Scheduling Issues in High-Performance Computing

L.W. Dowdy*, E. Rosti†, G. Serazzi‡, E. Smirni§

Abstract

In this paper we consider the problem of scheduling computational resources across a range of high-performance systems, from tightly coupled parallel systems to loosely coupled ones like networks of workstations and geographically dispersed meta-computing environments. We review the role of architecture issues in the choice of scheduling discipline and we present a selected set of policies that address different aspects of the scheduling problem. This discussion serves as the motivation for addressing the success of academic research in scheduling as well as its common criticisms.

1 Introduction

Many traditional science disciplines as well as recent multimedia applications are increasingly dependent upon powerful high-performance systems for the execution of both computationally intensive and data intensive simulations of mathematical models and their visualization. Parallel computing in particular has emerged as an indispensable tool for problem solving in many scientific domains during the course of the past fifteen years.

Powered by the rapid development of high-performance parallel architectures and software, a wide variety of parallel systems is now available to the user community. This variety ranges from the traditional multiprocessor vector systems, to shared memory multi-threaded systems, to distributed shared memory MIMD systems, to distributed memory MIMD systems, to clusters or networks of stand-alone workstations (NOWs) or PCs, to even geographically dispersed meta-systems connected by high-speed Internet connections. Research and development efforts focusing on building faster and more flexible memories at all hierarchy levels, from cache to disks, and on building higher speed and larger bandwidth networks, both specialized and generic, contribute to the deployment of a wide spectrum of high-performance systems.

Complementary to hardware advances is the availability of transparent, highly portable, and robust runtime environments like PVM and MPI. Such environments hide the architecture details from the end-users and contribute to the portability and robustness of parallel codes across a variety of hardware substrates. The availability of such environments transforms every intranet into a high-performance system, thus increasing the user access to parallel systems, which used to be exclusively offered by super-computing centers in the late 80's. A step further in this direction is represented by meta-computing systems [11, 14]. Such systems provide concurrent use of a potentially large number of geographically distributed heterogeneous computational resources in order to solve large scale problems, which are otherwise intractable due to their diverse requirements in terms of computing power, memory, and secondary storage.

The considerable availability of different parallel systems as well as the diversity of the available hardware and software make the arbitration and management of resources among the user community a non-trivial problem. The number of users that attempt to use the system simultaneously, the variable parallelism of the applications and their respective computational and secondary storage needs, the meeting of the execution deadlines of applications, the presence of "unexpected" conditions in the system, are examples of issues that exacerbate the application scheduling problem.

Research in application scheduling for parallel systems has generated a large amount of results under different assumptions about the type of parallel workload, the goals of scheduling, and the type of parallel system [6, 7, 8, 9, 10]. Job scheduling research for traditional parallel systems typically focuses on ways to distribute processors to competing applications in order to optimize two conflicting functions: minimize the individual job response time (i.e., cater to the interests of the user, which is especially important for interactive systems) and maximize the system throughput (i.e., maximize system

*Department of Computer Science - Vanderbilt University - Tennessee, USA - dowdy@vuse.vanderbilt.edu

†Dipartimento di Scienze dell'Informazione - Università di Milano, Italy - rose@dsi.unimi.it

‡Dipartimento di Elettronica e Informazione - Politecnico di Milano and CESTIA CNR center, Italy - serazzi@elet.polimi.it

§Department of Computer Science - College of William and Mary, Virginia, USA - esmirni@cs.wm.edu

This work was partially supported by Italian PQE2000 project and M.U.R.S.T. 60% and by sub-contract 19X-SL131V from the Oak Ridge National Laboratory managed by Martin Marietta Energy Systems, Inc. for the U.S. Department of Energy under contract no. DE-AC05-84OR21400.

utilization, which is particularly important to amortize the cost of parallel supercomputers). Scheduling in loosely coupled parallel systems such as networks of workstations and heterogeneous distributed systems is often related to process dispatching and load balancing.

In this paper, we address the issue of scheduling resources in high-performance systems. In Section 2 we consider the critical factors of the computer architecture field that affect the design of scheduling algorithms in high-performance systems. We continue in Section 3 by presenting a short survey of the various scheduling solutions that appeared in the literature. Finally, we conclude in Section 4 discussing the successes and failures of the parallel scheduling field.

2 Design Parameters for a Scheduling Algorithm

In this section we consider the spectrum of parallel architectures where the scheduling problem applies. We concentrate on the system and workload parameters that directly affect the proposed solutions to the scheduling problem.

