adding a simple sliding-window technique and a simple, fuzzy classification technique to the original concept of M. Mitzenmacher's two choices ($d = 2$) in randomised load-balancing [121].

- ***Threshold policies.*** The node randomly selects a potential destination node for the job and probes it to determine its load index. If the load index at the proposed destination is less than or equal to a preset threshold value, that node becomes the job's receiver. Otherwise, another node is randomly selected and probed. Probing continues until a receiver is found or until the number of nodes probed is equal to a limit L_p. Threshold location policies are based on the result of the probing activity; if a receiver has been found, the job is sent there—otherwise the job is executed locally. The threshold policy was originated by Eager et al., who referred to it as "Sender' [55] and "Threshold" [62], and was also simulated by Zhou [56] and Kremien and Kramer [58]. It corresponds to "Algorithm 1" in the SAHAYOG tests [77] and to "Forward" in the studies by Mirchandaney et al. [91, 92].
- ***Lowest policies.*** Like threshold policies, lowest policies employ a threshold L_p. However, lowest policies differ from threshold policies in that it probes a group of nodes until a node with a zero load index is found, or until exactly L_p nodes have been probed. The lowest location policy is to select the probed node with the lowest load index as the execution site for the incoming job, provided that the load index at that node is less than a preset threshold value. Lowest policies have been simulated by Zhou [56] and are related to an algorithm studied by Theimer and Lantz [78]. The algorithm originated as "Shortest" in [62], where Eager et al. concluded that its performance was not sufficiently better than that of threshold policies to warrant the extra effort expended on information collection.
- ***Preferred list.*** Based on the topology of the system, each node orders all other nodes into a preferred list [124, 125]. A node is the *k*-th preferred node of one and only one other node, where *k* is an integer. If node *i* is the *k*-th preferred node of node *j*, then node *j* is also the *k*-th preferred node of node *i*. When a node is overloaded, it will contact the first node found in its preferred list, and attempts to transfer a task to that node. Although the preferred list of each node is generated statically, the actual preference of the node in transferring a task may change dynamically with the states of nodes in its preferred list. If a node's most

preferred node becomes overloaded, its second preferred node will become the most preferred.

- ***Least policies*.** To differentiate from the location policy lowest, we call this class of location policies "least". Least policies differ from lowest policies in that they do not need to probe nodes, and no threshold is used. The least location policy is to select the node with the smallest load index as the destination node for dispatching the jobs on the basis of the information on a specified balancing domain (e.g., [74, 118, 134]).

In a heterogeneous environment, a node with minimal load, i.e., queue length, does not mean the best transfer partner for a certain task. Node processing power and task transfer delay incurred among the node and remote nodes should also be considered in location policy.

2.3 Existing load-balancing algorithms

Two classes of well-known dynamic and distributed load-balancing algorithms are presented in this section. The focus is on the load-balancing algorithms utilising partial information to make decision. Although some algorithms are initially presented for parallel computers, they are applicable in a distributed computing system with more or less deficiencies. Thus, these are also introduced here.

Most load-balancing policies execute two activities that require communications: distribute its own load information and collect other nodes information and transfer tasks. If each node is required to interact with other nodes, it will have to use mechanisms – such as broadcast, global gathering, long-distance communication – which are not scalable and create intolerable overhead or congestion in systems with a large number of nodes.

To reduce this overhead, in many policies, a node only exchange information and transfer tasks to its physical and/or logical neighbours. These are usually called "neighbour-based" load-balancing algorithms. Clustering is another technique to tackle the problem. The nodes can be partitioned into clusters based on network transfer delay,

where load-balancing operates on two-level: intra-cluster and inter-cluster via cluster managers or brokers. These are usually called “cluster-based” load-balancing algorithms. We will give corresponding discussion to these two classes of algorithms below.

2.3.1 Neighbours-based load-balancing algorithms

The neighbours-based approach is a dynamic load-balancing technique that allows the nodes to communicate and transfer tasks with their neighbours only [65]. Each node balances the workload with its neighbours so that the whole system will be balanced after a number of iterations. Since this technique does not require a global coordinator, it is inherently local, fault tolerant and scalable. Hence, this approach is a natural choice for load-balancing in a highly dynamic environment [116]. Among of the neighbour-based algorithms, we are interested in a couple of typical representatives, described as follows.

2.3.1.1 The gradient model

The gradient model (GM) is a demand driven approach [66]. The basic concept is that underloaded nodes inform other nodes in the system of their state, and overloaded nodes respond by sending a portion of their load to the nearest lightly loaded node in the system. The resulting effect is a form of relaxation where tasks transferring through the system are guided by the proximity gradient and gravitate towards underloaded points. The scheme is based on two threshold parameters: the *Low-Water-Mark (LWM)* and the *High-Water-Mark (HWM).* A node’s state is considered light if its load is below the *LWM*, heavy if above the *HWM*, and moderate otherwise. A node’s *proximity* is defined as the shortest distance from itself to the nearest lightly loaded node in the system. All nodes are initialised with a proximity of W_{max}, a constant equal to the diameter of the system. The proximity of a node is set to zero if its state becomes light. A node’s proximity may not exceed W_{max}. A system is saturated, and does not require load-balancing if all nodes report a proximity of W_{max}. If the proximity of a node changes it must notify its near-neighbours. A gradient map of the proximities of underloaded nodes in the system serves to route tasks through the system in the direction of the nearest underloaded nodes. A task continues to transfer until it reaches an underloaded node or it reaches a node for which no neighbouring nodes report a lower proximity.

2.3.1.2 Adaptive contracting within neighbourhood

In the Adaptive Contracting Within Neighbourhood (ACWN) method [74], two parameters need to be specified to make the contracting decision, *min_hops* and *max_hops*. Here, *min_hops* specifies the minimum number of hops needed for a drifting task to travel before it settles into the standing state. This parameter is used to ensure a newly created task will travel certain distances to reduce the horizon effect. The other, *max_hops*, is the upper limit of travelling distance of a drifting task. It ensures that each newly created task will be sent only to a node within a fixed radius neighbourhood from its source node. It prevents unbounded message oscillations, and also induces locality which makes the communication between the creating and created tasks efficient. They keep track of the number of hops travelled so far for each task *c*, called *c.hops*. Thus, at each node, for a drifting task *c*, which is either created by themselves or received from other nodes, we have the following cases: if *c.hops* < *min_hops*, a node will contract task *c* to its least loaded neighbour no matter its own load; if *c.hops* > *max_hops*, task *c* will be retained locally, added to the local pool of messages, terminating its drifting state. Otherwise, the task will be contracted conditionally: if the load on the least-loaded neighbour is smaller than its own load, the task is contracted out to that neighbour. In this way, the newly generated task *c* travels along the steepest load gradient to a local minimum.

In ACWN, *min_hops* and *max_hops* are varied to adapt to the dynamic variations. Also, each node maintains a separate, independent, version of these parameters. Two additional parameters, *low_mark* and *high_mark*, are used to ascertain the current load status of a node and its neighbours. A node's state is considered light if its load is below the *low_mark*, heavy if above the *high_mark*, and moderate otherwise.

In the light-load state, ACWN tends to contract tasks out, since at least one neighbour is lightly loaded. In the moderate-load state, *min_hops* set to zero, so that any new work is kept locally unless a neighbour's load is smaller than a node's own load by the value of load-delta. In the heavy-load state, since all neighbours have sufficient work to do, it is not necessary to balance load between nodes. Therefore, we change *max_hops* to zero to retain newly created tasks locally.

This amounts to applying the saturation control technique. They observe from their experiments the almost same performance for *max_hops* ≥ 3. Also, the *low_mark* between 2 to 5 and *high_mark* around 8 were found to be satisfactory settings. They also show that ACWN performed consistently better than the gradient model.

2.3.1.3 Basic (Baseline) diffusion model

Diffusion was first presented as a method for load-balancing in [94]. Diffusion was also explored in [61] and was found to be superior to other load-balancing strategies in terms of its performance, robustness, and scalability. In the diffusion method, each node simultaneously sends workload to its neighbours with lower workload and receives workload from its neighbours with higher workload. Under the synchronous assumption, the diffusion method has been proven to converge in polynomial time for any initial workload distribution given the quiescent assumption that no new workload is generated and no existing workload is completed during execution of the algorithm [94]. Without the quiescent assumption, it is possible only to prove that the variance of the unbalanced workload is bounded [85]. Optimal parameters that maximise the convergence rate have been derived on mesh, torus, and n-D hypercube [84]. The convergence of the asynchronous diffusion method has also been proven [95]. A disadvantage of diffusion approach is that it requires many iterations to achieve load-balancing. Watts and Taylor [67] overcame this by using a fully implicit diffusion schemes with adaptive time steps. Another improved diffusion algorithm was derived based on Chebyshev polynomials and shows significantly faster convergence than baseline diffusion method, but at the additional cost of calculating two eigenvalues [96].

Diffusion algorithm in heterogeneous environment has been considered. Hui [98, 99] proposed an intuitive approach based on a hydrodynamic analogy, for a heterogeneous environment characterised by different computing powers and uniform communication. Diekmann et al. proposed diffusion schemes for a computational environment characterised by uniform computing powers and different communication parameters [100]. Elsasser et al. extended these schemes for computational environments that are heterogeneous both with respect to the processing performances and the communication speeds [101].

2.3.1.4 Sender initiated diffusion

The Sender Initiated Diffusion (SID) strategy [61] is a, local, nearest-neighbour diffusion approach which employs overlapping balancing domains to achieve global balancing. The scheme is purely distributed and asynchronous. Each node acts independently, apportioning excess load to deficient neighbours. Balancing is performed by each node whenever it receives a load update message from a neighbour indicating that the neighbour load is smaller than a preset threshold, L_{LOW}. Each node is limited to load information from within its own domain, which consists of itself and its immediate neighbours. All nodes inform their nearest-neighbours of their load levels and update this information throughout program execution. The profitability of load-balancing is determined by first computing the average load in the domain. Next, if a node's load exceeds the average load by a prespecified amount, $L_{threshold}$, it proceeds to implement the third phase of the load-balancing process. Task migration is performed by apportioning excess load to deficient neighbours. Balancing continues throughout program execution whenever a node's load exceeds the local average by more than a certain amount $L_{threshold}$.

2.3.1.5 Receiver initiated diffusion

The Receiver Initiated Diffusion (RID) strategy [61] can be thought of as the converse of the SID strategy in that it is a receiver initiated approach as opposed to a sender initiated approach. However, besides the fact that in the RID strategy underloaded nodes request load from overloaded neighbours, certain subtle differences exist between the strategies. First, the balancing process is initiated by any node whose load drops below a prespecified threshold (L_{Low}). Second, upon receipt of a load request, a node will fulfill the request only up to an amount equal to half of its current load (this reduces the effect of the aging of the data upon which the request was based). Finally, in the receiver-initiated approach, the underloaded nodes in the system take on the majority of the load-balancing overhead, which can be significant when the task granularity is fine.

As with the SID strategy, each node is limited to load information from within its own domain, which consists of itself and its immediate neighbours. All nodes inform their nearest-neighbours of their load levels and update this information throughout

program execution. When a node's load drops below the prespecified threshold L_{LOW}, the profitability of load-balancing is determined by first computing the average load in the domain. If a node's load is below the average load by more than a prespecified amount, $L_{threshold}$, it proceeds to implement the third phase of the load-balancing process. Task migration is performed by requesting proportionate amounts of load from overloaded neighbours. However, upon receipt of a load request, a node will fulfill the request only up to an amount equal to half of its current load. Balancing is activated whenever a node's load drops below a prespecified threshold and there are no outstanding load requests.

2.3.1.6 Estimated load information scheduling algorithm

In a decentralised dynamic load scheduling algorithm, the Estimated Load Information Scheduling Algorithm (ELISA), the problem of frequent exchange of information is alleviated by estimating the load, based on system-state information received at sufficiently large intervals of time [69]. The algorithm was designed to reduce communication delays by reducing the need for status exchange.

The basic idea behind ELISA is that at periodic intervals of time, the *status exchange interval*, the nodes in the system exchange their status information, which consists of the queue length at the instant of information exchange and an estimate of the arrival rate. The instant at which this information exchange takes place is a *status exchange epoch.* Each status exchange interval is further divided into equal subintervals—*estimation intervals*. The points of division are *estimation epochs.* At the estimation epochs, every node estimates the load in the nodes belonging to its buddy set, which consists of the immediate neighbours only (that is, those nodes which are one hop away). The status exchange epochs and the estimation epochs together constitute the set of transfer epochs. At the transfer epochs, rescheduling of jobs is carried out. Thus, the decision to transfer jobs is taken and the actual transfer of jobs is done at the transfer epochs. By making the interval between status exchange epochs large, and by restricting the exchange of information to the buddy set, the communication overheads are kept at a low value. Finally, by transferring jobs only at the transfer epochs, overheads on the scheduler are also kept low.

The load scheduling decision is taken as follows: from the estimated queue lengths of the nodes in its buddy set, and the accurate knowledge of its own queue length, each node computes the average load on itself and its buddy set. Nodes in the buddy set, whose estimated queue length is less than the estimated average queue length by more than a threshold, form the active set. The node under consideration transfers jobs to the nodes in the active set until its queue length is not greater than the estimated average queue length. The value of threshold, which is fixed *a priori,* is of importance to the performance of ELISA.

A Modified ELISA algorithm is presented in the paper [117]. Their proposed algorithm considers job migration cost, which is primarily influenced by the available bandwidth between the sender and receiver nodes, when taking decision for load-balancing. The job will be transferred only if its expected finish time on destination node is less than expected finish time on source node. In the sense, the algorithm is similar to our earlier proposed algorithm Performance-driven Neighbours-based algorithm (P-NB), which will be introduced in the section 5.3.

2.3.1.7 Recursive search

Arora et al. [119] proposed a highly decentralised, sender-initiated and scalable algorithm for scheduling tasks and load-balancing resources in heterogeneous grid environments. Whenever a job is submitted to a node, a decision needs to be made as to whether the job needs to be transferred according to a preset threshold reflecting by the job queue length. If the job needs to be transferred to another node, a request is sent to all neighbouring nodes. Each node, having received a request to send the status of its resources, packs the information about their current utilisation and sends it back to the requesting node along the route the request came. This route is piggybacked to the node, which needs to transfer load. Besides replying to requests, a node recursively pings its neighbours for their resource status if the total round-trip delay between the sender and its neighbour would be less than the time for which the internal job queue is emptied at the requesting node. This allows the time required to look for additional resources to be hidden under processing. This is a remarkable property of their algorithm. Their goal was to assign each node a job that utilises its resource in the best possible manner.

2.3.1.8 Discussion

Although a variety of different decentralised algorithms have been studied, most approaches are not applicable where a system is comprised of heterogeneous nodes separated by a wide-area broadband network. Previous research [101, 117] that considered a collection of heterogeneous nodes and different communication capability among nodes, did not achieve better utilisation of powerful processing nodes in a heterogeneous system. However, in a computational grid, there are numerous nodes capable of providing computing resources; some are frequently underutilised and able to provide powerful computing resources, but others are not. Therefore, when selecting nodes for distributing tasks, if only a neighbouring node is chosen, the redistribution of tasks may frequently occur, lowering the execution performance of the system. These solutions, except the work in [117], do not address the issue of the communication overhead incurred by frequent message transfer for making better load distribution, even though the information is restricted to a small domain, such as neighbours.

2.3.2 Cluster-based load-balancing algorithms

Cluster-based load-balancing algorithms have been the subject of several studies (e.g., [22, 83, 104, 135]). The nodes can be partitioned into clusters on the basis of network transfer delay. One node is designated as the cluster manager, which gathers the load information for other clusters, determines the cluster destination, and determines the destination within a cluster for a job that has been transferred from another cluster. Each node communicates only with its cluster manager. These studies involve the following problems that make them inapplicable for large-scale computing grid environments.

- Each cluster manager needs to have the load information of all other clusters and all nodes of its cluster for making load-balancing decisions. This introduces considerable communication overhead.
- Consulting the cluster managers for job dispatching results in non-negligible overhead and network delay.
- Job migration cost is not considered for load-balancing decision.

- The centralised intra-cluster scheme creates fault-tolerance problems due to single point of failure.

2.4 Summary

This chapter has provided an extensive overview of existing load-balancing methods, with a focus on decentralised load-balancing approaches utilising partial information to make decisions. As discussed in Section 2.3, existing decentralised techniques, which rely on neighbours or clustering, are not applicable in a large-scale heterogeneous computational grid. The survey pointed out opportunities for improving the performance of decentralised load-balancing algorithms, in some cases incorporating good features of neighbour-based and cluster-based models. In Chapters 4–6, we will describe new methods to tackle the issues that the existing approaches do not address—issues that are especially relevant to large-scale heterogeneous computational grids.

3

System model

This chapter presents a scalable, extensible system model for load balancing in a computational grid. The system model lays the groundwork for the load-balancing algorithms discussed in the next few chapters. The model is composed of a (1) grid architecture model, (2) job queue model, (3) communication model, (4) job model, (5) job migration model, and (6) performance objective. The grid architecture model provides a representation and organisation of system resources. The job queue model provides a two-level architecture for the job-waiting queue at each grid site. The communication model provides an estimate of expected communication costs for message exchange and job transfer among grid sites. The job model provides a representation for jobs, and defines the job information needed by the load-balancing algorithms. The job migration model considers techniques for reducing the opportunities for site thrashing and job starvation. The performance metric for evaluating our load-balancing algorithms is given in the performance objective.

3.1 Architecture model

It is assumed that the grid system consists of a collection of sites *S* connected by a communication network (Figure 3.1). The set *S* contains *n* sites, labelled as $s_1,\ldots,s_n$. Logically, the architecture is hierarchical and is divided into four levels: the grid, site, cluster and node levels. The capacity of resource management is different at different levels. The node can be a workstation or a processor. The other three levels are now discussed.

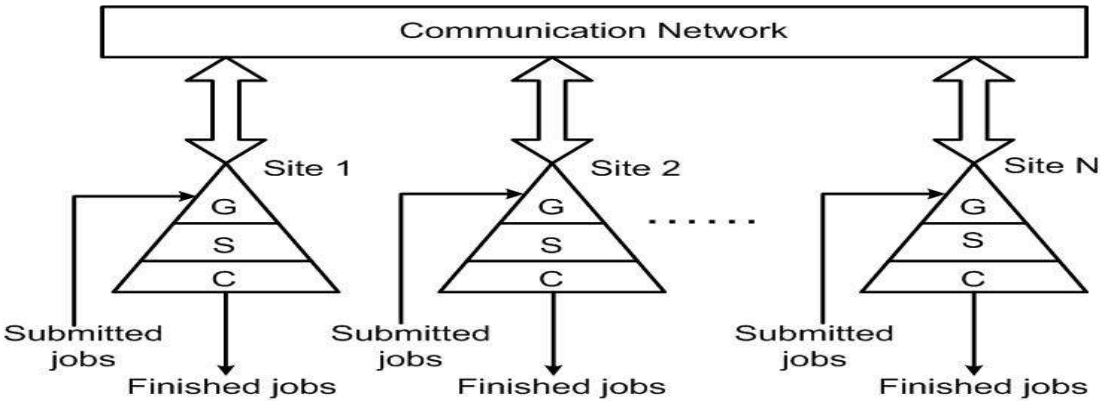

Figure 3.1: Logical view of the grid architecture;
G, S, C are grid, site and cluster levels, respectively.

3.1.1 Cluster level

The cluster level contains a cluster of processors. The processors in a processor cluster share communication bandwidth and are protected by firewalls from the outside world. Processor clusters include tightly coupled multiprocessors such as a Sequent (in which processors communicate via shared-memory), distributed-memory multicomputers such as a Paragon, and loosely coupled workstations such as a Sun 4 cluster (in which processors communicate via message passing).

The management of jobs at cluster level has been addressed by many research and commercial systems, including: Condor [35], Load Sharing Facility (LSF) [30], Portable Batch System (PBS) [31], LoadLeveler [33], Sun grid Engine/CODEINE [32], Maui [29], MOSIX [36], COSY [34]. A comprehensive review of seven commercial packages and 12 research packages is given in [42].

3.1.2 Site level

The Site is an organisational entity. Each site contains a processor cluster. Each site has a broker denoted by the circle (Figure 3.2). On the one hand, each site s_i can be regarded as a whole system, and all of its nodes have a common objective. On the other hand, a site s_i can fully centrally control the resources of its nodes, but cannot directly operate the resources of nodes in other sites. In this view, all nodes are cooperative within the same site.

The site model can be extended to support sophisticated architecture. For example, a site may contain multiple administration domains. Each site has the freedom to choose the number of hierarchical levels and of clusters or resources belonging to each level, such that these numbers will best satisfy its management goals.

To clarify the statement and emphasise our main ideas in the dissertation, we will simplify the model of grid site to one computing node with a single processor. Our scheduling can be easily extended to accommodate these complicated cases.

System heterogeneity can be of different kinds—for example, processor speed, memory and disk I/O. A simpler and more practical solution is to use CPU speed alone. It is reasonable to assume that a machine with a powerful CPU will have matching memory and I/O resources. The sites in the grid system may have different processing power. Processing power of a site s_i is denoted as APW_i. For $i \neq j$, APW_i may be different from APW_j. APW_i is presented as the number of computational units that the site can execute per unit of time. The processing power of a grid site s_i is measured by the average processing power across all processors within the grid site s_i if that site has more than one processor. The most common measure of heterogeneity used in literature is the ratio of processing power of the system nodes [54]. APW_i means the ratio of the average processing power of site s_i to the average processing power of the slowest site s_j in the system—in other words, a job that takes one unit of time on the site s_i requires APW_i units of time on the site s_j.

3.1.3 Grid level

All sites at the grid level are organised in a fully distributed way. There is no central broker in the computational grid. The sites themselves are in a completely connected graph (Figure 3.1). The grid sites are mutually independent. Each grid site communicates only with a subset of grid sites while maintaining load information.

3.1.4 Role of site brokers

The site broker handles all communications with other site brokers via core grid middleware on behalf of the local site, and acts as a grid scheduler. It handles all communication with local scheduler on behalf of remote sites. Site brokers are software processes that can run on a computer node in a cluster or on a separate server node. When the node fails, a predetermined backup node becomes the site broker. The focus of this dissertation is on the design of algorithmic mechanisms for grid schedulers.

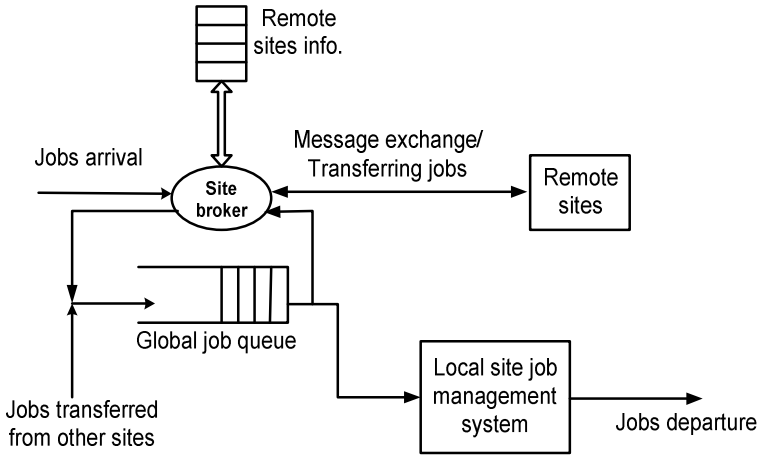

Figure 3.2: Logical structure of a processing site

3.2 Job queue model

We assume that there is a global job-waiting queue at each site that holds those jobs waiting to be assigned to local job management system or a remote grid site (Figure 3.2). Jobs that are submitted to the site are first placed in this queue. The site broker will

determine that the jobs in the global job-waiting queue are processed at local site or at remote sites. If a job is determined to be processed at the local site, it will be transferred to the underlying job management system at cluster level within the site. We use $GJQ(s_i)$ to denote the global job-waiting queue in site s_i. The jobs in the global job-waiting queue are processed in a "first-come-first-serve" order.

The job-waiting queue at site level is different from the job-waiting queue at cluster level. For the following reasons, we used a job-waiting queue at site level.

- The implementation complexity of pulling a job from the job-waiting queue managed by cluster-level job management system can be reduced.
- Different load-balancing algorithms can be implemented at site level and have not any interference with job management system at cluster level. This incurs no extra work for the underlying job management system.

This approach leads to a flexible and portable solution to the existing grid job management system. It is a compromise between the benefits obtained from load-balancing algorithms applied at site level and the implementation complexity introduced in modifying the job management system running at cluster level. Although a trend is starting to occur as vendors adopt a grid perspective to scheduling, by combining pairs of local and grid schedulers into a single scheduler [9, 30, 31, 37], these systems do not interoperate and are not yet widely used.

3.3 Communication model

The sites *S* are fully interconnected, such that there is at least one communication path between any two sites in *S*. The only way that inter-site communication can occur is through message passing. There is a non-trivial transfer delay on the communication network between the sites. The transfer delay is different between different pairs of sites. The underlying network protocol guarantees that messages sent across the network are received in the order sent. The sites are interconnected by point-to-point links. There is no efficient broadcasting service available.

In general, the network performance between any site pair(s_i, s_j) is represented as two parameters: a transfer delay TD_{ij} and a data transmission rate BW_{ij}. The communication time for sending a message of Z bytes between these sites is then given by $TD_{ij}+\frac{Z}{BWij}$, where $\frac{Z}{BWij}$ is the transmission time. The two parameters abstractly represent the total time for traversing all of the links on the path between s_i and s_j. BW_{ij} is presented as effective data transferring rate in bytes per time of unit, or is characterised in terms of Kb/s. TD_{ij} includes a startup cost and delays incurred by contention at intermediate links on the path between s_i and s_j. TD_{ij} and BW_{ij} can be dynamically forecast by what is known as the Network Weather Service [38]. Other research has been proposed on estimating host distance between any two IP addresses [39–41].

3.4 Job model

For any site, $s_i \in S$, jobs are arriving at s_i. We assume that the arrival of jobs is a random process with an average delay, λ^{-1}, between two successive arrivals (e.g., the arrivals could be a Poisson process with rate, λ; that is, the delay between two successive arrivals follows an Exponential law with the same rate of change). The jobs are assumed to be computationally intensive, mutually independent, and can be executed at any site. Job execution is not time-shared, but dedicated. As soon as a job arrives, it must be assigned to exactly one site for processing. When a job is completed, the executing site will return the results to the originating site of the job. We use J to denote the set of all jobs generated at S, $J = \{j_1,\ldots, j_r\}$. The following parameters related to the job are created automatically by the system:

- bornSite(j_i) denotes the originating site of the job j_i
- exeSite(j_i) denotes the executing site of the job j_i
- arrTime(j_i) denotes the arrival time of job j_i, which is the time when the job is generated at bornSite(j_i)
- endTime(j_i) denotes the finish time of j_i; this includes the job communication time from bornSite(j_i) to exeSite(j_i), waiting time queued at the exeSite(j_i), processing times at the exeSite(j_i), and the communication time it takes to return the

processing results from exeSite(j_i) to bornSite(j_i)

- respTime(j_i) denotes the finish time of j_i. respTime(j_i) ≡ endTime(j_i) − arrTime(j_i).

Each job j_x that arrives at a grid site s_i is represented in two parameters: the amount of computation and the amount of communication. The values for these two parameters may be unknown or can be estimated from prediction techniques. The amount of computation normally has one of the following formats.

