A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems

Thomas L. Casavant

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

Jon G. Kuhl

Department of Electrical and Computer Engineering
University of Iowa
Iowa City, Iowa

ABSTRACT

One measure of usefulness of a general-purpose distributed computing system is the system's ability to provide a level of performance commensurate to the degree of multiplicity of resources present in the system. Many different approaches and metrics of performance have been proposed in an attempt to achieve this goal in existing systems. In addition, analogous problem formulations exist in other fields such as control theory, operations research and production management. However, due to the wide variety of approaches to this problem, it is difficult to meaningfully compare different systems since there is no uniform means for qualitatively or quantitatively evaluating them. It is difficult to successfully build upon existing work or identify areas worthy of additional effort without some understanding of the relationships between past efforts. In this paper, a taxonomy of approaches to the resource management problem is presented in an attempt to provide a common terminology and classification mechanism necessary in addressing this problem. The taxonomy, while presented and discussed in terms of *distributed scheduling*, is also applicable to most types of resource management. As an illustration of the usefulness of the taxonomy an annotated bibliography is given which classifies a large number of distributed scheduling approaches according to the taxonomy.

May 1, 1996

A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems

Thomas L. Casavant

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

Jon G. Kuhl

Department of Electrical and Computer Engineering
University of Iowa
Iowa City, Iowa

1. Introduction

The study of distributed computing has grown to include a large range of applications[16, 17, 31, 32, 37, 54, 55]. However, at the core of all the efforts to exploit the potential power of distributed computation are issues related to the management and allocation of system resources relative to the computational load of the system. This is particularly true of attempts to construct large **general-purpose** multiprocessors [3, 8, 25, 26, 44, 45, 46, 50, 61, 67].

The notion that a loosely-coupled collection of processors could function as a more powerful general-purpose computing facility has existed for quite some time. A large body of work has focused on the problem of managing the resources of a system in such a way as to effectively exploit this power. The result of this effort has been the proposal of a variety of widely differing techniques and methodologies for distributed resource management. Along with these competing proposals has come the inevitable proliferation of inconsistent and even contradictory terminology, as well as a number of slightly differing problem formulations, assumptions etc. Thus, it is difficult to analyze the relative merits of alternative schemes in a meaningful fashion. It is also difficult to focus common effort on approaches and areas of study which seem most likely to prove fruitful.

This paper attempts to tie the area of **distributed scheduling** together under a common, uniform set of terminology. In addition, a taxonomy is given which allows the classification of distributed scheduling algorithms according to a reasonably small set of salient features. This allows a convenient means of quickly describing the central aspects of a particular approach, as well as a basis for comparison of commonly classified schemes.

Earlier work has attempted to classify certain aspects of the scheduling problem. In [9], Casey gives the basis of a hierarchical categorization. The taxonomy presented here agrees with the nature of Casey's categorization. However, a large number of additional fundamental distinguishing features are included which differentiate between existing approaches. Hence, the taxonomy given here provides a more detailed and complete look at the basic issues addressed in that work. Such detail is deemed necessary to allow meaningful comparisons of different approaches. In contrast to the taxonomy of Casey, Wang[65] provides a taxonomy of load-sharing schemes. Wang's taxonomy succinctly describes the range of approaches to the load-sharing problem. The categorization presented describes solutions as being either *source initiative* or *server initiative*. In addition, solutions are characterized along a continuous range according to the degree of information dependency involved. The taxonomy presented here takes a much broader view of the distributed scheduling problem in which load-sharing is only one of several possible *basic* strategies available to a system designer. Thus the classifications discussed by Wang describe only a narrow category within the taxonomy.

Among existing taxonomies, one can find examples of flat and hierarchical classification schemes. The taxonomy proposed here is a hybrid of these two -- hierarchical as long as possible in order to reduce the total number of classes, and flat when the descriptors of the system may be chosen in an arbitrary order. The levels in the hierarchy have been chosen in order to keep the description of the taxonomy itself small, and do not necessarily reflect any ordering of importance among characteristics. In other words, the descriptors comprising the taxonomy do not attempt to hierarchically order the characteristics of scheduling systems from more to less general. This point should be stressed especially with respect to the positioning of the flat portion of the taxonomy near the bottom of the hierarchy. For example, load balancing is a characteristic which pervades a large number of distributed scheduling systems, yet for the sake of reducing the size of the description of the taxonomy, it has been placed in the flat portion of the taxonomy and, for the sake of brevity, the flat portion has been placed near the bottom of the hierarchy.

The remainder of the paper is organized as follows. In section 2, the scheduling problem is defined as it applies to distributed resource management. In addition, a taxonomy is presented which serves to allow *qualitative* description and comparison of distributed scheduling systems. Section 3 will present examples from the literature to demonstrate the use of the taxonomy in qualitatively describing and comparing existing systems. Section 4 presents a discussion of issues raised by the taxonomy and also suggests areas in need of additional work.

In addition to the work discussed in the text of the paper, an extensive annotated bibliography is given in an appendix. This appendix further demonstrates the effectiveness of the taxonomy in allowing standardized description of existing systems.

2. The Scheduling Problem and Describing its Solutions

The *general scheduling* problem has been described a number of times and in a number of different ways in the literature[12, 22, 63] and is usually a restatement of the classical notions of job sequencing[13] in the study of production management[7]. For the purposes of distributed process scheduling, we take a broader view of the scheduling function as a *resource management resource*. This management resource is basically a mechanism or policy used to efficiently and effectively manage the access to and use of a resource by its various consumers. Hence, we may view every instance of the scheduling problem as consisting of three main components.

1) Consumer(s).
2) Resource(s).
3) Policy.

Like other management or control problems, understanding the functioning of a scheduler may best be done by observing the effect it has on its environment. In this case, one can observe the behavior of the scheduler in terms of how the *policy* affects the *resources* and *consumers*. Note that although there is only one policy, the scheduler may be viewed in terms of how it affects either or both resources and consumers. This relationship between the scheduler, policies, consumers and resources is shown in figure 1.

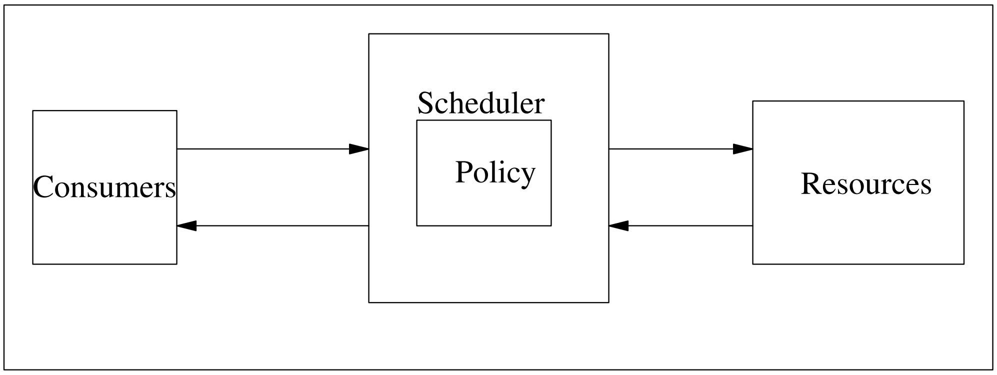

Figure 1. Scheduling System

In light of this description of the scheduling problem, there are two properties which must be considered in evaluating any scheduling system i) the satisfaction of the consumers with how well the scheduler manages the resource in question(performance), and ii) the satisfaction of the consumers in terms of how difficult or costly it is to access the management resource itself(efficiency). In other words, the consumers want to be able to quickly and efficiently access the actual resource in question, but do not desire to be hindered by overhead problems associated with using the management function itself.

One by-product of this statement of the general scheduling problem is the unification of two terms in common use in the literature. There is often an implicit distinction between the terms *scheduling* and *allocation*. However, it can be argued that these are merely alternative formulations of the same problem, with allocation posed in terms of *resource allocation*(from the resources' point of view), and *scheduling* viewed from the consumer's point of view. In this sense, allocation and scheduling are merely two terms describing the same general mechanism, but described from different viewpoints.

2.1. The Classification Scheme

The usefulness of the four-category taxonomy of computer architecture presented by Flynn[20] has been well demonstrated by the ability to compare systems through their relation to that taxonomy. The goal of the taxonomy given here is to provide a commonly accepted set of terms and to provide a mechanism to allow comparison of past work in the area of distributed scheduling in a qualitative way. In addition, it is hoped that the categories and their relationships to each other have been chosen carefully enough to indicate areas in need of future work as well as to help classify future work.