2.1 High-Performance Architectures

With the term high-performance architectures we refer to a broad range of systems that either deliver high peak performance in terms of single job execution time, or allow to execute applications with significant resource requirements in terms of computing power, memory space, and secondary storage. They span from highly integrated and tightly coupled systems comprised of tens or hundreds of homogeneous processing elements (PEs)[1] to distributed systems comprising possibly thousands of heterogeneous commodity stand-alone processing elements loosely connected on local-area or wide-area general purpose networks. A schematic representation of this view of the architecture spectrum is given in Figure 1.

High-end vector processors and/or custom-made processors contribute to the high cost and possibly higher peak performance of those systems at the left part of the spectrum. If high specialization characterizes the left end of the architecture spectrum, high hardware variability (and subsequently high variance in both peak and delivered performance) characterizes its right part.

[1] In the context of this document, a processing element consists of the CPU, memory, and the interprocessor communication component.

With respect to memory organization, there is a host of possibilities ranging from purely distributed memory to various levels of logically shared but physically distributed memory systems, with memory latency being a function of the data location. The amount of main memory increases from the left to the right of the spectrum, as the amount of local memory of several workstations and high-end PCs is often significantly larger than what is provided on the individual PEs of MPPs and supercomputers. Input/output (I/O), being the last level of memory hierarchy, is another area where a variety of technologies that attempt to lessen the disparity between increases in processor speed and memory access time exists. Mass storage systems can be again highly specialized (left side of the spectrum) or generic ones (right side of the spectrum).

With respect to interconnection networks, there is a wide variety to the degree of specialization spanning from highly specialized custom-made designs to generic high-speed networks. Hypercube based, mesh based, fat tree based, multistage, bus based, uniform and non-uniform latency networks are just examples of the interconnects used in parallel systems. Moving towards the right end of the spectrum, network speeds decrease as arbitrary topologies may be organized in LAN or WAN fashion.

2.2 The Scheduling Perspective

The problem of scheduling in parallel systems is the composite problem of deciding *where* and *when* an application should execute, i.e., on which PEs (also indicated as the processor allocation or job scheduling problem) and in what order (also indicated as the process dispatching problem) the application processes (or threads) will run. Two-level scheduling strategies offer solutions to the composite problem of processor allocation and process dispatching. Single-level scheduling strategies focus on either of the problems.

Regardless of what part of the scheduling problem we focus our attention, both system and workload features need to be taken into consideration when designing a scheduling strategy. Although the scheduling problem is conceptually the same across different systems and parallel workloads, the feasibility and performance of possible solutions are very sensitive to the underlying architecture and workload changes. There is no such thing as policy portability across the architecture spectrum if we want to guarantee optimal policy performance. In other

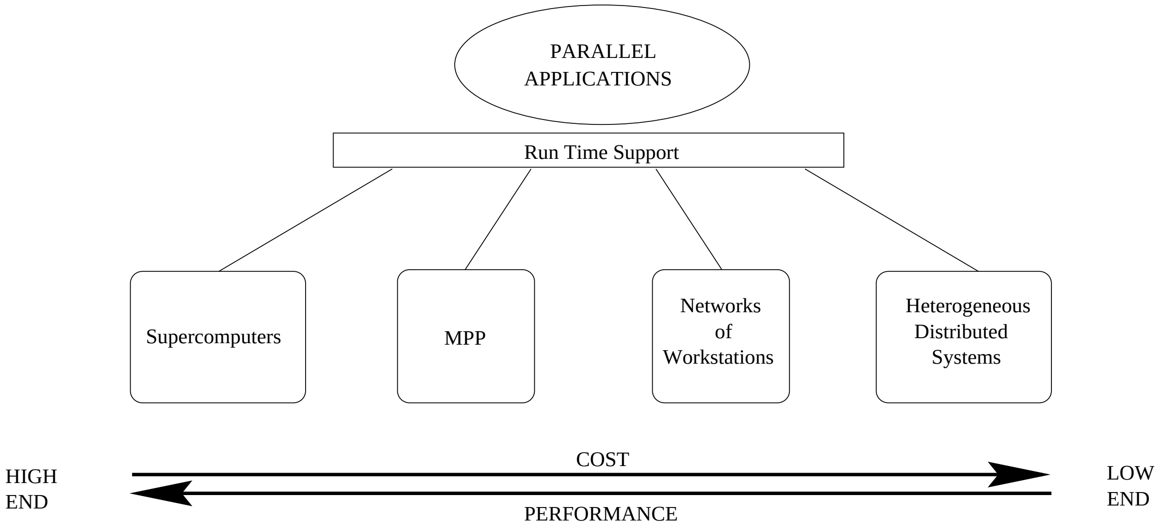

Figure 1: High performance computing architecture spectrum.

