New Challenges of Parallel Job Scheduling

Eitan Frachtenberg[1] and Uwe Schwiegelshohn[2]

[1] Powerset, Inc.
eitan@powerset.com
[2] University Dortmund
uwe.schwiegelshohn@udo.edu

Abstract. The workshop on job scheduling strategies for parallel processing (JSSPP) studies the myriad aspects of managing resources on parallel and distributed computers. These studies typically focus on large-scale computing environments, where allocation and management of computing resources present numerous challenges. Traditionally, such systems consisted of massively parallel supercomputers, or more recently, large clusters of commodity processor nodes. These systems are characterized by architectures that are largely homogeneous and workloads that are dominated by both computation and communication-intensive applications. Indeed, the large majority of the articles in the first ten JSSPP workshops dealt with such systems and addressed issues such as queuing systems and supercomputer workloads.
In this paper, we discuss some of the recent developments in parallel computing technologies that depart from this traditional domain of problems. In particular, we identify several recent and influential technologies that could have a significant impact on the future of research on parallel scheduling. We discuss some of the more specific research challenges that these technologies introduce to the JSSPP community, and propose to enhance the scope of future JSSPP workshops to include these topics.

1 Introduction

The last few years have brought about many radical changes in the technologies and platforms that exhibit the same salient challenges that JSSPP focuses on—all in the family of flexible allocation and management of multiple computer resources. These technologies, however, depart from the traditional supercomputer model of relatively homogeneous architectures and applications, and add new dimensions to those that are already being studied within JSSPP's scope. This paper therefore has two main goals: (1) To present some of the major technological changes and to discuss the additional dimensions they add to the set of JSSPP challenges; and, (2) to promote and suggest research topics inspired by these dimensions in the JSSPP community.

Such dimensions include, for example, reliability and resource allocation across multiple sites (Grids), workloads that are a mixture of parallel, sequential, and interactive applications on multi-core desktops, and data-intensive applications on Web servers that exhibit little or no communication. Although the

traditional topics of interest of JSSPP are still relevant and will likely continue to attract high quality papers to the workshop, we feel the need to introduce these new topics to JSSPP's scope at this time, for two main reasons:

1. The field of parallel job scheduling, while still evolving, is showing signs of maturity.
2. New technologies are exhibiting many characteristics of the problems that the JSSPP community tackles. We believe that the JSSPP community's expertise can produce meaningful contributions for these technologies.

By introducing these new but related topics to the scope of JSSPP, we hope to expand its impact and attractiveness to researchers with little or no past exposure to traditional parallel job scheduling. After discussing these topics with our peers, we present here a nonexhaustive list of research topics and questions to which we believe the JSSPP community can add significant contributions.

Paper organization We have loosely grouped these topics into four technological trends:

- Commodity parallel computers (Section 2): Parallel environments mainly consisting of desktops and laptops with multi-core chips
- Grids (Section 3): Large-scale, heterogeneous, distributed, partially shared computing environments
- Web servers (Section 4): Large-scale, latency-sensitive online services, and the offline data infrastructure behind it
- Virtualization (Section: 5): Resource management inside and among multiple virtual machines

These categories are not sorted, and in fact, under the umbrella of parallel scheduling, have much more in common with each other than what sets them apart. Section 6 discusses the similarities and overarching scheduling considerations that affect most or all of these contemporary technologies. Finally, Section 7 concludes the paper.

2 Commodity Parallel Computers

The largest shift towards parallel computing is actually occurring right now. A large majority of the desktop and notebook computers sold today for everyday use employs dual-core and quad-core chips. Several server, console, and special-purpose processors even contain between 8 and 96 cores, and the trend to increase on-chip parallelism is expected to continue in the foreseeable future [20].

As MIT's Leiserson writes: [41]

> The Age of Serial Computing is over. With the advent of multi-core processors, parallel-computing technology that was once relegated to universities and research labs is now emerging as mainstream.

Commodity hardware is growing increasingly more complex, with advances such as chip heterogeneity and specialization, deeper memory hierarchies, fine-grain power management, and most importantly, chip parallelism. Similarly, commodity software and workloads are becoming more concurrent and diverse, encompassing spreadsheets, content creation and presentation, 3D games, and computationally intensive business and scientific programs, among others. With this new complexity in hardware and software, process scheduling in the operating system (OS) becomes more challenging. Nevertheless, many commodity OS schedulers are based on design principles that are 30 years old [20]. This disparity may soon lead to significant performance degradation. Particularly, modern parallel architectures such as multi-core chips require more than scalable OSs: parallel programs need parallel-aware scheduling [21]. Although the effort to produce more scalable scheduling is already producing some results (both in task scheduling [41] and process scheduling [24]), there is still much research and implementation work needed before commodity parallel computing fulfills its performance promises.