The taxonomy will be kept as small as possible by proceeding in a hierarchical fashion for as long as possible, but some choices of characteristics may be made independent of previous design choices, and thus will be specified as a set of descriptors from which a subset may be chosen. The taxonomy, while discussed and presented in terms of distributed process scheduling, is applicable to a larger set of resources. In fact, the taxonomy could usefully be employed to classify any set of resource management systems. However, we will focus our attention on the area of process management since it is in this area which we hope to derive relationships useful in determining potential areas for future work.

2.1.1. Hierarchical Classification

The structure of the hierarchical portion of the taxonomy is shown in figure 2. A discussion of the hierarchical portion then follows.

Local vs. Global

At the highest level, we may distinguish between *local* and *global* scheduling. Local scheduling is involved with the assignment of processes to the time-slices of a

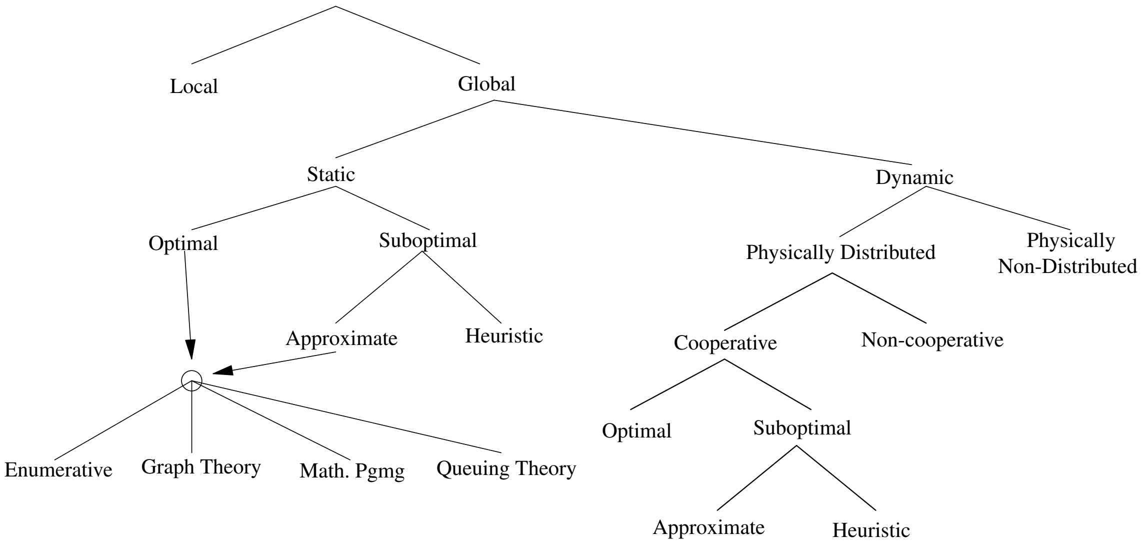

Figure 2. Task Scheduling Characteristics

single processor. Since the area of scheduling on single-processor systems [12, 62] as well as the area of sequencing or job-shop scheduling[13, 18] has been actively studied for a number of years, this taxonomy will focus on global scheduling. Global scheduling is the problem of deciding *where* to execute a process, and the job of local scheduling is left to the operating system of the processor to which the process is ultimately allocated. This allows the processors in a multiprocessor increased autonomy while reducing the responsibility (and consequently overhead) of the global scheduling mechanism. Note that this does not imply that global scheduling must be done by a single central authority, but rather, we view the problems of local and global scheduling as separate issues, and (at least logically) separate mechanisms are at work solving each.

Static vs. Dynamic

The next level in the hierarchy (beneath global scheduling) is a choice between *static* and *dynamic* scheduling. This choice indicates the time at which the scheduling or assignment decisions are made.

In the case of static scheduling, information regarding the total mix of processes in the system as well as all the independent subtasks involved in a job or task force[26, 44] is assumed to be available by the time the program object modules are linked into load modules. Hence, each executable image in a system has a static assignment to a particular processor, and each time that process image is submitted for execution, it is assigned to that processor. A more relaxed definition of static scheduling may include algorithms that schedule task forces for a particular hardware configuration. Over a period of time, the topology of the system may change, but characteristics describing the task force remain the same. Hence, the scheduler may generate a new assignment of processes to processors to serve as the schedule until the topology changes again.

Note here that the term *static scheduling* as used in this paper has the same meaning as *deterministic scheduling* in [22] and *task scheduling* in [56]. These alternative terms will not be used, however, in an attempt to develop a consistent set of terms and taxonomy.

Optimal vs. Sub-Optimal

In the case that all information regarding the state of the system as well as the resource needs of a process are known, an *optimal* assignment can be made based on some criterion function [5, 14, 21, 35, 40, 48]. Examples of optimization measures are minimizing total process completion time, maximizing utilization of resources in the system, or maximizing system throughput. In the event that these problems are computationally infeasible, *sub-optimal* solutions may be tried[2, 34, 47]. Within the realm of sub-optimal solutions to the scheduling problem, we may think of two general categories.

Approximate vs. Heuristic

The first is to use the same formal computational model for the algorithm, but instead of searching the entire solution space for an optimal solution, we are satisfied when we find a "good" one. We will categorize these solutions as *sub-optimal-approximate*. The assumption that a *good* solution can be recognized may not be so insignificant, but in the cases where a metric is available for evaluating a solution, this technique can be used to decrease the time taken to find an acceptable solution (schedule). The factors which determine whether this approach is worthy of pursuit include:

1) Availability of a function to evaluate a solution.
2) The time required to evaluate a solution.
3) The ability to judge according to some metric the value of an optimal solution.
4) Availability of a mechanism for intelligently pruning the solution space.

The second branch beneath the sub-optimal category is labeled *heuristic*[15, 30, 66]. This branch represents the category of static algorithms which make the most realistic assumptions about *a priori* knowledge concerning process and system loading characteristics. It also represents the solutions to the static scheduling problem which require the most reasonable amount of time and other system resources to perform their function. The most distinguishing feature of heuristic schedulers is that they make use of special parameters which effect the system in indirect ways. Often, the parameter being monitored is correlated to system performance in an indirect instead of a direct way, and this alternate parameter is much simpler to monitor or calculate. For example, clustering groups of processes which communicate heavily on the same processor and physically separating processes which would benefit from parallelism[52] directly decreases the overhead involved in passing information between processors, while reducing the interference among processes which may run without synchronization with one another. This result has an impact on the overall service that users receive, but cannot be *directly* related (in a quantitative way) to system performance as the user sees it. Hence, our intuition, if nothing else, leads us to believe that taking the aforementioned actions when possible will improve system performance. However, we may not be able to *prove* that a first-order relationship between the mechanism employed and the desired result exists.

Optimal and Sub-Optimal Approximate Techniques

Regardless of whether a static solution is optimal or sub-optimal-approximate, there are four basic categories of task allocation algorithms which can be used to arrive at an assignment of processes to processors.

1) Solution Space Enumeration and Search[48].

2) Graph Theoretic[4, 57, 58].
3) Mathematical Programming[5, 14, 21, 35, 40].
4) Queuing Theoretic[10, 28, 29].

Dynamic Solutions

In the dynamic scheduling problem, the more realistic assumption is made that very little *a priori* knowledge is available about the resource needs of a process. It is also unknown in what environment the process will execute during its lifetime. In the static case, a decision is made for a process image before it is ever executed, while in the dynamic case no decision is made until a process begins its life in the dynamic environment of the system. Since it is the responsibility of the running system to decide where a process is to execute, it is only natural to next ask where the decision itself is to be made.

Distributed vs. Non-Distributed

The next issue (beneath dynamic solutions) involves whether the responsibility for the task of global dynamic scheduling should physically reside in a single processor[44] (*physically non-distributed*) or whether the work involved in making decisions should be *physically distributed* among the processors[17]. Here the concerne is with the logical *authority* of the decision-making process.

Cooperative vs. Non-Cooperative

Within the realm of distributed dynamic global scheduling, we may also distinguish between those mechanisms which involve cooperation between the distributed components(*cooperative*) and those in which the individual processors make decisions independent of the actions of the other processors(*non-cooperative*). The question here is one of the degree of *autonomy* which each processor has in determining how its own resources should be used. In the non-cooperative case individual processors act alone as autonomous entities and arrive at decisions regarding the use of their resources independent of the effect of their decision on the rest of the system. In the cooperative case each processor has the responsibility to carry out its own portion of the scheduling task, but all processors are working toward a common system-wide goal. In other words, each processor's local operating system is concerned with making decisions in concert with the other processors in the system in order to achieve some global goal, instead of making decisions based on the way in which the decision will affect local performance only. As in the static case, the taxonomy tree has reached a point where we may consider optimal, sub-optimal-approximate, and sub-optimal-heuristic solutions. The same discussion as was presented for the static case applies here as well.