- An expected execution time $ETC(j_x, s_{std})$, that is, the time that would be taken at a standard platform (with a *APW* equal to 1) for processing that job. On a site s_i with APW_i, the expected execution time of a job $ETC(j_x, s_i)$ will therefore be $ETC(j_x, s_{std})/APW_i$. We assume that the expected execution time $ETC(j_x, s_{std})$ follows a type of probabilistic distribution (for instance, an Exponential, Hyperexponential or Bounded Pareto distribution).
- The number of computation unit in a job j_x is denoted as NCU_x. Thus, the expected execution time for the job j_x on site s_i is $\frac{NCU_X}{APW_i}$.

In a grid environment, the related file of a job needs to be transferred through much slower internet links if the job is scheduled to run in a remote site. Therefore, the cost of file transfers or the amount of communication must be considered in the scheduling algorithm. The amount of communication is calculated in one of two ways.

A. The file size of a job j_x includes input file size A_{1x} and output file size A_{2x}. Assume that, on average, A_{1x} bytes are required to profile a job and that A_{2x} bytes are required to return a response for the job. A_{1x} and A_{2x} are represented as the number of packets needed to be transferred. Thus, the communication time for job j_x needed for transfer purpose is denoted as follows:

$$\text{commTime}(j_x) = T_{com}^{j_x}(s_i, s_j) + T_{com}^{j_x}(s_j, s_i)$$

$$T_{com}^{j_x}(s_i, s_j) = TD_{ij} + \frac{A_{1x}}{BW_{ij}}$$

$$T_{com}^{j_x}(s_j, s_i) = TD_{ji} + \frac{A_{2x}}{BW_{ji}}$$

where $T_{com}^{j_x}(s_i, s_j)$ denotes the communication time of the job j_x from s_i to s_j, and $T_{com}^{j_x}(s_j, s_i)$ denotes the communication it takes to return the processing results from s_j to s_i.

However, due to the changes in the load situations that might occur during the transmission of the job, this job may have to make several moves before it reaches its final destination where it will be processed. Thus, we assume that the job j_x has been transferred from the site s_i to the site s_j through the path $s_i = s_{x_1}, s_{x_2}, \ldots, s_{x_k} = s_j$, where $s_i = s_{x_1}$ and $s_j = s_{x_k}$. The communication cost is given by the following formula:

$$T_{com}^{j_x}(s_i, s_j) = \sum_{q=1}^{k-1} T_{com}^{j_x}(s_{x_q}, s_{x_{q+1}})$$

B. The communication time for running a job in a remote site is set to the computation time divided by *CCR*, where *CCR* is the computation to the communication ratio. By using a range of *CCR* values, different communication time incurred in transit can be accommodated. The computation time is the expected execution time $ETC(j_x, s_{std})$. The communication time means the total of the communication time of transferring a job from its bornsite(j_i) to its final exesite(j_i) and the communication time of sending the execution results from its exesite(j_i) to its bornsite(j_i).

3.5 Job migration model

Because each site scheduler acts independently, there is a small probability that a job can shuttle between sites. This can be prevented in various ways. One approach used throughout our simulation makes the job join not at the end of the queue, but at the position where it is computed by an ageing scheme if the job had arrived at that queue [52]. This can significantly reduce the probability that the job will be transferred again and can guarantee the minimising of its response time. The ageing scheme is given below.

$\forall s_x \in S, \ \forall j_x \in J,$ a job j_x is transferred to the site s_x, its position in the global job queue of the site s_x is defined as

$$POS = \lfloor \frac{QLx}{\zeta} \rfloor$$

$$\zeta = 1 + \frac{age}{\eta}$$

where

- *POS* is the position of the job j_x in the global job queue at site s_x.
- QL_x is the queue length of global job queue at site s_x.
- ζ is an ageing factor and is computed for each job. The ageing factor is used to enhance the probability that an "older" job will start before the jobs that would otherwise start.
- The age of a job is set to 1 when it is moved for the first time, and is incremented by 1 each time the job is moved again.
- η is a constant that can be adjusted empirically to change the extent to which ageing affects the operation of the scheduler.

The approach promotes the position of transferred job in the global job queue of that site s_x, instead of adding it at the end of the queue. This can considerably reduce the probability that the job will be transferred again, and guarantees the minimisation of its response time. We used the approach throughout our simulation to improve the performance of the proposed algorithm.

A more conservative approach was used to reduce the rate at which jobs are moved from one site to another. This can be achieved by restricting the maximum number of jobs transmitted between sites to one job at any given time. This approach is more robust and requires minimal processing time at each site.

3.6 Performance objective

Our major objective is to minimise the average (overall) response time for a collection of jobs, here denoted as *ART*. Minimising the *ART* of the jobs submitted for processing in a parallel/distributed system is a critical performance metric for improving the overall

performance of the system. Many load-balancing algorithms have striven to meet this objective of minimising the *ART* [45, 54, 56, 97, 118, 128].

The average response time for a collection of jobs is defined by:

$$ART = \frac{\sum_{i=1}^{u} responseTime(j_i)}{u}$$

where u represents the total number of jobs completed for evaluation purpose. Note that $u < r$.

To evaluate the performance of our algorithms that developed in Chapter 4–6, we define the improvement factor of algorithm F over another algorithm G as follows in terms of average response time of jobs:

$$\frac{ART(G) - ART(F)}{ART(G)}$$

where *ART(F)* denotes the average response time of jobs using algorithm F. *ART(G)* denotes the average response time of jobs using algorithm G. A positive value of the improvement factor indicates an improvement, while a negative value implies degradation. The value of the improvement factor is presented in terms of percent (%).

3.7 Summary

This chapter has described both a model for presenting grid resource architecture, and a model for presenting job queue. Then a communication model and job model are also presented. These two models define the information needed to construct cost functions for computation and communication. The migration considerations and major performance objectives were then discussed. The system model forms the cornerstone of our load-balancing algorithms that are described in the next three chapters.

4

Decentralised and desirability-aware load-balancing algorithm

This chapter presents a novel load-balancing algorithm for heterogeneous grid systems, with consideration of site desirability. It provides two definitions of site desirability: processing power and communication delay. Using site desirability, a set of partners and neighbours are formed for each site. For each site s_i in the grid, our algorithm uses the desirability of other sites to s_i to form k number of partners and p number of neighbours for s_i. The corresponding approaches for constructing partners and neighbours are also given. A new job arriving at s_i is immediately distributed to s_i or its partner sites. Continuous load adjustment is employed among neighbour sites. To reduce or minimise the state-collection overhead in our load-balancing algorithm, state information exchange is performed via Mutual Information Feedback (MIF). Our algorithms are dynamic, sender-initiated and decentralised.

Section 4.1 describes in detail a desirability-aware load-balancing algorithm. In Section 4.2, the performance of our algorithm is evaluated in a series of simulations.

4.1 Desirability-aware load-balancing algorithm

4.1.1 Load index

Most algorithms in the literature have used the instantaneous run-queue length (the number of jobs being served or waiting for service at the sampling instant) as the load index [56, 136]. The load index is easily obtained and calculated with minimum overhead. Thomas Kunz [136] reported that the simple CPU queue length load index is the most effective. If a site has more than one CPU, a simple modification is to divide the total queue length by the number of CPUs.

4.1.2 Site desirability

Our objective is achieved by using site desirability to guide load assignments. Site desirability is based on how site characteristics will affect the performance of future load balancing. We give two definitions of the desirability of s_j to s_i: desirability based on the average processing power of s_j, and desirability based on transfer delay between s_j and s_i.

4.1.3 Site-clustering algorithm

Here, the site desirability of average processing power accounts for the site cluster. Our site-clustering algorithm uses a set of reference sites of size m. The reference sites are chosen at random, with the only condition that the reference sites be separated from each other by a big enough difference in processing power to avoid the situation where two identical or very close reference sites are chosen. Similar approaches have been widely used to generate proximity information [e.g., 41, 137]. This information is based on the intuition that sites close to each other in processing power are likely to be similar distances from several selected sites.

These reference sites are sorted by *APW* in descending order before applying sites clustering approach. For each grid site s_i, the clustering algorithm first measures the difference in *APW* of site s_i to the reference sites and calculate a reference vector $<d_1, d_2,\ldots, d_m>$. Two grid sites with similar reference vector are 'close' to each other in terms of average processing power. The grid sites are then clustered into $C_1, C_2, \ldots, C_m$ clusters. Finally, empty clusters in $C_1, C_2, \ldots, C_m$ are removed so that we have C_1,

C_2, …, C_q ($q \leq m$), which are also in decreasing order of *APW*. We denote the cluster ID containing s_i as Ω_i, with values of positive integers between 1 and q. Sufficient reference sites will be required to reduce the probability of false clustering where sites that have very different processing power have similar/close reference vectors.

The approach outlined above is a coarse-grained approximation and is not effective in differentiating closely located sites. Nevertheless, our simulation results show that the method works well for our load-balancing scheme. This is largely because our load-balancing scheme does not require very precise measurements. The clusters generated are then used to generate partner sites, as described in 4.1.4.

Algorithm 4.1 (Procedure FindPartners(s_i, k))

Find all sites $s_j \in S$ ($i \neq j$) with $\Omega_j < \Omega_i$. Denote this set of sites as Q_i
If $\gamma \geq k$ /* γ is the size of Q_i */
  Select k sites from Q_i randomly and add them to $PSet_i$
Else {
  $l \leftarrow k - \gamma$
  Add Q_i to $PSet_i$.
  $v \leftarrow \Omega_i$
  While $l > 0$ {
    $Q_i \leftarrow Q_i \cup C_v$
    If $\gamma^+ \geq l$ { /* γ^+ is the size of C_v */
      Select l sites from C_v randomly and add them to $PSet_i$
      Break
    }
    Else {
      Add C_v to $PSet_i$.
      $l \leftarrow l - \gamma^+$
      $v \leftarrow v+1$
    }
  }
}

4.1.4 Partners

Each site s_i automatically maintains k number of partner sites $PSet_i$, which the site scheduler will use to select a partner site for processing new arriving jobs. When a site joins the grid system, it will determine its partners. We employed a simple heuristic to

find partner sites in terms of their processing power. It is natural to consider more powerful sites as partners. Here we consider the heterogeneity of sites. The optimal partners are sites with lightly loaded and greater average processing powers. Pseudo-code for our partners' selection procedure is given in Algorithm 4.1. Q_i is a preferred collection of sites of s_i and are also used in our Partners Adjustment Policy. The sites in Q_i have greater or comparable processing power to site s_i. In the algorithm, the set of preferred sites Q_i may be updated as necessary. Although the approach described here does not guarantee the finding of optimal partners, the methodology provides a scalable and performance-efficient approach to the initial formation of partner sites.

4.1.5 Neighbours

Each site s_i maintains p number of neighbouring sites, $NSet_i$, which the site scheduler will use to select a neighbouring site for offloading jobs. This can reduce the cost of load movement, and enable quick response to load imbalances. Neighbours for each site are formed in terms of the site desirability of transfer delay. For s_i, s_j is considered as its neighbouring site as long as the transfer delay between s_j and s_i is within ε times of the transfer delay with high probability between s_i and the nearest site. For each site, the other sites are sorted by transfer delay in ascending order. After this process, the first-ranked site is chosen as the nearest site. This is described as follows:

$$\varepsilon = \frac{TDji}{TDnearest}$$

where TD_{ji} denotes the transfer delay from site s_j to s_i. $TD_{nearest}$ denotes the transfer delay from the nearest site of site s_i to itself.

Any number of set relationships between $PSet_i$ and $NSet_i$ is possible, including intersect, disjoint and include.

4.1.6 Partners Adjustment Policy

The dynamic Partners Adjustment Policy is triggered whenever a site s_i receives load information message from a neighbour or partner. If a site s_j in the preferred sites Q_i of s_i is found in the message, it will be involved in the partner adjustment of s_i. It is possible that s_j becomes a partner site of site s_i if its load is lower than the highest load in the partner sites of s_i. Algorithm 4.2 describes the procedure of Partners

Adjustment Policy when s_j receives an information message from its neighbour or partner site s_i.

Algorithm 4.2 (Procedure PartnersAdjustment (s_j, k)):

$S_1 \leftarrow \phi$
$\forall s_y \in Y$: If ($s_y \notin NSet_j \cup PSet_j$) AND ($s_y \in Q_j$) $S_1 \leftarrow S_1 \cup s_y$
If $S_1 \neq \phi$ {
    $S_1 \leftarrow S_1 \cup PSet_j$
    Sort S_1 by LD in ascending order
    Remove all sites from $PSet_j$
    Select the first k sites from S_1 and add them to $PSet_j$
}

4.1.7 Information policy

We use MIF for load-state information exchange. The information exchange is restricted to partners and neighbours. Algorithm 4.3 describes the procedure when a site s_i transfers a job j_x to its neighbour or partner site s_j for processing. s_i appends the load information of itself and ω_P (a small positive integer) random neighbours or partners to the job transfer request TR sent to s_j by piggybacking. s_j then updates the corresponding load information in its state object by comparing the timestamps, if the sites contained in the transfer request belong to its neighbours or partners. Similarly, s_j inserts its current load information, and ω_P random sites from its $NSet_j$ and $PSet_j$ in the job acknowledge AR or completions reply CR to s_i, so s_i can update its state objects.

For any site $s_i \in S$, if the state object element $O_i[j]$ ($\forall s_j \in NSet_i \cup PSet_i$, $i \neq j$) has not been updated for a predefined period T_P, then the load-balancing scheduler will send an information exchange message to s_j. The procedure is the same as the algorithm 4.3.

The MIF method is an alternative to the periodic information exchange method and its correspondingly high messaging overhead. Under the MIF method, the processing site will return its current load and the load of ω_P random sites, along with the ACK message or completion reply CR, back to the forwarding site. As such, the overhead is minimal.

The MIF method has another advantage: the rate of load dissemination is directly tied to the job arrival rate. An increase in the job arrival rate means that each node receives initial job requests more frequently, which means that each node forwards job requests more frequently and in turn receives load information more frequently. Therefore, the load dissemination rate is automatically adjusted to the request rate.

Algorithm 4.3 (**procedure of job transfer and information exchange**):

Steps processed in s_i:

1. $Y \leftarrow s_i + \{\omega_P$ random sites from $NSet_i \cup PSet_i - s_j\}$
   // s_i select neighbours or partners for information exchange
2. $\forall s_y \in Y$, s_i appends ($O_i[y].LD$, $O_i[y]$.LT) to the job transfer reques*t TR*
3. s_i appends bornsite(j_x) to *TR*
4. s_i sends message *TR* to s_j

Steps processed in s_j:
Upon receiving *TR*:

1. $\forall s_y \in Y$: If ($O_i[y].LT > O_j[y].LT$) AND ($s_y \in NSet_j \cup PSet_j$) $O_j[y] \leftarrow O_i[y]$
   // s_j updates the state object using s_i's info
2. $Z \leftarrow s_j + \{\omega_P$ random sites from $NSet_j \cup PSet_j - s_i\}$
3. $\forall s_z \in Z$, s_j appends ($O_j[z].LD$, $O_j[z].LT$) to the acknowledge reply *AR*
4. s_j sends messag*e AR* to s_i

Upon completion of job j_x:

1. $Z \leftarrow s_j + \{\omega_P$ random sites from $NSet_j \cup PSet_j - s_i\}$
2. $\forall s_z \in Z$, s_j appends ($O_j[z].LD$, $O_j[z].LT$) to the completion reply *CR*
3. s_j sends message *CR* to bornsite(j_x) = s_i

Steps processed in s_i:
Upon receiving the reply *AR* or *CR*:

$\forall s_z \in Z$: If ($O_j[z].LT > O_i[z].LT$) AND ($s_z \in NSet_i \cup PSet_i$) $O_i[z] \leftarrow O_j[z]$

4.1.8 Transfer policy and location policy

Our transfer and location policies are a combination of two policies—Instantaneous Distribution Policy (IDP) and Load Adjustment Policy (LAP). These are described below.

4.1.8.1 Instantaneous Distribution Policy

When a new job arrives at site s_i, the policy decides whether it is to be sent to the

global job queue of s_i or one of its partner sites. If the existing partner sites are already overloaded, it is placed in the global job queue of s_i and involved in load balancing performed by another policy at a later time (Section 4.1.8.2). The policy has two advantages: First, the policy try to control the job processing rate on each site in the system; Second, the policy makes more powerful sites carry more loads, and jobs executed at fast sites are more likely to execute at a high speed. If there are two partner sites with the same minimum load, the nearest partner site is chosen. Algorithm 4.4 describes the IDP for s_i.

Algorithm 4.4 (Instantaneous Distribution Policy):

$\forall j_x \in J$ with bornSite(j_x) = $s_i \in S$
    Let $LD_{min} \leftarrow$ Min{$O_i[k].LD \mid s_k \in s_i + PSet_i$} /* the minimum load among site s_i and its $PSet_i$ */
    If ($O_i[i].LD - LD_{min} < \theta$) /* θ is a positive real constant close to zero */
        $GJQ(s_i) \leftarrow$ enqueue(j_x) /* put the job j_x in the global job queue $GJQ(s_i)$ */
    Else {
        Transfer the job j_x to the partner site s_j having LD_{min}
        Update $O_i[j].LD$
    }

Algorithm 4.5 (Load Adjustment Policy):

If $O_i[i].LD > LD_{avg}$ {
    $j_x \leftarrow$ dequeue(GJQ(s_i))
    Transfer the job j_x to a neighbour site s_j where $O_i[j].LD$ = Min{$O_i[k].LD \mid s_k \in NSet_i$}
}

4.1.8.2 Load Adjustment Policy

The Load Adjustment Policy for a site s_i tries to continuously reduce load difference among s_i and its neighbours $NSet_i$ by transferring jobs from heavily loaded sites to lightly loaded neighbouring sites. The policy is triggered whenever s_i receives load information from a neighbour. The policy will use the most recent load status information to decide whether a transfer is initiated. An adaptive threshold policy is used so that the thresholds are adjusted as the system load changes. Sites with loads

that are higher than the average load are considered as senders. Once s_i makes a transfer decision, the last job waiting in $GJQ(s_i)$ is considered first for transfer. If there are two neighbouring sites with the same minimum load, the faster neighbouring site is chosen. The algorithm 4.5 describes the LAP for a site s_i.

4.2 Performance evaluation

We consider only sender-initiated algorithms. In the simulation, our algorithm (labelled as DA) is compared with the Neighbours-based load-balancing algorithm (e.g., [105, 106]) (labelled as NB). For the NB, each site is limited to load information from within its own domain, which consists of itself and its neighbours. If the load of a site load exceeds the computed average load in its domain, load balancing is initiated. We select the algorithm because it represents a typical class of decentralised approaches and bears similarity to our work. The definition of neighbours and approach for information update in the NB is the same as ours.

4.2.1 Simulation model

In this section, we use simulations to study the performance of the algorithms under different system parameters. Nine assumptions were devised for the simulation model. These are:

- All of the work is carried out on a grid system that consists of n sites. The average processing power of sites is assigned in a specified range. According to the different degrees of heterogeneity in the average processing power of sites, several different heterogeneous systems are produced.
- The reference sites are chosen randomly and are separated from each other at least D_f percent difference in processing power.
- Jobs arrive at each site s_i, i=1, 2,…, n according to a Poisson process with rate $\lambda_i = \lambda \times P_i$, where P_i=1/n. The actual inter arrival time of jobs is adjusted to give the required overall average system loading (see last bullet point).
- The expected execution times of jobs are assumed to follow an exponential distribution with a mean of X time unit.
- The transfer delay that may be incurred between any site pairs in the grid system is chosen from a lognormal distribution with a mean of τ time unit and

a standard deviation σ_c.

- The *CCR* is chosen randomly in a specified range.
- The partner set of each site need to be provided before our algorithm starts to run. It is based on the method described in Section 4.1.3.
- The neighbours of each site are fed to the simulator before the algorithms starts to run. They are based on the transfer delay generated from the distribution of mean transfer delay. The sites are chosen as its neighbours, according to the method described in Section 4.1.4.
- Let ρ be the required average system load for our simulation, which is the average job arrival rate divided by the average job processing rate. Using this definition, we adjust the job mean inter-arrival time $1/\lambda$ required to get the desired ρ.

Table 4.1: Simulation parameters (tu = time unit, pt = percent)

Simulation parameter	Value
Size of system, *n*	32
The number of reference sites, *m*	12
The difference of *APW* among reference sites, D_f	10 pt
Mean processing time of system jobs, *X*	1.0 tu
The computation to communication ratio, *CCR*	{0.1, 02, 0.5, 1, 2, 5, 10}
Mean transfer delay, τ	0.05 tu
Standard deviation of transfer delay, σ_c	50 pt
Distance coefficient to from a site to its nearest site, ε	1.5
Period for periodic information exchange, T_p	10 tu
Number of partners, *k*	4
Number of random partners/neighbours for information update, ω_P	2

Table 4.1 shows the values of the parameters used in the simulations. Table 4.2 shows heterogeneous system configurations, in which the third column contains the *APW* of each of the four site types. The second column contains four site types and the number of sites in the system corresponding to each site type. An exception for heterogeneous system HS3 is that a value is first randomly chosen from a range of [10, 100], and then the value is divided by 10 to generate the *APW* of a site. We used the first 2000 jobs to bring the system to a steady state. We traced the arrival time, processing time

and finish time from j_{2000} to j_{9999}. Here u equals to 8000 (for evaluation purpose). The simulation does not end until all jobs between j_{2000} and j_{9999} have completed. For each site, we recorded the number of completed jobs. After each simulation run, we computed the average response time of jobs (*ART*). We carried out each measurement five times with different random seeds.

Table 4.2: Heterogeneous system configurations

Heterogeneous systems	Sites split (fraction)	*APW*
HS1	3/8, 1/4, 1/4, 1/8	[1, 2, 5, 10]
HS2	12, 8, 6, 6	[1, 1.5, 2, 3]
HS3	Random	[10, 100] / 10

4.2.2 Effect of system heterogeneity

We carry out a series of simulations with the algorithms described above for three different heterogeneous systems shown in Table 4.2, under different system utilisation parameter ρ.

We first considered only situations where the fastest sites have up to 10 times higher relative processing power than the slowest site, because this is true of most of the current heterogeneous grid systems. In Table 4.2, we present a highly heterogeneous system configuration HS1 with four different processing powers. We varied the system loading by varying the mean inter-arrival time (initiation time) of the jobs, $1/\lambda$. Results are shown in Figure 4.1 and Table 4.3.

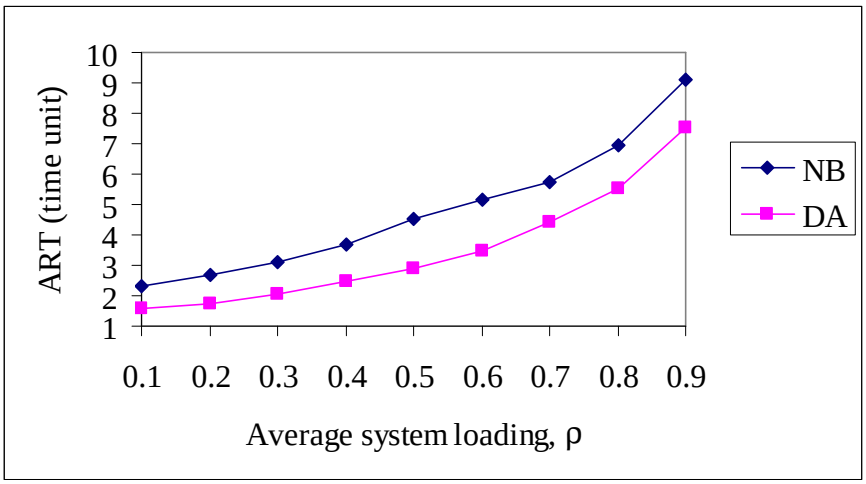

Figure 4.1: Effect of system with high heterogeneity

Table 4.3: Improvement factor (in percent) of DA over NB in Figure 4.1

	Average system loading, ρ				
	0.1	0.2	0.3	0.4	0.5
NB	33.34	35.32	34.41	33.69	36.92
	Average system loading, ρ				Average (0.1–0.9)
	0.6	0.7	0.8	0.9	
NB	32.62	22.96	20.72	17.14	29.68

We can conclude that NB behaves poorly in a highly heterogeneous system. DA gives the minimum *ART* across all values of ρ. At light or medium system loading (10–60%), DA performs significantly better than NB. For example, at system loading of 50%, the *ART* using DA is 36.92% less than NB and the difference reaches the highest point. When the system loading becomes high, the difference between the *ART* of NB and DA decreases. At high system loading of 90%, DA yields the *ART*, which is 17.14% less than NB. DA has an average improvement factor of 29.68% over NB. Analysis of the results revealed the following reasons for the relative performance of each algorithm in terms of the *ART*.

- When the system loading is light or moderate for DA, IDP plays a crucial role and LAP makes little influence on the *ART* of the jobs. NB transfers a job to an idle neighbouring site, which can be much slower in a highly heterogeneous system than a faster non-neighbouring site that has only a small amount of jobs in the queue (or that is currently processing a job and has an empty queue).
- At high system loading, NB tends to dispatch jobs to a neighbouring site independently of its relative processing power. In contrast, DA first dispatches new jobs to faster partner sites with minimum load, which means that it is more likely for faster sites to shorten the queue length and reduce the *ART*. As a result, the *ART* is smaller compared with NB, which contributes to a better *ART*.

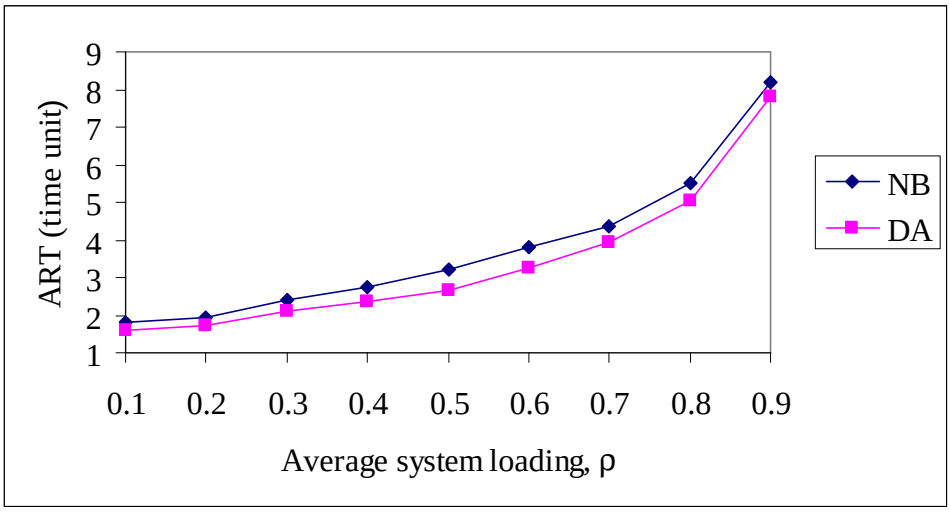

Figure 4.2: Effect of system with low heterogeneity

Table 4.4: Improvement factor (in percent) of DA over NB in Figure 4.2

	Average system loading, ρ				
	0.1	0.2	0.3	0.4	0.5
NB	17.50	20.97	14.58	17.25	19.17
	Average system loading, ρ				Average (0.1–0.9)
	0.6	0.7	0.8	0.9	
NB	18.30	15.54	8.86	5.73	15.32

Secondly, we focused our analysis on the case where the system is much less heterogeneous. We consider a low heterogeneous system HS2 that the processing power of the fastest site is only three times as high as the processing power of the slowest site. By observing the results shown in Figure 4.2 and Table 4.4, we conclude that DA has a lower *ART* than NB under all loads, but that the difference is not significant. DA has an average improvement factor of 15.32% over NB. The conclusion is due to the fact that IDP makes main contribution at low or medium system loading.

Lastly, we studied a system with randomly generated heterogeneity. We randomly generated one heterogeneous grid system HS3 and computed the *ART* by using NB and DA. On average, DA performs better than NB (Figure 4.3 and Table 4.5), by 22.41%.

