# Distributed Systems' Load Management: Basic Issues

Luís Paulo Santos
Dep. de Informática
Universidade do Minho
*psantos@di.uminho.pt*

14th of April, 2000

**Abstract**

Effective load management is a critical requirement for the efficient execution of parallel applications on distributed systems. This report discusses some fundamental issues which must be considered on the design of any load manager. The problem of load management is characterized as a scheduling problem and the scope of this discussion is confined to that of application level scheduling. It is claimed that effective application level scheduling requires dynamic schedulers, and that the scheduler must be aware of its uncertainty about the current system's state. Furthermore, an analysis of the costs associated with scheduling is presented.

## Introduction

Parallel or distributed systems are composed of a multiplicity of heterogeneous resources, which cooperate to solve one or more computational jobs. In such systems, an application component (hereby referred to as a task) may be waiting for service at one resource, while other resources are idle [2, 10]. This idle–while–waiting condition suggests that the capabilities of such systems may be more efficiently exploited by adequately mapping the tasks onto the available resources. This mapping, which can be either static or dynamic, is the load manager's role.

This report presents some relevant issues which should be taken into account when designing or evaluating a given load management strategy. It begins by characterizing the load management problem as a scheduling problem and by identifying its main components. Section 2 confines the scope of this discussion to application level scheduling. Section 3 discusses the tradeoffs between static and dynamic approaches, whereas section 4 presents some problems and metrics related to the scheduler's evaluation. Finally, section 5 warns the reader of the unavoidable uncertainty about the environment that an application level scheduler has to face.

## 1 The Scheduling Problem

The problem of mapping a set of tasks onto a set of resources is a scheduling problem. Scheduling resources on a parallel or distributed system is a two–dimensional scheduling problem [2, 5, 18]. Local, or intranode, scheduling is concerned with scheduling within a single node. Global, or internode, scheduling is a level above this, and is concerned with scheduling among nodes. This report addresses global scheduling, assuming that local scheduling is performed by each node's operating system.

The tasks's events, system's characteristics and performance goals are the input to the scheduler, or load manager, and the mapping, or schedule, is its output. The scheduling problem has five components: the tasks (or relevant events related to them), the environment, the requirements, the scheduler and the schedule.

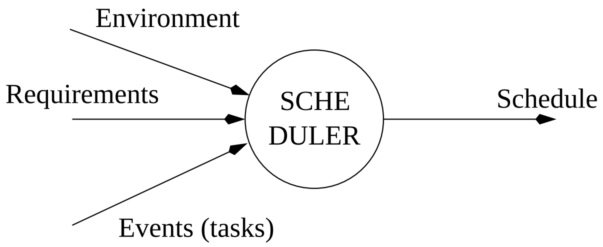


Figure 1: The scheduling problem

The environment refers to all system's characteristics that impact on the schedule, such as the available resources' absolute capabilities and current state.

The events are messages related to the workload. These can inform the scheduler of the arrival of new tasks, removal of old ones or just report a task's current state.

The requirements specify which goals the scheduler must pursue. These can include maximization of system's throughput, minimization of execution time, predictability of tasks's response times (e.g. real time systems), system's reliability (fault-tolerance) or minimization of resources' idle time.

The schedule may be either an ordered list of pairs (tasks, resources) generated *a priori* by the scheduler if all tasks and systems' characteristics are previously known, or it may be dynamically generated at run time by a set of rules. These rules specify correcting actions that redistribute tasks over the system such that some requirement is more tightly met.

On the context of load management, decision making policies can be broadly classified as static or dynamic. Static policies generate the schedule before execution time, based on the system's properties and on the tasks' requirements. This is also called the mapping problem, because a mapping function, which assigns tasks to resources before the execution begins, must be defined [1, 6]. This approach can be effective when the workload can be sufficiently well characterized before the actual execution, but it fails to adjust to fluctuations in system's load. Dynamic policies, on the other hand, generate the schedule at run time, using a set of rules to specify correcting actions that redistribute work over the system. Among dynamic policies three different approaches can be distinguished: (i) those that do not consider the system's state at each instant, deciding as if they were blind, (ii) those that use system's state information as input to their set of rules, hoping to make better decisions, and (iii) those that go a step beyond by using system's state inputs to modify either its rules' parameters or the rules themselves; these are usually classified as adaptive policies. Throughout this report only dynamically generated schedules will be considered, since no previous knowledge about incoming tasks's requirements is assumed.