In addition to the hierarchical portion of the taxonomy already discussed, there are a number of other distinguishing characteristics which scheduling systems may have. The following sections will deal with characteristics which do not fit uniquely under any particular branch of the tree-structured taxonomy given thus far, but are still important in the way that they describe the behavior of a scheduler. In other words, the following could be branches beneath several of the leaves shown in figure 2 and in the interest of clarity are not repeated under each leaf, but are presented here as a flat extension to the scheme presented thus far. It should also be noted that these attributes represent a *set* of characteristics, and any particular scheduling subsystem may possess some subset of this set. Finally, the placement of these characteristics near the bottom of the tree is not intended to be an indication of their relative importance or any other relation to other categories of the hierarchical portion. Their position was determined primarily to reduce the size of the description of the taxonomy.

2.1.2. Flat Classification Characteristics

2.1.2.1. Adaptive vs. Non-Adaptive

An adaptive solution to the scheduling problem is one in which the algorithms and parameters used to implement the scheduling policy change dynamically according to the previous and current behavior of the system in response to previous decisions made by the scheduling system. An example of such an adaptive scheduler would be one which takes many parameters into consideration in making its decisions[52]. In response to the behavior of the system, the scheduler may start to ignore one parameter or reduce the importance of that parameter if it believes that parameter is either providing information which is inconsistent with the rest of the inputs or is not providing any information regarding the change in system state in relation to the values of the other parameters being observed. A second example of adaptive scheduling would be one which is based on the stochastic learning automata model[39]. An analogy may be drawn here between the notion of an adaptive scheduler and adaptive control[38], although the usefulness of such an analogy for purposes of performance analysis and implementation are questionable[51]. In contrast to an adaptive scheduler, a non-adaptive scheduler would be one which does not necessarily modify its basic control mechanism on the basis of the history of system activity. An example would be a scheduler which always weighs its inputs in the same way regardless of the history of the system's behavior.

2.1.2.2. Load Balancing

This category of policies, which has received a great deal of attention recently[10, 11, 36, 40, 41, 42, 46, 53], approaches the problem with the philosophy that being fair to the hardware resources of the system is good for the users of that system. The basic idea is to attempt to balance(in some sense) the load on all processors in such a way as to allow progress by all processes on all nodes to proceed at approximately the same rate. This solution is most effective when the nodes of a system are homogeneous since this allows all nodes to know a great deal about the structure of the other nodes. Normally, information would be passed about the network periodically or on demand[1, 60] in order to allow all nodes to obtain a local estimate concerning the global state of the system. Then the nodes act together in order to remove work from heavily loaded nodes and place it at lightly loaded nodes. This is a class of solutions which relies heavily on the assumption that the information at each node is quite accurate in order to prevent processes from endlessly being circulated about the system without making much progress. Another concern here is deciding on the basic unit used to measure the load on individual nodes.

As was pointed out in section 1, the placement of this characteristic near the bottom of the hierarchy in the flat portion of the taxonomy is not related to its relative importance or generality compared with characteristics at higher levels. In fact, it might be observed that at the point that a choice is made between optimal and sub-optimal characteristics, that a specific objective or cost function must have already been made. However, the purpose of the hierarchy is not so much to describe relationships between classes of the taxonomy, but to reduce the size of the overall description of the taxonomy so as to make it more useful in comparing different approaches to solving the scheduling problem.

2.1.2.3. Bidding

In this class of policy mechanisms, a basic protocol framework exists which describes the way in which processes are assigned to processors. The resulting scheduler is one which is usually cooperative in the sense that enough information is exchanged (between nodes with tasks to execute and nodes which may be able to execute tasks) so that an assignment of tasks to processors can be made which is beneficial to all nodes in the system as a whole.

To illustrate the basic mechanism of bidding, the framework and terminology of [49] will be used. Each node in the network is responsible for two roles with respect to the bidding process: *manager* and *contractor*. The manager represents the task in need of a location to execute, and the contractor represents a node which is able to do work for other nodes. Note that a single node takes on both of these roles, and that there are no nodes which are strictly managers or contractors alone. The manager announces the existence of a task in need of execution by a *task announcement*, then receives *bids* from the other nodes (contractors). A wide variety of possibilities exist concerning the type and amount of information exchanged in order to make decisions[53, 59]. The amount and type of information exchanged are the major factors in determining the effectiveness and performance of a scheduler employing the notion of bidding. A very important feature of this class of schedulers is that all nodes generally have full autonomy in the sense that the manager ultimately has the power to decide where to send a task from among those nodes which respond with bids. In addition, the contractors are also autonomous since they are never forced to accept work if they do not choose to do so.

2.1.2.4. Probabilistic

This classification has existed in scheduling systems for some time[13]. The basic idea for this scheme is motivated by the fact that in many assignment problems the number of permutations of the available work and the number of mappings to processors so large, that in order to analytically examine the entire solution space would require a prohibitive amount of time.

Instead, the idea of randomly (according to some known distribution) choosing some process as the next to assign is used. Repeatedly using this method, a number of different schedules may be generated, and then this set is analyzed to choose the best from among those randomly generated. The fact that an important attribute is used to bias the random choosing process would lead one to expect that the schedule would be better than one chosen entirely at random. The argument that this method actually produces a good selection is based on the expectation that enough variation is introduced by the random choosing to allow a *good* solution to get into the randomly chosen set.

An alternative view of probabilistic schedulers are those which employ the principles of decision theory in the form of team theory[24]. These would be classified as probabilistic since sub-optimal decisions are influenced by prior probabilities derived from *best-guesses* to the actual states of nature. In addition, these prior probabilities are used to determine (utilizing some random experiment) the next action (or scheduling decision).

2.1.2.5. One-time Assignment vs. Dynamic Reassignment

In this classification, we consider the entities to be scheduled. If the entities are *jobs* in the traditional batch processing sense of the term[19, 23], then we consider the single point in time in which a decision is made as to where and when the job is to execute. While this technique technically corresponds to a dynamic approach, it is static in the sense that once a decision is made to place and execute a job, no further decisions are made concerning the job. We would characterize this class as one-time assignments. Notice that in this mechanism, the only information usable by the scheduler to make its decision is the information given it by the user or submitter of the job. This information might include estimated execution time or other system resource demands. One critical point here is the fact that once users of a system understand the underlying scheduling mechanism, they may present false information to the system in order to receive better response. This point fringes on the area of psychological behavior, but human interaction is an important design factor to consider in this case since the behavior of the scheduler itself is trying to mimic a general philosophy. Hence, the interaction of this philosophy with the system's users must be considered.

In contrast, solutions in the dynamic reassignment class try to improve on earlier decisions by using information on smaller computation units - the executing subtasks of jobs or task forces. This category represents the set of systems which 1) do not trust their users to provide accurate descriptive information, and 2) use dynamically created information to adapt to changing demands of user processes. This adaptation takes the form of migrating processes (including current process state information). There is clearly a price to be paid in terms of overhead, and this price must be carefully weighed against possible benefits.

An interesting analogy exists between the differentiation made here and the question of preemption vs. non-preemption in uniprocessor scheduling systems. Here, the difference lies in whether to move a process from one place to another once an assignment has been made, while in the uniprocessor case the question is whether to remove the running process from the processor once a decision has been made to let it run.

3. Examples

In this section, examples will be taken from the published literature to demonstrate their relationships to one another with respect to the taxonomy detailed in section 2. The purpose of this section is twofold. The first is to show that many different scheduling algorithms can fit into the taxonomy and the second is to show that the categories of the taxonomy actually correspond, in most cases, to methods which have been examined.

3.1. Global Static

In [48], we see an example of an optimal, enumerative approach to the task assignment problem. The criterion function is defined in terms of optimizing the amount of time a task will require for all interprocess communication and execution, where the tasks submitted by users are assumed to be broken into suitable modules before execution. The cost function is called a *minimax criterion* since it is intended to minimize the maximum execution and communication time required by any single processor involved in the assignment. Graphs are then used to represent the module to processor assignments and the assignments are then transformed to a type of graph matching known as weak homomorphisms. The optimal search of this solution space can then be done using the $A*$ algorithm from artificial intelligence[43]. The solution also achieves a certain degree of processor load balancing as well.