words, although the underlying physical architecture may be hidden from the user as run time environments give the illusion of the same parallel system, scheduling must be done with caution as solutions need to be carefully tailored to both hardware and software. The complexity of solutions increases even more when additional conditions are taken into account. For example, consider the scheduling applications on heterogeneous geographically dispersed distributed systems. Site autonomy, i.e., different administrative domains operating and owning the resources, considerably contribute to the increase of the problem complexity.

Memory organization at all levels of the memory hierarchy is critical to both application performance and application scheduling in multiprogrammed parallel systems. Memory becomes critical for two reasons: size and access mode. The size of available memory, whether real or virtual, affects the number of jobs that can simultaneously be in execution on each PE, or the minimum number of PEs that the application requires for its execution. In spite of the limited size of each PE's main memory, supercomputers and MPPs have little or no support for virtual memory, mostly because of its high cost. On the other hand, NOWs and heterogeneous distributed systems are usually provided with larger RAMs and virtual memory. A related issue is that of the memory access mode. The use of either logically shared memory, with uniform or non-uniform access times, or purely distributed memory affects both the PE allocation scheme and the process dispatching scheme. Cache coherence schemes simplify the processor allocation problem and the related problem of load balancing, because they lessen the degree of affinity of a process with a given PE. In this case, process migration is inexpensive and can become an integral part for a scheduling strategy.

The second level of scheduling, i.e., process dispatching, is also affected by the memory structure. There are policies that integrate the two decisions of where and when a process will execute. Such integrated policies have been proposed for parallel systems on the left side of the spectrum. As we move towards the right part of the spectrum the two decisions are in general disjoint because the PE allocation is usually done by the runtime environment while the dispatching policy is part of the local operating system.

The interconnection network is another critical architecture component that may affect scheduling decisions. The type of interconnect defines whether arbitrary partitions of the pool of PEs are possible or topological constraints apply, such as node contiguity, substructure partitioning as in hypercubes, or fixed numbers, e.g., powers of two. For loosely coupled systems such constraints hardly ever apply since the interconnect is usually the Internet, or some TCP/IP routed local or wide area network. This implies that the network load and the consequent increase in communication latency should be taken into account when designing a scheduling algorithm.

We conclude the discussion on the importance of architecture in scheduling decisions by addressing the I/O component. There are parallel applications with so large memory requirements that they require out-of-core solution algorithms that explicitly access data from secondary storage devices. When scheduling such applications, explicit support for scheduling I/O resources is also needed. Especially for cases where I/O resources are shared by multiple applications, perturbation in I/O access times is inevitable, further delaying the execution times of each parallel application. Scheduling policy design should consider the coordinated allocation of both PEs and disks.

The last factor we consider, but not least important, is the flexibility and requirements of workload that the high-performance system is expected to execute. The distinction among rigid jobs, i.e., jobs that can run only on the requested number of PEs, moldable jobs, i.e., jobs that accept any number of PEs and will keep them for the entire execution, evolving jobs, i.e., jobs whose parallelism level changes during execution and is explicitly indicated to the system at each phase for PE reallocation, and malleable jobs, i.e., jobs that tolerate PE reassignment during execution, is also important to scheduling policy design.

3 Algorithms

In this section we present an overview of research results in the area of scheduling that appeared in the literature. We emphasize scheduling and their prerequisites in tightly coupled systems as the scheduling problem on these systems has been studied for a longer period of time but we also review solutions in loosely coupled environments (e.g., networks of workstations and meta-computing systems).

3.1 Tightly-Coupled Multiprocessor Systems

We present a brief overview of the scheduling policies for tightly coupled multiprocessor systems that appeared in the literature (see [6] for a complete and detailed coverage of proposed scheduling policies). We concentrate on interactive environments and present a classification of the proposed scheduling techniques. We focus on the variability of the proposed approaches, as well as on the necessary information for policy implementation (i.e., system and workload parameters). The diversity of existing architectures (although scheduling policies evolved in "parallel" with advances in the parallel systems area), the fact that parallel systems are shared by many users with *different* needs, along with the conflicting goals of scheduling contribute to the common belief that there are no unique solutions to the problem. Here, we concentrate on a few notable approaches.