The scheduler, responsible for generating the schedule, must select, at each instant, the best action to take, based on what it knows about the environment and on its performance requirements. According to this statement the scheduler is an autonomous agent which perceives the surrounding environment through its sensors and acts upon it through its effectors [20].

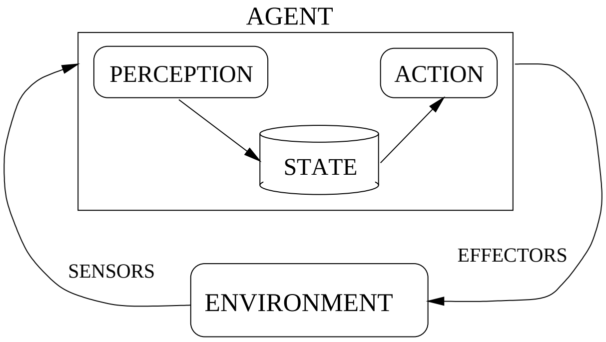


Figure 2: An agent that perceives its environment and acts upon it

In this context, autonomous means that the agent determines its behaviour and can act without the intervention of other systems. This concept can be further extended if the scheduling agent's behaviour depends, at least partially, on its previous experience, i.e., if it exhibits some learning capacity.

If the agent tries to achieve its performance objectives by selecting the most adequate action for each environmental state, then it can be classified as an intelligent agent. This does not mean that the agent is omniscient and never fails. Rather, it means that the agent operates with flexibility and rationality, by choosing the action that will, with higher probability, optimize its performance, given the agent's effectual capabilities and the information it has about the environment.

# 2 Application Level Scheduling

Most distributed systems are simultaneously shared by many users and associated applications. These compete for the same set of resources, each seeking to achieve its own performance goals. Applications on these shared systems must schedule the execution of concurrent tasks on resources whose performance varies dynamically due to the presence of other applications sharing the same resources [14].

The present work is concerned with application level scheduling, defined as scheduling performed by the application itself in order to meet its own performance objectives, while in competition with any other applications that may be sharing the same distributed system [4]. Application level scheduling is the opposite of system level scheduling, where a single scheduler controls all resources and applications, and makes all allocation decisions.

Effective application level scheduling involves the integration of application-specific and system-specific data, and it depends on the dynamic interactions between the application and the computing system. The scheduler embedded within each application must effectively combine the data it receives about the environment and the tasks' states in order to meet its performance requirements.

Berman et al. [4] identify four fundamental application level scheduling principles:

- **Effective scheduling requires both application and system specific data** – the quality of the schedules generated by the scheduler is highly correlated with the knowledge it has about the system's characteristics and the structure and requirements of the application;

- **Dynamic information is required to determine the system's state** – using updated system's state data the scheduler can detect load imbalances and act to more efficiently exploit the available resources. Static scheduling policies may be inadequate on shared distributed systems, namely when the system's state and global workload vary widely during execution time due to the other users' activities.

- **Effective scheduling requires both application and system performance prediction** – prediction provides the basis for effective scheduling. The scheduler makes its decisions based on predictions about the tasks's requirements and the resources' capacities on the near future. Accurate prediction can be difficult, since the system's state and the application requirements vary in forms that are beyond the scheduler's control. To make predictions the scheduler must have some model of the application and system's behaviour. This model needs to accurately represent the dynamic performance variation of the application on the underlying resources in a manner that allows the scheduler to adapt application execution to the current system's state. Simple models can make the prediction task easier but may fail to provide accurate predictions. A tradeoff must be found between the complexity of the model and the quality of the predictions it provides.

- **All resources can be evaluated strictly in terms of the performance they deliver to the application** – from the application's perspective each resource is evaluated in terms of how much it benefits the application's execution. If the computing system is shared between several applications, some resources, e.g. processors, are probably time-shared by these applications. Each application must evaluate each resource in terms of the percentage of that resource that will be allocated to it.

## 3 Static vs. Dynamic Policies

The degree of flexibility the scheduler must exhibit depends on how demanding are its performance requirements and on the complexity of the environment it is supposed to manage. Russel, Norvig and Weiss [11, 20] suggest that environments can be classified according to the following properties:

**Accessible vs. Inaccessible** – an environment is accessible if the agent's sensors give it complete, accurate and updated information about the environment's state;

**Deterministic vs. Non-Deterministic** – an environment is deterministic if its next state is completely determined by its actual state and the agent's selected action; on a deterministic environment each action has a single guaranteed effect, which is known by the agent;

**Static vs. Dynamic** – a static environment remains unchanged except by effect of actions triggered by the agent; if the environment changes in ways beyond the agent's control, due to other processes operating on it, then the environment is dynamic.