[4] gives a good demonstration of the usefulness of the taxonomy in that the paper describes the algorithm given as a solution to the optimal dynamic assignment problem for a two processor system. However, in attempting to make an objective comparison of this paper with other *dynamic* systems, we see that the algorithm proposed is actually a static one. In terms of the taxonomy of section 2, we would categorize this as a static, optimal, graph theoretical approach in which the *a priori* assumptions are expanded to

include more information about the set of tasks to be executed. The way in which reassignment of tasks is performed during process execution is decided upon before any of the program modules begin execution. Instead of making reassignment *decisions* during execution, the stronger assumption is simply made that all information about the dynamic needs of a collection of program modules is available *a priori*. This assumption says that if a collection of modules possess a certain communication pattern at the beginning of their execution, and this pattern is completely predictable, that this pattern may change over the course of execution and that these variations are predictable as well. Costs of relocation are also assumed to be available, and this assumption appears to be quite reasonable.

The model presented in [35] represents an example of an optimum mathematical programming formulation employing a branch and bound technique to search the solution space. The goals of the solution are to minimize inter-processor communications, balance the utilization of all processors, and satisfy all other engineering application requirements. The model given defines a cost function which includes inter-processor communication costs and processor execution costs. The assignment is then represented by a set of zero-one variables, and the total execution cost is then represented by a summation of all costs incurred in the assignment. In addition to the above, the problem is subject to constraints which allow the solution to satisfy the load balancing and engineering application requirements. The algorithm then used to search the solution space (consisting of all potential assignments) is derived from the basic branch and bound technique.

Again, in [10], we see an example of the use of the taxonomy in comparing the proposed system to other approaches. The title of the paper - "Load Balancing in Distributed Systems" - indicates that the goal of the solution is to balance the load among the processors in the system in some way. However, the solution actually fits into the static, optimal, queuing theoretical class. The goal of the solution is to minimize the execution time of the entire program to maximize performance and the algorithm is derived from results in Markov Decision Theory. In contrast to the definition of load balancing given in section 2, where the goal was to even the load and utilization of system resources, the approach in this paper is consumer oriented.

An interesting approximate mathematical programming solution, motivated from the viewpoint of fault-tolerance, is presented in [2]. The algorithm is suggested by the computational complexity of the optimal solution to the same problem. In the basic solution to a mathematical programming problem, the state space is either implicitly or explicitly enumerated and searched. One approximation method mentioned in this paper[64] involves first removing the integer constraint, solving the continuous optimization problem, discretizing the continuous solution, and obtaining a bound on the discretization error. Whereas this bound is with respect to the continuous optimum, the algorithm proposed in this paper directly uses an approximation to solve the discrete problem and bound its performance with respect to the discrete optimum.

The last static example to be given here appears in [66]. This paper gives a heuristic-based approach to the problem by using extractable data and synchronization requirements of the different subtasks. The three primary heuristics used are:

1) Loss of Parallelism
2) Synchronization
3) Data Sources

The way in which loss of parallelism is used is to assign tasks to nodes one at a time in order to affect the least loss of parallelism based on the number of units required for execution by the task currently under consideration. The synchronization constraints are phrased in terms of *firing conditions* which are used to describe precedence relationships between subtasks. Finally, data source information is used in much the same way a functional program uses precedence relations between parallel portions of a computation which take the roles of varying classes of suppliers of variables to other subtasks. The final heuristic algorithm involves weighting each of the previous heuristics, and combining them. A distinguishing feature of the algorithm is its use of a greedy approach to finding a solution, when at the time decisions are made, there can be no guarantee that a decision is optimal. Hence, an optimal solution would more carefully search the solution space using a back track or branch and bound method, as well as using exact optimization criterion instead of the heuristics suggested.

3.2. Global Dynamic

Among the dynamic solutions presented in the literature, the majority fit into the general category of physically distributed, cooperative, sub-optimal, heuristic. There are, however, examples for some of the other classes.

First, in the category of physically non-distributed, one of the best examples is the experimental system developed for the Cm* architecture - Medusa[44]. In this system, the functions of the operating system (e.g. - file system, scheduler) are physically partitioned and placed at different places in the system. Hence, the scheduling function is placed at a particular place and is accessed by all users at that location.

Another rare example exists in the physically distributed *non-cooperative class*. In this example[27], random level-order scheduling is employed at all nodes independently in a tightly-coupled MIMD machine. Hence, the overhead involved in this algorithm is minimized since no information need be exchanged to make random decisions. The mechanism suggested is thought to work best in moderate to heavily loaded systems since in these cases, a random policy is thought to give a reasonably balanced load on all processors. In contrast to a cooperative solution, this algorithm does not detect or try to avoid system overloading by sharing loading information among processors, but makes the assumption that it will be under heavy load most of the time and bases all of its decisions on that assumption. Clearly, here, the processors are not necessarily concerned with the utilization of their own resources, but neither are they concerned with the effect their individual decisions will have on the other processors in the system.

It should be pointed out that although the above two algorithms (and many others) are given in terms relating to general-purpose distributed processing systems, that they do not strictly adhere to the definition of distributed data processing system as given in [17].

In [57], another rare example exists in the form of a physically distributed, cooperative, *optimal* solution in a dynamic environment. The solution is given for the two-processor case in which critical load factors are calculated prior to program execution. The method employed is to use a graph theoretical approach to solving for load factors for each process on each processor. These load factors are then used at run time to determine when a task could run better if placed on the other processor.

The final class (and largest in terms of amount of existing work) is the class of physically distributed, cooperative, sub-optimal, heuristic solutions.

In [53] a solution is given which is adaptive, load balancing, and makes one-time assignments of *jobs* to processors. No *a priori* assumptions are made about the characteristics of the jobs to be scheduled. One major restriction of these algorithms is the fact that they only consider assignment of jobs to processors and once a job becomes an active process, no reassignment of processes is considered regardless of the possible benefit. This is very defensible, though, if the overhead involved in moving a process is very high (which may be the case in many circumstances). Whereas this solution cannot exactly be considered as a bidding approach, exchange of information occurs between processes in order for the algorithms to function. The first algorithm (a copy of which resides at each host) compares its own *busyness* with its estimate of the busyness of the least busy host. If the difference exceeds the bias (or threshold) designated at the current time, one job is moved from the job queue of the busier host to the less busy one. The second algorithm, allows each host to compare itself with all other hosts and involves two biases. If the difference exceeds bias1 but not bias2, then one job is moved. If the difference exceeds bias2, then two jobs are moved. There is also an upper limit set on the number of jobs which can move at once in the entire system. The third algorithm is the same as algorithm one except that an anti-thrashing mechanism is added to account for the fact that a delay is present between the time a decision is made to move a job, and the time it arrives at the destination. All three algorithms had an adaptive feature added which would turn off all parts of the respective algorithm except the monitoring of load when system load was below a particular minimum threshold. This had the effect of stopping *processor thrashing* whenever it was practically impossible to balance the system load due to lack of work to balance. In the high load case, the algorithm was turned off to reduce extraneous overhead when the algorithm couldn't affect any improvement in the system under any redistribution of jobs. This last feature also supports the notion in the non-cooperative example given earlier that the load is usually automatically balanced as a side effect of heavy loading. The remainder of the paper focuses on simulation results to reveal the impact of modifying the biasing parameters.

The work reported in [6] is an example of an algorithm which employs the heuristic of load-balancing, and probabilistically estimates the remaining processing times of processes in the system. The remaining processing time for a process was estimated by one of the following methods:

memoryless: $Re(t) = E\{S\}$
pastrepeats: $Re(t) = t$
distribution: $Re(t) = E\{S - t \mid S > t\}$
optimal: $Re(t) = R(t)$

where R(t) is the remaining time needed given that t seconds have already elapsed, S is the service time random variable, and Re(t) is the scheduler's estimate of R(t). The algorithm then basically uses the first three methods to predict response times in order to obtain an expected delay measure which in turn is used by pairs of processors to balance their load on a pairwise basis. This mechanism is adopted by all pairs on a dynamic basis to balance the system load.

Another adaptive algorithm is discussed in [52] and is based on the bidding concept. The heuristic mentioned here utilizes prior information concerning the known characteristics of processes such as resource requirements, process priority, special resource needs, precedence constraints, and the need for clustering and distributed groups. The basic algorithm periodically evaluates each process at a current node to decide whether to transmit bid requests for a particular process. The bid requests include information needed for contractor nodes to make decisions regarding how well they may be able to execute the process in question. The manager receives bids and compares them to the local evaluation and will transfer the process if the difference between the best bid and the local estimate is above a certain threshold. The key to the algorithm is the formulation of a function to be used in a modified McCulloch-Pitts neuron. The neuron (implemented as a subroutine) evaluates the current performance of individual processes. Several different functions were proposed, simulated and discussed in this paper. The adaptive nature of this algorithm is in the fact that it dynamically modifies the number of hops that a bid request is allowed to travel depending on current conditions. The most significant result was that the information regarding process clustering and distributed groups seemed to have had little impact on the overall performance of the system.