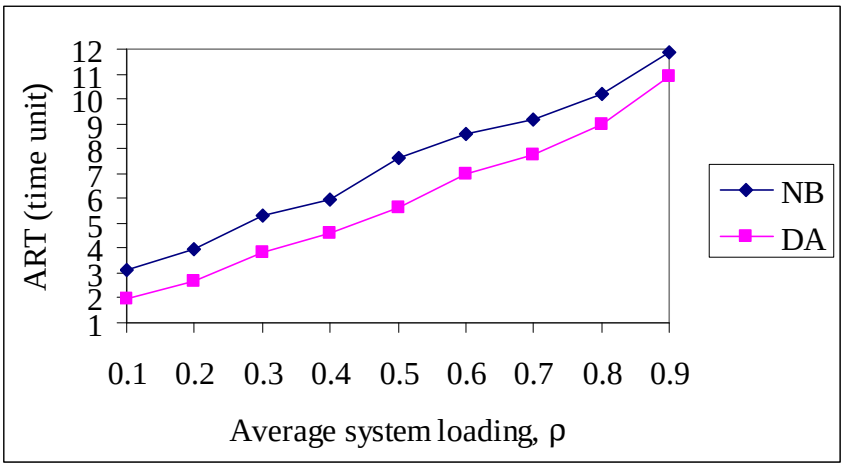

Figure 4.3: Effect of system with randomly generated heterogeneity

Table 4.5: Improvement factor (in percent) of DA over NB in Figure 4.3

	Average system loading, ρ				
	0.1	0.2	0.3	0.4	0.5
NB	38.29	32.07	28.07	23.09	26.41
	Average system loading, ρ				Average (0.1–0.9)
	0.6	0.7	0.8	0.9	
NB	18.45	15.44	11.50	8.33	22.41

4.2.3 Job completion statistics

One of the aims of this study was to obtain insight into how best to allocate the workload among the component sites of a heterogeneous grid system. This could then be used to develop heuristics for designing and evaluating load-balancing algorithms. This section looks at the utilisations of the different class of sites in the HS1 and HS2 heterogeneous systems, at overall system loadings of 0.5 and 0.9, for the two algorithms DA and NB, as shown in Figures 4.4–4.7. The fractions of total jobs are computed as the total number of jobs that completed at each type of sites divided by *u*.

At the average system loading of 0.5 for HS1 and HS2 heterogeneous systems, DA tends to underload the less powerful sites, and thus allocates a disproportionate share of the work to the more powerful sites. The difference between the algorithms is more marked for the highly heterogeneous system HS1, where there are more powerful sites. However, NB tends to overload the weaker sites (50% system loading); this is because NB algorithm does not consider heterogeneity in average processing power among sites while assigning load. At the system loading of 0.9 for HS1 and

HS2 heterogeneous systems, NB reduces load at weaker sites and increases load at more powerful sites. On the contrary, DA increases some loads at weaker sites and reduces some loads at more powerful sites, because jobs may transfer to a less powerful neighbouring site.

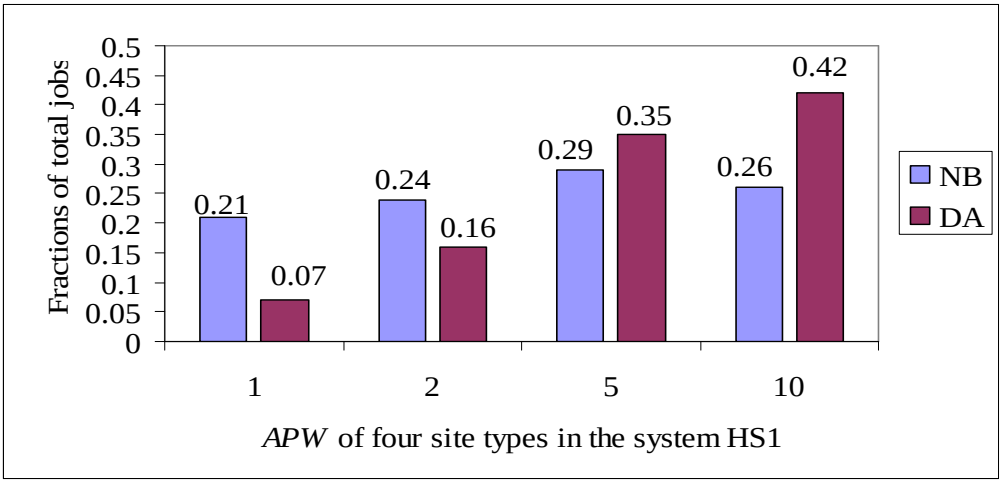

Figure 4.4: Job completion statistics at the system loading of 0.5 in the system HS1

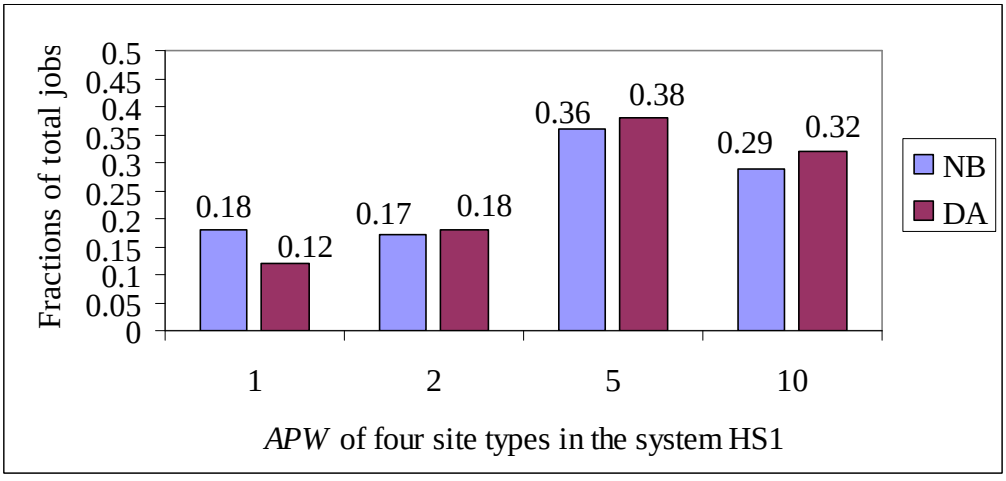

Figure 4.5: Job completion statistics at the system loading of 0.9 in the system HS1

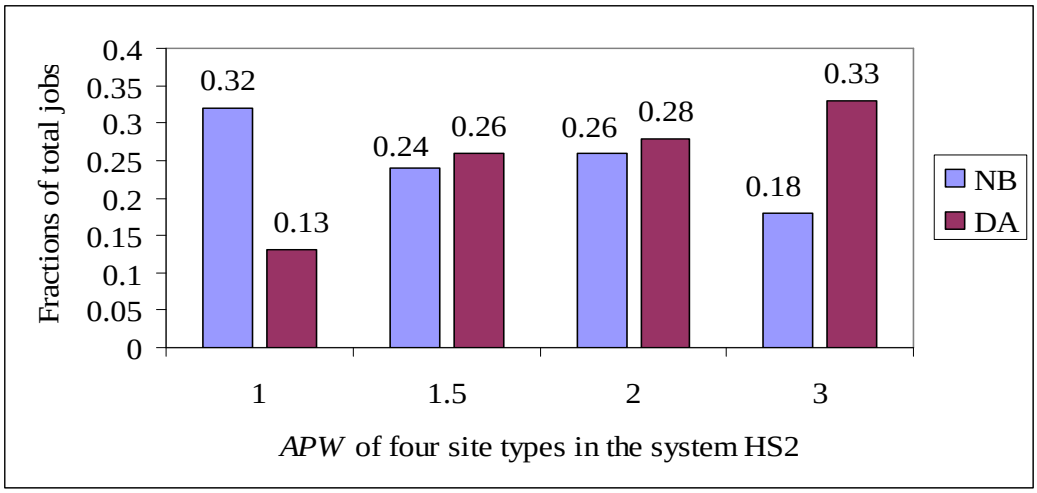

Figure 4.6: Job completion statistics at the system loading of 0.5 in the system HS2

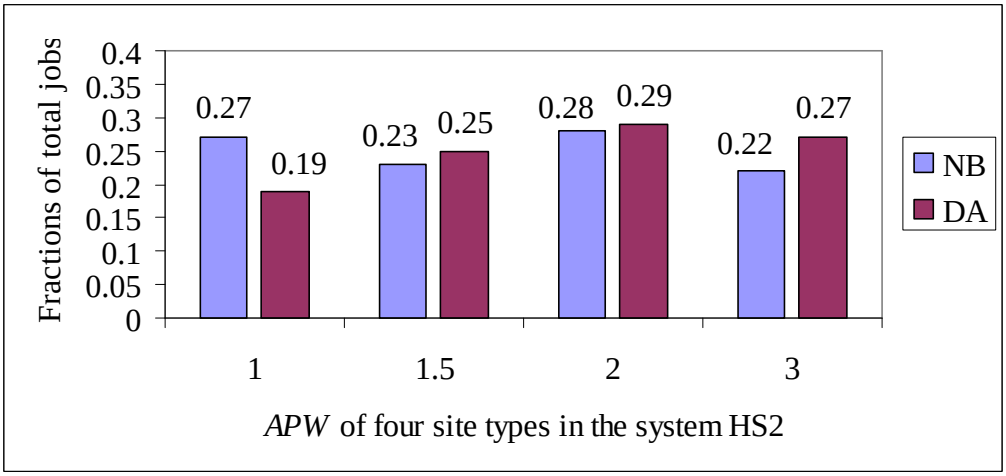

Figure 4.7: Job completion statistics at the system loading of 0.9 in the system HS2

4.2.4 Scalability

To explore how the size of the system (in terms of computing sites) affects the performance of the algorithms according to *ART*, we tried to simulate them at system loadings of 0.5 and 0.9 while increasing the number of the sites. Results are presented for system sizes from 16 to 450. We created 18 different systems, with sizes of 16, 25, 32, 50, 64, 80, 100, 120, 140, 160, 200, 220, 250, 280, 300, 350, 400, and 450. For each system size, we used the heterogeneous system HS3. The performance of both algorithms at the system loading of 0.5 and 0.9 is given in Figure 4.8–4.9 and Table 4.6–4.7. The improvement that DA offers to NB varies from 13.47–27.33% at the

system loading of 0.5, and from 3.78–12.53% at the system loading of 0.9, depending mainly on the heterogeneity of system. DA had an average improvement of 19.23% and 7.88% over NB at system loading of 0.5 and 0.9, respectively.

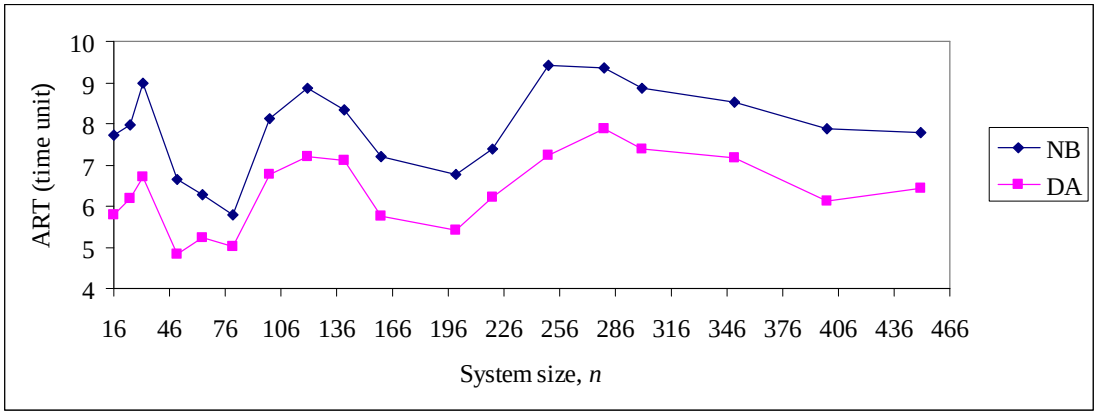

Figure 4.8: Evaluation of scalability with different size at the system loading of 0.5

Table 4.6: Improvement factor (in percent) of DA over NB in Figure 4.8

	System size, *n*						
	16	25	32	50	64	80	100
NB	24.97	22.21	25.31	27.33	16.56	13.47	16.62
	System size, *n*						
	120	140	160	200	220	250	280
NB	18.53	14.75	20.24	19.68	15.47	23.44	15.53
	System size, *n*				Average		
	300	350	400	450			
NB	16.80	15.73	22.21	17.33	19.23		

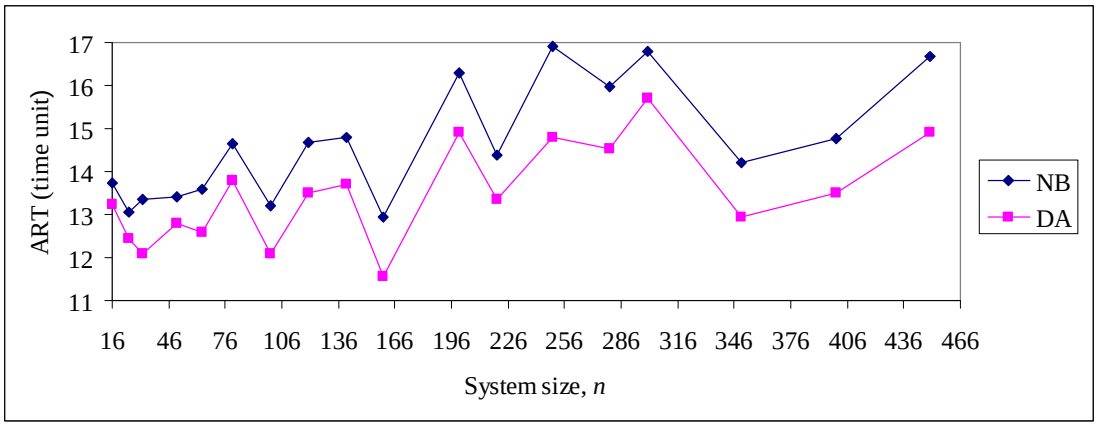

Figure 4.9: Evaluation of scalability with different size at the system loading of 0.9

Table 4.7: Improvement factor (in percent) of DA over NB in Figure 4.9

	System size, *n*						
	16	25	32	50	64	80	100
NB	3.78	4.59	9.43	4.48	7.21	5.87	8.40
	System size, *n*						
	120	140	160	200	220	250	280
NB	7.98	7.31	10.60	8.58	7.30	12.53	9.02
	System size, *n*				Average		
	300	350	400	450			
NB	6.55	8.94	8.60	10.73	7.88		

4.2.5 Effect with different job arrival patterns

All the results discussed in the previous simulations are generated under the assumption that all sites have the same job arrival rate. In reality, job arrival rates usually differ from one site to another. To evaluate the effect of different job arrival rate on the *ART*, we have conducted another simulation, in which we randomly chose ten of the sites as lightly loaded site (ρ = 0.3), eleven of the sites as moderately loaded sites (ρ = 0.6) and eleven of the sites as highly loaded sites (ρ = 0.9). The HS1, HS2 and HS3 system setting were used as the base configurations in the section. The results shown in the Figure 4.10 and Table 4.8 illustrate that DA has an average improvement of 27.13%, 11.19% and 19.23% over NB in the HS1 system, the HS2

system and the HS3 system, respectively.

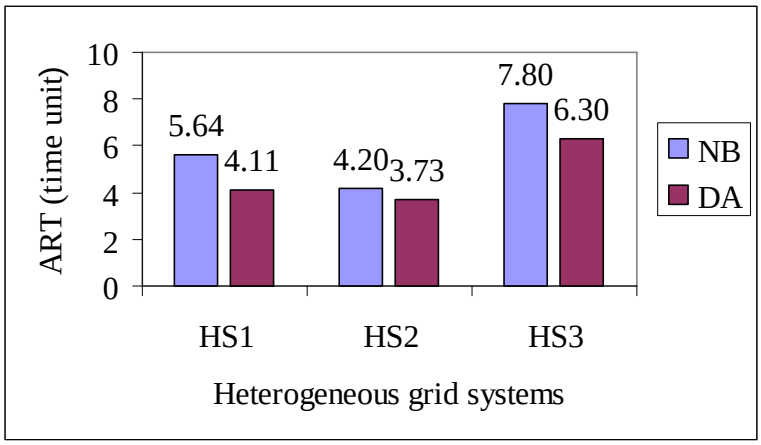

Figure 4.10: Effect with different job arrival patterns in three different kinds of heterogeneous grid systems

Table 4.8: Improvement factor (in percent) of DA over NB in Figure 4.10

	HS1	HS2	HS3
NB	27.13	11.19	19.23

4.2.6 Sensitivity to the variance in job inter-arrival times

The burstiness of the job arrivals may cause performance deterioration [48, 56]. The burstiness can be measured by the coefficient of variation (*CV*) of the job inter-arrival times (the *CV* of a random variable is calculated as its standard deviation divided by its mean) [48, 56]. For a Poisson arrival, the *CV* of the inter-arrival times is 1. However, the job arrivals in real environments tend to be burstier than this. Job traces in a real computing system are analysed in [56], which showed that the *CV* of the inter-arrival times is 2.64. Job arrivals can be modelled using a Hyperexponential distribution [48].

In this simulation, the system HS3 is used. The job arrivals are modelled by a two-stage hyperexponential distribution, in which the *CV* can be adjusted by changing the distribution parameters. Figure 4.11 and 4.12 show the effect of the *CV* of the inter-arrival times on the *ART*, where the *CV* increases from 1.0–3.0 with increments of 0.5, while the average arrival rate remains unchanged. We show only the results for

the case where the average system loading is 0.5 and 0.9 as the results for other workload levels demonstrate similar patterns.

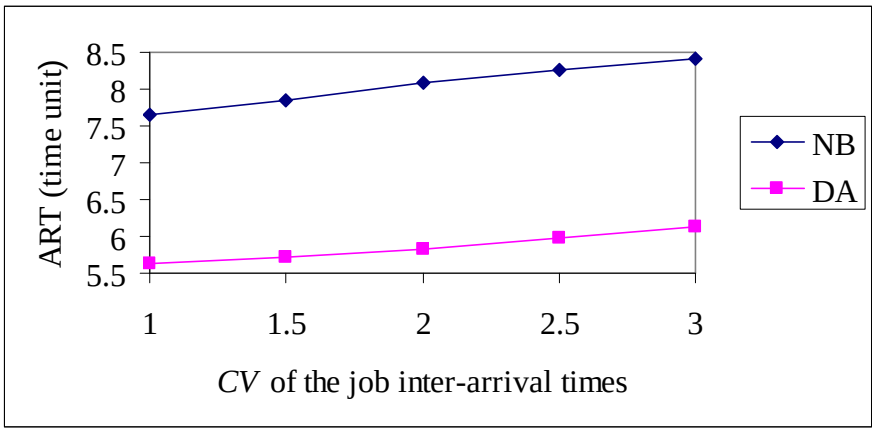

Figure 4.11: Sensitivity to the variance in job inter-arrival times (at the system loading of 0.5)

Table 4.9: Improvement factor (in percent) of DA over NB in Figure 4.11

	CV of the job inter-arrival times					Average
	1	1.5	2	2.5	3	
NB	26.41	27.04	27.85	27.81	27.20	27.26

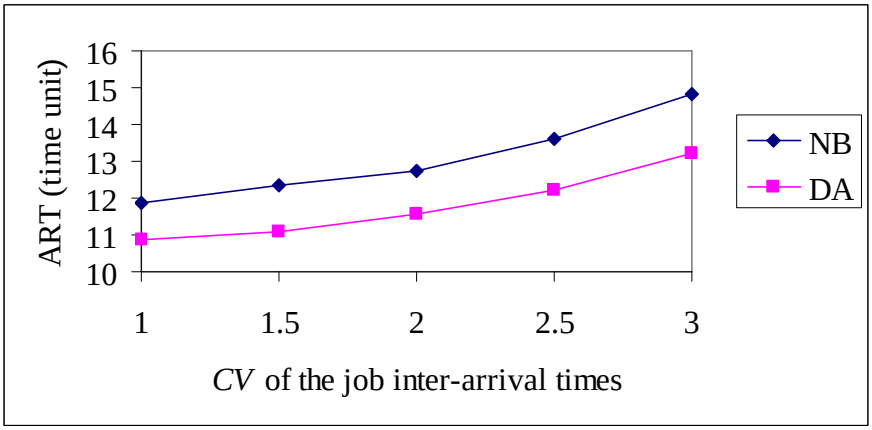

Figure 4.12: Sensitivity to the variance in job inter-arrival times (at the system loading of 0.9)

Table 4.10: Improvement factor (in percent) of DA over NB in Figure 4.12

	CV of the job inter-arrival times					Average
	1	1.5	2	2.5	3	
NB	8.33	10.12	9.04	10.21	10.85	9.71

It can be seen in Figure 4.11 and 4.12 that, under these two algorithms, the *ART* increases as the *CV* increases, as is to be expected. When the system loading is 0.5 (Table 4.9), DA outperforms NB by 27.26%, while there still is an average improvement factor of 9.71% over NB at the system loading of 0.9 (Table 4.10). These results suggest that the burstiness of job arrivals does not notably impair the advantages of the DA workload allocation strategy.

Figure 4.13 shows the *ART* as a function of the workload level when the *CV* is set to 3.0. The performance curves of DA and NB demonstrate similar patterns to those previously seen in Figure 4.3. When the system loading is 0.1 (Table 4.11), DA outperforms NB by 38.87%, while the advantage is 10.85% when the system loading is 0.9. These results indicate once again that DA consistently performs better than NB, even if there is higher burstiness in job arrivals.

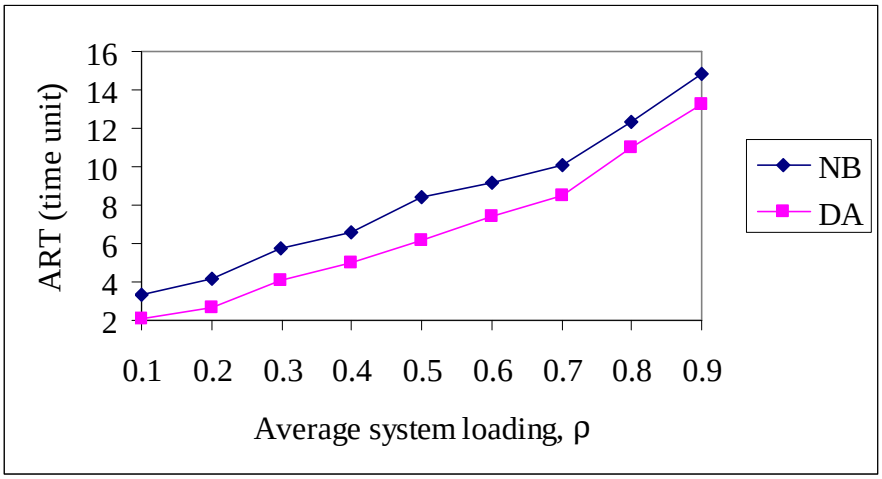

Figure 4.13: Sensitivity of performance to different system loading (*CV* equals to 3)

Table 4.11: Improvement factor (in percent) of DA over NB in Figure 4.13

	Average system loading, ρ				
	0.1	0.2	0.3	0.4	0.5
NB	38.87	36.43	28.72	24.32	27.20
	Average system loading, ρ				Average (0.1–0.9)
	0.6	0.7	0.8	0.9	
NB	19.32	15.71	10.56	10.85	23.55

4.2.7 Sensitivity to highly variable job sizes

We assume Exponentially distributed job sizes in previous simulations. The studies [80, 109] have found that job size distributions exhibit a heavy-tailed property in most computing systems. In this section, we examine jobs with a Bounded Pareto distribution that has considerably more variability in job size. We use a Bounded Pareto distribution to bound the mean job size while retaining a large variance of job sizes.

The probability density function of the Bounded Pareto Distribution *B(min, max,*α*)* is defined as follows [109]:

$$f(x) = \frac{\alpha \, min^{\alpha}}{1-(min/max)^{\alpha}} x^{-\alpha-1} \qquad min \le \alpha \le max$$

where *min* and *max* are the lower and upper bounds of job size, respectively, and α is a parameter that reflects the variability of job size.

Figure 4.14 shows performance under a Bounded Pareto workload (α = 1.2, *min* = 0.203901, max = 1000) with three different values for the arrival rate λ and the *max* value. This means that the maximum job size is 1000 times the average job size; *min* was chosen to set the mean job size at 1.0 for these values of α and *max*.

Once again, DA performs substantially better than NB over a range of situations of system HS3 loading. When the system loading is 0.1 (Table 4.12), DA outperforms NB by 39.44%, while the advantage is 8.73% when the system loading is 0.9. This is mainly because, the large jobs are dispatched to faster site for processing with high probability. For NB, the larger jobs may be placed at slower sites for

processing. This suggests that site selection may play a more important role under workloads with highly variable job sizes. Thus, NB is more sensitive to the highly variable job sizes and it is more likely to reduce the *ART* for DA.

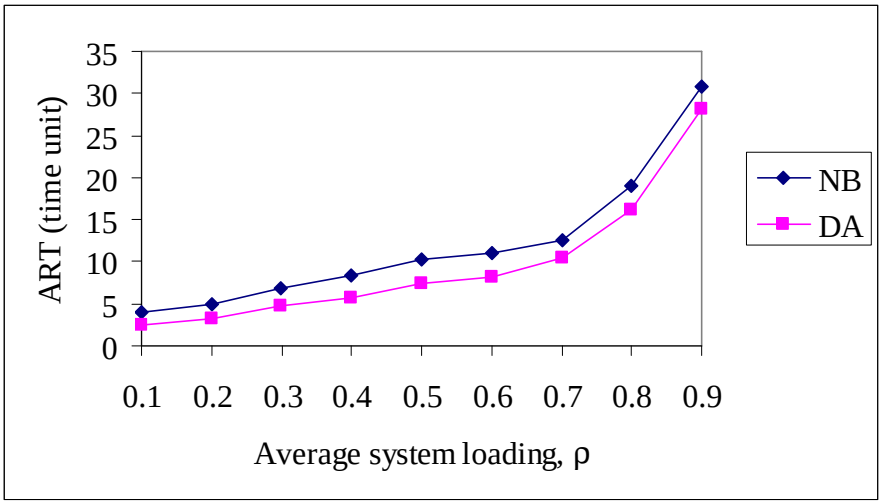

Figure 4.14: Sensitivity of performance to highly variable job size

Table 4.12: Improvement factor (in percent) of DA over NB in Figure 4.14

	Average system loading, ρ				
	0.1	0.2	0.3	0.4	0.5
NB	39.44	36.91	29.31	31.33	27.84
	Average system loading, ρ				Average (0.1–0.9)
	0.6	0.7	0.8	0.9	
NB	26.02	16.35	14.68	8.73	25.62

4.2.8 Sensitivity to the transfer delay τ

The delay in load information update is an important issue in grid load balancing. The lag time in disseminating load information is typically much larger than that in a LAN and can vary greatly due to network congestion. The probability that the messages carry up-to-date information depends on the distance between the sending and receiving site and on the load dynamics at the sending site. Therefore, it is necessary to study the effect of transfer delay on the load balancing.

In this simulation, we compared information exchange policy via MIF with Periodic information exchange policy. The Periodic information policy here means each site disseminates its load information at regular interval to all its neighbouring sites and all sites that has chosen them as partners. The value of time interval for

Periodic information policy was set to 1 time unit.