The environments where a scheduling agent is required to act are, usually, inaccessible, non-deterministic and dynamic.

**Dynamic:** on a shared distributed system many applications, possibly launched by several users, exhibiting varying and irregular workload and communication patterns, are allowed to co-exist and start and terminate at any instant. This is well beyond the scheduler's control as it depends on the work performed by the resources it is managing and on the users' activity.

**Inaccessible:** the environment may be so complex that the scheduler can not consider all the relevant aspects on its internal model of the world. Furthermore, the information available to the agent is inaccurate because it is expensive, or even impossible, to acquire with great precision. Also this information is often outdated because the system is continuously changing – information aging.

**Non-deterministic:** the environment changes while the agent is deliberating and its selected actions are being performed. The scheduler can not be completely sure of its actions' consequences. If an environment is inaccessible it may appear to be non-deterministic as the agent can not keep track of its inaccessible aspects. Non-determinism is a consequence of the environment's previous properties: dynamic and inaccessible.

A scheduler operating on a distributed shared system must deal with a large number of activities that it can neither anticipate nor control and that it is, probably, unable to characterize completely. It has been shown that adaptive policies provide good performance when the system state changes widely during execution time [6, 16, 23] and that different load management policies are best suited to handle different workload patterns [3, 22]. This suggests that in order to achieve its performance requirements and deal effectively with the environment's properties, the scheduling agent must use an adaptive decision making policy.

Becker and Waldmann [3] identify three opportunities for adaptive scheduling:

**Correction of profile and load predictions** - sophisticated approaches enable the exploitation of applications workload profile predictions. The challenge for the scheduler is to generate or improve predictions about near future workload profiles. By comparing the expected to the real system behaviour the exactness and importance of these predictions can be evaluated and corrected.

**Determination of diffuse factors in the execution model** - dynamic decision making and information management incur overheads at runtime; therefore they must be based on simple execution models. Important side effects are often neglected or the correlations between the considered items and the desired results are not strong enough. Using feedback the scheduler can increase its knowledge about the system's behaviour and correct its initial world's model.

**Control of the relationship between overhead and profit** - dynamic load management incurs parallelism, communication and computation overheads. Adaptive approaches must minimize the efforts for load management and automatically optimize the relationship between cost and benefit.

# 4 The scheduler's evaluation

In order to generate and apply the schedule, the scheduling agent itself consumes the resources it is managing. It consumes both CPU time and communication bandwidth to exchange control and information messages between the system's nodes, to compute the metrics required to describe the system's state, to deliberate on which is the best action to take and to execute these actions. The scheduler must carefully balance the costs, or overheads, it imposes on the system with the achieved benefits. Imposing low overheads might prevent the scheduler from meeting its performance objectives due to improper decision making, whereas imposing to much overheads will lead to the same undesired final result, since too much resources are dedicated to the managing function.

## 4.1 Performance and efficiency

To evaluate the scheduler it is not enough to evaluate the quality of its decisions. The resources it consumes must also be taken into account. The scheduler must be assessed by the relation between the performance obtained with its decisions and the efficiency with which that performance is achieved.

The performance is the scheduler's global objective degree of achievement, or effectiveness, and it is closely related to the quality of its decisions. Efficiency is a measure of the costs, in terms of consumed resources, and it is closely related to the level of intrusion the scheduler imposes on the system.

The desired system behaviour is to have the highest level of performance, while incurring the least overhead [7]. A scheduler with high performance and poor efficiency, i.e., high overheads, is preferable to one which is more efficient but achieves less performance. The scheduler's main objective is to fulfill its performance requirements.

Evaluation of performance and efficiency is difficult due to their inherent entanglement. To correctly balance these factors the scheduler must perform, either explicitly or implicitly, a profitability determination analysis [19, 21], which estimates the potential gains obtainable with a given action and weights it with its overheads.

## 4.2 Cost Analysis

The design of an efficient scheduler requires a careful analysis of the costs involved, the reasons why these costs exist and what are the potential benefits associated with them.

The costs induced by the scheduler depend on the environment where it operates and on its performance objectives. On a distributed computing system the scheduler costs include CPU time, memory space and communication bandwidth. With respect to the performance objective, the following analysis will deal essentially with minimization of application execution time.