The final example to be discussed here[55] is based on a heuristic derived from the area of Bayesian decision theory[33]. The algorithm uses no *a priori* knowledge regarding task characteristics, and is dynamic in the sense that the probability distributions which allow maximizing decisions to be made based on the most likely current state of nature are updated dynamically. Monitor nodes make observations every p seconds and update probabilities. Every d seconds the scheduler itself is invoked to approximate the current state of nature and make the appropriate maximizing action. It was found that the parameters p and d could be tuned to obtain maximum performance for a minimum cost.

4. Discussion

In this section, we will attempt to demonstrate the application of the qualitative description tool presented earlier to a role beyond that of classifying existing systems. In particular, we will utilize two behavior characteristics -- *performance* and *efficiency*, in conjunction with the classification mechanism presented in the taxonomy, to identify general qualities of scheduling systems which will lend themselves to managing large numbers of processors. In addition, the uniform terminology presented will be employed to to show that some earlier-thought-to-be-synonymous notions are actually distinct, and to show that the distinctions are valuable. Also, in at least one case, two earlier-thought-to-be-different notions will be shown to be much the same.

4.1. Decentralized vs. Distributed Scheduling

When considering the decision-making policy of a scheduling system, there are two fundamental components -- responsibility and authority. When responsibility for making and carrying out policy decisions is shared among the entities in a distributed system, we say that the scheduler is *distributed*. When authority is distributed to the entities of a resource management system, we call this *decentralized*. This differentiation exists in many other organizational structures. Any system which possesses decentralized authority must have distributed responsibility, but it is possible to allocate responsibility for gathering information and carrying out policy decisions, without giving the authority to change past or make future decisions.

4.2. Dynamic vs. Adaptive Scheduling

The terms *dynamic scheduling* and *adaptive scheduling* are quite often attached to various proposed algorithms in the literature, but there appears to be some confusion as to the actual difference between these two concepts. The more common property to find in a scheduler (or resource management subsystem) is the dynamic property. In a dynamic situation, the scheduler takes into account the current state of affairs as it perceives them in the system. This is done during the normal operation of the system under a dynamic and unpredictable load. In an adaptive system, the scheduling policy itself reflects changes in its environment -- the running system. Notice that the difference here is one of level in the hierarchical solution to the scheduling problem. Whereas a dynamic solution takes environmental inputs into account when making its decisions, an adaptive solution takes environmental stimuli into account to modify the scheduling policy itself.

4.3. The Resource/Consumer Dichotomy in Performance Analysis

As is the case in describing the actions or qualitative behavior of a resource management subsystem, the performance of the scheduling mechanisms employed may be viewed from either the resource or consumer point of view. When considering performance from the consumer (or user) point of view, the metric involved is often one of minimizing individual program completion times -- *response*. Alternately, the resource point of view also considers the rate of process execution in evaluating performance, but from the view of total system throughput. In contrast to response, *throughput* is concerned with seeing that *all* users are treated fairly and that *all* are making progress. Notice that the resource view of maximizing resource utilization is compatible with the desire for maximum system throughput. Another way of stating this, however, is that all users, when considered as a single collective user, are treated best in this environment of maximizing system throughput or maximizing resource utilization. This is the basic philosophy of load-balancing mechanisms. There is an inherent conflict, though, in trying to optimize both response and throughput.

4.4. Focusing on Future Directions

In this section, the earlier presented taxonomy, in conjunction with two terms used to quantitatively describe system behavior, will be used to discuss possibilities for distributed scheduling in the environment of a large system of loosely-coupled processors.

In previous work related to the scheduling problem, the basic notion of performance has been concerned with evaluating the way in which users' individual needs are being satisfied. The metrics most commonly applied are *response* and *throughput*[23]. While these terms accurately characterize the goals of the system in terms of how well users are served, they are difficult to measure during the normal operation of a system. In addition to this problem, the metrics do not lend themselves well to direct interpretation as to the action to be performed to increase performance when it is not at an acceptable level.

These metrics are also difficult to apply when analysis or simulation of such systems is attempted. The reason for this is that two important aspects of scheduling are necessarily intertwined. These two aspects are *performance* and *efficiency*. Performance is the part of a system's behavior that encompasses how well the resource to be managed is being used to the benefit of all users of the system. Efficiency, though, is concerned with the added cost (or overhead) associated with the resource management facility itself. In terms of these two criteria, we may think of desirable system behavior as that which has the highest level of performance possible, while incurring the least overhead in doing it. Clearly, the exact combination of these two which brings about the most desirable behavior is dependent on many factors and in many ways resembles the space/time trade-off present in common algorithm design. The point to be made here is that simultaneous evaluation of efficiency and performance is very difficult due to this inherent entanglement. What we suggest is a methodology for designing scheduling systems in which efficiency and performance are separately observable.

Current and future investigations will involve studies to better understand the relationships between performance, efficiency, and their components as they effect quantitative behavior. It is hoped that a much better understanding can be gained regarding the costs and benefits of alternative distributed scheduling strategies.

5. Conclusion

This paper has sought to bring together the ideas and work in the area of resource management generated in the last 10 to 15 years. The intention has been to provide a suitable framework for comparing past work in the area of resource management, while providing a tool for classifying and discussing future work. This has been done through the presentation of common terminology and a taxonomy on the mechanisms employed in computer system resource management. While the taxonomy could be used to discuss many different types of resource management, the attention of the paper and included examples have been on the application of the taxonomy to the processing resource.

Finally, recommendations regarding possible fruitful areas for future research in the area of scheduling in large scale general-purpose distributed computer systems have been discussed.

As is the case in any survey, there are many pieces of work to be considered. It is hoped that the examples presented fairly represent the true state of research in this area, while it is acknowledged that not *all* such examples have been discussed. In addition to the references at the end of this paper, the appendix contains an annotated bibliography for work not explicitly mentioned in the text but which have aided in the construction of this taxonomy through the support of additional examples. The exclusion of any particular results has not been intentional nor should it be construed as a judgment of the merit of that work. Decisions as to which papers to use as examples were made purely on the basis of their applicability to the context of the discussion in which they appear.

REFERENCES

[1] A.K. Agrawala, S.K. Tripathi, G. Ricart, *Adaptive Routing Using a Virtual Waiting Time Technique,* IEEE Trans. on Software Eng., Vol. SE-8, No. 1, Jan. 1982, pp. 76-81.

[2] J.A. Bannister, K.S. Trivedi, *Task Allocation in Fault-Tolerant Distributed Systems,* Acta Informatica, Vol. 20, 1983, pp. 261-281, Springer-Verlag.

[3] J.F. Bartlett, *A NonStop Kernel,* Proc. 8th Symp. on O.S. Principles, Dec. 1981, pp. 22-29.

[4] S.H. Bokhari, *Dual Processor Scheduling with Dynamic Reassignment,* IEEE Trans. on Software Eng., Vol. SE-5, No. 4, Jul. 1979, pp. 326-334.

[5] S.H. Bokhari, *A Shortest Tree Algorithm for Optimal Assignments Across Space and Time in a Distributed Processor System,* IEEE Trans. on Software Eng., Vol. SE-7, No. 6, Nov. 1981, pp. 335-341.

[6] R.M. Bryant, R.A. Finkel, *A Stable Distributed Scheduling Algorithm,* Proc. 2nd Intl. Conf. on Dist. Comp., Apr. 1981, pp. 314-323.

[7] E.S. Buffa, *Modern Production Management 5ed,* 1977, John Wiley & Sons, New York.

[8] T.L. Casavant, J.G. Kuhl, *Design of a Loosely-Coupled Distributed Multiprocessing Network,* 1984 Intl. Conf on Parallel Proc., Aug. 1984, pp. 42-45.

[9] L.M. Casey, *Decentralized Scheduling,* Australian Computing Journal, Vol. 13, May 1981, pp. 58-63.

[10] T.C.K. Chou, J.A. Abraham, *Load Balancing in Distributed Systems,* IEEE Trans. on Software Eng., Vol. SE-8, No. 4, Jul. 1982, pp. 401-412.

[11] Y.C. Chow, W.H. Kohler, *Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System,* IEEE Trans. on Comp., Vol. C-28, No. 5, May 1979, pp. 354-361.

[12] E.G. Coffman, P.J. Denning, *Operating Systems Theory,* 1973, Prentice-Hall, Englewood Cliffs, N.J.