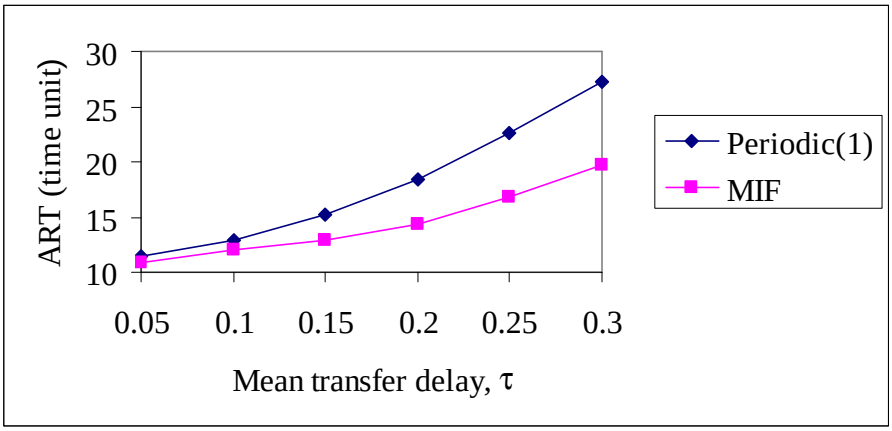

Figure 4.15: Sensitivity of information exchange policy MIF to the transfer delay

Table 4.13: Improvement factor (in percent) of MIF over Periodic in Figure 4.15

	Average system loading, ρ						Average
	0.05	0.1	0.15	0.2	0.25	0.3	
Periodic (1)	4.47	7.48	14.96	21.63	25.49	27.50	16.92

The system HS3 loading is kept same while the mean transfer delay τ is changed from 0.05 to 0.30 at a step of 0.05. We show only the result for the case where the system loading is 0.9 as the difference between these two information policies is insignificant under the light or moderate system loading. Figure 4.15 shows that the *ART* for using both information policies increases when the mean transfer delay increases. This is because the transfer delay between sites and rapid changes in state of site sometimes make the status information messages obsolete by the time they reach their destination. It is possible that the scheduler has not received the updated load information from a remote site when a new job arrives. Figure 4.15 illustrate the following points:

The increasing rate of MIF policy is very smaller than that of Periodic policy. It is especially apparent when the mean transfer delay exceeds 0.20. The MIF policy has an average improvement of 16.92% over Periodic policy (Table 4.13). For Periodic policy, a lightly loaded site can quickly become overloaded because a few schedulers send jobs to it before the new load information is available. This is known as the Herd effect [72] and often leads to incorrect job distribution and poor

performance. The effect becomes more serious if the transfer delay is significant. In contrast, the MIF policy appears to include a "random" component that makes the per-site updates desynchronise the sites enough to reduce the herd effect; this may benefit jobs arriving in the future, thereby reducing the *ART*. The stochastic approach can avoid system instability when all sites transfer the jobs to the site estimated to have minimum load.

4.2.9 Sensitivity to the number of random sites ω_P

We investigated the sensitivity of MIF policy to the number of random sites for information update, ω_P with different values. We varied the number of random sites while keeping the HS3 system loading at a constant ρ = 0.9. It is intuitive that the larger the number of sites included in the information exchange message from each site, the better the performance. However, the simulation results illustrate that including more than two sites did not further improve system performance (Figure 4.16). Therefore, we selected two sites in the information exchange message from each site. We did not give the result for the case where the system loading ρ is light or moderate, as there is little difference by applying different number of random sites.

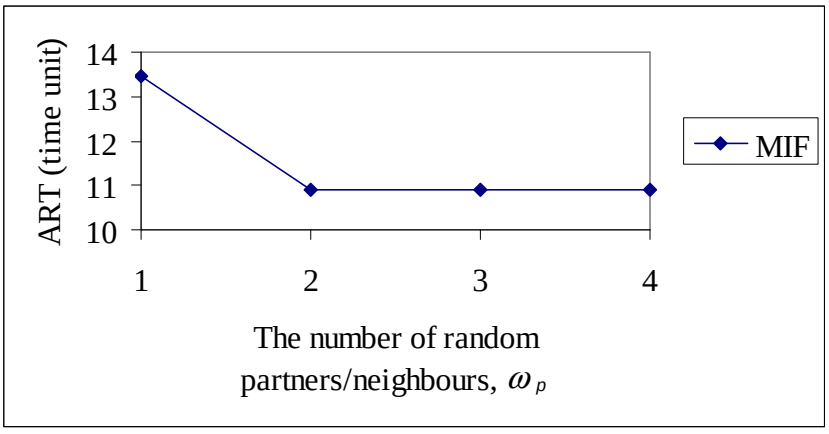

Figure 4.16: Sensitivity of MIF to the number of random sites

4.2.10 Sensitivity to interval T_p

The relationship was explored for *ART* and T_p (Figure 4.17), which shows results for the MIF policy for updated intervals of several values of T_p in the system HS3. The system loading was kept at 0.9. The best performance is yielded when the update interval is shortest (T_p equal to 1). This is due to the quality of the information on

which the load-balancing algorithm acts, which improves in proportion to the frequency with which information is distributed. However, it is clear that the improvement is slight for T_p of 1, 5, and more than 10. If a site has not received the updated information from another site at time interval T_p, it sends a request message to that site, which replies with the required information. This generates a high volume of messages when a shorter interval is used.

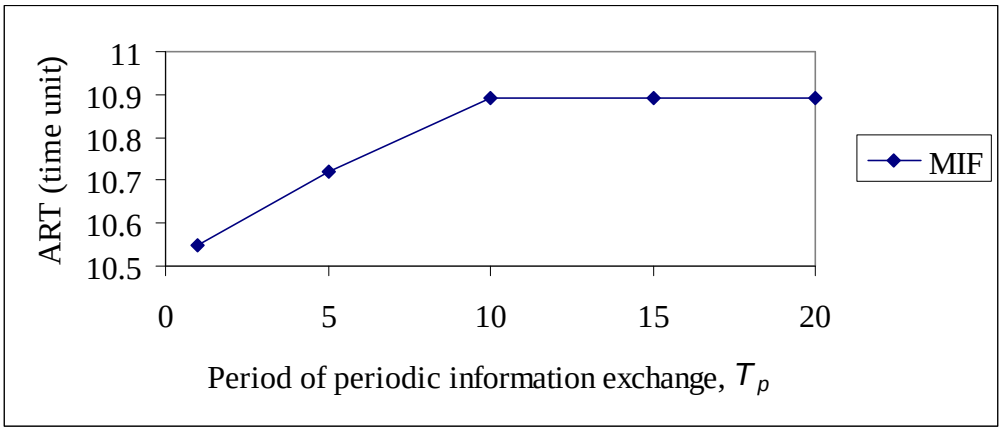

Figure 4.17: Sensitivity of information exchange policy MIF to the period for periodic information exchange

We suggest that an update interval of 10 is appropriate for the MIF policy, because it is a compromise between the performance obtained by the policy and the network overhead introduced in the system by the policy.

4.2.11 Sensitivity to the number of partner sites

We varied the number of partners of a site while keeping the system HS3 loading ρ at a constant 0.5 and 0.9. Figure 4.18 illustrates this from 1–8 partner sites.

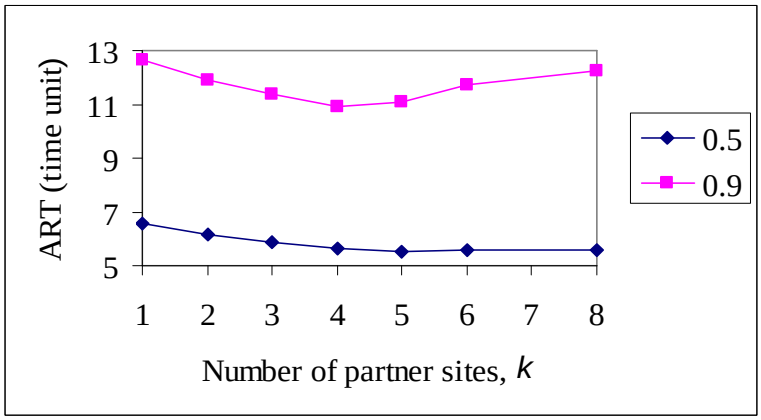

Figure 4.18: Sensitivity of algorithm DA to the number of partner sites

The *ART* goes down as the number of partner increases. However, the improvements come at a decreasing rate. In moving from 4 to 8 partner sites, the benefits of load balancing are very few or do not exist, and there seems to be a saturation point. This suggests that a small number of partner sites are more effective with respect to load balancing.

4.2.12 Sensitivity to distance coefficient ε

The HS3 system was employed as the base configuration. All runs were performed at system loadings of 0.5 and 0.9. The system loading was kept the same while the value of ε was changed from 1.2 to 1.75. Figure 4.20 illustrates the following points. The *ART* goes down as the value of ε increases. However, the improvements come at a decreasing rate. In moving from a value of 1.5 to 1.75, the benefits of load balancing do not exist, but there seems to be a saturation point. This suggests that a value of 1.5 can be more effective with respect to load balancing. At a system loading of 0.5, the same conclusion was drawn. Thus, we do not give the result here.

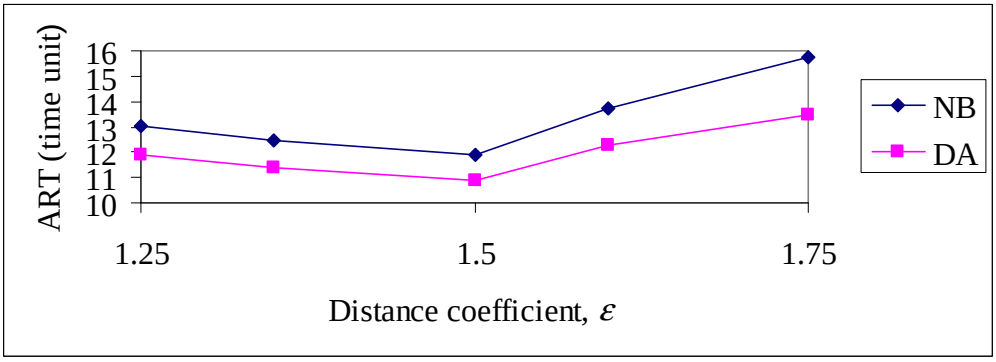

Figure 4.19: Sensitivity of algorithm DA to distance coefficient ε

4.2.13 Sensitivity to the number of reference sites

We varied the number reference sites at different system size while keeping the HS3 system loading ρ at constants 0.5 and 0.9 (Figures 4.20 and 4.21).

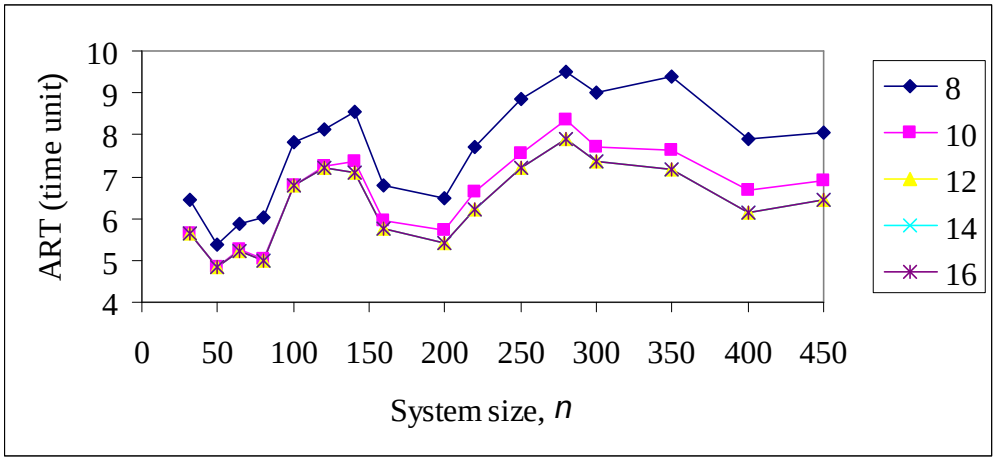

Figure 4.20: Sensitivity of algorithm DA to the number of reference sites (at system loading of 0.5)

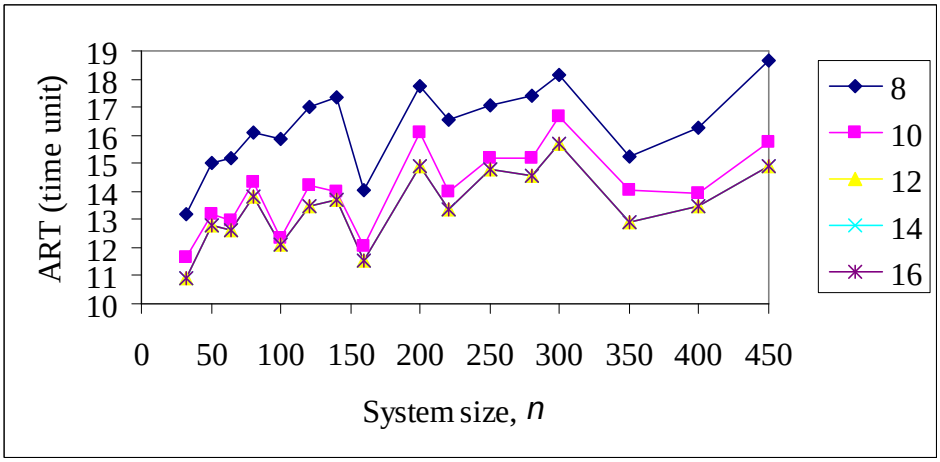

Figure 4.21: Sensitivity of algorithm DA to the number of reference sites (at system loading of 0.5)

At a low system loading of 0.5 (Figures 4.20), the *ART* goes down with increasing numbers of reference sites. However, this improvement stops from 12 to 16 reference sites. There was a very small difference in the *ART* between 10 and 12 reference sites. As a small number of reference sites are used for sites clustering, the less powerful sites may be partitioned into a reference site with higher processing power, because there is no reference site closer to the less powerful sites. Thus, when a site cannot find a site from a cluster for which the ID is greater, there is a high probability that the site will find less powerful sites as its partner sites, because it randomly select partner

sites from the cluster it belongs to.

At a high system loading of 0.9 (Figures 4.21), there is a similar trend to the low system loading of 0.5. Except for the reasons mentioned above, the Partner Adjustment Policy also increase the possibility of choosing less powerful sites as partner sites, because the policy plays a important role with the increasing of the system loading.

4.2.14 Effectiveness of load balancing

This section presents the effectiveness of load balancing, showing when it is beneficial to perform load balancing. The effectiveness of load balancing is given by:

$$E = \frac{ART_{DA}}{ART_{Local}}$$

where ART_{DA} is the *ART* run by DA. ART_{Local} is the *ART* completely processed at their originating sites (without running load-balancing algorithm).

Where E is less than 1, this means that the *ART* after load balancing is more than that before load balancing, and thus that the load balancing is not effective. Where the value of E is 1, it means that there is no change in the *ART*. Where the value of E is more than one, it means that the *ART* after load balancing is less than that before load balancing; thus, the load balancing is effective.

To calculate the effectiveness of load balancing, the DA algorithm was executed on a heterogeneous grid system HS3. The setting of system loading is the same as one in the Section 4.2.5. The *ART* with and without load balancing was calculated for different computation and communication costs, within a range of {1, 2, 3, 4, 5, 6, 7}. Figure 4.22 presents the effectiveness of DA as the ratio of computation costs to communication costs changes. The graph clearly shows that as this ratio changes from 1 to 7, the effectiveness of load balancing increases. Figure 4.23 plots the crossover point, clearly showing when it is beneficial to perform DA. When the computation to communication ratio *CCR* is less than 1.75, the load balancing is ineffective; but when this ratio is higher than 1.75, DA becomes very effective.

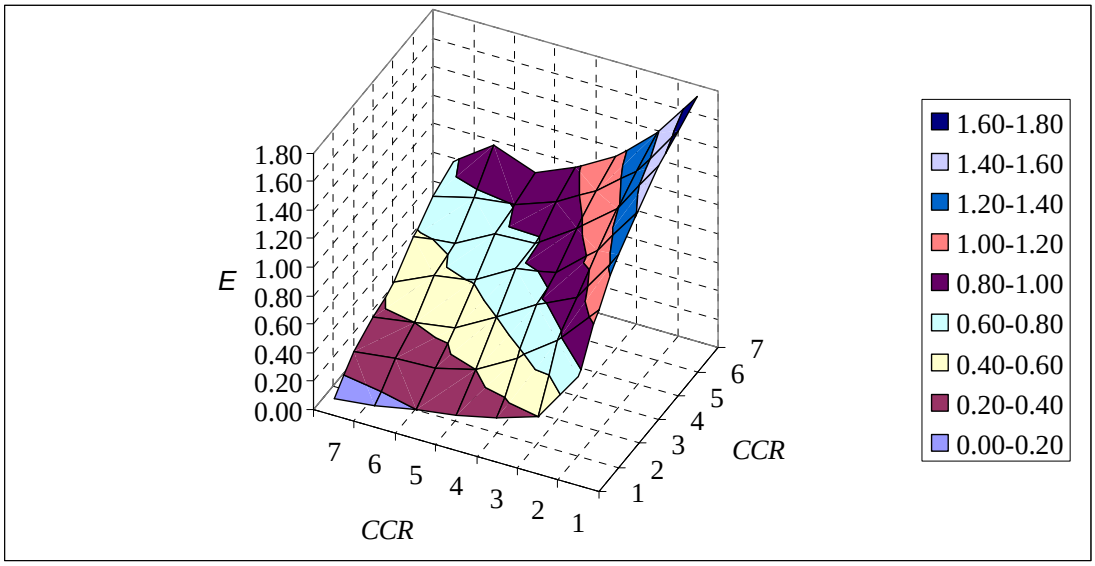

Figure 4.22: Sensitivity of algorithm DA to the number of reference sites (at system loading of 0.9)

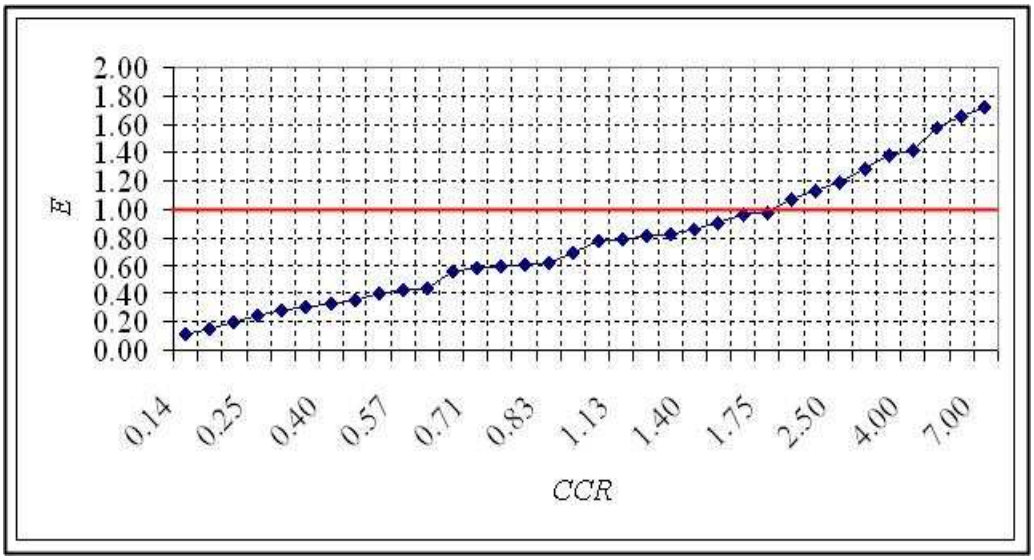

Figure 4.23: Effectiveness of algorithm DA as the ratio of computation to communication cost of jobs increases

4.3 Summary

We have proposed a desirability-aware decentralised dynamic load-balancing algorithm, which considers scalability of grid system, heterogeneous processing

power of grid sites and considerable communication overheads involved in information collection.

We conducted extensive simulation studies to analyse the performance of the load-balancing algorithm. The algorithm was compared with the neighbours-based load-balancing algorithm. Our experiment results show that the algorithm performs better than the neighbours-based algorithm, and reduces the average response time of jobs over a wide range of system parameters.

5

Decentralised and performance-driven load-balancing algorithm

This chapter presents a dynamic and decentralised load-balancing algorithm for computationally intensive jobs in the heterogeneous grid computing systems. The time spent by a job in the system is the main issue that needs to be minimised. This is an extended work of a desirability-aware load-balancing algorithm by considering performance benefit that jobs can gain in the load distribution triggered by Instantaneous Distribution Policy (IDP) and Load Adjustment Policy (LAP). Here, we propose another load adjustment policy, Augmented Load Adjustment Policy (ALAP); this policy determines whether there is a better placement beyond the neighbourhood of a site, where IDP has failed to show that a less powerful site in that neighbourhood is a performance bottleneck.

Section 5.1 describes in detail a performance-driven load-balancing algorithm. In Section 5.2, the performance of our algorithm is evaluated in a series of simulations.

5.1 Performance driven load-balancing algorithm

The proposed algorithm uses the desirability of sites to guide load assignments (Chapter 4).

5.1.1 Load index

$\forall s_i \in S$, the load index of s_i at a particular instant of time t is defined as

$$LD_{i,t} = TET_{i,t} + RET_{i,t}$$

where $TET_{i,t}$ is the total estimated job execution time of all jobs currently waiting in the job queue of s_i at time instant t, and $RET_{i,t}$ is the estimated remaining time of the job currently being processed by s_i at time instant t.

The assumption that the expected job execution times can be estimated is commonly made when studying scheduling and load-balancing techniques for heterogeneous computer systems [13, 52, 53, 97, 133]. Some approaches for doing this estimation are discussed in [38, 138–141].

5.1.2 Region and region load

The region r_i of a site s_i includes the site s_i and its neighbours. The region load is measured by the average load across all sites within the region. The load of region r_i at a particular instant of time t is defined as:

$$RD_{i,t} = \frac{\sum_{j=1}^{q} LD_{j,t}}{q}$$

where $LD_{j,t}$ is the load of site s_j at time instant t , q is the number of grid sites in the region r_i.

The average load between a region r_i and another remote region r_j at a particular instant of time t is defined as:

$$\Lambda_{i,t} = \frac{\sum_{k=i,j} RD_{k,t}}{d}$$

where $RD_{k,t}$ is the load of region r_k at time instant t , d equals to 2.

5.1.3 Threshold policy

Two kinds of threshold policies can be considered for a load-balancing algorithm: a fixed threshold policy and an adaptive threshold policy. As the former name suggests, a fixed threshold policy has predetermined thresholds that will not change when the system load changes. As the latter name suggests, an adaptive threshold policy has thresholds that are adjusted as the system load changes [87, 93]. Hence, an adaptive threshold policy is applied to load balancing between regions.

Based on the average load value calculated earlier, a region r_i can be defined as:

- A heavy region if $\Pi_{i,t} > H \times \Lambda_{i,t}$
- A light region if $\Pi_{i,t} < L \times \Lambda_{i,t}$
- A normal region if $\Pi_{i,t} \geq L \times \Lambda_{i,t}$ and $\Pi_{i,t} \leq H \times \Lambda_{i,t}$

where H and L are two threshold parameters, which are used to determine whether a region is a heavy or light load. The magnitude of the threshold parameters reflects the sensitivity of the system to load fluctuation. H and L are set to 1.2 and 0.8, respectively [87, 93].

5.1.4 Execution cost

We include dynamic communication cost in the cost calculation. This is because dynamic and considerable communication cost may have a significant influence on the performance of a load-balancing algorithm in the grid environment. It may be more efficient to send a job to a site with heavier load but smaller communication cost.

$\forall\ s_i, s_j \in S$, the execution cost of sending a job $j_x \in J$ from s_i to s_j at time instant t is estimated by s_i as

If $TRAN\_IN(j_x, s_i, s_j, t) \geq LD_{j,t}$

$$EC(j_x, s_i, s_j, t) = TRAN\_IN(j_x, s_i, s_j, t) + ETC(j_x, s_j) + TRAN\_OUT(j_x, s_j, s_i, t)$$

Else

$$EC(j_x, s_i, s_j, t) = LD_{j,t} + ETC(j_x, s_j) + TRAN\_OUT(j_x, s_j, s_i, t)$$

where $TRAN\_IN(j_x, s_i, s_j, t)$ measures how long it takes to transfer a job from s_i to

s_j. $TRAN\_OUT(j_x, s_j, s_i, t)$ measures how long it takes to transfer a job result from s_j to s_i. $LD_{j,t}$ is the recent load index of s_j at the time instant t that are recorded in s_i.

5.1.5 Performance benefit

The performance benefit associated with a job j_x is based on the idea that better migration can be done by assigning a job to a grid site that would "benefit" most in terms of expected response time if that grid site is assigned to it. If a job j_x is transferred from its current site s_i to a remote site s_j for execution, the value of performance benefit of a job, labelled as B_x, is computed as follows:

$$\forall\ s_i, s_j \in S, s_i \neq s_j$$

$$B_x = EC(j_x, s_i, s_i, t) - EC(j_x, s_i, s_j, t)$$

where $EC(j_x, s_i, s_j, t)$ is the estimated execution cost at s_j while $EC(j_x, s_i, s_i, t)$ is the estimated execution cost at s_i.

5.1.6 Information policy

We use the same Mutual Information Feedback policy for state information exchange, as shown in Algorithm 4.3 in Chapter 4. The state object O_i of a site s_i includes a new property, called Region Load, *RD*. The $O_i[j].RD$ is maintained through message exchanges with neighbours by appending it to the message.

5.1.7 Transfer and location policy

Our transfer and location policy is a hybrid of three policies: IDP, LAP and ALAP. The policies use the most recent load status information to decide whether a migration is initiated. These are described below.

5.1.7.1 Instantaneous Distribution Policy

This is a static load-balancing policy. When a new job arrives at a site s_i, the policy decides whether it is to be sent to s_i or other partner sites. The decision depends on whether it can gain performance benefit if it is distributed to a partner site. The policy

aims to control the job processing rate on each site in the system. Algorithm 5.1 describes the *IDP* for s_i.

Algorithm 5.1 (Instantaneous Distribution Policy):

$\forall j_x \in J$ with bornSite(j_x) = $s_i \in$ S
For each s_j in $PSet_i$ {
    Calculate $EC(j_x, s_i, s_j, t)$
    Calculate related benefit value B_x
}
Find the partner site s_j that gives the maximum B_x
If $B_x > \theta$ { /* θ is a positive real constant close to zero */
    Transfer the job j_x to the partner site s_j
    Update load index of site s_j recorded at the site s_i
}
Else
    $GJQ(s_i) \leftarrow$ enqueue(j_x) /* put the job j_x in the job queue $GJQ(s_i)$ */

5.1.7.2 Load Adjustment Policy

The LDP for a site s_i tries to continuously reduce load difference among s_i and its neighbours $NSet_i$ by transferring jobs from heavily loaded sites to lightly loaded neighbouring sites. The LDP is triggered whenever s_i receives load information from a neighbour. The load-balancing algorithm will use the most recent load status information to decide whether a migration is initiated. The job that benefits most in the global job queue $GJQ(s_i)$ is considered first for migration. If two neighbouring sites give the same performance benefit, the faster neighbouring site is chosen. The LDP for s_i is described in Algorithm 5.2.