Some costs result directly from the scheduler's activity. These are referred to as direct costs (see table 1), and they represent the resources the scheduler consumes. They depend on the scheduler's strategy complexity and on the frequency with which it triggers its various mechanisms. Control messages, for instance, have no obvious positive impact on performance and should, therefore, be avoided.

Load management policies are often described in terms of four sub-policies – information, transfer, selection and location policies – and the transfer mechanism used to move the tasks around the distributed system [16]. It is useful, for the scheduler's designer, to understand how direct costs are distributed over these sub-policies:

**Information policy** – to decide when, where and what information must be collected about the system's state. It is responsible for the metrics estimation and for all information messages. It consumes both CPU time and communication bandwidth.

**Transfer policy** – to determine whether a node is in a suitable state to participate on a task transfer. The scheduling agent must make this decision based on the information available about the environment. It consumes CPU time.

**Location policy** – to identify the various partners enrolled on a given action, e.g., workload transfer. This is also a decision, so it consumes CPU time.

**Selection policy** – to decide what work must be transferred between the partners identified by the location policy. It consumes CPU time.

| Scheduling activity | Comments |
|---|---|
| Metrics estimation to describe the system's state. | These metrics are the scheduler's environment and events inputs. The means used to compute them constitute the scheduling agent's sensory apparatus.<br><br>**Benefits** – the more accurate and diverse these metrics are, the more knowledge is available about the system's state, hence the more potential for good decision making.<br><br>**Costs** – this consumes both CPU time, required to compute the metrics, and communication bandwidth, since some metrics may be related to the network subsystem. |
| Information messages | In a distributed system the system's state metrics are computed across the whole system and must be communicated to the scheduler. The scheduler itself may be distributed and exchange state information among its components.<br><br>**Benefits** – closely related to the estimation of system's metrics.<br><br>**Costs** – communication bandwidth and CPU time. Many distributed systems – hardware and software – require messages to be packed/unpacked and eventually routed by the computing nodes, which may consume CPU time. |
| Control messages | Depending on the protocol used by the scheduler, it may need to exchange control messages between its various components and the processes it is managing.<br><br>**Costs** – communication bandwidth and CPU time. |
| Selection of the most adequate action | Based on what it knows about the environment's state and on its performance requirements, the scheduler must select the best action to take at each instant.<br><br>**Benefits** – the better the quality of the decision making process, the higher the degree of performance achievement.<br><br>**Costs** – CPU time. |
| Execution of the selected actions | After selecting which action to take, the scheduler must execute it. The set of available actions constitute the scheduler's effectual capabilities. This set depends on the scheduler's design, but it may include: do nothing, transfer work, change the scheduling policy, collect more information about the system's state.<br><br>**Costs** – most actions, aside 'do nothing', require the engagement of one or more nodes' CPUs. They may also consume communication bandwidth because the scheduler must communicate with the appropriate nodes and work might be transferred between them. |

Table 1: Direct costs induced by the scheduler

**Transfer mechanism** – the mechanism used to redistribute the workload across the system. This may be a costly operation, depending on how it is implemented and on the system's characteristics. It consumes both CPU time and communication bandwidth.

A further source of overheads is the scheduler's internal coordination. Since the scheduler may itself be distributed, it may need to transfer control messages between its various components. These must be reduced as far as possible, since they consume communication bandwidth and CPU time.

Besides direct costs, there are indirect costs that are consequences of the scheduler's selected actions. Indirect costs depend on the application and distributed system being managed. For instance, a task's transfer from one node to another, on a distributed memory parallel system, may require the additional transfer of data to complete the job. Or it may require the replication of work already performed by the source node. Indirect costs may prevent the scheduler from achieving its performance objectives, and should, therefore, be taken into account on the decision making process. Indirect costs are related to the quality of the decisions made by the scheduling agent. An high level of indirect costs, that exceeds the benefits attained with the selected actions, suggests that the scheduler is not making the correct decisions.

The scheduler's evaluation must be done in terms of the level of performance achievement, rather than using some second–order metrics. An example of such metric is the standard deviation of CPU time across all nodes, used to evaluate some load balancing schedulers. The performance objective is, usually, to reduce execution time, not to balance the system's load. A well balanced system, presenting low standard deviation, may have a longer execution time than an unbalanced one, due to the overheads, both direct and indirect, imposed by the scheduler. Although second-order metrics may be important indicators about the scheduler's behaviour, the designer can not forget which are its real performance objectives.