[13] R.W. Conway, W.L. Maxwell, L.W. Miller, *Theory of Scheduling,* 1967, Addison-Wesley, Reading, Mass.

[14] K.W. Doty, P.L. McEntire, J.G. O'Reilly, *Task Allocation in a Distributed Computer System,* IEEE InfoCom, 1982, pp. 33-38.

[15] K. Efe, *Heuristic Models of Task Assignment Scheduling in Distributed Systems* Computer, Vol. 15, Jun. 1982, pp. 50-56.

[16] C.S. Ellis, J.A. Feldman, J.E. Heliotis, *Language Constructs and Support Systems for Distributed Computing,* ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Aug. 1982, pp. 1-9.

[17] P.H. Enslow Jr., *What is a "Distributed" Data Processing System,* Computer, Vol. 11, No. 1, Jan. 1978, pp. 13-21.

[18] J.R. Evans et al, *Applied Production and Operations Management,* 1984, West Publishing Co., St. Paul, Minn.

[19] I. Flores, *OSMVT,* 1973, Allyn and Bacon, Boston MA.

[20] M.J. Flynn, *Very High-Speed Computing Systems,* Proceedings of IEEE, Vol. 54, Dec. 1966, pp. 1901-1909.

[21] A. Gabrielian, D.B. Tyler, *Optimal Object Allocation in Distributed Computer Systems,* Proc. 4th Intl. Conf. on Dist. Comp. Systems, May 1984, pp. 84-95.

[22] M.J. Gonzalez, *Deterministic Processor Scheduling,* ACM Computing Surveys, Vol. 9, No. 3, Sep. 1977, pp. 173-204.

[23] H. Hellerman, T.F. Conroy, *Computer System Performance,* 1975, McGraw-Hill, New York.

[24] Y. Ho, *Team Decision Theory and Information Structures,* Proceedings of the IEEE, Vol. 68, No. 6, Jun. 1980, pp. 644-654.

[25] E.D. Jensen, *The Honeywell Experimental Distributed Processor -- An Overview,* Computer, Vol. 11, Jan. 1978, pp. 28-38.

[26] A.K. Jones et al, *StarOS, a Multiprocessor Operating System for the Support of Task Forces,* Proc. 7th Symp. on Operating System Prin., Dec. 1979, pp. 117-127.

[27] D. Klappholz, H.C. Park, *Parallelized Process Scheduling for a Tightly-Coupled MIMD Machine,* 1984 Intl. Conf. on Parallel Proc., Aug. 1984, pp. 315-321.

[28] L. Kleinrock, *Queuing Systems, Vol 2: Computer Applications,* 1976, Wiley, New York.

[29] L. Kleinrock, A. Nilsson, *On Optimal Scheduling Algorithms for Time-Shared Systems,* JACM, Vol. 28, No. 3, Jul. 1981, pp. 477-486.

[30] C.P. Kruskal, A. Weiss, *Allocating Independent Subtasks on Parallel Processors Extended Abstract,* 1984 Intl. Conf. on Parallel Proc., Aug. 1984, pp. 236-240.

[31] R.E. Larsen, *Tutorial: Distributed Control,* 1979, IEEE Press, New York.

[32] G. Le Lann, *Motivations, Objectives and Characterizations of Distributed Systems,* Lecture Notes in Computer Science - 105, 1981, pp. 1-9, Springer-Verlag.

[33] B.W. Lindgren, *Elements of Decision Theory,* 1971, MacMillan, New York.

[34] V.M. Lo, *Heuristic Algorithms for Task Assignment in Distributed Systems,* Proc. 4th Intl. Conf. on Dist. Comp. Systems, May 1984, pp. 30-39.

[35] P.Y.R. Ma, E.Y.S. Lee, J. Tsuchiya, *A Task Allocation Model for Distributed Computing Systems,* IEEE Transactions on Computers, Vol. C-31, No. 1, Jan. 1982, pp. 41-47.

[36] R. Manner, *Hardware Task/Processor Scheduling in a Polyprocessor Environment,* IEEE Trans. on Comp., Vol. C-33, No. 7, Jul. 1984, pp. 626-636.

[37] P.L. McEntire, J.G. O'Reilly, R.E. Larson, *Distributed Computing: Concepts and Implementations,* 1984, IEEE Press, New York.

[38] E. Mishkin, L. Braun Jr., *Adaptive Control Systems,* 1961, McGraw-Hill, New York.

[39] K. Narendra, *Learning Automata - A Survey,* IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-4, No. 4, Jul. 1974, pp. 323-334.

[40] L.M. Ni, K. Hwang, *Optimal Load Balancing Strategies for a Multiple Processor System,* Proc. Intl. Conf. on Parallel Proc., 1981, pp. 352-357.

[41] L.M. Ni, K. Abani, *Nonpreemptive Load Balancing in a Class of Local Area Networks,* Proc. Comp. Networking Symp., Dec. 1981, pp. 113-118.

[42] L.M. Ni, K. Hwang, *Optimal Load Balancing in a Multiple Processor System with Many Job Classes,* IEEE Transactions on Software Engineering, Vol. SE-11, No. 5, May. 1985, pp. 491-496.

[43] N.J. Nilsson, *Principles of Artificial Intelligence,* 1980, Tioga Publishing Co., Palo Alto, CA.

[44] J. Ousterhout, D. Scelza, P. Sindhu, *Medusa: An Experiment in Distributed Operating System Structure,* CACM, Vol. 23, No. 2, Feb. 1980, pp. 92-105.

[45] G. Popek et al, *LOCUS: A Network Transparent, High Reliability Distributed System,* Proc. 8th Symp. on O.S. Principles, Dec. 1981, pp. 169-177.

[46] M.L. Powell, B.P. Miller, *Process Migration in DEMOS/MP,* Proc. 9th Symp. on Operating Systems Principles, OS Review, Vol. 17, No. 5, Oct. 1983, pp. 110-119.

[47] C.C. Price, S. Krishnaprasad, *Software Allocation Models for Distributed Computing Systems,* Proc. 4th Intl. Conf. on Dist. Comp. Systems, May 1984, pp. 40-48.

[48] C. Shen, W. Tsai, *A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems Using a Minimax Criterion,* IEEE Trans. on Comp., Vol. C-34, No. 3, Mar. 1985, pp. 197-203.

[49] R.G. Smith, *The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver,* IEEE Trans. on Comp., Vol. C-29, No. 12, Dec. 1980, pp. 1104-1113.

[50] M.H. Solomon, R.A. Finkel, *The ROSCOE Distributed Operating System,* Proc. 7th Symp. on O.S. Principles, Dec. 1979, pp. 108-114.

[51] J.A. Stankovic et al, *An Evaluation of the Applicability of Different Mathematical Approaches to the Analysis of Decentralized Control Algorithms,* Proceedings of IEEE COMPSAC 82, Nov. 1982, pp. 62-69.

[52] J.A. Stankovic, I. S. Sidhu, *An Adaptive Bidding Algorithm for Processes, Clusters and Distributed Groups,* Proc. 4th Intl. Conf. on Dist. Comp. Systems, May 1984, pp. 49-59.

[53] J.A. Stankovic, *Simulations of Three Adaptive, Decentralized Controlled, Job Scheduling Algorithms,* Computer Networks, Vol. 8, No. 3, Jun. 1984, pp. 199-217.

[54] J.A. Stankovic, *A Perspective on Distributed Computer Systems,* IEEE Trans. on Comp., Vol. C-33, No. 12, Dec. 1984, pp. 1102-1115.

[55] J.A. Stankovic, *An Application of Bayesian Decision Theory to Decentralized Control of Job Scheduling,* IEEE Trans. on Comp., Vol. C-34, No. 2, Feb. 1985, pp. 117-130.

[56] J.A. Stankovic et al, *A Review of Current Research and Critical Issues in Distributed System Software,* IEEE Computer Society Distributed Processing Technical Committee Newsletter, Vol. 7, No. 1, Mar. 1985, pp. 14-47.

[57] H.S. Stone, *Critical Load Factors in Two-Processor Distributed Systems,* IEEE Trans. on Software Eng., Vol. SE-4, No. 3, May 1978, pp. 254-258.

[58] H.S. Stone, S.H. Bokhari, *Control of Distributed Processes,* Computer, Vol. 11, Jul. 1978, pp. 97-106.

[59] H. Sullivan, T. Bashkow, *A Large-Scale Homogeneous, Fully Distributed Machine - II,* Proc. 4th Symp. on Computer Architecture, Mar. 1977, pp. 118-124.