Algorithm 5.2 (Load Adjustment Policy):

aFlag = true /* flag for initiating ALAP algorithm */
For each s_x in $NSet_i$
    For each Job j_x in $GJQ(s_i)$
        Calculate $EC(j_x, s_i, s_x, t)$
    End for
End for
For each Job j_x in $GJQ(s_i)$ {
    Find the site s_y that gives the minimum execution cost
    Calculate related benefit value B_x
}
Sort the jobs in $GJQ(s_i)$ in ascending order by their benefit value
Select the Job j_y with the biggest benefit value B_y
Find the neighbouring site s_j that gives the maximum B_y to j_y
If $B_y > \theta$ { /* θ is a positive real constant close to zero */
    Remove the job j_y from $GJQ(s_i)$
    Transfer the job j_y to the neighbouring site s_j
    Update load index of site s_j recorded at the site s_i
    aFlag = false
}
Else {
    If aFlag = true
        Call ALAP algorithm
}

5.1.7.3 Augmented Load Adjustment Policy

If a site s_i receives the load information from a neighbour s_j, after the application of the LAP, no load adjustment occurs; this implies that s_i is the best suited site for the jobs in its global job waiting queue, with respect to the neighbourhood. If a better placement can be found, it should be beyond the neighbourhood of s_i, $NSet_i$. This also implies that may s_j operate like a bottleneck to LAP, and this could be the triggering point of the ALAP. The sample of a less powerful site causing the bottleneck of LAP to its overlapping neighbourhoods structure, is given below.

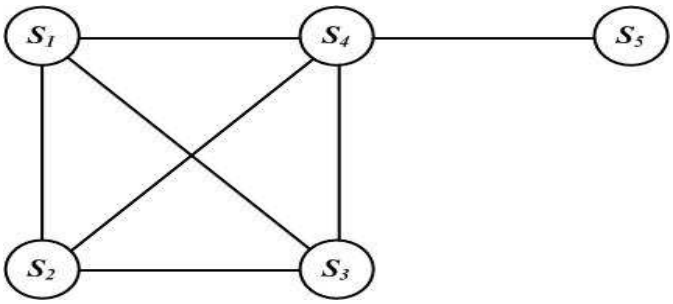

Figure 5.1: A heterogeneous grid system of five sites (special case)

A heterogeneous grid system of five sites $\{s_1, s_2, s_3, s_4, s_5\}$ is assumed (Figure 5.1). Here, we do not consider partner sites. They are of average processing powers (*APW*) 3, 1, 1, 1 and 6. The sites are assumed idle. For clarity, the communication cost between sites is the same here, taking one time unit for each transferred job at each link. Assume that there are three jobs $\{j_1, j_2, j_3\}$ at s_1. They consist of 6, 9 and 12 computational units, respectively. According to the LAP algorithm, no job could migrate to s_4, because s_4 is less powerful site and no performance could be gained from execution at the site s_4. This means that the more powerful s_5 is still idle. As a result, the response time of jobs j_1, j_2 and j_3 is 2, 5, and 11 time units, respectively. This leads to an average response time of 6 time units for these three jobs. The job j_3 can benefit most compared with other two jobs if it migrates to s_5 for processing. If this happens, the response times of j_1, j_2, and j_3 can be changed into 2, 5, and 6 time units, respectively. Thus, the three jobs have an average response time of 4.3 time units, an improvement of 28.33%.

How to trigger the ALAP is a critical problem. Refer to Algorithm 5.5 for the triggering policy we have incorporated into our algorithm. It is based on the simple heuristic that the heavier the load in a region, the less inclined it will be to accept future loads. Triggering is initiated if the region load of s_i is heavy and the region load of s_j is light. A request message from s_i is then sent to s_j. Having received a request to send the status of its neighbours back to the requesting s_i, s_j packs the information about their current load and sends it back. If the information of the neighbours of s_i is included in the requested information, the neighbours of s_i are excluded and the remaining sites are considered as potential sites for load adjustment; these are called

augmented neighbours of s_i. Then, we apply ALAP among the site and its augmented neighbours, as shown in Algorithm 5.4.

Algorithm 5.4 (Augmented Load Adjustment Policy):

NeedForTriggering (s_i, s_j)
If (NeedForTriggering return true) {
$ANSet_i \leftarrow$ Find the augmented neighbours of site s_i
For each s_x in $ANSet_i${
    For each Job j_x in $GJQ(s_i)$
        Calculate $EC(j_x, s_i, s_x, t)$
}
For each Job j_x in $GJQ(s_i)$ {
    Find the site s_y that gives the minimum execution cost
    Calculate related benefit value B_x
}
Sort the jobs in $GJQ(s_i)$ in ascending order by their benefit value
Select the Job j_y with the biggest benefit value B_y
Find the neighbouring site s_j that gives the maximum B_y to j_y
If $B_y > \theta$ { /* θ is a positive real constant close to zero */
    Remove the job j_y from $GJQ(s_i)$
    Transfer the job j_y to the augmented neighbouring site s_j
}
}

Algorithm 5.5 (NeedForTriggering (s_i, s_j)):

Calculate $RD_{i,t}$
Calculate $\Lambda_{i,t}$
If r_i is heavy AND r_j is light
    Return true
Else
    Return false

5.2 Performance evaluation

In the simulations, our algorithm (labelled as P-DA) is compared with the following

algorithms.

- ***Central.*** This is a dynamic load-balancing algorithm. A central load balancer (Central) coordinates load distribution among sites. It tries to balance the load by assigning each job to the computing site that yields the earliest completion time. On receipt of a message notifying a job arrival, the Central finds the site that gives the shortest completion time, and if the site is different from the job's current site, the job is sent to that site. Otherwise, the job is executed at its current location. If several sites give the same shortest completion time for the job, one of them is selected randomly. The Central updates its load vector to reflect the load distribution decision, and the originating site is informed of the execution site. The originating site subsequently moves the job to its destination if necessary. Central has been studied in [56, 87] and in [113] (where it was referred to as CENTEX) and implemented in Utopia [22]. It is related to the LBC algorithm [60] and to an algorithm studied by Theimer and Lantz in [78]. Central has also been studied in the computational grids [43]. Central assumes that the load balancer retrieves the current load value of all other grid sites without cost, but considers the transfer cost of the job from a site to another site. We selected the algorithm because it represents a typical class of centralised approaches, and used it as an estimate for a load-balancing algorithm with "perfect" information.
- ***Performance-driven neighbour-based.*** It is labelled as P-BA. In this algorithm, the performance heterogeneity in processing power among grid sites is not considered. A new job is sent to local site or one of neighbouring sites immediately (IDP). Then continuous load adjustment is employed among neighbouring sites (LAP). A job is assigned or redistributed to the site that it would benefit most. State information exchange is done via MIF among a site and its neighbour sites. The algorithm is a partial version of algorithm P-DA. It is regarded as an improved version of traditional neighbour-based algorithms because the remote execution cost has been considered. We selected the algorithm because it represents a typical class of decentralised approaches.

Table 5.1: Simulation parameters (time unit = tu, pt = percent)

Simulation parameter	Value
Size of system, n	32
Number of computational unit in a job j_x, NCU_x	[100, 30000]
Number of packets in a job j_x, A_{1x}	[1, 100]
Ratio of the computation unit to the number of packet for the job, R_f	[1, 300]
Mean transfer delay, τ	0.05 tu
Standard deviation of transfer delay, σ_c	50 pt
Bandwidth between any two sites, BW_{ij}	[1, 100]
Distance ratio from a site to its nearest site, R_d	1.5
Updated interval, T_p	10 tu
Number of partners, k	4
Number of random partners/neighbours for information update, ω_P	2

5.2.1 Simulation model

In this section, we studied the performance of the algorithms under different system parameters via simulations. Eleven assumptions were devised for the simulation model.

- All of the work is carried out on a grid system that consists of n sites. The average processing power of sites APW is randomly assigned in the range [P1, P2]. According to the different degrees of heterogeneity in the average processing power of sites, several different heterogeneous systems are yielded.
- Jobs arrive at each site s_i, i=1, 2,…, n according to a Poisson process, with rate $\lambda_i = \lambda \times P_i$, where P_i=1/n. The actual inter arrival time of jobs is adjusted to give the required overall average system loading (see last bullet point).
- The number of computational units in a job is randomly chosen in the interval [W1, W2]. The number of packets in a job is chosen in the range [F1, F2] at random. We choose $A_{1x} = A_{2x}$.
- The transfer delay that may be incurred between any site pairs in the grid system is chosen from a lognormal distribution with a mean of τ time unit and standard

deviation σ_c.

- The bandwidth between any two sites is chosen randomly in the range [B1, B2].
- The ratio between the computational unit for a computing intensive job and the packets of the same job needed to migrate is chosen randomly as $R_f = \frac{NCUx}{A_{1x}} \in$ [Y1, Y2].
- Once a job is created, several attributes are assigned to it. These attributes include Job ID, the file size (number of packets), the creation time and the required computation time. These attributes remain unchanged throughout the lifetime of the job.
- The parameters of network between different pairs of sites need to be provided for the simulator, including transfer delay and bandwidth.
- For the algorithm P-DA, the partner set of each site needs to be provided before our algorithm runs. The partner set is based on the processing power generated from the distribution of average processing power. The sites are selected as its partners according to the method described in Section 4.1.4.
- For algorithms P-NB and P-DA, the neighbours of each site are fed to the simulator before the algorithms run. The neighbours are based on the transfer delay generated from the distribution of mean transfer delay. The sites are selected as its neighbours according to the method described in Section 4.1.5.
- Let ρ be the required average system load for our simulation, which is the average job arrival rate divided by the average job processing rate. Using this definition, we adjust the job mean inter-arrival time $1/\lambda$ needed to obtain the desired ρ.

Table 1 displays the values of the parameters used in the simulations. Considering a 100 MB/s fast ethernet network, and jobs such as the calculation of prime numbers or Fibonacci numbers, the ratio R_f chosen in Table 1 is very close to the reality. The same model can be used for internet connection using either low-speed dial-up lines (average 44 KB/s), to high-speed fibre-optic connections (2–100 MBs). Thus, the simulation results are as close to the reality as possible.

Heterogeneous systems introduce additional parameters that make performance evaluation much more difficult than for homogeneous systems. To simplify, we considered three types of heterogeneous systems: HS1, HS2, and HS3. Table 2 shows heterogeneous system configurations. The third column contains the APW of each of the four site types. The second column contains four site types and the fractions of sites in the system corresponding to each site type. An exception for heterogeneous system HS3, the APW of each site is generated randomly in a range of U[10, 100].

Table 5.2: Heterogeneous system configurations

Heterogeneous Systems	Sites split (fraction)	APW
HS1	3/8, 1/4, 1/4, 1/8	10, 20, 50, 100
HS2	3/8, 1/4, 1/4, 1/8	10, 15, 20, 30
HS3	Random / no split	[10, 100]

Unless explicitly mentioned otherwise, the default system parameters are applied to all simulations in the simulation model. For each simulation run, to eliminate the start-up transients, we ignored the first 1000 jobs. After the warm-up time, we traced the jobs' arrival, processing and finish times from j_{1000} to j_{4999}. Here u equals to 5000 (for evaluation purpose). After each simulation run, we computed the average response time of jobs (*ART*). We conducted each measurement five times with different random seeds.

5.2.2 Effect of system heterogeneity

Intuitively, heterogeneous system reinforces the advantages of using load balancing, since is expected that the presence of different job processing power at different sites will lead to an increased probability of load-balancing success. In this simulation, we created three heterogeneous systems by using different processing power at the various sites (Table 2). The performance of the three load-balancing algorithms is compared for these three heterogeneous systems under different system loading ranging from 0.1 to 0.9. Figures 5.2–5.4 and Table 5.3-5.5 summarise the results of this simulation.

At all of the loads tested, the general trend is a decrease in performance with increasing system load. Up to an approximate load level of $\rho = 0.6$, the performance decrease is slight, but after this point it deteriorates rapidly.

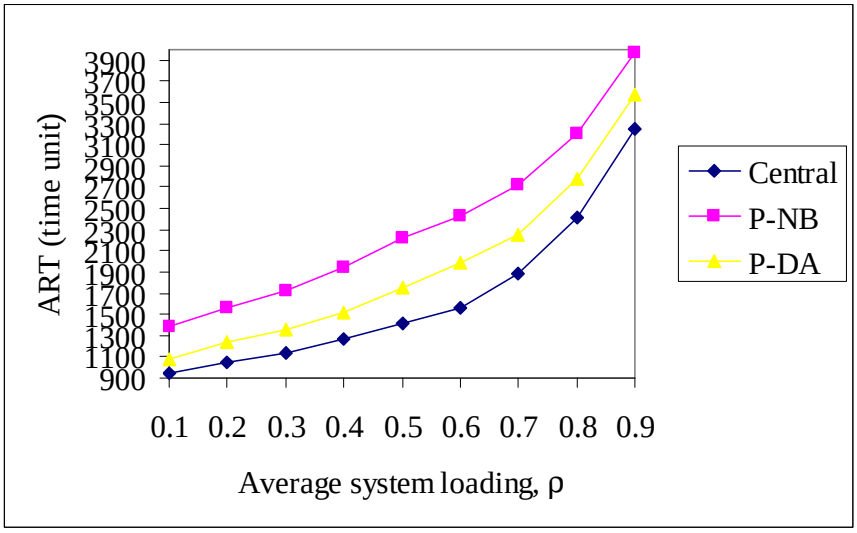

Figure 5.2: Effect of system with high heterogeneity

Table 5.3: Improvement factor (in percent) of P-DA over other algorithms in Figure 5.2

	Average system loading, ρ						Average (0.1–0.6)
	0.1	0.2	0.3	0.4	0.5	0.6	
P-NB	22.16	20.58	21.34	22.18	21.46	18.24	20.99
Central	-13.78	-17.55	-20.35	-20.00	-23.83	-27.72	-20.54

	Average system loading, ρ			Average (0.7–0.9)	Average (0.1–0.9)
	0.7	0.8	0.9		
P-NB	17.26	13.32	10.08	13.55	18.51
Central	-19.00	-15.31	-9.81	-14.71	-18.59

For the highly heterogeneous system HS1, P-DA has an average improvement factor of 18.51% over P-NB. At light or medium loads (10–60%), it is seen that P-DA is better than P-NB by a substantial margin of 20.99%. When the system loading is light or moderate, for P-DA, IDP plays a crucial role and the LAP has little influence on the average response time of the jobs. For P-NB, the less powerful neighbouring sites with either idle or very light load may become the bottleneck, because a better placement can be found beyond the neighbourhood of a site. Thus, the more powerful non-neighbouring sites are not exploited at low or moderate system loading. However, this advantage of P-DA decreases as the loading increases. At high loadings, remarkably, even at a system

loading as high as 0.9, P-DA still yields an improvement of 10.08% in *ART* than P-NB. One reason for this is that new jobs are first dispatched to faster partners, giving the performance benefit, which means that it is more likely for jobs to shorten *ART* at faster sites. Another reason is that ALAP of P-DA contributes to a reduction in *ART*; hence, P-DA gives a better performance. As may be expected, P-DA performs worse than the nonrealistic Central. However, P-DA generates close performance to the Central. For instance, at $\rho = 0.9$, P-DA is 9.81% worse than Central.

For the low heterogeneous system HS2, P-DA is notably still 11.03% better than P-NB; at light and medium loads (10–60%), P-DA is 12.33% better; at high system loading (70–90%), P-DA is 8.43% better. The performance improvement is due to the fact that the IDP and ALAP of P-DA make the main contributions. Another interesting observation is that, as expected, the comparison against the nonrealistic Central at high system load is not significant. For instance, P-DA at $\rho = 0.9$ is about 11.14% worse than the Central.

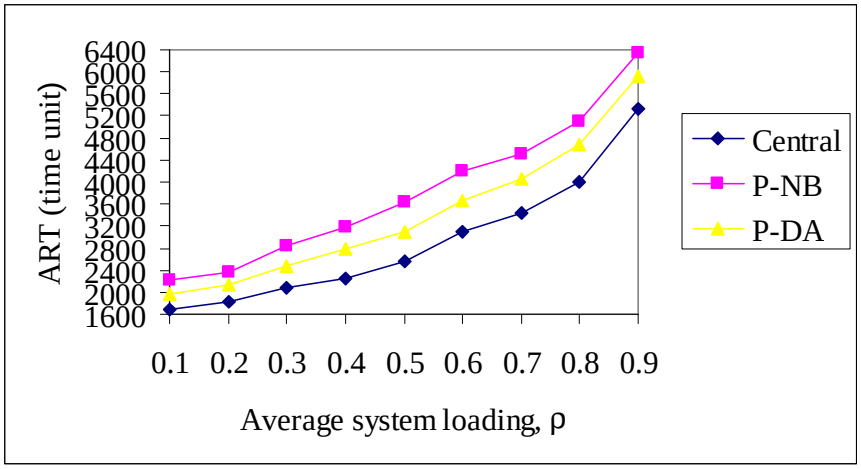

Figure 5.3: Effect of system with low heterogeneity

Table 5.4: Improvement factor (in percent) of P-DA over other algorithms in Figure 5.3

	Average system loading, ρ						Average (0.1–0.6)
	0.1	0.2	0.3	0.4	0.5	0.6	
P-NB	11.29	10.16	12.60	12.32	14.92	12.66	12.33
Central	-17.66	-16.32	-20.00	-23.45	-20.55	-18.38	-19.39

	Average system loading, ρ			Average (0.7–0.9)	Average (0.1–0.9)
	0.7	0.8	0.9		
P-NB	9.74	8.71	6.83	8.43	11.03
Central	-18.20	-16.68	-11.14	-15.34	-18.04

For systems with randomly generated heterogeneity, the same conclusion is drawn. P-DA performs 17.31% better than P-NB, with a performance close to the optimal value of the Central at high system loading.

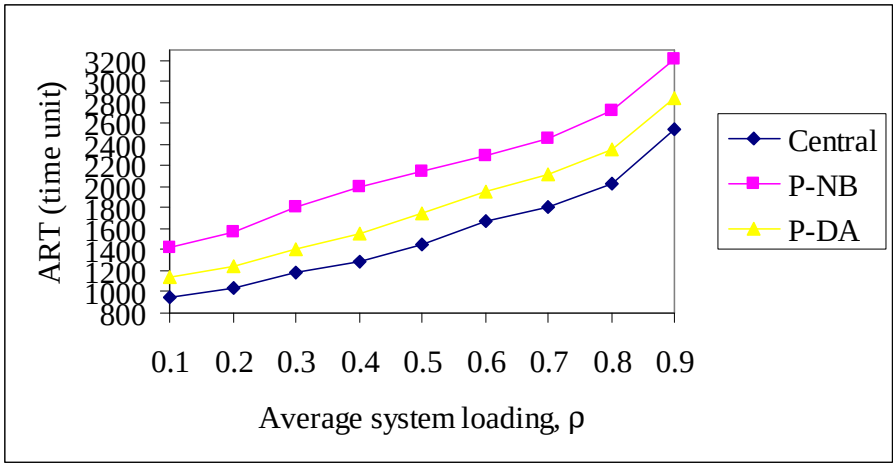

Figure 5.4: Effect of system with randomly generated heterogeneity

Table 5.5: Improvement factor (in percent) of P-DA over other algorithms in Figure 5.4

	Average system loading, ρ						Average (0.1–0.6)
	0.1	0.2	0.3	0.4	0.5	0.6	
P-NB	19.23	20.89	21.61	22.27	18.13	14.96	20.99
Central	-21.63	-19.88	-19.17	-19.97	-20.58	-16.49	-19.62

	Average system loading, ρ			Average (0.7–0.9)	Average (0.1–0.9)
	0.7	0.8	0.9		
P-NB	13.82	13.52	11.37	13.55	17.31
Central	-17.78	-16.26	-12.01	-15.35	-18.20

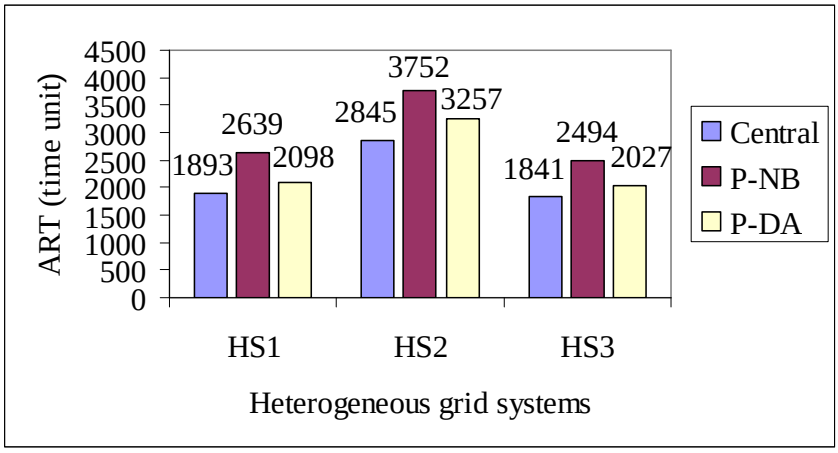

Figure 5.5: Effect of different job arrival patterns in three different kinds of heterogeneous grid systems: HS1, HS2 and HS3

Table 5.6: Improvement factor (in percent) of P-DA over other algorithms in Figure 5.5

	HS1	HS2	HS3
P-NB	20.50	13.19	18.73
Central	-10.83	-14.48	-10.10

5.2.3 Effect with different job arrival patterns

Our observations in the preceding section were in the context of uniform job arrival rates,

but the average job arrival rate in grid systems is often heterogeneous. In the simulation, to reduce complexity we considered three classes of job arrivals, each with a different average job arrival rate. We randomly chose 10 of the sites as lightly loaded (ρ = 0.3), 11 of the sites as moderately loaded sites (ρ = 0.6), and 11 as highly loaded (ρ = 0.9). The HS1, HS2 and HS3 system settings were used as the base configurations in the section. The *ART* with P-DA is an average improvement of 20.50%, 13.19% and 18.73% over P-NB in the HS1, HS2 and HS3 systems, respectively (Figure 5.5 and Table 5.6). However, P-DA was on average 10.83%, 14.48% and 10.10% less than Central in the HS1 system, HS2 system and HS3 system, respectively.

5.2.4 Sensitivity to the coefficient of variation in the inter-arrival times of jobs

The simulation examines the effect of the coefficient of variation (*CV*) in inter-arrival times of jobs to the *ART* in a heterogeneous system HS3. Figures 5.6 and 5.7 provide the *ART* for different *CV*s of job inter-arrival rate. The system utilisation is a constant of 0.5 and 0.9. The *CV*s of job inter-arrival rate has little effect on the *ART* when the system loading is uniformly low, and there is a significant effect when the system loading is high.

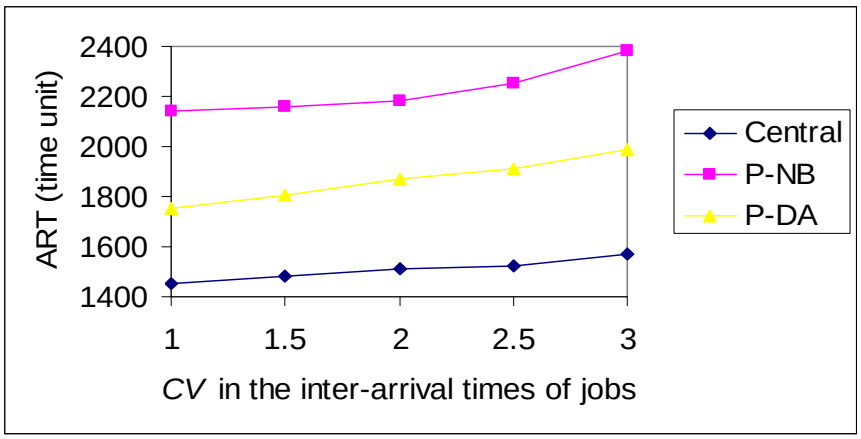

Figure 5.6: Sensitivity to the variance in the inter-arrival times of jobs (system loading equals to 0.5)

Table 5.7: Improvement factor (in percent) of P-DA over other algorithms in Figure 5.6

	1	1.5	2	2.5	3	Average
P-NB	18.13	19.30	14.51	15.26	16.69	16.78
Central	-20.58	-22.00	-23.63	-25.23	-26.32	-23.55

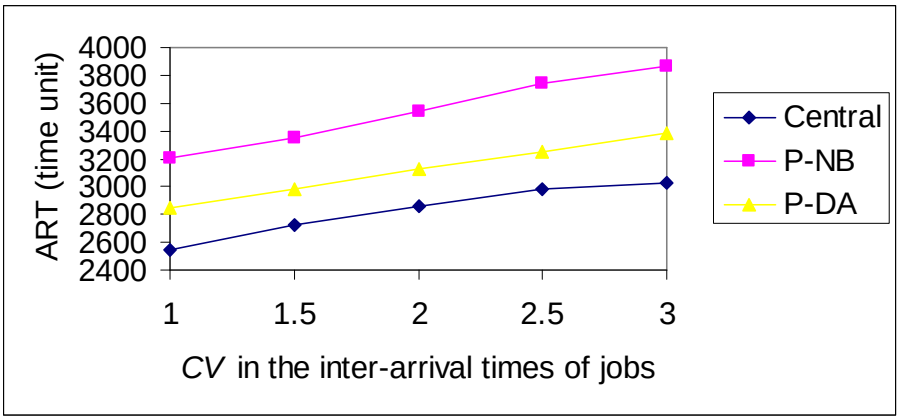

Figure 5.7: Sensitivity to the variance in the inter-arrival times of jobs (system loading 0.9)

Table 5.8: Improvement factor (in percent) of P-DA over other algorithms in Figure 5.7

	1	1.5	2	2.5	3	Average
P-NB	11.37	11.18	11.68	13.15	12.68	12.01
Central	-11.96	-9.16	-9.44	-9.06	-11.55	-10.23

When the system loading is 0.5 (Table 5.7), DA outperforms NB by 16.78%, while there still is an average improvement factor of 12.01% over NB at the system loading of 0.9 (Table 5.8). P-DA is that at the instant clustered job arrival at a site, it initiates load distribution immediately to powerful partner sites. The ALAP of P-DA algorithm plays a major role because jobs can be dispatched to the powerful sites beyond neighbours. P-NB, on the other hand, attempts load distribution only among its neighbouring sites and does not consider the heterogeneous processing power of sites. This delays the job execution and is especially apparent under low and moderate loading.

Figure 5.8 presents the *ART* for three algorithms under different system loading when the inter-arrival times of jobs is set to 3.0. The simulation results for P-DA with gradually increasing *CV* lead to similar conclusions, i.e., the burstiness in job arrivals does not significantly weaken the advantage of P-DA over P-NB in terms of the *ART*, and the performance difference between P-DA and Central remains small at high system loading (Table 5.9).

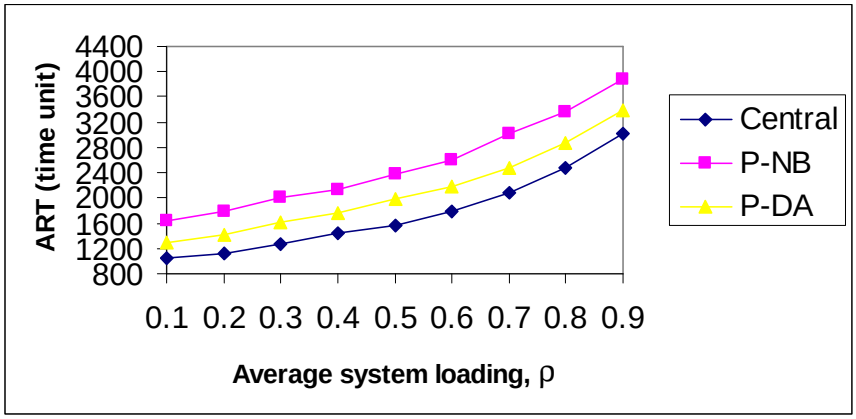

Figure 5.8: Sensitivity of performance to different system utilisation (*CV* of 3)

Table 5.9: Improvement factor (in percent) of P-DA over other algorithms in Figure 5.8

	Average system loading, ρ				
	0.1	0.2	0.3	0.4	0.5
P-NB	20.25	20.67	20.22	17.72	16.69
Central	-25.00	-26.79	-25.78	-22.22	-26.32
	Average system loading, ρ				Average (0.1–0.9)
	0.6	0.7	0.8	0.9	
P-NB	15.77	17.61	14.29	12.68	17.32
Central	-22.35	-19.23	-16.13	-11.55	-21.71

5.2.5 Job completion statistics

This section looks at the utilisations of the different class of sites in the HS1 and HS2 heterogeneous systems, at overall system loadings of 0.5 and 0.9 for Central, P-NB and

P-DA algorithms, as shown in Figures 5.9–5.12. The fractions of total jobs are computed as the total number of jobs that completed at each type of site divided by *u*.