High overheads are often associated with instability. Instability occurs when the scheduler is excessively active, but failing to obtain any performance benefits. Instability is related to the amount of schedulable resources being consumed by the scheduler while the system's state is changing but not moving towards a more desirable state. Casavant [5] identifies four causes of instability and their effects on the system's behaviour:

**Intolerance** – reaction to small imbalances in load distribution may cause a state where tasks are continuously transferred between a given set of nodes – processor trashing;

**Overresponse** – attempts to react too quickly to local imbalance conditions may originate transfers of excessive workload between resources;

**High load** – as the overall load present in the system increases, so does the opportunity for load imbalances and to overreactions; in a system with a low load's level the opportunity for large load imbalances is not present;

**Invalid assumption** – this results when the scheduler's environment violates certain assumptions made by the designer; the rules embedded within the scheduling agent are not able to deal with the environment's behaviour and instability may result.

Stankovic [17] claims that load management algorithms must include specific mechanisms to avoid all forms of instability and handle efficiently the many cases where it can occur. It is generally accepted that scheduling activities must be restricted to a small percentage of the system's total activity, otherwise instability will arise [9, 23]. Kremien and Kramer [9] propose two measures to evaluate a scheduler's stability:

**% of remote execution** – related to the number of task transfers decided by the scheduler. This figure must be minimized.

**Hit–ratio** – the ratio of successful workload transfer decisions to the total number of decisions. This ratio must be maximized.

To measure the scheduler's error margin it is not enough to measure its unsuccessful workload migration decisions. All those situations where a migration would have been beneficial, in terms of the scheduler's performance objectives, but was not decided, must also be considered. The scheduler's error should be measured by:

**Miss–ratio** – % of situations where a migration was decided but should not have been;

**Missed–opportunity–ratio** – % of situations where a migration was not decided but should have been.

These ratios may be difficult to measure because they require the identification of all the wrong decisions made by the scheduler.

## 5 Incompleteness of Information and Uncertainty

To meet its performance requirements, an adaptive or dynamic scheduler must collect a reasonable amount of information about the system's state and workload profile. The more information, or knowledge, accumulated, the more accurate the decisions become. However, the environment where the scheduler has to operate is often inaccessible. Its sensors can not give complete, exact and up-to-date information about the system's state. The scheduling agent does not have access to the whole knowledge about its environment. In every situation there will be important questions to which no categorical answer can be found. The agent has to act under uncertainty.

The incompleteness of information has three main reasons [11]:

**Theoretical ignorance** – there is no complete theory for load management which describes all the relevant aspects of system's state and workload profile that should be accounted for.

**Laziness** – there are too much different factors influencing the system's behaviour, so it is neither feasible nor efficient to account for all of them. The scheduler has to make its decisions based on a simpler internal model of the world. Any computational representation of a real world problem must be a simplification of the objects and relations that hold on the universe being considered. Many of these objects and relations must be neglected or summarized so that a manageable execution model is obtained. This inescapable incompleteness in representation leads to unavoidable uncertainties about the state of the world and the consequences of actions (non–deterministic environment) [8].

**Practical ignorance** – the knowledge available to the decision maker is inaccurate, because it is too expensive, or even impossible, to collect exact information about all factors considered in the execution model. Also, the information gets obsolete, because the system's state is continuously changing (dynamic environment). This is usually referred to as information aging.

In every instant the agent is uncertain about the current system's state, the tasks' requirements and their current execution progress. To make its decisions the scheduler uses its internal world's model to predict application and system's behaviour on the near future. However, the system's state, for instance, may vary over time, specially in distributed shared systems. Some schedulers' performance models explicitly integrate uncertainty by quantifying real world entities not as deterministic single-valued variables, but as stochastic variables allowed to range over an interval of values with a given probability distribution [13, 14, 15, 12].

Often also the relationships between real world entities have to be modeled stochastically, either because they vary over time or because they are not exactly known. One such case is the uncertainty associated with the consequences of the scheduler's actions for each of the world' states.

In order to meet its performance requirements the scheduling agent must be able to deal effectively with this uncertainty about past, present and future system's states and workload profiles.

# 6 Summary

This report characterized the problem of managing a distributed system's load as a scheduling problem and stressed some specific characteristics of application level scheduling. Particularly, it is claimed that in order to be effective, an application level scheduler on a distributed shared system must follow a dynamic strategy, where input about the current system's state and tasks' execution progress is considered to make future allocation decisions. Additionally, a scheduling agent must be aware of its uncertainty about many aspects of the surrounding environment and, possibly, include this uncertainty explicitly on its execution model.