[60] A.S. Tanenbaum, *Computer Networks,* 1981, Prentice-Hall, Englewood Cliffs, N.J.

[61] D.P. Tsay, M.T. Liu, *MIKE: A Network Operating System for the Distributed Double-Loop Computer Network,* IEEE Transactions on Software Engineering, Vol. SE-9, No. 2, Mar. 1983, pp. 143-154.

[62] D.C. Tsichritzis, P.A. Bernstein, *Operating Systems,* 1974, Academic Press, New York.

[63] K. Vairavan, R.A. DeMillo, *On the Computational Complexity of a Generalized Scheduling Problem,* IEEE Transactions on Computers, Vol. C-25, No. 11, Nov. 1976, pp. 1067-1073.

[64] R.A. Wagner, K.S. Trivedi, *Hardware Configuration Selection Through Discretizing a Continuous Variable Solution,* Proc. 7th IFIP Symp. on Comp. Performance Modeling, Measurement and Evaluation, Toronto, Canada, 1980, pp. 127-142.

[65] Y.T. Wang, R.J.T. Morris, *Load Sharing in Distributed Systems,* IEEE Trans. on Comp., Vol. C-34, No. 3, Mar. 1985, pp. 204-217.

[66] M.O. Ward, D.J. Romero, *Assigning Parallel-Executable, Intercommunicating Subtasks to Processors,* 1984 Intl. Conf. on Parallel Proc., Aug. 1984, pp. 392-394.

[67] L.D. Wittie, A.M. Van Tilborg, *MICROS: A Distributed Operating System for MICRONET, A Reconfigurable Network Computer,* IEEE Transactions on Computers, Vol. C-29, No. 12, Dec. 1980, pp. 1133-1144.

APPENDIX

Annotated Bibliography

APPLICATION OF TAXONOMY TO EXAMPLES FROM LITERATURE

This appendix contains references to additional examples not included in section 3 well as abbreviated descriptions of those examples discussed in detail in the text of the paper. The purpose is to demonstrate the use of the taxonomy of section 2 in classifying a large number of examples from the literature.

1. G.R. Andrews, D.P. Dobkin, P.J. Downey, *Distributed Allocation with Pools of Servers,* ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Aug. 1982, pp. 73-83.
   Global, dynamic, distributed(however in a limited sense), cooperative, sub-optimal, heuristic, bidding, non-adaptive, dynamic reassignment.

2. J.A. Bannister, K.S. Trivedi, *Task Allocation in Fault-Tolerant Distributed Systems,* Acta Informatica, Vol. 20, 1983, pp. 261-281, Springer-Verlag.
   Global, static, sub-optimal, approximate, mathematical programming.

3. F. Berman, L. Snyder, *On Mapping Parallel Algorithms into Parallel Architectures,* 1984 Intl. Conf. on Parallel Proc., Aug. 1984, pp. 307-309.
   Global, static, optimal, graph theory.

4. S.H. Bokhari, *Dual Processor Scheduling with Dynamic Reassignment,* IEEE Trans. on Software Eng., Vol. SE-5, No. 4, Jul. 1979, pp. 326-334.
   Global, static, optimal, graph theoretic.

5. S.H. Bokhari, *A Shortest Tree Algorithm for Optimal Assignments Across Space and Time in a Distributed Processor System,* IEEE Trans. on Software Eng., Vol. SE-7, No. 6, Nov. 1981, pp. 335-341.
   Global, static, optimal, mathematical programming, intended for tree-structured applications.

6. R.M. Bryant, R.A. Finkel, *A Stable Distributed Scheduling Algorithm,* Proc. 2nd Intl. Conf. on Dist. Comp., Apr. 1981, pp. 314-323.
   Global, dynamic, physically distributed, cooperative, sub-optimal, heuristic, probabilistic, load-balancing.

7. T.L. Casavant, J.G. Kuhl, *Design of a Loosely-Coupled Distributed Multiprocessing Network,* 1984 Intl. Conf on Parallel Proc., Aug. 1984, pp. 42-45.
   Global, dynamic, physically distributed, cooperative, sub-optimal, heuristic, load-balancing, bidding, dynamic reassignment.

8. L.M. Casey, *Decentralized Scheduling,* Australian Computing Journal, Vol. 13, May 1981, pp. 58-63.
   Global, dynamic, physically distributed, cooperative, sub-optimal, heuristic, load-balancing.

9. T.C.K. Chou, J.A. Abraham, *Load Balancing in Distributed Systems,* IEEE Trans. on Software Eng., Vol. SE-8, No. 4, Jul. 1982, pp. 401-412.
Global, static, optimal, queuing theoretical.

10. T.C.K. Chou, J.A. Abraham, *Load Redistribution Under Failure in Distributed Systems,* IEEE Transactions on Computers, Vol. C-32, No. 9, Sep. 1983, pp. 799-808.
Global, dynamic(but with static parings of supporting and supported processors), distributed, cooperative, sub-optimal, provides 3 separate heuristic mechanisms, motivated from fault recovery aspect.

11. Y.C. Chow, W.H. Kohler, *Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System,* IEEE Trans. on Comp., Vol. C-28, No. 5, May 1979, pp. 354-361.
Global, dynamic, physically distributed, cooperative, sub-optimal, heuristic, load-balancing, (part of the heuristic approach is based on results from queuing theory).

12. W.W. Chu et al, *Task Allocation in Distributed Data Processing,* Computer, Vol. 13, No. 11, Nov. 1980, pp. 57-69.
Global, static, optimal, sub-optimal, heuristic, heuristic approached based on graph theory and mathematical programming are discussed.

13. K.W. Doty, P.L. McEntire, J.G. O'Reilly, *Task Allocation in a Distributed Computer System,* IEEE InfoCom, 1982, pp. 33-38.
Global, static, optimal, mathematical programming(non-linear spatial dynamic programming).

14. K. Efe, *Heuristic Models of Task Assignment Scheduling in Distributed Systems* Computer, Vol. 15, Jun. 1982, pp. 50-56.
Global, static, sub-optimal, heuristic, load-balancing.

15. J.A.B. Fortes, F. Parisi-Presicce, *Optimal Linear Schedules for the Parallel Execution of Algorithms,* 1984 Intl. Conf on Parallel Proc., Aug. 1984, pp. 322-329.
Global, static, optimal, Uses results from mathematical programming for a large class of data-dependency driven applications.

16. A. Gabrielian, D.B. Tyler, *Optimal Object Allocation in Distributed Computer Systems,* Proc. 4th Intl. Conf. on Dist. Comp. Systems, May 1984, pp. 84-95.
Global, static, optimal, mathematical programming, uses a heuristic to obtain a solution close to optimal, employs backtracking to find optimal one from that.

17. C. Gao, J.W.S. Liu, M. Railey, *Load Balancing Algorithms in Homogeneous Distributed Systems,* 1984 Intl. Conf. on Parallel Proc., Aug. 1984, pp. 302-306.
Global, dynamic, distributed, cooperative, sub-optimal, heuristic, probabilistic.

18. W. Huen et al, *TECHNEC, A Network Computer for DIstributed Task Control,* Proceedings 1st Rocky Mountain Symposium on Microcomputers, Aug. 1977, pp. 233-237.
Global, static, sub-optimal, heuristic.

19. K. Hwang et al, *A Unix-Based Local Computer Network with Load Balancing,* Computer, Vol. 15, No. 4, Apr. 1982, pp. 55-65.
Global, dynamic, physically distributed, cooperative, sub-optimal, heuristic, load-balancing.

20. D. Klappholz, H.C. Park, *Parallelized Process Scheduling for a Tightly-Coupled MIMD Machine,* 1984 Intl. Conf. on Parallel Proc., Aug. 1984, pp. 315-321.
Global, dynamic, physically distributed, non-cooperative.

21. C.P. Kruskal, A. Weiss, *Allocating Independent Subtasks on Parallel Processors Extended Abstract,* 1984 Intl. Conf. on Parallel Proc., Aug. 1984, pp. 236-240.
Global, static, sub-optimal, but optimal for a set of optimistic assumptions, heuristic, problem stated in terms of queuing theory.

22. V.M. Lo, *Heuristic Algorithms for Task Assignment in Distributed Systems,* Proc. 4th Intl. Conf. on Dist. Comp. Systems, May 1984, pp. 30-39.
Global, static, sub-optimal, approximate, graph theoretic.

23. V.M. Lo, *Task Assignment to Minimize Completion Time,* 5th International Conference on Distributed Computing Systems, May 1985, pp. 329-336.
Global, static, optimal, mathematical programming for some special cases, but in general is sub-optimal, heuristic using the LPT algorithm.