We begin by briefly outlining the workload characteristics and system parameters that are needed for scheduling decisions. We continue by describing the basic ideas in processor-only scheduling algorithms (i.e., algorithms that consider only the processor resources) and outline the issues in combined resource scheduling for processors and memory and/or I/O resources.

3.1.1 Workload Characterization and Scheduling Parameters

Many scheduling policies base their allocation decisions on information related to the parallel workload characteristics and the state of the system, as reflected by available resources and the workload demands. Since examining policy alternatives by experimentation only is prohibitively expensive, if not infeasible, it is necessary to use powerful workload abstractions that can serve as input to the analytic models and/or simulations.

The characterization studies that appeared in the literature are distinguished by the granularity of observation. Single application characterization studies usually offer information about how well the application can utilize the existing processors. Examples of such measures are reported in Table 1 and include the application speedup (and its various functional forms), the average and maximum parallelism, the processor working set, the variance in parallelism, the CPU versus I/O intensity of the application. Any information that the user may disclose with respect to the scalability of each application can be of great use for scheduling decisions (see subsection 3.1.2).

For a scheduling policy to be effective, it not only needs to strive to optimize performance for workloads with different scalabilities, but it should also adjust resource allocations according to the system load and the workload mix. Composite workload characterization considers the behavior of *all* applications in the system concentrating on their interarrival and service time distributions. Recent studies that concentrated on production parallel workloads at various supercomputing centers report similarities in the job size distribution and job interarrival times. These studies indicate that the popular exponential assumptions about the interarrival and service distribution of the parallel workload do not hold. Indeed, they report that the coefficient of variation of these measures is greater than one.

Apart from workload related information, system related information is necessary for scheduling decisions in order to adjust resource allocation according to the system load. Such information is summarized in Table 1 and includes measures that can be obtained by monitoring the system, e.g., the multiprogramming level (MPL) on each PE, the number of waiting jobs at system level, the accumulated wait time of the jobs in the system. Policies that rely solely on information that can be easily retrieved by simple system monitoring are of great importance with respect to implementation on actual systems [27].

Workload/Application	System
avg/max parallelism	queue length
execution rate	num. of idle processors
speedup	num. of running jobs
proc. working set	local MPL
I/O activity	cumulated wait time

Table 1: Parameters used and usable in scheduling algorithms.

3.1.2 Algorithms for Processor-Only Scheduling

The earliest works in parallel scheduling were based on *time-sharing* (i.e., rotation of resources among the set of parallel jobs in time) but initial investigation of such policies quickly confirmed that strategies that proved successful in uniprocessor systems do not readily address the needs of a parallel environment. A lot of attention has been put on *space-sharing* as an alternative way to share resources in parallel systems, where the parallel system is split in disjoint partitions, each dedicated to a parallel application. By combining the best of the time-sharing and the space-sharing worlds, *gang scheduling* offers the possibility to time-share the partitions in a parallel system.

Early investigations of simple *static* space-sharing policies, where the system is partitioned in a number of equal sized partitions that remain fixed indicated that the partition size should adjust according to the state of the system, as performance is sensitive to system load and application scalability [24, 26, 27]. However, if these two factors are known in advance (and are guaranteed to remain unchanged), the ideal partition size can be determined and equipartitioning proves to be an excellent solution. Thus, static policies provide a base-line comparison for further policy development.

Building on the idea of equipartitioning, but striving for more flexibility, *adaptive* policies have been proposed. With adaptive policies, the job partition size is determined before the job starts execution and can be a function of the system load (as reflected by the size of the waiting queue) and of the workload parallel characteristics. The common characteristics of these policies are that they target smaller partitions as the system load increases and larger ones as the load decreases, but they suggest an upper bound to the partition size so as to ensure that the workload can make good use of its assigned processors. The most notable examples of adaptive policies require knowledge of the workload scalability in the form of the PWS measure [13], the average parallelism [3, 31], the minimum and maximum parallelism, the variance in parallelism [26]. Adaptive policies are attractive because they are easy to implement but their performance benefits are limited by their ability to quickly respond to workload changes.

Dynamic policies have been proposed as an alternative to the adaptive ones. They are flexible enough to allow the scheduler to reduce or augment the number of processors assigned to each job in response to environment or workload changes. With dynamic policies, the job partition size can be modified during execution, adjusting to the overall system load or to each running job's instantaneous parallelism [18]. Dynamic policies introduce costs related to data and job migration, processor preemption and coordination, and application reconfiguration. Their performance and implementation feasibility are a function of the underlying architecture and programming model as costs may outweigh benefits [29] (e.g., like in distributed memory systems). Dynamic policies that limit the number of preemptions strive to accommodate such problems [3].