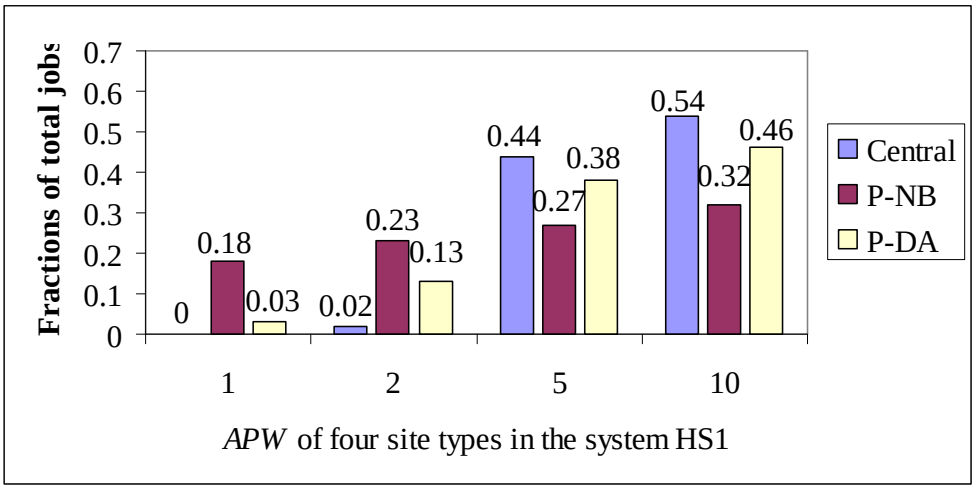

Figure 5.9: Job completion statistics at the system loading of 0.5 in the system HS1

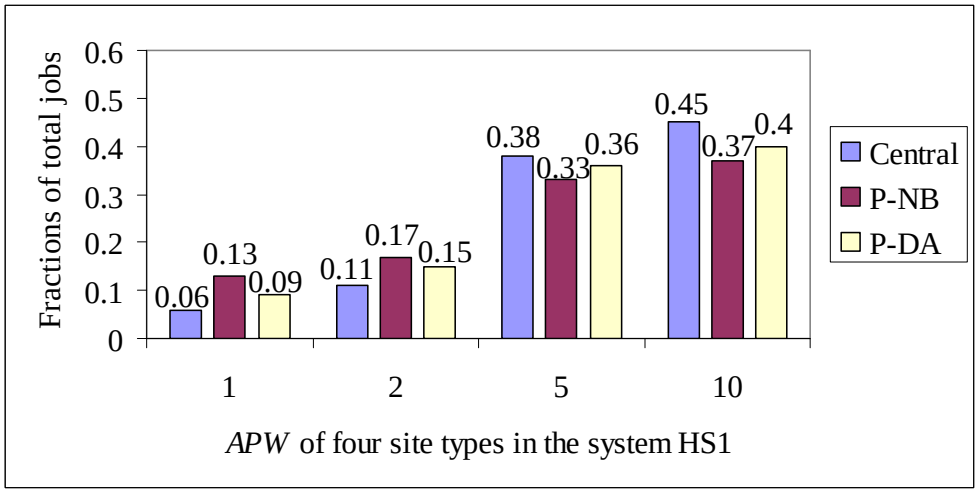

Figure 5.10: Job completion statistics at the system loading of 0.9 in the system HS1

At the system loading of 0.5, for both HS1 and HS2, P-DA tends to underload the less powerful sites, and consequently allocates a disproportionate share of the work to more powerful sites. The difference between the algorithms is more marked for the highly heterogeneous HS1, where there are more powerful sites. However, P-NB tends to assign loads to less powerful neighbouring sites. This is because P-NB does not consider heterogeneity in *APW* among sites while assigning load. At the system loading of 0.9, for both HS1 and HS2, although P-NB reduces load at weaker sites and loads more at more

powerful sites, there remains the issue of neglecting processing power. On the contrary, P-DA increases loads at less powerful sites and at more powerful sites, because its LAP can migrate jobs to a less powerful neighbouring site when a more powerful site cannot provide performance benefit for the queued jobs at the site. However, due to the major contribution from the IDP of P-DA, the algorithm still has a higher throughput for each class of sites than P-NB. The results correspond to the analysis in Section 5.2.2. P-DA has close load assignments to Central for both system loadings in two heterogeneous systems.

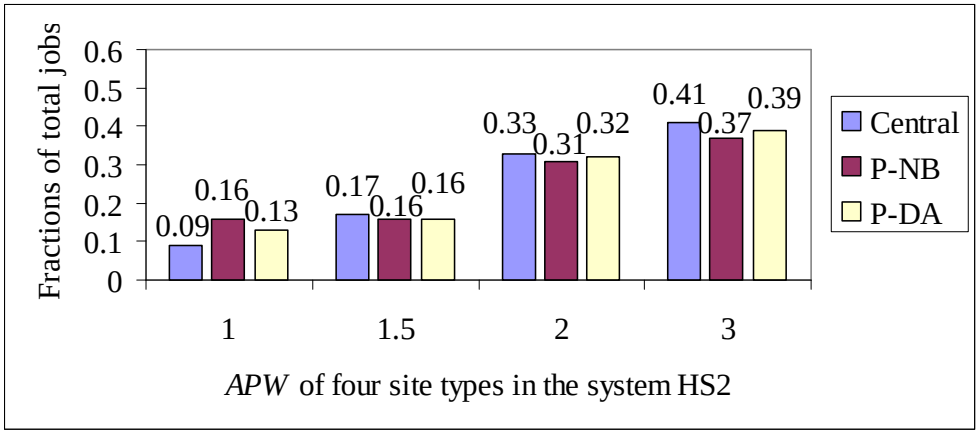

Figure 5.11: Job completion statistics at the system loading of 0.5 in the system HS2

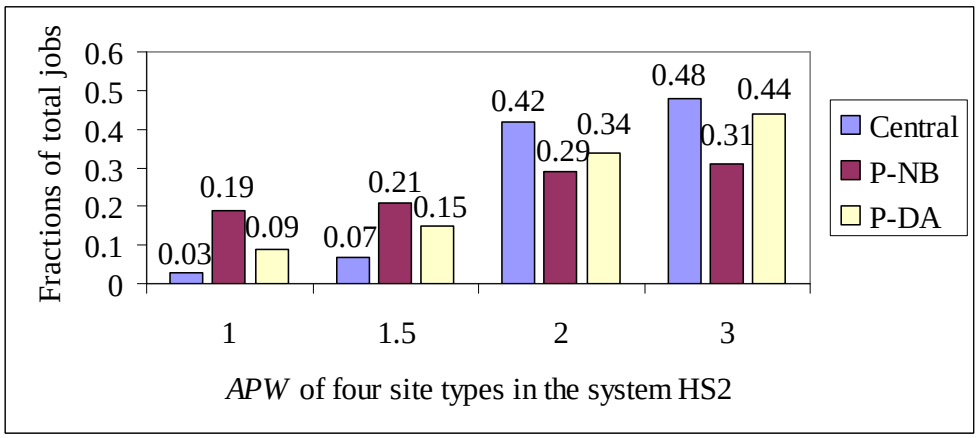

Figure 5.12: Job completion statistics at the system loading of 0.9 in the system HS2

5.2.6 Scalability

Figures 5.13 and 5.14 present the simulation results for Scenarios 1a and 1b, respectively. The system size employed in the scenarios varies from 32 to 512. In Scenario 1a, an under-utilisation system is simulated, with overall system loading ρ = 0.5. In Scenario 1b, the overall system loading is selected as ρ = 0.9.

The results for Scenario 1a show that P-DA performs 15.52% better than P-NB and worse than the nonrealistic Central by about 15.18% (Table 5.10). The results for Scenario 1b show that P-DA has 9.60% better than P-NB (Table 5.11). Although P-DA is worse than the nonrealistic Central, the difference is smaller – 9.43%. In both scenarios, P-DA is efficient in load balancing. The advantage of P-DA over P-NB does not depend on the size of grid.

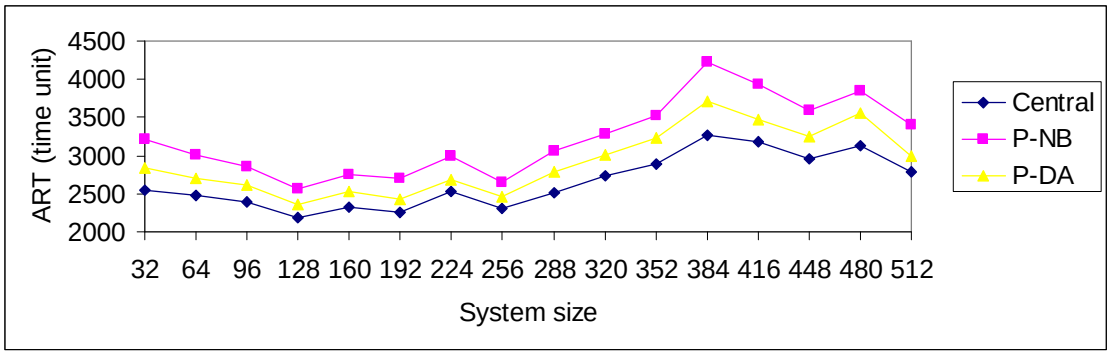

Figure 5.13: Evaluation of scalability with different size at the system load of 0.5

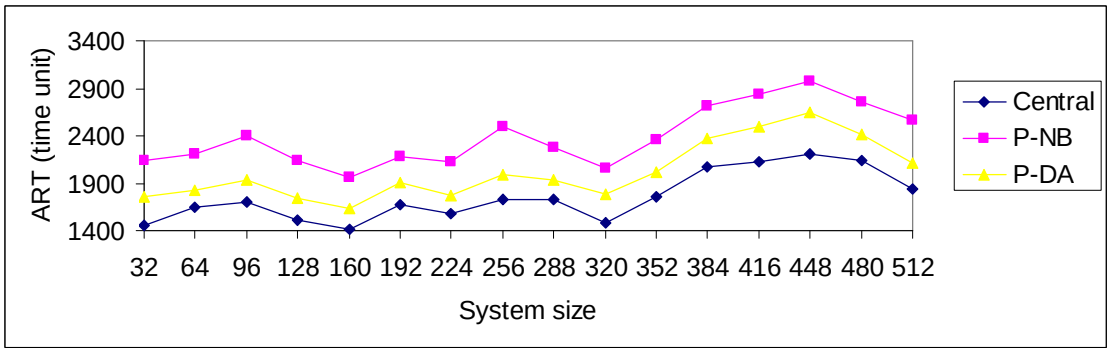

Figure 5.14: Evaluation of scalability with different size at the system load of 0.9

Table 5.10: Improvement factor (in percent) of P-DA over other algorithms in Figure 5.13

	System size, *n*					
	32	64	96	128	160	192
P-NB	18.13	17.49	19.09	18.73	16.96	12.63
Central	-20.58	-11.21	-13.61	-15.12	-14.79	-13.82
	System size, *n*					
	224	256	288	320	352	384
P-NB	16.32	20.20	15.09	13.24	14.93	12.64
Central	-12.49	-14.59	-11.72	-20.47	-14.12	-14.94

	System size, *n*				Average
	416	448	480	512	
P-NB	11.75	10.83	12.74	17.46	15.52
Central	-17.36	-20.34	-12.88	-14.84	-15.18

Table 5.11: Improvement factor (in percent) of P-DA over other algorithms in Figure 5.14

	System size, *n*					
	32	64	96	128	160	192
P-NB	11.37	10.23	8.78	8.16	8.41	10.17
Central	-12.01	-9.35	-8.58	-7.2	-8.36	-6.94
	System size, *n*					
	224	256	288	320	352	384
P-NB	10.12	7.11	9.05	8.70	8.24	12.20
Central	-5.71	-6.74	-10.83	-9.88	-12.35	-13.58

	System size, *n*				Average
	416	448	480	512	
P-NB	11.85	9.54	7.43	12.20	9.60
Central	-8.82	-9.91	-13.61	-7.06	-9.43

5.2.7 Sensitivity to accuracy of estimation of job execution cost

Estimated job execution cost is needed in P-DA. We assessed the impact of incorrect job execution cost estimation on P-DA. Suppose that the actual execution cost of job j_i is x_i and that the predictive error, denoted by y_i, is a random variable in the range $[-ax_i, bx_i]$

following some probability density function, $g_i(y_i)$, where the possible value fields of a and b are [0, 100%] and [0, ∞], respectively. The parameters a and b represent the range of estimated errors. It is assumed that the estimated errors of different jobs are independent random variables. The estimated execution cost of job j_i, denoted by z_i, is computed as follows:

$$z_i = x_i + y_i$$

The estimated error (y_i) follows uniform probability distribution. Therefore, the relation between the estimated execution cost (z_i) and the actual execution cost is expressed linearly.

To calculate the effect of accuracy of estimation of job execution cost, P-DA was executed in a heterogeneous grid system HS3. The setting of the system loading is the same as the one in Section 5.2.3. Figure 5.15 shows the performance for P-DA when job execution cost is accurately estimated and estimated with a range size of errors. The Figure also shows the performance of the algorithm without load balancing. a and b increase from 10% to 90%, in increments of 10 points. This results in the range of estimation errors for the actual execution time of x increasing from $[-0.1x, 0.1x]$ to $[-0.9x, 0.9x]$, while the average estimation errors remains unchanged (at 0).

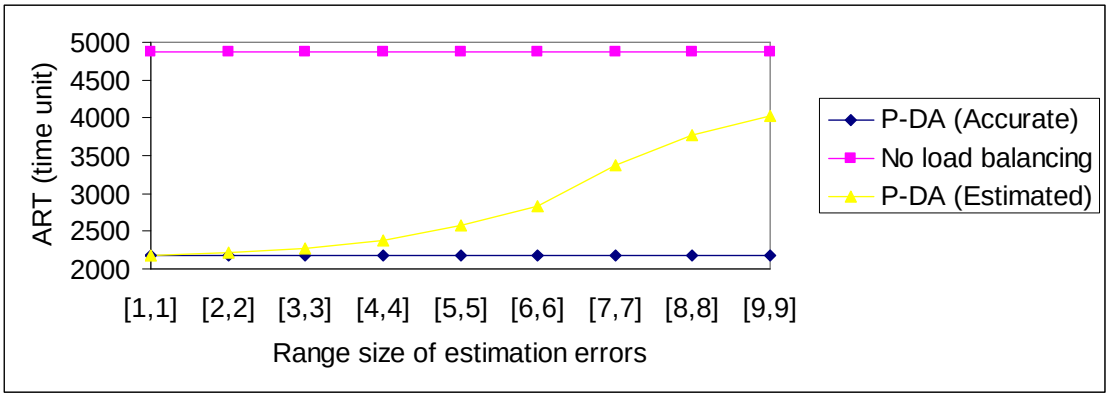

Figure 5.15: The impact of range size of estimation errors on the performance of algorithm P-DA

Figure 5.15 shows that a and b increase, there is an overall increase in average response time. However, this does not affect the performance significantly as a and b change from

[1, 1] to [3, 3]. This is because the jobs are first dispatched to more powerful partner sites and may be processed fast even if there is a low estimation error rate. With the increasing of a and b, the advantage of P-DA is offset by highly inaccurate information, especially when the error is large. However, the performance of algorithm P-DA is still better than the situation without load balancing when a and b reaches to [9, 9].

5.3 Summary

A performance-driven, decentralised load-balancing algorithm has been proposed for computational grids. The proposed algorithm still uses the desirability of sites to guide load assignments while taking into account the performance benefit that jobs can gain as a migration decisions are made.

The algorithm has been extensively evaluated using simulations. The algorithm consistently outperforms another performance-driven neighbour-based algorithm (P-NB) by requiring shorter average response times of jobs. The results of simulations show that the performance of the algorithm is similar to that of unrealistic algorithm Central.

6

Performance-driven and region-based load-balancing algorithm

This chapter presents another performance-driven load-balancing algorithm based on the regional grids, where grid sites are clustered into regions around a set of well-known broker sites in terms of network transfer delay, and the regional brokers are organised in a fully decentralised fashion. For each regional grid, our algorithm integrates static Instantaneous Distribution Policy (IDP) and dynamic Load Adjustment Policy (LAP) to make load distribution and redistribution driven by the performance benefit that jobs can gain. The LAP also considers load redistribution across regional grids. To keep intra-region and inter-region communication at low levels, our information policy combines Mutual Information Feedback (MIF) inside regions with the random polling of a remote regional broker site, performed by each regional broker site at a set time interval. Our algorithm achieves a balance between the inherent efficiency of a centralised approach, and the autonomy, load balancing and fault tolerant features offered by distributed approach.

Section 6.1 describes in detail a performance-driven, region-based load-balancing algorithm. In Section 6.2, the performance of our algorithm is evaluated in a series of simulations.

6.1 Performance-driven and region-based load-balancing algorithm

The algorithm is partially based on research that was presented in Chapter 5. The definitions used in this chapter for site load index, region load, average region load, threshold policy, execution cost and performance benefit are the same those used in Chapter 5, although here the definition of region is different.

6.1.1 Region construction

We assume that there is a known set of h sites in the computational grid. A grid site s_i measures the relative distances to the set of h sites and sorts the obtained vector $<d_1, d_2,\ldots, d_h>$ in order of increasing transfer delay. Based on these transfer delays, s_i has an associated ordering of sites. This ordering represents the region that s_i belongs to.

The set of h sites is called the "regional brokers". The regions are denoted as $r_1,\ldots, r_h$. The region is actually a subset of S. If a site s_i is located in a certain region r_a, we call the region r_a, as a local or current region of s_i. Each regional broker site does not function as a central load balancer in the region; it is responsible for exchanging information with other broker sites and forwarding the received information to potential sender in its region.

6.1.2 Information policy

The information policy comprises two parts: intra-region information and inter-region information policy.

Algorithm 6.1 (**Modified procedure of job transfer and information exchange**):

$s_i \in r_a, s_j \in r_b$

Steps processed in s_i:

If $r_a = r_b$ {/* sites s_i and s_j are in the same region

1. $Y \leftarrow s_i + \{\omega_P$ random sites from $r_a - s_j\}$
   // s_i select neighbours or partners for information exchange
2. $\forall s_y \in Y$, s_i appends ($O_i[y].LD$, $O_i[y].LT$) to the job transfer request TR
3. s_i appends bornsite(j_x) to TR

}

4. s_i sends job transfer request message TR to s_j

Steps processed in s_j:

Upon receiving TR:

If $r_b = r_a$ { /* sites s_i and s_j are in the same region

1. $\forall s_y \in Y$: If ($O_i[y].LT > O_j[y].LT$) $O_j[y] \leftarrow O_i[y]$
   // s_j updates the state object using s_i's info
2. $Z \leftarrow s_j + \{\omega_P$ random sites from $r_g - s_i\}$
3. $\forall s_z \in Z$, s_j appends ($O_j[z].LD$, $O_j[z].LT$) to the acknowledge reply AR

}

4. s_j sends acknowledge reply message AR to s_i

Upon completion of job j_x:

If $r_b = r_a$ { /* sites s_i and s_j are in the same region

1. $Z \leftarrow s_j + \{\omega_P$ random sites from $r_g - s_i\}$
2. $\forall s_z \in Z$, s_j appends ($O_j[z].LD$, $O_j[z].LT$) to the completion reply CR

}

3. s_j sends completion reply message CR to bornsite(j_x)= s_i

Steps processed in s_i:

Upon receiving the reply AR or CR:

If $r_a = r_b$ { /* sites s_i and s_j are in the same region

$\forall s_z \in Z$: If ($O_j[z].LT > O_i[z].LT$) $O_i[z] \leftarrow O_j[z]$

}

6.1.2.1 Intra-region information policy

We use the same MIF policy for state information exchange, as shown in Algorithm 4.3 (Chapter 4). However, the following changes are made.

- For a site, ω_p number of random sites for information exchange is chosen from its local region instead of partners or neighbours.
- Whenever a site receives a message from or sends a message to another site beyond its local region, it does not need to update the load information of the ω_p number of random sites that have been stored in its local site, or appends them to the outgoing message.

The modified procedure of job transfer and information exchange is described in Algorithm 6.1.

6.1.2.2 Inter-region information policy

Each regional broker site collects the load information from remote regions. The regional broker site is not a central storage source of load information of its local region. Its roles are described in this section. Let us assume that both s_u and s_v are regional broker sites. Algorithm 6.2 outlines the procedure of processing messages at a regional broker site s_u. In the description of Algorithm 6.2, $message_{uv}$ denotes a message sent from s_u to s_v. There are three types of message used in the algorithm, which are *Request*, *Heavy*, and *Light*. Except the type of *Heavy* message, s_u appends the load information of its region and least loaded site of its region to the messages.

s_u sends a $Request_{uv}$ message to petition s_v for the load status of its region at a periodic interval of time T_S. At most, one polling is performed at a time interval T_S to reduce the communication overhead across regions.

When s_u receives a $Request_{vu}$ message from s_v, it will append the load information of the least loaded site s_l of its local region in the $Light_{uv}$ message sent to s_v, if the local region is a light region and the remote region is a heavy region. Otherwise, unless a $Heavy_{uv}$ message is sent to s_v, it needs to forward the message as *Light* message to the heaviest loaded site s_h in its current region according to information stored at s_u, if the local region is a heavy region and the remote region is a light region.

As soon as s_u has received the $Light_{vu}$ message from s_v, it forwards the message to the heaviest loaded site s_h in its current region, according to information stored in the

broker site s_u. If the received message is $Heavy_{vu}$, the regional broker site s_u does not need to act.

Algorithm 6.2 (Procedure of processing messages at the regional broker site s_u):

```
Switch (msgType) {
    case msgType = "Heavy_vu":
        Do nothing
    case msgType = "Light_vu":
        Forward the least loaded site of the remote region to the heaviest loaded
         site of local region
     case msgType = "Request_vu":
        Compute RD_u,t  / * the load of local region
        Compute Λ_u,t  /* the average load between local and remote region
        Determine the load status of local region
        If the local region is light region {
            Add the load information of least loaded site to the Light message
            Send the Light message to the regional broker site s_v
        }
        Else {
            Send a Heavy message to the regional broker site s_v
            If (local region is heavy region) AND (the remote region is light
            region) then
            Forward the least loaded site of the remote region to the heaviest
            loaded site of local region
            End if
        }
  }
```

6.1.2.3 Fault-tolerance for regional broker sites

To guard against the possibility that the regional broker sites may fail, a second site is given the responsibility for each region. If a site finds that its regional broker site fails, this is reported to the backup broker site. The site can act as regional broker site immediately and notifies other regional broker sites of the failure. Any intra-region sites may be assigned this duty. The load information does not need to be replicated from nonfunctional regional broker site to the new regional broker site, because each site has had load information of all sites of its local region in terms of our intra-region

information policy. But there may be a difference with respect to the load information of the remaining sites of its local region, due to distributed information storage.

6.1.3 Transfer policy and location policy

As in Chapters 4 and 5, our transfer and location policies adopt a hybrid policy, which is a combination of IDP and LAP. The policies will use the most recent load status information to decide whether a migration is initiated. These are described below.

6.1.3.1 Instantaneous Distribution Policy

This is a static load-balancing policy. When a new job arrives at site s_i, the policy decides whether it is to be sent to the site s_i or other sites of its local region. The decision depends on whether it can obtain performance benefit if it is distributed to one of the sites in the local region. The policy aims to control the job processing rate on each site in the system. Algorithm 6.3 describes the Instantaneous Distribution Policy for a site s_i.

Algorithm 6.3 (Instantaneous distribution policy)

$\forall j_x \in J$ with bornSite(j_x) = $s_i \in S$, $s_i, s_j \in r_g$
For each s_j in r_g {
    Calculate $EC(j_x, s_i, s_j, t)$
    Calculate related benefit value B_x
}
Find the site s_j that gives the maximum B_x
If $B_x > \theta$ { /* θ is a positive real constant close to zero */
    Transfer the job j_x to the site s_j
    Update load index of site s_j recorded at the site s_i
}
Else
    $GJQ(s_i) \leftarrow$ enqueue(j_x) /* put the job j_x in the job queue $GJQ(s_i)$ */

Algorithm 6.4 (Load adjustment policy at site s_i):

$s_i \in r^+$ /* s_i belongs to region r^+

Switch (msgType) {

Case msgType="*Light*": /* upon receiving the information of least loaded site of a remote region forwarded by local regional broker */

$s^+ \leftarrow$ the least loaded site in the remote region

Case msgType="*TR*" or "*AR*" or "*CR*": /* upon receiving the messages sent from other sites of local region */

$s^+ \leftarrow$ all sites of the local region

}

$j^+ \leftarrow$ all waiting jobs that enqueue at the $GJQ(s_i)$

Found = true

While Found = true {

For each s_x in s^+ {

For each Job j_x in j^+

Calculate $EC(j_x, s_i, s_x, t)$

}

For each Job j_x in j^+ {

Find the site s_y that gives the minimum execution cost

Calculate related benefit value B_x

}

Sort the jobs in j^+ in ascending order by their benefit value

Select the Job j_y with the biggest benefit value B_y

Find the site s_j that gives the maximum B_y to j_y

If $B_y > \theta$ { /* θ is a positive real constant close to zero */

$j^+ \leftarrow j^+ - j_y$ // Remove the job j_y from j^+

if $s_j \in r^+$ Update load index of site s_j recorded at the site s_i

Transfer the job j_y to the site s_j

}

Else

Found=False

}

6.1.3.2 Load Adjustment Policy

This is a dynamic load-balancing policy, which is triggered at each site. Algorithm 6.3 describes the Load Adjustment Policy for a site s_i. The job that's benefits most in the

global job queue $GJQ(s_i)$ is considered first for migration. The policy has three major advantages.

- Whenever s_i receives updated load information of other sites in its current region, the policy is triggered. This means that the policy can continuously reduce load differences among s_i and other sites in its current region by migrating jobs from heavily loaded to lightly loaded sites. Thus, intra-region-wide dynamic job scheduling and load balancing may be implemented.
- Whenever s_i receives the information of the least loaded site of a remote region forwarded by the local regional broker site, the policy is triggered. This means that the policy can reduce load difference between s_i and another site of remote region by transferring jobs from the heavily loaded local site to the lightly loaded remote site. Thus, inter-region-wide dynamic job scheduling and load balancing is implemented.
- Whenever s_i has migrated some of its jobs to a remote region, it reports its current load status to its regional broker site and ω_P heavily loaded sites in its current region, which means that a multi-round intra-region-wide load balancing can be triggered.

6.2 Simulations

In the simulation, our algorithm (P-RB) is compared with the Minimum Completion Time Algorithm (MCT), a distributed algorithm. Each site tries to balance the load by assigning each job to the computing site that yields the earliest completion time. It is assumed that each grid site retrieves the current load value of all other grid sites without cost, although the transfer cost of the job is considered. During the job transfer, the load of the destination site may be changed. On receipt of a message notifying a job arrival, MCT finds the site that gives the shortest completion time, and if the site is different from the job's current site, the job is sent to that site; otherwise, the job is executed at its current location. If several sites give the same shortest completion time for the job, one of them is selected randomly. The originating site subsequently moves the job to its destination if necessary. MCT has been studied in [42] and [118] (where it is referred to

as Shortest Expected Delay) and implemented in AppLes [13] and Nimrod/G [14]. We selected the algorithm because it represents a typical class of distributed approaches with "perfect" information of all sites in the system at the scheduling instant.