Furthermore, an analysis of the costs associated with scheduling was presented and some considerations about how to evaluate the scheduler were discussed. The scheduler's designer must be aware that the scheduling function also consumes the resources it is managing. Therefore, it is not enough to assess the performance achieved with a given strategy; it is also necessary to assess the efficiency with which that performance was obtained.

## References


[1] Ahmad, I., and Ghafoor, A. Semi-distributed load balancing for massively parallel multicomputer systems. *IEEE Transactions on Software Engineering 17*, 10 (Oct. 1991), 987–1004.

[2] Baumgartner, K., and Wah, B. Computer scheduling algorithms: Past, present and future. *Information Sciences 57, 58* (Dec. 1991), 319–345.

[3] Becker, W., and Waldmann, G. Adaption in dynamic load balancing: Potential and techniques. In *Fachtagung Arbeitsplatz-Rechensysteme* (Hanover, Germany, May 1995).

[4] Berman, F., Wolski, R., Figueira, S., Schopf, J., and Schao, G. Application level scheduling on distributed heterogeneous networks. In *SuperComputing'96* (Pittsburgh, USA, 1996).

[5] Casavant, T., and Kuhl, J. Effects of response and stability on scheduling in distributed computing systems. *IEEE Transactions on Software Engineering* (Nov. 1988), 1578–1588.

[6] Casavant, T., and Kuhl, J. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on Software Engineering* (Feb. 1988), 141–154.

[7] Diekmann, R., Monien, B., and Preis, R. Load balancing strategies for distributed memory machines. *World Scientific* (1997).

[8] Horvitz, E., Breese, J., and Henrion, M. Decision theory in expert systems and artificial intelligence. Tech. rep., July 1988.

[9] Kremien, O., and Kramer, J. Methodical analysis of adaptive load sharing algorithms. *IEEE Parallel and Distributed Systems* (Nov. 1992).

[10] Livney, M., and Melman, M. Load balancing in homogeneous broadcast distributed systems. In *Proceedings of the ACM Computer Network Performance Symposium* (1982), pp. 47–55.

[11] Russel, S., and Norvig, P. *Artificial Intelligence: A Modern Approach*, 1st ed. Prentice Hall, 1995.

[12] Schopf, J. A practical methodology for defining histograms for predictions and scheduling. In *Parallel Computing (ParCo'99)* (Aug. 1999).

[13] Schopf, J., and Berman, F. Performance prediction using intervals. Tech. Rep. CS97-541, University of California, San Diego, May 1997.

[14] Schopf, J., and Berman, F. Stochastic scheduling. In *SuperComputing'99* (Portland, OR, USA, 1999).

[15] Schopf, J., and Berman, F. Using stochastic intervals to predict apllication behavior on contended resources. In *Workshop on Advances in Parallel Computing (ISPAN'99)* (1999).

[16] Shivaratri, N., Krueger, P., and Singhal, M. Load distributing for locally distributed systems. *IEEE Computer* (Dec. 1992), 33–44.

[17] Stankovic, J. Stability and distributed scheduling algorithms. *IEEE Transactions on Software Engineering* (Oct. 1985), 1141–1152.

[18] Wang, Y., and Morris, R. Load sharing in distributed systems. *IEEE Transactions on Computers C-34*, 2 (Mar. 1985), 204–217.

[19] Watts, J., and Taylor, S. A practical approach to dynamic load balancing. *IEEE Transactions on Parallel and Distributed Systems 9*, 3 (Mar. 1998), 235–248.

[20] Weiss, G. *MultiAgent Systems: A Modern Approach to Distributed Artificial Intelligence*, 1st ed. The MIT Press, 1999.

[21] Willebeek-LeMair, M., and Reeves, A. Strategies for dynamic load balancing on highly paralel computers. *IEEE Transactions on Parallel and Distributed Systems 4* (Sept. 1993), 979–993.

[22] Zaki, M., Li, W., and Parthasarathy, S. Customized dynamic load balancing for a network of workstations. *Journal of Parallel and Distributed Computing 43* (1997), 156–162.

[23] Zhou, S. A trace driven simulation study of dynamic load balancing. *IEEE Transactions on Software Engineering* (Sept. 1988), 1327–1341.