We conclude the discussion on processor-only scheduling by discussing *gang-scheduling*, which effectively integrates space-sharing and time-sharing. In Ousterhout's seminal paper [20] on coscheduling the issue of integrating processor allocation with process dispatching (i.e., both levels of scheduling) was proposed. Gang-scheduling ensures that all running processes of a parallel job execute in the same time quantum [20, 16, 29] and it is a particularly useful scheme for guaranteeing good interactive response times.

3.1.3 Considerations for Dual Resource Allocation Policies

Another important aspect in scheduling is that of coordinating the allocation of dual resources, thus making a hard problem even harder. Given that in distributed memory systems there is limited memory capacity in each processor, and given that most scheduling policies decrease the job partition as load increases, the danger of performance losses due to excessive paging is eminent. The important trade-off between processor and memory allocation has been addressed by several processor-memory scheduling policies. A general consensus has been reached that processor allocation should be done so as to accommodate for potential memory constraints [22, 21, 25].

Another issue that has been traditionally overlooked and only recently examined is that many applications also contend for shared I/O resources. Since I/O resources belong to the slowest level of memory hierarchy, their efficient management becomes critical for high performance. In traditional space-sharing policies, each application has the impression of a dedicated system image once it starts executing. The dedicated image may be true for the processing elements, but it is not true for the parallel I/O system. If there are other I/O intensive jobs, then all applications compete for access to the secondary storage resources. As a result, there may be significant perturbations in the expected I/O service times and consequently in the job execution times. A recent study explicitly considered the I/O characteristics of parallel applications and demonstrated performance trends that are significantly different from those previously reported where application I/O characteristics were not explicitly considered [23].

3.2 Networks of Workstations

Scheduling in loosely coupled systems such as networks of workstations becomes increasingly complicated. Hardware heterogeneity, variability in interconnect speeds, coexistence of parallel and sequential workloads, and severe fluctuations in the expected system load are among the factors that further contribute to the difficulty of devising efficient scheduling algorithms in NOW environments. Via measurements, it is confirmed that in academic settings there is significant availability of idle (or lightly loaded) workstations during the course of a workday [1]. Taking advantage of workstation cycles for parallel computation in such environments should be done with caution. The fluctuations in load of each workstation due to sequential jobs suggest that load balancing, checkpointing, and process migration should be addressed as part of process co-scheduling of each parallel application.

There is a host of scheduling strategies proposed for such environments [2, 28] that concentrate on different alternatives to implement co-scheduling. The applications that are run in such environments are message passing applications. Since implementing any type of co-scheduling will require a large number of messages to be sent, it can become prohibitively expensive. Co-scheduling is thus implemented in an implicit manner, triggered by communication events that make indirect references to what processes may be currently running on the other workstations in the network. The majority of these algorithms is based on the idea of changing the process priorities on each workstation according to indications that processes of the same application may be active on other workstations. It is natural to tie the implementation of such policies within the context of user-level messaging platforms.

3.3 Heterogeneous Distributed Systems

The variety of architectures and components integrated in meta-computing environments raise plenty of new problems, together with an enormous amount of computing power. The system heterogeneity with respect to both hardware and software platforms coupled with the system's operational domain that may span from local to wide-area networks such as the Internet itself, constitute a significantly complex environment. The objectives of meta-computing vary from achieving high computational speedup by means of several cooperating supercomputers [12], to supporting high throughput computations by using idle resources on a network [17]. Thus, scheduling in meta-computing environments must be aware of functionalities such as negotiation of resources with site managers, multiple language support, interoperability between heterogeneous hardware and software components, co-allocation for parallel processing, on-line control of geographically distributed computations. In short, scheduling in such systems must account for issues that are not present in homogeneous high-performance systems.

Resource management in heterogeneous distributed systems and in meta-systems has a dramatic effect on application performance. Individual application requirements are usually expressed in an integrated fashion together with their task structure. Depending upon the size of the operational domain, different scheduling algorithms apply. A possible classification distinguishes *batch scheduling* and *wide-area scheduling* policies [5].

Batch scheduling policies are usually applied to a networked set of computers that belong to a single administrative domain. Users may directly select the resource (among the ones in the operational domain) to which a request is to be sent or describe the resource requirements of each task using a suitable interface or language. Several meta-computing environments adopt this type of approach for the resource management problem [4, 5, 15].