6.2.1 Simulation model

We studied the performance of the algorithms under different system parameters via simulations. Twelve assumptions were devised for the simulation model.

- All of the work is carried out on a grid system consisting of n sites. The average processing power (APW) for each site is assigned in the range [P1, P2]
- It is assumed that the simulated grid system includes a fixed h number of regions. The size of each region n_r is randomly assigned in the range [R1, R2]. The maximum system size is the product of h and R2; the minimum system size is the product of h and R1.
- Jobs arrive at each site s_i, i = 1, 2,…, n according to a Poisson process, with rate $\lambda_i = \lambda \times P_i$, where $P_i = 1/n$. The actual inter-arrival time of jobs is adjusted to give the required overall average system loading (see last bullet point).
- The number of computational unit in a job j_x is randomly chosen within the interval [W1, W2]. The number of packets in a job j_x is randomly chosen within the range [F1, F2] ($A_{1x} = A_{2x}$).
- The transfer delay TD_{ij} that may be incurred between any site pairs across different regions is chosen from a lognormal distribution, with a mean of τ_{ex} time unit and a standard deviation σ_{ex}.
- The transfer delay TD_{ij} that may be incurred between any site pairs in a region is chosen from a lognormal distribution with a mean of τ_{in} time unit and a standard deviation σ_{in}.
- The bandwidth between any two sites is chosen randomly in the range [B1, B2].
- The ratio between the computational unit for a computing intensive job j_x and the packets of the same job needed to transfer is chosen randomly as $R_f = \frac{NCUx}{A_{1x}} \in$

[Y1, Y2].

- The number and size of the regions need to be fed into the simulator before the algorithms start to run.
- The network parameters between different pairs of sites need to be fed into the simulator, including transfer delay and bandwidth.
- Once a job is created, several attributes are assigned to it. These attributes include job ID, creation time, required computational unit, file size (number of packets), and ratio R_f. These attributes remain unchanged throughout the lifetime of the job in the system.
- Let ρ be the required average system load for our simulation, which is the average job arrival rate divided by the average job processing rate. Using this definition, we adjust the job mean inter-arrival time $1/\lambda$ needed to obtain the desired ρ.

Table 6.1: Simulation parameters (time unit = tu, pt = percent)

Simulation parameter	Value
Number of regions or regional broker sites, h	12
Size of each region, n_r	[10, 30]
Average processing power of site s_i, APW_i	[10, 100]
Number of computational unit in a job j_x, NCU_x	[100, 30000]
Number of packets in a job j_x, A_{1x}	[1, 100]
Ratio of the computation unit to the number of packet for the job, R_f	[1, 300]
Mean transfer delay within a region, τ_{in}	0.05 tu
Standard deviation of transfer delay, σ_{in}	50 pt
Mean transfer delay across regions, τ_{ex}	0.1 tu
Standard deviation of transfer delay, σ_{ex}	50 pt
Bandwidth between any two sites, BW_{ij}	[1, 100]
Period for periodic intra-region information exchange, T_p	10 tu
Polling interval for inter-region information exchange, T_S	1 tu
Number of random sites for information update in a region, ω_P	2

Table 1 displays the values of the parameters used in the simulations. Unless explicitly mentioned, the default system parameters are applied to all simulations. For each simulation run, to eliminate the start-up transients, we ignore the first 2000 jobs. After the

warm-up time, we trace the jobs' arrival, processing and finish times from j_{2000} to j_{9999}. Here u equals to 8000 (for evaluation purpose). After each simulation run, we computed the average response time of jobs (*ART*). We carried out each measurement five times with different random seeds.

6.2.2 Effect of system loading

We conducted a series of simulations with MCT and P-RB under medium and high system loading, ranging from 0.5 to 0.9. Figure 6.1 and Table 6.2 summarise the results of this simulation.

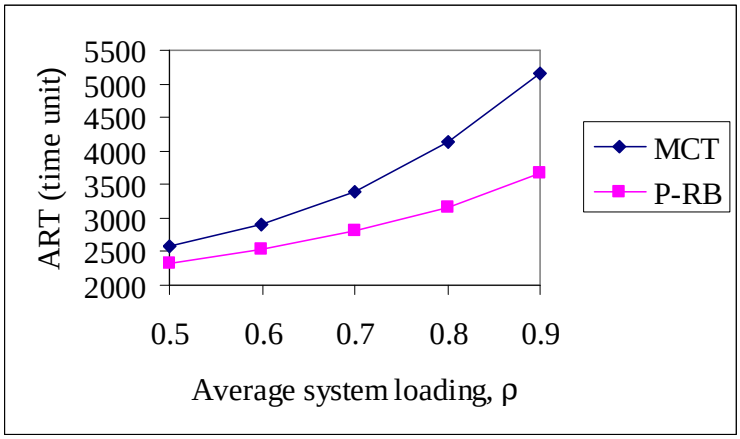

Figure 6.1: Effect of system loading

Table 6.2: Improvement factor (in percent) of P-RA over MCT in Figure 6.1

	Average system loading, ρ					Average
	0.5	0.6	0.7	0.8	0.9	
NB	9.60	12.95	17.48	23.80	28.93	18.55

At all of the loads tested, the general trend is of a decrease in performance with increasing system load. When the system loading is high, the difference in the *ART* between MCT and P-RB is high. P-RB has an average improvement factor of 18.55% over MCT. At medium or high system loading, MCT may result in the transfer of multiple jobs to the site that provides the shortest job completion time. The jobs have to

wait longer to reach the execution stage, and increase the queuing time component of the response time of jobs. MCT and P-RB have different response to this situation. MCT takes no action, but waits at the site for processing. On the contrary, the LAP in our algorithm plays a critical role. The jobs may be involved in the load balancing of the local region. Another reason is that inter-region load balancing in the algorithm happens only when a light region is found. The IDP also has an effect on the *ART* at medium and high system loading. We suggest that it may be better for a job to be processed in a site of the local region at high system load, because each site maintains more accurate load information of other sites in its local region than a site in a remote region.

6.2.3 Effect with different job arrival patterns

Our observations in the preceding section were in the context of uniform job arrival rates, but the mean job arrival rate in grid systems is often heterogeneous. In the simulation, to reduce the simulation complexity, we consider three classes of job arrivals, each with a different average job arrival rate. We randomly chose a third of the sites as lightly loaded site (ρ = 0.3), a third of the sites as moderately loaded sites (ρ = 0.6) and other sites as highly loaded sites (ρ = 0.9). Figure 6.2 shows that *ART* with P-RB has an 18.21% improvement over those with MCT.

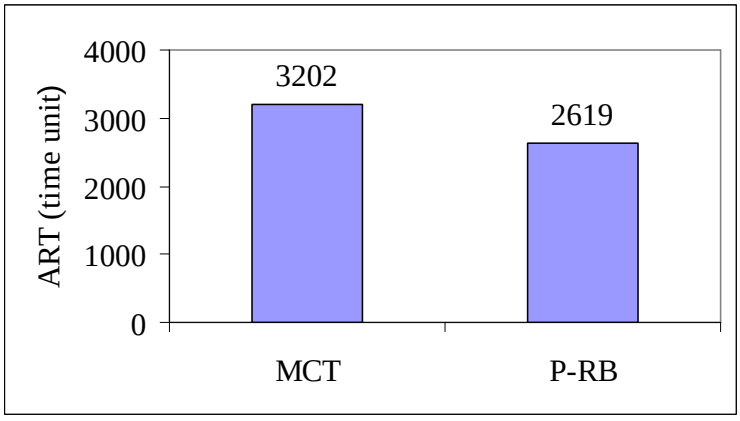

Figure 6.2: Effect with different job arrival patterns

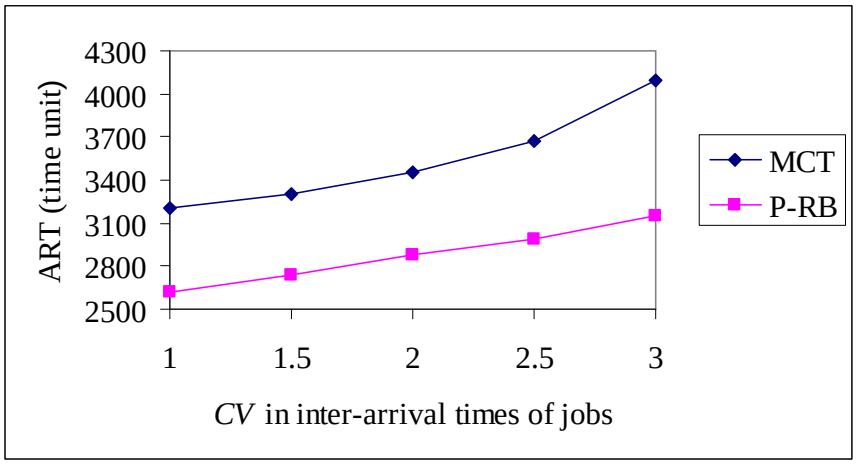

Figure 6.3: Sensitivity to the variance in the inter-arrival times of jobs

Table 6.3: Improvement factor (in percent) of P-RA over MCT in Figure 6.3

	1	1.5	2	2.5	3	Average
MCT	18.21	17.01	16.69	18.53	23.04	18.69

6.2.4 Sensitivity to the coefficient of variation in the inter-arrival times of jobs

The simulation examines the effect of the coefficient of variation (*CV*) in inter-arrival time to the *ART* in the heterogeneous system. The setting of system loading was the same as that in Section 6.2.3. Figure 6.3 shows the *ART* for different *CV*s of job inter-arrival rate. The figure shows that the advantage of P-RB over P-NB has not been weakened in terms of *ART*. P-RB has an average improvement of 18.69% over MCT (Table 6.3).

A higher *CV* in inter-arrival time implies that job arrivals at each node in the system are clustered; the higher the *CV*, the more clustered the job arrival process. Clustered arrivals mean that there are large gaps between job arrivals, but that the average job arrival rate remains the same. The more clustered the arrival process, the more attempts the site makes at load distribution. For P-RB, this means that each site

receives load information more frequently and accurately from other sites in its local region.

The long gaps between arrivals provide more opportunities for P-RB in distributing the load during the gap. When there are long gaps between arrivals, the probability that a site will move into a lightly loaded state increases. Thus, the probability that the LAP in P-RB will locate a lightly loaded site within and across regions increases with increasing *CV* in the inter-arrival times. In contrast, MCT does not make load adjustments, even though the jobs may be dispatched to overloaded sites or some sites have become lightly loaded during the long gaps.

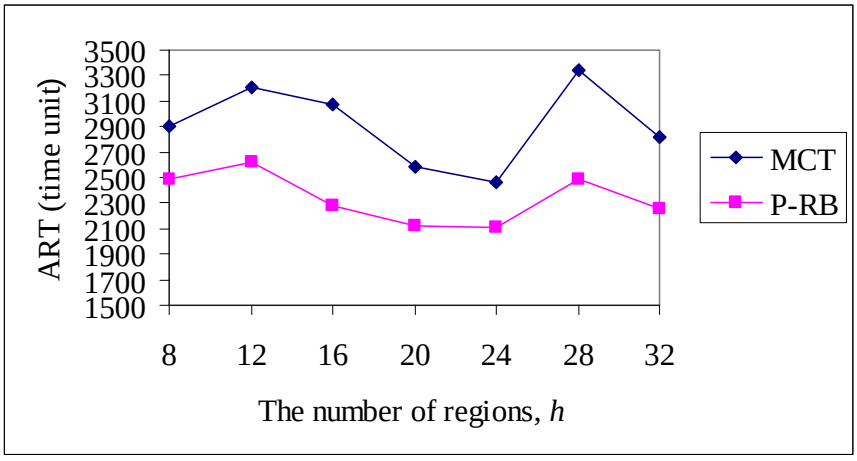

Figure 6.4: Evaluation of scalability with different number of regions

Table 6.4: Improvement factor (in percent) of P-RA over MCT in Figure 6.4

	8	12	16	20	24	28	32	Average
MCT	14.08	18.21	25.64	17.75	14.39	25.51	19.86	19.35

6.2.5 Sensitivity to the number of regions

This section presents the performance of both algorithms to the different number *h* of regions. We varied the number of regions from 8 to 32, in increments of 4. The setting of system loading was the same as that in Section 6.2.3. All other parameters were set to their default values (Table 6.1). The simulation results are presented in Figure 6.4 and

Table 6.4. The results show that P-RB performs 19.35% better than MCT. Figure 6.4 shows that the advantage of P-RB over MCT does not depend on the number of regions.

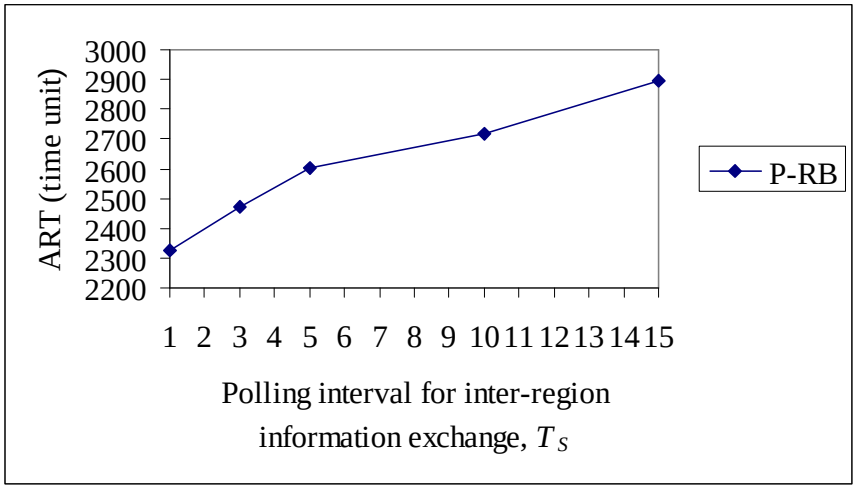

Figure 6.5: Sensitivity of performance to different polling intervals at system loading 0.5

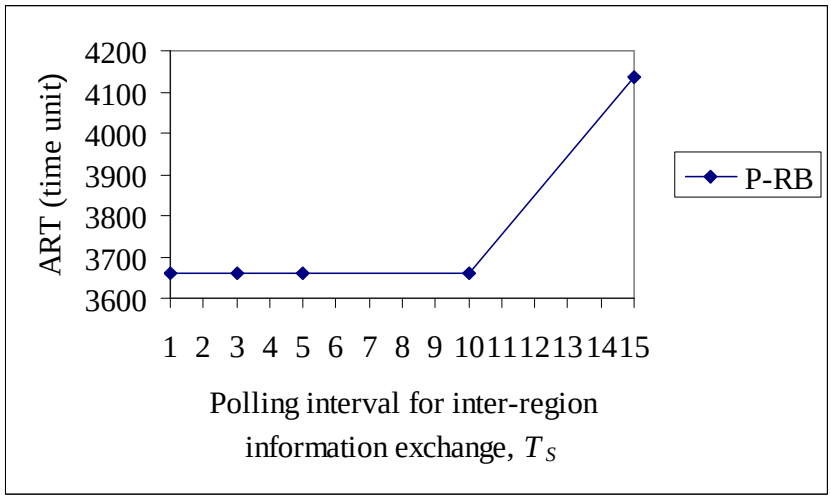

Figure 6.6: Sensitivity of performance to different polling intervals at system loading 0.9

6.2.6 Sensitivity to polling interval T_S

The parameter T_S controls the trade-off between inter-region network traffic and the quality of load balancing: intuitively, a smaller value of T_S provides a better balance at the expense of greater inter-region network traffic. To observe the impact of this

parameter on the performance, we conducted simulations for various values in the range 1–15. We set the system loading to 0.5 and 0.9. Selected results are shown in Figures 6.5 and 6.6.

At the low system loading (ρ = 0.5), the average response time increases as we move from left to right. This is because more lightly loaded sites are available in the system at low system loading. At the high system loading (ρ = 0.9), performance is not sensitive to a range of values (from 1 to 10) since lightly loaded sites become fewer and fewer. This suggests that at high system loading it is more preferable to make load balancing inside regions than across regions.

6.2.7 Sensitivity to accuracy of estimation of job execution cost

P-RB requires estimated job execution cost. We assessed the impact of incorrect job execution cost estimation on the algorithm. The evaluation method is the same as described in Section 5.3.8. The setting of system loading is the same as one in Section 6.2.3. Figure 6.7 shows the performance for P-RB when job execution cost is estimated (1) accurately, and (2) with a range size of prediction errors. The Figure also shows the performance of the algorithm without load balancing.

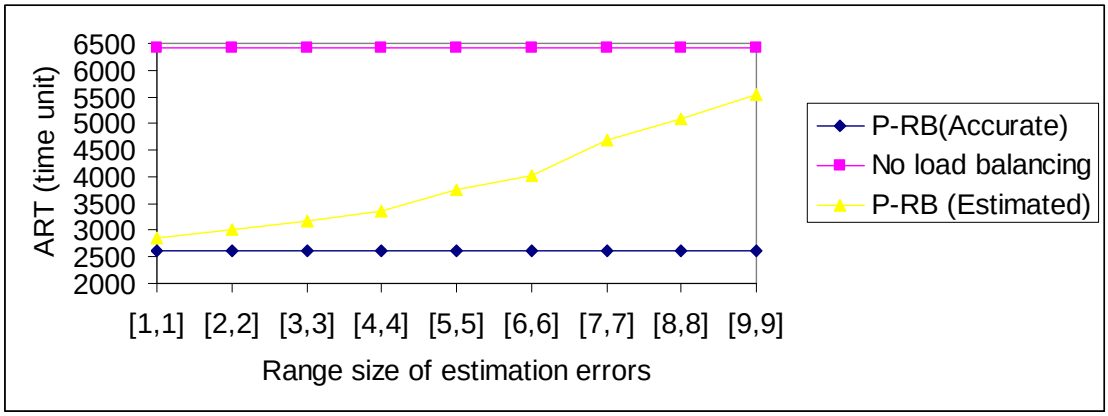

Figure 6.7: The impact of range size of prediction errors on the performance of algorithm P-RB

In Figure 6.7, *a* and *b* increase from 10% to 90%, in increments of 10%. This results in the range of predicted error for the actual execution cost of *x* increasing from [−0.1*x*, 0.1*x*] to [−0.9*x*, 0.9*x*], while the average predicted error remains unchanged (at 0). The Figure

shows that as *a* and *b* increase, there is an overall increase in the *ART*; this was due to highly inaccurate information, especially when the error is large. But the performance of P-RB is still better than the situation without load balancing when a and b reach [9, 9].

6.3 Summary

A performance-driven, region-based load-balancing algorithm has been proposed for computational grids. Our algorithm does not require a centralised decision scheme. Instead, we developed a decentralised load-balancing mechanism for the intra-region and inter-region load balancing directly in the sites. Intra-region communication can be minimised by MIF. To control inter-region communication, at most a remote regional broker site is queried at a given time interval.

We extensively evaluated the algorithm using simulations. The results have shown that it performs better than MCT, by requiring shorter average response times of jobs.

7

Conclusion and future work

This dissertation has studied the issue of load-balancing in large-scale heterogeneous computational grids. To explore the solution space for load-balancing in such environments, we designed a survey for load-balancing solutions. We have developed a system model to study load balancing problems in computational grid environments. In particular, we have developed three decentralised algorithms for job dispatching and load-balancing that use only partial information. All of them are scalable, dynamic, decentralised and sender-initiated. We have built decentralised schemes that are capable of efficient load assignment and redistribution to minimise the average response time of jobs, despite the scalability of grid systems, the heterogeneous processing power of grid sites, and considerable communication overheads involved in information collection. This chapter concludes the dissertation by summarising the major contributions and describing future research directions.

Section 7.1 highlights the main contributions. Section 7.2 focuses on future directions, which are extensions of our past and current research on decentralised load-balancing support for heterogeneous computational grids.

7.1 Main contributions

7.1.1 Desirability-aware load-balancing algorithm

Chapter 4 presented a new desirability-aware load balancing algorithm for heterogeneous computational grids.

We gave two criteria for site desirability: processing power and transfer delay. For each site s_i in a grid, our algorithm uses the desirability of other sites to s_i to form k number of partners and p number of neighbours for the site s_i. Partners are sites with comparable or greater processing power, and neighbours are nearby sites with low transfer delays. We have designed an approach for constructing the partner sites for each site when a site joins the grid. We determined an approach to enable the set of partners for a site to be updated dynamically at runtime based on feedback information, and a relatively simple approach to form neighbouring sites for each grid site.

Rather than using the conventional periodic or polling approaches, state information exchange between a site and its partners or neighbours is performed via Mutual Information Feedback (MIF) to reduce communication overheads.

The algorithm comprises two specific policies for load distribution: Instantaneous Distribution Policy (IDP) and Load Adjustment Policy respectively (LAP). When a new job arrives at a site, it either remains at that site or is immediately allocated by IDP to it or to one of its partner sites. Due to the likely fluctuating behaviour of grid resources, continuous load adjustment is employed among neighbour sites under the guidance of LAP to better exploit the grid environment.

Extensive simulation studies were conducted to analyse the performance of our load-balancing algorithm. The algorithm was compared to the Nearest Neighbour load-balancing algorithm. The results show that our algorithm performs better than the Nearest Neighbor algorithm, and reduces the average job response time over a wide range of system parameters.

Our algorithm (which considers the heterogeneity of sites) makes more powerful sites carry higher loads, because jobs executed at fast sites are more likely to execute at high speed. From the system perspective, our load-balancing scheme – which takes into account the different network transfer delay between sites – enables quick responses to load imbalances. In other words, the desirability-aware approach is "greedy" in the sense that it tries, at each step, to make jobs assignments at lightly loaded sites.

7.1.2 Performance-driven desirability-aware load-balancing algorithm

Chapter 5 presented a novel performance-driven load-balancing algorithm for heterogeneous computational grids while considering the site desirability.

The accumulated job execution time is defined as load index of a site. We included the dynamic communication cost in the cost calculation for job execution at a remote site, and how to determine the performance benefit that a job can gain for execution at a remote site. The state information exchange is done via MIF.

The algorithm uses site desirability to guide load assignments (Chapter 4), and integrates three dynamic approaches to make load distribution and redistribution driven by the performance benefit that jobs can gain: IDP, LAP, and Augmented Load Adjustment Policy (ALAP).

- ***IDP***. A new job arriving at a site is immediately allocated to that site or one of its partner sites, giving the job maximum performance benefit.
- ***LAP***. The load adjustment policy aims to continuously reduce load difference among a site and its neighbours by transferring the job that benefits most in the global job queue of that site.
- ***ALAP***. LAP causes a bottleneck in less powerful sites and their overlapping neighbourhoods. The load adjustment can be further accomplished by transferring to an augmented neighbour the job that benefits most from being in the global job queue. How to find the augmented neighbour and how to trigger the ALAP were described.

The performance-driven approach is “greedy” in the sense that it tries, at each step, to make job assignments at the site that can provide most performance benefit. We extensively evaluated the algorithm using simulations. Those results showed that our algorithm outperforms the Performance-driven Neighbours-based algorithm, while having a closer performance to the unrealistic algorithm, Central.

7.1.3 Performance-driven Region-based load-balancing algorithm

Chapter 6 presented a Performance-driven Region-based load-balancing algorithm for heterogeneous computational grids by applying clustering approach. The algorithm is partially based on research that was presented in Chapter 5.

The grid sites are clustered into regions around a set of known broker sites in terms of network transfer delay; the regional brokers are organised in a fully decentralised fashion. We developed a decentralised load-balancing mechanism for the intra-region and inter-region load balancing directly in the sites. For each regional grid, our algorithm integrates static IDP and dynamic LAP to make load distribution and redistribution driven by the performance benefit that jobs can gain. The LAP also considers load redistribution across regional grids. The intra-region communication is minimised by MIF. To control inter-region communication, the random polling of a remote regional broker site is performed by each regional broker site at a set time interval.

We used simulations to extensively evaluate the algorithm, and showed that it performs better than Minimum Completion Time algorithm (MCT).

7.1.4 Discussion

Optimising workload allocation for heterogeneous grid systems is not an easy task. The assignment of jobs to processing sites is done in such a way as to minimise the average response time of jobs while minimising the overhead from communication delay. Owing to the dynamic nature of the grid computing environment, designing an ideal load-balancing algorithm on it remains a challenge. We hope our algorithms can serve as examples for continuing work on research into decentralised load-balancing solutions.

7.2 Future work

In the course of designing and evaluating decentralised load-balancing schemes for heterogeneous computational grids, we have found several interesting issues that need further investigation. These open issues are as follows.

7.2.1 Replication as fault-tolerant strategy

Our work points to the need to address the problem of efficient utilisation and satisfactory response time, and the problem of fault-tolerance for job scheduling and load-balancing in computational grids. Therefore, as grids are increasingly used for jobs requiring high levels of performance and reliability, the ability to tolerate failures while effectively exploiting resources in a scalable and transparent manner must be an integral part of grid computing resource management systems.

One future direction is to integrate job replication strategy and our load-balancing algorithms. Replication strategy has been widely used for job scheduling in computational grids. It attains good performance without relying on information about the grid or the job, although consuming a few more cycles. It can provide fault-tolerance and decreased completion time.

7.2.2 Incorporation of security concerns

Grids are mostly formed with resources owned by many organisations and thus are not dedicated for certain users. As such, jobs that are dispatched to a remote site may experience security and reliability problems if the site is attacked by malicious users, such that the jobs it is executing are destroyed. A grid job scheduler must be security-driven, in that it must consider the risk involved in dispatching jobs to remote sites. Each grid site can be modelled by a parameter called the security level that a grid site can offer to remote jobs. Applying the notion of security into our load balancing algorithms is clearly a research opportunity.

7.2.3 Consideration of resource availability

In the grid environment, numerous sites are capable of providing computing resources. Some of these sites are frequently idle and able to constantly share computing resources; some, however, are not. Therefore, when selecting sites for distributing jobs, if an inefficient site is chosen, redistribution of jobs may frequently occur, and thus a reduction in the execution performance of the system. How to select efficient sites is an issue worthy of further investigation. This is an interesting research direction – to integrate the idle-time concept with our load-balancing algorithms.

Most computing sites have daily routines with few idle day-time cycles and large chunks of idle night or early-morning time cycles. In addition, the computing sites are geographically distributed in different time zones on the grids. During a 24-hour cycle, the area that contains the most idle computers can change. The jobs may be distributed to sites located in idle night or early-morning time zones around the global.

7.2.4 Applying economic models for load balancing

The economic approaches have recently attracted considerable attention for job scheduling and load balancing in large distributed system environments. The goal of grid economy is not necessarily to determine the best resource for the execution of every job, but to improve the distribution of the overall workload to maximise the number of jobs that can simultaneously achieve a Quality of Service (QoS) objective, while limiting the massive complexity and the computational overhead of the scheduling process.

This opens the possibility of investigating the dynamic adjustment of resource prices, thus enabling a site broker to make load-balancing decisions on the basis of economic models – for example, the use of queue waiting time as a stimulus for price adjustment.

7.2.5 Load-balancing scheme for data grid

A data grid is a collection of geographically dispersed storage resources over a wide area network. The goal of a data grid system is to provide a large virtual storage framework with unlimited power through collaboration among individuals, institutions and resources.

We expect that heterogeneity will be a big challenge for data-intensive applications running on data grids, where interconnections are relatively slow and network latencies high. Some data sites may be over-utilised, while others may be under-utilised. In designing an efficient load-balancing mechanism for data grids, the performance of our load-balancing algorithms in such environments needs to be investigated.