24. P.Y.R. Ma, E.Y.S. Lee, J. Tsuchiya, *A Task Allocation Model for Distributed Computing Systems,* IEEE Transactions on Computers, Vol. C-31, No. 1, Jan. 1982, pp. 41-47.
Global, static, optimal, mathematical programming(branch and bound).

25. S. Majumdar, M.L. Green, *A Distributed Real Time Resource Manager,* Proc. IEEE Symp. on Distributed Data Acquisition, Computing and Control, 1980, pp. 185-193.
Global, dynamic, distributed, cooperative, sub-optimal, heuristic, load balancing, non-adaptive.

26. R. Manner, *Hardware Task/Processor Scheduling in a Polyprocessor Environment,* IEEE Trans. on Comp., Vol. C-33, No. 7, Jul. 1984, pp. 626-636.
Global, dynamic, distributed control and responsibility, but centralized information in hardware on bus lines. Cooperative, optimal, (priority) load balancing.

27. L.M. Ni, K. Hwang, *Optimal Load Balancing for a Multiple Processor System,* Proc. Intl. Conf. on Parallel Proc., 1981, pp. 352-357.
Global, static, optimal, mathematical programming.

28. L.M. Ni, K. Abani, *Nonpreemptive Load Balancing in a Class of Local Area Networks,* Proc. Comp. Networking Symp., Dec. 1981, pp. 113-118.
Global, dynamic, distributed, cooperative, optimal and sub-optimal solutions given - mathematical programming, and adaptive load balancing respectively.

29. J. Ousterhout, D. Scelza, P. Sindhu, *Medusa: An Experiment in Distributed Operating System Structure,* CACM, Vol. 23, No. 2, Feb. 1980, pp. 92-105.
Global, dynamic, physically non-distributed.

30. M.L. Powell, B.P. Miller, *Process Migration in DEMOS/MP,* Proc. 9th Symp. on Operating Systems Principles, OS Review, Vol. 17, No. 5, Oct. 1983, pp. 110-119.
Global, dynamic, distributed, cooperative, sub-optimal, heuristic, load balancing but no specific decision rule given.

31. C.C. Price, S. Krishnaprasad, *Software Allocation Models for Distributed Computing Systems,* Proc. 4th Intl. Conf. on Dist. Comp. Systems, May 1984, pp. 40-48.
Global, static, optimal, mathematical programming, but also suggest heuristics.

32. C.V. Ramamoorthy et al, *Optimal Scheduling Strategies in a Multiprocessor System,* IEEE Transactions on Computers, Vol. C-21, No. 2, Feb. 1972, pp. 137-146.
Global, static, optimal solution presented for comparison with the heuristic one also presented. Graph theory is employed in the sense that it uses task precedence graphs.

33. K. Ramamritham, J.A. Stankovic, *Dynamic Task Scheduling in Distributed Hard Real-Time Systems,* Proc. 4th Intl. Conf. on Dist. Comp. Systems, May 1984, pp. 96-107.
Global, dynamic, distributed, cooperative, sub-optimal, heuristic, bidding, one-time assignments, (a real time guarantee is applied before migration).

34. J. Reif, P. Spirakis, *Real-Time Resource Allocation in a Distributed System,* ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Aug. 1982, pp. 84-94.
Global, dynamic, distributed, non-cooperative, probabilistic.

35. S. Sahni, *Scheduling Multipipeline and Multiprocessor Computers,* 1984 Intl. Conf. on Parallel Processing, Aug. 1984, pp. 333-337.
Global, static, sub-optimal, heuristic.

36. T.G. Saponis, P.L. Crews, *A Model for Decentralized Control in a Fully Distributed Processing System,* Fall COMPCON, 1980, pp. 307-312.
Global, static, sub-optimal, heuristic based on load balancing. Also intended for applications of the nature of coupled recurrence systems.

37. C.C. Shen, W.H. Tsai, *A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems Using a Minimax Criterion,* IEEE Trans. on Comp., Vol. C-34, No. 3, Mar. 1985, pp. 197-203.
Global, static, optimal, enumerative.

38. J.A. Stankovic, *The Analysis of a Decentralized Control Algorithm for Job Scheduling Utilizing Bayesian Decision Theory,* Proc. Intl. Conf. on Parallel Proc., 1981, pp. 333-337.
Global, dynamic, distributed, cooperative, sub-optimal, heuristic, one-time assignment, probabilistic.

39. J.A. Stankovic, *A Heuristic for Cooperation Among Decentralized Controllers,* IEEE INFOCOM 1983, Apr. 1983, pp. 331-339.
Global, dynamic, distributed, cooperative, sub-optimal, heuristic, one-time assignment, probabilistic.

40. J.A. Stankovic, I. S. Sidhu, *An Adaptive Bidding Algorithm for Processes, Clusters and Distributed Groups,* Proc. 4th Intl. Conf. on Dist. Comp. Systems, May 1984, pp. 49-59.
Global, dynamic, physically distributed, cooperative, sub-optimal, heuristic, adaptive, bidding, additional heuristics regarding clusters and distributed groups.

41. J.A. Stankovic, *Simulations of Three Adaptive, Decentralized Controlled, Job Scheduling Algorithms,* Computer Networks, Vol. 8, No. 3, Jun. 1984, pp. 199-217.
Global, dynamic, physically distributed, cooperative, sub-optimal, heuristic, adaptive, load-balancing, one-time assignment. Three variants of this basic approach given.

42. J.A. Stankovic, *An Application of Bayesian Decision Theory to Decentralized Control of Job Scheduling,* IEEE Trans. on Comp., Vol. C-34, No. 2, Feb. 1985, pp. 117-130.
Global, dynamic, physically distributed, cooperative, sub-optimal, heuristic based on results from Bayesian Decision Theory.

43. J.A. Stankovic, *Stability and Distributed Scheduling Algorithms,* Proc. ACM National Conference, New Orleans, Mar. 1985.
Here there are two separate algorithms specified. The first is a Global, dynamic, physically distributed, cooperative, heuristic, adaptive, dynamic reassignment example based on stochastic learning automata. The second is a Global, dynamic, physically distributed, cooperative, heuristic, bidding, one-time assignment approach.

44. H.S. Stone, *Critical Load Factors in Two-Processor Distributed Systems,* IEEE Trans. on Software Eng., Vol. SE-4, No. 3, May 1978, pp. 254-258.
Global, dynamic, physically distributed, cooperative, optimal, (graph theory based).

45. H.S. Stone, S.H. Bokhari, *Control of Distributed Processes,* Computer, Vol. 11, Jul. 1978, pp. 97-106.
Global, static, optimal, graph theoretical.

46. H. Sullivan, T. Bashkow, *A Large-Scale Homogeneous, Fully Distributed Machine - I,* Proc. 4th Symp. on Computer Architecture, Mar. 1977, pp. 105-117.
Global, dynamic, physically distributed, cooperative, sub-optimal, heuristic, bidding.

47. A.M. VanTilborg, L.D. Wittie, *Wave Scheduling--Decentralized Scheduling of Task Forces in Multicomputers,* IEEE Trans. on Comp., C-33, 9, Sep. 1984, pp. 835-844.
Global, dynamic, distributed, cooperative, sub-optimal, heuristic, probabilistic, adaptive. Assumes tree-structured (logically) task-forces.

48. R.A. Wagner, K.S. Trivedi, *Hardware Configuration Selection Through Discretizing a Continuous Variable Solution,* Proc. 7th IFIP Symp. on Comp. Performance Modeling, Measurement and Evaluation, Toronto, Canada, 1980, pp. 127-142.
Global, static, sub-optimal, approximate, mathematical programming.

49. Y.T. Wang, R.J.T. Morris, *Load Sharing in Distributed Systems,* IEEE Trans. on Comp., Vol. C-34, No. 3, Mar. 1985, pp. 204-217.
Global, dynamic, physically distributed, cooperative, sub-optimal, heuristic, one-time assignment, load-balancing.

50. M.O. Ward, D.J. Romero, *Assigning Parallel-Executable, Intercommunicating Subtasks to Processors,* 1984 Intl. Conf. on Parallel Proc., Aug. 1984, pp. 392-394.
Global, static, sub-optimal, heuristic.

51. L.D. Wittie, A.M. Van Tilborg, *MICROS, A Distributed Operating System for MICRONET, A Reconfigurable Network Computer,* IEEE Transactions on Computers, Vol. C-29, No. 12, Dec. 1980, pp. 1133-1144.
Global, dynamic, physically distributed, cooperative, sub-optimal, heuristic, load-balancing(also with respect to message traffic).