A convenient organization adopted in many wide-area scheduling policies is a hierarchy of managers, that cooperate in mapping high-level program requests into requests to target systems [30, 19]. System managers schedule resource requests at a global level, while local managers are responsible for resource allocation at the single host level. Object-oriented models are also used, where specialized objects negotiate the resource allocation [14].

The various scheduling models implemented address to different degrees the characteristics that a meta-computing environment should have. Site autonomy, extensible core, scalable architecture, easy-to-use seamless computational environment, high-performance via parallelism, single persistent name space, security for users and resource owners, management and exploitation of resource heterogeneity, multiple language support, inter-operability, and fault-tolerance are the fundamental characteristics of modern meta-computing environments [14].

4 The Future of High Performance Scheduling: Bright or Dim?

Processor scheduling in high-performance systems, with particular emphasis on parallel systems, is an area that has received considerable attention over the past decade. However, despite the amount of attention and the number of new proposed scheduling algorithms, the true impact of this research area remains questionable. The evidence for this skepticism is the gap (chasm?) between the theoretical solutions proposed by the academic community and the practical implementations observed in real systems. This is further evidenced by the relative lack of private and public funding for such research.

It is noteworthy that most parallel machines at supercomputing centers use the most simplistic FCFS algorithms when they operate in interactive mode. For batch production, simple solutions like NQS are typical. Anecdotal evidence exits that users behave mischievously in order to beat the primitive scheduler, steal computational cycles, and weasle in ahead of other users in the waiting queues. Thus, users mistrust the scheduler and believe that the policies are not optimal, particularly from their selfish viewpoint. It is paradoxical that the most sophisticated systems do not implement equally sophisticated scheduling algorithms. It is natural to conclude that either the benefits of implementing more sophisticated algorithms are not worth the cost, or that the theoretical academic community has failed at validating and demonstrating the worth of these algorithms to the satisfaction of the industrial community. The conclusion is that few of the results from the past decade of research on high-performance scheduling has enjoyed industrial success. If industrial implementation is used as a metric of success, the conclusion is that the scheduling research field has been unsuccessful, despite the efforts, energy, and results of dozens of researchers.

There are several views that help explain the existence of this paradox. These views (over-stated here for contrast), and corrective actions are summarized briefly.

Views:

- *Scheduling in high-performance environments is not needed.* Typically, high-performance systems are modularly designed. The addition of processing elements, memory modules, and communication capacity can be accomplished easily and at relatively little cost. Therefore, the systems are either: 1) "fast enough", or 2) can be made faster by adding more hardware. In either case, complicated resource allocation policies are not needed. *Scheduling effort in high-performance environments is not worth it.*

Performance analysis and tuning studies typically yield performance improvements in the 10%–50% range. Hardware enhancements typically yield performance improvements of an order of magnitude. These hardware enhancements seem to be occurring with regularity every few years. Therefore, scheduling in high-performance environments is not worth it.

- *Analysis of scheduling policies in high performance environments is too hard to be practical.* High-performance environments are characterized by a variety of hardware resources with different degrees of parallelism and by a mix of sequential and highly parallel workloads. The formal analysis of resource allocation in these systems is complex and requires the construction of non-traditional models. Whenever new models are constructed, many simplifying assumptions are made. These assumptions and the related scheduling models must be validated. Validation requires detailed experimentation and sensitivity analysis. This requires access to a prototype environment which can be used for hands-on, dedicated, stand-alone, and controlled experiments. Validation of the models is too hard, with too little payoff, to justify it. It is impractical.

- *Experimental evaluation of scheduling in high-performance environments is not rewarding.* Experimental performance analysis is dirty and it requires a lot of work. Experiments often must be repeated several times, and always point towards other experiments that "would be nice to see what happens if ...". The time frame required to begin producing good results is long. The academic researcher must go through an iterative process which involves obtaining a suitable testbed system, instrumenting the system, capturing and characterizing the workload, constructing different scheduling strategies, evaluating their performance via experimentation, and analyzing the data. Given the rate of advances in the computer architecture field, this iterative process can become endless. Industrial researchers will be driven to more "quick-and-dirty" analysis due to product deadline constraints. Experimental evaluation of scheduling in high-performance environments is not rewarding.

The above views are overlapping, negative, and extreme. However, they hint at the essence of the scheduling problem. Also, in stating these views, the appropriate corrective actions become apparent.