References

1. I. Foster, C. Kesselman (Eds.), The Grid: blueprint for a new computing infrastructure, Morgan-Kaufmann Publishers, 1st Edition 1999, 2nd Edition 2003.
2. W. M. Jones, L. W. Pang, D. Stanzione, W. B. III. Ligon, Job communication characterization and its impact on meta-scheduling co-allocated jobs in a mini-grid**,** in: Proceedings of the 18th International Parallel and Distributed Processing Symposium, 26-30 April 2004, pp:253-260.
3. I. Foster, C. Kesselman, S. Tuecke, The anatomy of the Grid: enabling scalable virtual organizations, The International Journal of High Performance Computing Applications 15 (3) (2001) 200–222.
4. M. Anirban, K. Kennedy, C. Koelbel, G. Marin, J. Mellor-Crummey, B. Liu, L. Johnsson, Scheduling strategies for mapping application workflows onto the grid, in: Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing (HPDC-14), 24-27 July 2005, pp:125 – 134.
5. R. Buyya (ed.), High performance cluster computing: architectures and systems, vol. 1 and vol. 2, Prentice-Hall: Englewood Cliffs, NJ, 1999.
6. Community scheduler framework. http://www.globus.org/toolkit/docs/4.0/contributions/csf.

7. Foster, I., Kesselman, C., and Tuecke, S., The anatomy of the Grid: enabling scalable virtual organizations, International Journal of High Performance Computing Applications, 15 (3) (2001) 200–222.
8. Platform Enterprise Grid Orchestrator (EGO). http://www.platform.com/Products/Platform.Enterprise.Grid.Orchestrator/Product.Information/.
9. Moab grid suite. http://www.clusterresources.com/pages/products/moab-grid-suite.php.
10. Globus, http://www.globus.org/.
11. I. Foster, C. Kesselman, Globus: A metacomputing infrastructure toolkit, International Journal of Supercomputer Applications, 11 (2) (1997) 115–128.
12. Chapin, S., Karpovich, J., and Grimshaw, A. (1999). The Legion resource management system. In: Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing, April 16, 1999, San Juan, Puerto Rico, Lecture Notes in Computer Science (LNCS), Vol. 1659, pp. 162-178.
13. H. Casanova, A. Legrand, D. Zagorodnov and F. Berman, Heuristics for scheduling parameter sweep applications in Grid environments, in: Proceedings of the 9th Heterogeneous Computing workshop (HCW'2000), May 1, 2000, Cancun, Mexico, pp. 349-363.
14. R. Buyya, J. Abramson, and J. Giddy, Nimrod/G: architecture for a resource management and scheduling system in a global computational Grid, in: Proceedings of 4th IEEE Conference on High-Performance Computing in the Asia-Pacific Region, 14-17 May 2000, Beijing, China, pp. 283-289.
15. M. Sato, H. Nakada, S. Sekiguchi, S. Matsuoka, U. Nagashima and H. Takagi, Ninf: a network based information library for global world-wide computing infrastructure, in: Proceedings of the International Conference on High Performance Computing and Networking Europe (HPCN Europe), Vienna, Austria, 28-30 April 1997, Lecture Notes in Computer Science (LNCS), Vol. 1225, pp. 491-502.
16. J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke, Condor-G: a computation management agent for multi-institutional Grids, in: Proceedings of

the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10), San Francisco, CA, USA, 7-9 August 2001, pp. 55-63.

17. S. Venugopal, R. Buyya and L. Winton, A Grid service broker for scheduling e-science applications on global data Grids, Concurrency and Computation: Practice and Experience, 18(6) (2005) 685-699
18. T. Hey and A. E. Trefethen, The UK e-science core programme and the Grid, Future Generation Computer Systems, 18(8) (2002) 1017-1031.
19. W. Hoschek , J. Jaen-Martinez, A. Samar , H. Stockinger , K. Stockinger, Data management in an international data Grid project, in: Proceedings of the first IEEE/ACM International Workshop on Grid Computing, Bangalore, India, 7 December 2000, Lecture Notes in Computer Science (LNCS), Vol. 1971, pp. 77-90.
20. Grid Physics Network (GriPhyN).http://www.griphyn.org/
21. W. Johnston, D. Gannon, and B. Nitzberg. Grids as production computing environments: The engineering aspects of NASA's Information Power Grid. In: Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing, Redondo Beach, CA, USA, 3-6 August 1999, pp. 197-204.
22. S. Zhou, X. Zheng, J. Wang, P. Delisle, Utopia: a load sharing facility for large heterogeneous distributed computer systems, Software - Practice and Experience, 23(12) (1993) 1305-1336.
23. J.M. Schopf, M. D'Arcy, N. Miller, L. Pearlman, I. Foster, C. Kesselman, Monitoring and discovery in a Web services framework: Functionality and performance of the Globus Toolkit's MDS4, in: Technical Report ANL/MCS-P1248-0405, Argonne National Laboratory, Argonne, IL, 2005.
24. K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman, Grid information services for distributed resource sharing. In: Proceedings of the 10th IEEE International Symposium on High-Performance Distributed Computing (HPDC 2001), San Francisco, CA, USA, 6–9 August 2001, pp. 181-194.
25. X. Zhang, J.L. Freschl J, J.M. Schopf, A performance study of monitoring and information services for distributed systems. In: Proceedings of the 12th IEEE

International Symposium on High-Performance Distributed Computing (HPDC 2003), Seattle, WA, USA, 22–24 June 2003.

26. X. Zhang, J.M. Schopf, Performance analysis of the Globus Toolkit monitoring and discovery service, MDS2, in: Proceedings of the International Workshop on Middleware Performance (MP 2004) at IPCCC 2004, April 2004.
27. G. Coulouris, J. Dollimore, K. Kinderberg, Distributed Systems: Concepts and Design. Addison-Wesley: Reading, MA, 1995.
28. V. Subramani, R. Kettimuthu, S. Srinivasan, and P. Sadayappan, Distributed job scheduling on computational grids using multiple simultaneous requests, in: Proceedings of the 11th International Symposium for High Performance Distributed Computing, 23-26 July 2002, pp. 359-366.
29. Maui. http://www.clusterresources.com/pages/products/maui-cluster-scheduler.php.
30. Load Sharing Facility. http://www.platform.com/Products/Platform.LSF.Family/.
31. Portable Batch System. http://www.openpbs.org/.
32. Sun Grid Engine / CODEINE. http://www.sun.com/software/gridware/index.xml.
33. LoadLeveler. http://www-03.ibm.com/systems/clusters/software/loadleveler.html.
34. COSY. http://www.ccrl-nece.de/~falk/COSY/cosy.shtml.
35. Condor. http://www.cs.wisc.edu/condor/.
36. MOSIX. http://www.mosix.org/.
37. A. Barak A. and A. Shiloh, The MOSIX2 management system for linux clusters and organizational Grids, white paper, March 2007.
38. R. Wolski, N. Spring, J. Hayes, The network weather service: A distributed resource performance forecasting service for metacomputing, Journal of Future Generation Computing Systems, 15 (5–6) (1999) 757–768.
39. P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, L. Zhang, IDMaps: a global internet host distance estimation service, IEEE/ACM Transactions on Networking, 9 (5) (2001) 525–540.
40. A. Agrawal, H. Casanova, Clustering hosts in P2P and global computing platforms, in: Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, 12–15 May 2003, pp. 367–373.

41. S. Ratnasamy, M. Handley, R.M. Karp, S. Shenker, Topologically-aware overlay construction and server selection, in: Proceedings of IEEE INFOCOM, 23–27 June 2002, volume 3, pp. 1190–1199.
42. M.A. Baker, G.C. Fox, and H.W. Yau, Review of cluster management software, NHSE Review, July 1996, available at http:// http://nhse.cs.rice.edu/NHSEreview/CMS/.
43. H. Kameda, J. Li, C. Kim, Y. Zhang, Optimal load balancing in distributed computer systems, Springer, London, 1997
44. J. Li, H. Kameda, Load balancing problems for multiclass jobs in distributed/parallel computer systems, IEEE Transactions on Computers 47 (3) (1998) 322–332
45. X. Tang, S.T. Chanson, Optimizing static job scheduling in a network of heterogeneous computers, in: Proceedings of the International Conference on Parallel Processing, 21-24 August 2000, pp. 373–382.
46. D. Grosu, A.T. Chronopoulos, M.Y. Leung, Load balancing in distributed systems: an approach using cooperative games, in: Proceedings of the International Parallel and Distributed Processing Symposium, 15-19 April 2002, pp. 52–61, IEEE Computer Society Press.
47. D. Grosu, A.T. Chronopoulos, Algorithmic mechanism design for load balancing in distributed systems, IEEE Transactions on Systems, Man and Cybernetics - Part B, 34(1) (2004) 77-84.
48. Z. Zeng and B. Veeravalli, Design and analysis of a non-preemptive decentralized load balancing algorithm for multi-class jobs in distributed networks, Computer Communications, 27(7) (2004) 679-694.
49. D. Grosu, A.T. Chronopoulos, Noncooperative load balancing in distributed systems, Journal of Parallel and Distributed Computing, 65(9) (2005) 1022-1034.
50. S. Penmatsa, A.T. Chronopoulos, Cooperative load balancing for a network of heterogeneous computers, in: Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium, 25-29 April 2006 Page(s):8.

51. S. F. El-Zogdhy, H. Kameda, and J. Li, Numerical studies on a paradox for non-cooperative static load balancing in distributed computer systems, Computers and Operations Research, 33(2) (2006) 345-355.
52. M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, R. Freund, Dynamic mapping and scheduling of a class of independent tasks onto heterogeneous computing systems, Journal of Parallel and Distributed Computing, 59 (2) (1999) 107-131.
53. V. Berten, J. Goossens, and E. Jeannot, On the distribution of sequential jobs in random brokering for heterogeneous computational Grids, IEEE Transactions on Parallel and Distributed Systems, 17 (2) (2006) 113-124.
54. S.P. Dandamudi, Sensitivity evaluation of dynamic load sharing in distributed systems, IEEE Concurrency, 6 (3) (1998) 62-72.
55. Eager D. L., Lazowska E. D., and Zahorjan J. A comparison of receiver-initiated and sender-initiated adaptive load sharing. Performance Evaluation, 6(1) (1986) 53–68.
56. S. Zhou, A trace-driven simulation study of dynamic load balancing, IEEE Transactions on Software Engineering, 14 (9) (1988) 1327–1341.
57. P. Krueger and N.G. Shivaratri, Adaptive location policy for global scheduling. IEEE Transaction Software Engineering, 20(6) (1994) 432–444.
58. O. Kremien and J. Kramer, Methodical analysis of adaptive load sharing algorithms. IEEE Transactions on Parallel and Distributed Systems, 3(6) (1992)747–760.
59. N.G. Shivaratri, P. Krueger, M. Singhal, Load distributing for locally distributed systems, Computer, 25 (12) (1992) 33–44.
60. H.-C. Lin, C. S. Raghavendra, A dynamic load-balancing policy with a central job dispatcher (LBC), IEEE Transactions on Software Engineering, 18 (2) (1992) 145–158.
61. M.H. Willebeek-LeMair, A.P. Reeves, Strategies for dynamic load balancing on highly parallel computers, IEEE Transactions on Parallel and Distributed Systems, 4 (9) (1993) 979–993

62. Eager D. L., Lazowska E. D., and Zahorjan J. Adaptive load sharing in homogeneous distributed systems. IEEE Transactions on Software Engineering, 12 (5) (1986) 662–675.
63. K. Benmohammed-Mahieddine, P.M. Dew, and M. Kara, A periodic symmetrically initiated load balancing algorithm for distributed systems. In: Proceedings of the 14th International Conference on Distributed Computing Systems, 21-24 June 1994, Poznan, Poland, pp. 616–623.
64. M.J. Zaki, W. Li, S. Parthasarathy, Customized dynamic load balancing for a network of workstations, Journal of Parallel and Distributed Computing 43 (2) (1997) 156–162.
65. C.Z. Xu and F.C.M. Lau, Iterative dynamic load balancing in multicomputers, Journal of the Operational Research Society, 45 (7) (1994) 786-796.
66. F. C. H. Lin and R. M. Keller, The gradient model load balancing method, IEEE Transaction on Software Engineering, 13 (1) (1987) 32-38.
67. J. Watts, S. Taylor, A practical approach to dynamic load balancing, IEEE Transactions on Parallel and Distributed Systems 9 (3) (1998) 235–248.
68. K. Antonis, J. Garofalakis, P. Spirakis, A competitive symmetrical transfer policy for load sharing, in: Proceedings of the 4th International Euro-Par Conference on Parallel Processing**,** 1-4 September 1998, UK, Lecture Notes in Computer Science (LNCS), Vol. 1470, pp. 352-355.
69. L. Anand, D. Ghose, V. Mani, ELISA: an estimated load information scheduling algorithm for distributed computing systems, Computers and Mathematics with Applications 37 (8) (1999) 57–85.
70. M. C. Luis and D. S. Isaac, Rate of change load balancing in distributed and parallel systems, Parallel Computing, 26 (9) (2000) 1213–1230.
71. M. Mitzenmacher, How useful is old information?, IEEE Transactions on Parallel and Distributed Systems, 11 (1) (2000) 6–20.
72. M. Dahlin, Interpreting stale load information, IEEE Transactions on Parallel and Distributed Systems, 11(10) (2000) 1033–1047.

73. Y. Amir, B. Awerbuch, A. Barak, R. Sean Borgstrom, A. Keren, An opportunity cost approach for job assignment in a scalable computing cluster, IEEE Transactions on Parallel and Distributed Systems, 11 (7) (2000) 760–768.
74. W. Shu and L.V. KaleÂ, A dynamic scheduling strategy for the chare-kernel system, in: Proceedings of the 1989 ACM/IEEE conference on Supercomputing, November 1989, Reno, Nevada, USA, pp. 389-398.
75. K. Antonis, J. Garofalakis, I. Mourtos, and P. Spirakis, A hierarchical adaptive distributed algorithm for load balancing, Journal of Parallel and Distributed Computing, 64 (1) (2004) 151-162.
76. Z. Zeng and B. Veeravalli, Design and performance evaluation of queue-and-rate-adjustment dynamic load balancing policies for distributed networks, IEEE Transactions on Computers, 55 (11) (2006) 1410-1422.
77. P. Dikshit, S. K. Tripathi and P. Jalote, SAHAYOG: A test bed for evaluating dynamic load sharing policies, Software - Practice and Experience, 19 (5) (1989) 411–435.
78. M. M. Theimer and K. A. Lantz, Finding idle machines in a workstation-based distributed system, IEEE Transactions on Software Engineering, 15(11) (1989) 1444–1458.
79. H.Shan, L.Oliker, and R.Biswas, Job superscheduler architecture and performance in computational grid environments, in: Proceedings of the ACM/IEEE conference on Supercomputing, 15-21 November 2003.
80. M. Harchol-Balter, A. B. Downey, Exploiting process lifetime distributions for dynamic load balancing, ACM Transactions on Computer Systems, 15 (3) (1997) 253-285.
81. W. Zhu, P. Socko, B. Kiepuszewski, Migration impact on load balancing—an experience on Amoeba, ACM SIGOPS Operating Systems Review, 31(1) (1997) 43–53.
82. D.L. Eager, E.D. Lazowska, J. Zahorjan, The limited performance benefits of migrating active processes for load sharing, ACM SIGMETRICS Performance Evaluation Review, 16 (1) (1988) 63–72 .

83. S. Lu and L. Xie. A scalable load balancing system for nows. ACMSIGOPS Operating Systems Review, 32 (3) (1998) 55–63.
84. C.Z. Xu and F.C.M. Lau, Optimal parameters for load balancing with the diffusion method in mesh networks, Parallel Processing Letters, 4 (1-2) (1994) 139-147.
85. X. Qian, and Q. Yang, Load balancing on generalized hypercube and mesh multiprocessors with LAL, in: Proceedings of 11th International Conference on Distributed Computing Systems. 20-24 May 1991, pp. 402 –409.
86. B. Shirazi, A.R. Hurson, K. Kavi, Scheduling and Load Balancing in Parallel and Distributed Systems, IEEE book, May 1995.
87. Y. Lan and T. Yu, A dynamic central scheduler load-balancing mechanism, in: Proceedings of IEEE 14th Annual International Phoenix Conference on Computers and Communications, 28-31 March 1995, pp. 734-740.
88. Shivaratri N. G. and Krueger P. Two adaptive location policies for global scheduling algorithms. In: Proceedings of the 14th International Conference Distributed Computer Systems, 28 May – 01 June 1990, pp. 502–509.
89. A. Barak and O. La'adan, The MOSIX multicomputer operating system for high performance cluster computing, Future Generation Computer Systems, 13 (4-5) (1998) 361-372.
90. D. Z. Gu, L. Yang and L. R. Welch, A Predictive, Decentralized Load Balancing Approach, in: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, Denver, Colorado, 04-08 April 2005.
91. R. Mirchandaney, D. Towsley, J. A. Stankovic, A adaptive load sharing in heterogeneous distributed systems, Journal of Parallel and Distributed Computing, 9 (4) (1990) 331-346.
92. R. Mirchandaney, D. Towsley, J. A. Stankovic, A Analysis of the effects of delays on load sharing, Transactions on Computers, 38 (11) (1989) 1513-1525.
93. A.Y. Zomaya, and Teh Yee-Hwei, Observations on using genetic algorithms for dynamic load-balancing, IEEE Transactions on Parallel and Distributed Systems, 12 (9) (2001) 899 – 911.

94. G. Cybenko, Dynamic load balancing for distributed memory multi-processors, Journal of Parallel and Distributed Computing, 7 (1989) 279–301.
95. J. Song, A partially asynchronous and iterative algorithm for distributed load balancing, Parallel Computing, 20 (6) (1994) 853-868.
96. Y. F. Hu, R. J. Blake, An improved diffusion algorithm for dynamic load balancing, Parallel Computing 25 (4) (1999) 417–444.
97. L. He, S. A. Jarvis, D. P. Spooner, X. Chen, G. R. Nudd, Hybrid performance-based workload management for multiclusters and grids, IEE Proceedings Software, 151(5) (2004) 224-231.
98. C. C. Hui and S. T. Chanson, Theoretical analysis of the heterogeneous dynamic load balancing problem using a hydrodynamic approach, Journal of Parallel and Distributed Computing, 43 (2) (1997) 139–146.
99. C.C. Hui, S.T. Chanson, Hydrodynamic load balancing, IEEE Transactions on Parallel and Distributed Systems, 10 (11) (1999) 1118–1137.
100. R. Diekmann, A. Frommer, B. Monien, Efficient schemes for nearest neighbor load balancing, Parallel Computing, 25 (7) (1999) 789–812.
101. R. Elsasser, B. Monien, R. Preis, Diffusion schemes for load balancing on heterogeneous networks, Theory of Computing Systems, 35 (3) (2002) 305–320.
102. K. Benmohammed-Mahieddine, P. Dew, A Periodically symmetrically initiated load balancing algorithm for distributed systems, Operating Systems review 28 (1) (1994) 66–77.
103. Y.Wong, K.Leung, and K.Lee, A stochastic load balancing algorithm for i-computing, Concurrency and Computation: Practice and Experience, 15(1) (2003) 55-78.
104. D. J. Evans and W. U. N. Butt, Load balancing with network partitioning using host groups, Parallel Computing, 20(3) (1994) 325-345.
105. C. Xu, F. Lau, B. Monien, and R. Luling, Nearest neighbor algorithms for load balancing in parallel computers, Concurrency: Practice and Experience, 7 (7) (1995) 707-736.
106. P. Sanders, Analysis of nearest neighbor load balancing algorithms for random loads. Parallel Computing, 25 (8) (1999) 1013-1033.

107. T. Thanalapati and S. Dandamudi, An efficient adaptive scheduling scheme for distributed memory multicomputers, IEEE Transactions on Parallel and Distributed Systems, 12 (7) (2001) 758–768.

108. V. Stergios and K. C. Sevcik, Parallel application scheduling on networks of workstations, Journal of Parallel and Distributed Computing, 43(1) (1997) 1159–1166.

109. M. Harchol-Baker, M. E. Crovella, and C. D. Murta, On choosing a task assignment policy for a distributed server system, Journal of Parallel and Distributed Computing, 59 (2) (1999) 204-228.

110. H.Y. Sit, K.S. Ho, R.W.P. Luk, L.K. Ho, An adaptive clustering approach to dynamic load balancing, in: Proceedings of the 7th International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'04), 2004, pp. 415–420, IEEE Computer Society Press.

111. X. Deng, H. Liu, J. S. Long, and B. Xiao, Competitive analysis of network load balancing, Journal of Parallel and Distributed Computing, 40 (2) (1997) 162–172.

112. C. Xu and F. Lau, Load balancing in parallel computers. Theory and Practice. Kluwer Academic Publishers, 1997.

113. K. K. Goswami, M. Deverakonda and R. K. Iyer, Prediction-based dynamic load-sharing heuristics, IEEE Transactions on Parallel and Distributed Systems, 4 (6) (1993) 638–648.

114. P.K.K. Loh, W.J. Hsu, C. Wentong, N. Sriskanthan, How network topology affects dynamic load balancing, IEEE Parallel and Distributed Technology 4 (3) (1996) 25-35.

115. D. Arredondo, M. Errecalde, S. Flores, F. Piccoli, M. Printista, R. Gallard Embedded intelligent assistance for load distribution and balancing, in: Proceedings of the 9th International Conference on Parallel and Distributed Computing and Systems, October 1997, pp.188-195.

116. A. Corradi, L. Leonardi, F. Zambonelli, Diffusive load-balancing policies for dynamic applications, IEEE Concurrency, 7 (1) (1999) 22-31.

117. S. Ruchir, V. Bharadwaj, and M. Manoj, On the design of adaptive and de-centralized load balancing algorithms with load estimation for computational grid environments, To appear in IEEE Transactions on Parallel and Distributed Systems, 2007.

118. K. Y. Kabalan, W. W. Smari, and J. Y. Hakimian, Adaptive load sharing in heterogeneous systems: Policies, modifications, and simulation, International Journal of Simulation, Systems, Science and Technology, 3 (1-2) (2002) 89-100.

119. M. Arora, S.K. Das, R. Biswas, A de-centralized scheduling and load balancing algorithm for heterogeneous Grid environments, in: Proceedings of the International Conference on Parallel Processing Workshops, 18–21 August 2002, pp. 499–505.

120. R. V. van Nieuwpoort, T. Kielmann, and H. E. Bal, Efficient load balancing for wide-area divide-and-conquer applications, in: Proceedings of the Eighth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'01), Snowbird, UT, 18-19 June 2001, pp. 34-43.

121. W. Yibing, R. Hyatt, An improved algorithm of two choices in randomized dynamic load-balancing, in: Proceedings of the Fifth International Conference on Algorithms and Architectures for Parallel Processing, 23-25 October 2002, pp. 440-445.

122. M. Mitzenmacher, The power of two choices in randomized load balancing, IEEE Transactions on Parallel and Distributed Systems 12 (10) (2001) 1094–1104.

123. Y. Azar, A.Z. Broder, A.R. Karlin, and E. Upfal, Balanced allocations, SIAM Journal on Computing, 29 (1) (1999) 180-200.

124. K. G. Shin, C. Yi-Chieh, A coordinated location policy for load sharing in hypercube-connected multicomputers, IEEE Transactions on Computers, 44 (5) (1995) 669 – 682.

125. C. Hou and K. G. Shin, Implementation of decentralized load sharing in networked workstations using the condor package. Journal of Parallel and Distributed Computing, 40 (2) (1997) 173-184.

126. X. Zhang, Y. Qu, and L. Xiao, Improving distributed workload performance by sharing both CPU and memory resources, in: Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS'2000), Taipei, Taiwan, 10-13 April 2000, pp. 233-241.

127. K.Q. Yan, S.C. Wang, C.P. Chang and J.S. Lin, A hybrid load balancing policy underlying grid computing environment, Computer Standards & Interfaces, 29 (2) (2007) 161-173.

128. H. Chi-Chung, S.T. Chanson, Improved strategies for dynamic load balancing. IEEE Concurrency, 7 (3) (1999) 58 – 67.

129. S. T. Chanson, D. Wantao, H. Chi-Chung, T. Xueyan, T. Mingyan, Multidomain load balancing, in: Proceedings of International Conference on Network Protocols, 14-17 November 2000, pp. 315 – 324.

130. S. P. Dandamudi, K. C. M. Lo, A hierarchical load sharing policy for distributed systems, in: Proceedings of the Fifth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '97), 12-15 Janauary 1997, pp:3 – 10.

131. M. Avvenuti, L. Rizzo, and L. Vicisano, A hybrid approach to adaptive load sharing and its performance, Journal of Systems Architecture, 42 (9-10) (1997) 679-696.

132. C. Junwei, D. P. Spooner, S. A. Jarvis, S. Saini, G. R. Nudd, Agent-based grid load balancing using performance-driven task scheduling, in: Proceedings. International Parallel and Distributed Processing Symposium, 22-26 April 2003.

133. S. Vadhiyar, J. Dongarra, Self adaptivity in Grid computing, Concurrency and Computation: Practice and Experience 17 (2-4) (2005) 235-257.

134. E. A. Billard and J. C. Pasquale, Load balancing to adjust for proximity in some network topologies, Parallel Computing, 22 (14) (1997) 2007-2023.

135. K. Nishimura, H. Ueno, M. Yamamoto, H. Ikeda, A dynamic load balancing method based on network delay for large distributed systems, Electronics and Communications in Japan (Part I: Communications), 84 (6) (2001) 11-21.

136. T. Kunz, The influence of different workload descriptions on a heuristic load balancing scheme, IEEE Transactions on Software Engineering, 17 (7) (1991) 725–730.

137. Z. Xu, C. Tang, and Z. Zhang, Building Topology-Aware Overlays Using Global Soft-State, in: Proceedings of the 23rd International Conference of Distributed Computing Systems, 19–22 May 2003, pp. 500-508.

138. S. Xian-He, W. Ming, GHS: A performance system of Grid computing, in: Proceedings of the 19th IEEE International Symposium on Parallel and Distributed Processing, 4–8 April 2003.

139. G. R. Nudd, D. J. Kerbyson, E. Papaefstathiou, S. C. Perry, J. S. Harper, D. V. Wilcox, PACE – A toolset for the performance prediction of parallel and distributed systems, International Journal of High Performance Computing Applications, 14 (3) (2000) 228–251.

140. M.A. Iverson, F. Ozguner, L. Potter, Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment, IEEE Transactions on Computers, 48 (12) (1999) 1374–1379.

141. M. Dobber, R. D. van der Mei and G. Koole, A prediction method for job runtimes on shared processors: Survey, statistical analysis and new avenues Performance Evaluation, 64 (7-8) (2007) 755-781.

Publications from thesis

Journal Paper

- K. Lu, R. Subrata**,** and A. Y. Zomaya, On the performance-driven load distribution for heterogeneous computational Grids, Journal of Computer and System Science, 73 (8) (2007) 1191-1206 (Elsevier)

Conference Papers

- K. Lu, R. Subrata**,** and A. Y. Zomaya, An efficient load balancing algorithm for heterogeneous grid systems considering desirability of grid sites, in: Proceedings of the 25th IEEE International Conference on Performance, Computing, and Communications, 10–12 April 2006, Phoenix, Arizona, USA.

- K. Lu, R. Subrata**,** and A. Y. Zomaya, Towards decentralized load balancing in a computational grid environment, in: Proceedings of the first International Conference on Grid and Pervasive Computing, May 3-5, 2006, Taichung, Taiwan, Lecture Notes in Computer Science (LNCS), Vol. 3947, pp. 466-477, Springer-Verlag Press.

- K. Lu and A. Y. Zomaya, A hybrid policy for job scheduling and load balancing in heterogeneous computational grids, in: Proceedings of the 6th IEEE International Symposium on Parallel and Distributed Computing, 5-8 July 2007, Hagenberg, Austria.