Corrective Actions:

- *A holistic view of the system and of the scheduling problem is needed.* This involves first understanding and then modeling the resource requirements (i.e., computational, memory, communication, and I/O requirements) of parallel applications. Application abstractions that can capture the interaction of the critical architectural components as triggered by the application are very important. Synergistic scheduling solutions that take into account the management of all critical architecture parts are needed.

- *A significant amount of foundational work is needed.* This includes the construction of system monitoring tools that are viewed as essential parts of the normal operating system. On-the-fly workload characterization through non-intrusive performance monitoring tools are needed.

- *Simplicity should be stressed.* Simple assumptions, simple models, and simple experimentation of complex systems, and simple solutions are needed. It should be possible to analyze the major aspects of any system in a simple manner. That is, with a relatively simple model, it should be possible to predict the performance of a given parallel system with a given parallel workload within 80% accuracy. Simple scheduling policies that are equally robust under *different* workload types are needed.

- *Better interaction between different communities in the parallel research field is needed.* The scheduling community needs to interact more closely with the architecture, the operating system, and the parallel user communities. It is now clear that scheduling policies cannot be given simply in a vacuum. Solutions for allocation policies need to be tightly coupled with the system underlying architecture and need to be an integral part of every operating system. Input to and from the user community is important for the construction of realistic workload models. The issue of making inaccurate assumptions or assumptions that do not reflect the current practice is very common. Parallel programmers could adapt their programming style to develop moldable jobs that can run on different number of assigned processors if they have the right incentive. Additionally, the development of software for checkpointing the execution of parallel programs would help application programmers write transparently evolving, or even malleable, programs.

- *Unification and integration of the various scheduling results are needed.* Unification of results regarding the effectiveness of scheduling policies under a variety of assumptions would greatly help both system designers and users. An exhaustive comparison and evaluation of scheduling disciplines is needed under the various architecture substrates and workload conditions. Such work will provide answers and the guidelines for the construction of meta-schedulers that flexibly "adapt" their decision mechanisms according to changes in both system and workload environments.

Scheduling resources in high-performance systems is needed and is worthwhile. However, we must bear the burden of proof, which requires that we must get to work as no easy answers to the issues outlined here are provided. Yet awareness is the first step in the direction of problem solving.

References

[1] Acharya A., Edjlali G., Saltz J., "The Utility of Exploiting Idle Workstations for Parallel Computation", in *Proc. of the 1997 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pp 225-236, 1997.

[2] Arpaci-Dusseau A.C., Culler D.E., Mainwaring A.M., "Scheduling with Implicit Information in Distributed Systems", in *Proc. of the 1998 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pp 233-243, 1998.

[3] Chiang S.-H., Mansharamani R.K., Vernon M.K., "Use of application characteristics and limited preemption for run-to-completion parallel processor scheduling policies," *Proc. of the 1994 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pp 33-44, 1994.

[4] Cray Research, Document Number IN-2153 2/97.

[5] Czajkowski K., Foster I., Karonis N., Kesselman C., Martin S., Smith W., Tuecke S., "A Resource Management Architecture for Metacomputing Systems," *Job Scheduling Strategies for Parallel Processing*, Springer-Verlag, Lecture Notes in Computer Science, Vol 1459, pp 62-82, 1998.

[6] Feitelson D.G., "A Survey of Scheduling in Multiprogrammed Parallel Systems", Research Report RC 19790 (87657), IBM T.J. Watson Research Center, Oct. 1994, revised version August 1997.

[7] Feitelson D.G., Rudolph L., (Eds.), *Job Scheduling Strategies for Parallel Processing*, Springer-Verlag, Lecture Notes in Computer Science, Vol 949, 1995.

[8] Feitelson D.G., Rudolph L., (Eds.), *Job Scheduling Strategies for Parallel Processing*, Springer-Verlag, Lecture Notes in Computer Science, Vol 1162, 1996.

[9] Feitelson D.G., Rudolph L., (Eds.), *Job Scheduling Strategies for Parallel Processing*, Springer-Verlag, Lecture Notes in Computer Science, Vol 1291, 1997.

[10] Feitelson D.G., Rudolph L., (Eds.), *Job Scheduling Strategies for Parallel Processing*, Springer-Verlag, Lecture Notes in Computer Science, Vol 1459, 1998.

[11] Foster I., Kessellman C., "Globus: a metacomputing infrastructure toolkit," *International Journal of Supercomputer Applications*, Vol 11(2), pp 115-128, 1997.

[12] Ghafoor A. , Yang J., "A Distributed Heterogeneous Supercomputing Management System," *IEEE Computer*, Vol 26(6), pp 78-86, June 1993.

[13] Ghosal D., Serazzi G., Tripathi S.K., "Processor working set and its use in scheduling multiprocessor systems", *IEEE Trans. Software Eng.*, Vol 17(5), pp 443-453, May 1991.

[14] Grimshaw A.S., Wulf W.A., and the Legion team, "The Legion Vision of a Worldwide Virtual Computer," *Communications of the ACM*, Vol 40(1), pp 39-45, 1997.

[15] Henderson R., Tweten D., "Portable Batch System: external reference specification," Tech. Report, NASA Ames Research Center, 1996.

[16] Islam N., Prodromidis A., Squillante M.S., Fong L.L., Gopal A.S., "Extensible resource management for cluster computing", in *Proc. of the International Conference on Distributed Computer Systems*, 1997.

[17] Litzkow M., Livny M., Mutka M., "Condor - A Hunter of Idle Workstations," *Proc. 8th Intl. Conf. on Distributed Computing Systems*, pp 104-111, 1988.

[18] McCann C., Vaswani R., Zahorjan J., "A dynamic processor allocation policy for multi-programmed shared memory multiprocessors", *ACM Transactions on Computer Systems*, Vol 11(2), pp 146-178, May 1993.

[19] Neuman B.C., and Rao S., "The Prospero resource manager: A scalable framework for processor allocation in distributed systems," *Concurrency: Practice and Experience*, Vol 4(6), pp 339-355, 1994.

[20] Ousterhout J., "Scheduling techniques for concurrent systems," *Proc. 3rd. International Conference on Distributed Computing Systems*, pp 22-30, October 1982.

[21] Parsons E.W., Sevcik K.C., "Coordinated Allocation of Memory and Processors in Multiprocessors", in *Proc. of the 1996 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pp 57-67, 1996.

[22] Peris V.G., Squillante M.S., and Naik V.K., "Analysis of the impact of memory in distributed parallel processing systems", in *Proceedings of the 1994 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pp 5-18, 1994.

[23] Rosti E., Serazzi G., Smirni E., Squillante M.S., "The Impact of I/O on Program Behavior and Parallel Scheduling", in Proc. of the Joint *ACM SIGMETRICS'98 Conference on the Measurement and Modeling of Computer Systems* and *Performance'98*, pp 57-65, 1998.

[24] Rosti E., Smirni E., Dowdy L.W., Serazzi G., Carlson B.M., "Robust partitioning policies for multiprocessor systems," *Performance Evaluation*, Vol 19(2-3), Special Issue on Parallel Systems, pp 141-165, 1994.

[25] Setia, S., "The Interaction between Memory Allocation and Adaptive Partitioning in Message-Passing Multicomputers," in Feitelson D.G., Rudolph L., (Eds.), *Job Scheduling Strategies for Parallel Processing*, Springer-Verlag, Lecture Notes in Computer Science, Vol 949, pp 146-164, 1995.

[26] Sevcik K.C., "Characterization of Parallelism in Applications and their Use in Scheduling," in *Proceedings of the 1989 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pp 171-180, 1989.

[27] Smirni E., Rosti E., Dowdy L.W., Serazzi G., "A Methodology for the Evaluation of Multiprocessor Non-Preemptive Allocation Policies," *Journal of Systems Architecture*, Vol 44(9-10), pp 703-721, 1998.

[28] Sobalvarro P.G., Weihl W.E., "Demand-Based Coscheduling of Parallel Jobs on Multiprogrammed Multiprocessors" in Feitelson D.G., Rudolph L., (Eds.), *Job Scheduling Strategies for Parallel Processing*, Springer-Verlag, Lecture Notes in Computer Science, Vol 949, pp 106-126, 1995.

[29] Squillante M.S., "On the Benefits and Limitations of Dynamic Partitioning in Parallel Computer Systems" in Feitelson D.G., Rudolph L., (Eds.), *Job Scheduling Strategies for Parallel Processing*, Springer-Verlag, Lecture Notes in Computer Science, Vol 949, pp 219-238, 1995.

[30] Weissman J., "Gallop: The benefits of wide-area computing for parallel processing," Tech. Report, University of Texas at San Antonio, 1997.

[31] Zahorjan J., McCann C., "Processor Scheduling in Shared Memory Multiprocessors" in *Proc. of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pp 214-225, 1990.