Scheduling for a mixed workload The arrival of ubiquitous parallel hardware leads to complex workloads with complex scheduling requirements, especially as software becomes increasingly more parallel. Although the transition to parallel software on the desktop is not immediate, it is already taking place. Popular programming languages such as C++ and Java offer increasingly sophisticated support for parallel programming [41], while the emerging parallel hardware creates a stronger incentive for concurrency. Some contemporary applications already benefit from parallel computing power, for example, parallel searches in terabytes of data, 3D games, photo and video editing filters, and technical computing in science and industry. Consequently, the typical desktop workload, already highly variable and difficult to characterize, becomes even more complex as parallel desktop applications grow in number. The scheduler designer must now contend with an unpredictable mix of conflicting scheduling requirements, such as:

- Media applications that require few resources, but at precise intervals and with strict process inter-dependencies.
- Parallel applications that need synchronization and/or co-scheduling.
- Low-priority background tasks such as virus scanning.
- Interactive applications that require high responsiveness, like web browsers or word processors.

Taking a more general view, we observe that parallelization poses two principal challenges to the commodity scheduler: (1) processes competing over resources suffer from degraded performance when coscheduled, and (2) collaborating processes suffer from degraded performance when *not* coscheduled. To design effective schedulers for such mixed workload, we must first understand the workloads. The job scheduling community has a rich history in characterizing and modeling supercomputer workloads, and can employ some of the experiences and techniques from that effort to similarly describe parallel desktop workloads.

Priorities Within a mixed workload we must often deal with priorities, fairness, and user experience. Unlike a large computing center handling multiple users with conflicting priorities and with economic constraints desktops typically have a single user with his attention primarily focused on a single application at a time. This additional constraint on the scheduler requires that it does its best effort in guessing temporal user priorities and prioritizing processes accordingly without sacrificing other scheduling goals. The dynamic nature of personal computer usage makes this goal more difficult, because it requires the scheduler to respond quickly to changes in user attention. This challenge again is not a new one [12], but the introduction of parallel resources and synchronized parallel programs complicates it beyond the scope of most existing solutions.

Power management Power consumption in chips is one of the largest challenges faced by chip manufacturers today and has itself instigated the arrival of the multi-core processors. Since the temperature and power of a chip is directly related to the chip's power consumption, the chip clock's speed is limited by its operational thermal envelope. To further minimize and control the power output, modern chips can selectively turn off and/or throttle down the speed of unused logic. Increasingly, this fine-grain power control functionality is exposed to the OS. This is where scheduling can play a vital role in managing the trade-offs between performance and power consumption. Although some initial work has already shown that scheduling can have a significant impact on power consumption, there are as of yet no mainstream desktop schedulers that explicitly try to optimize this factor. As the number of cores increases in the future, and as the granularity control that the hardware exports to the OS grows finer, this research area will likely grow more complex and challenging. On the other hand, the rising importance of power management will also make the fruits of this research more rewarding.

Asymmetric cores heterogeneity Merely duplicating cores in a chip is not always a sufficient solution to the ever-increasing demand for performance. Limiting factors such as power budget, cooling capacity, and memory performance will still require innovative design solutions such as heterogeneous cores with selective shutdown, use of specialized coprocessors, and moving computation closer to memory. The highly popular Cell processor, for example, comprises nine cores of two different designs and purposes [27]. Other emerging architectures include a relatively large number of special-purpose computing cores, such as the ClearSpeed 96-core chip for mathematical processing, and the Azul 24-core chip for Java applications [31,46]. Memory hierarchies are also growing more complex due to the use of multi-core and hyper-threaded chips. Such computers are essentially nonuniform memory access (NUMA) machines, and as such, may impose special scheduling requirements [6]. On another end of the computing spectrum, ubiquitous computers, such as mobile phones, portable music and video players, and media-convergence appliances that have strict minimum service requirements on a low-power, low-performance platform could further stress the resource management requirements.

These architectures will require OS support to allocate their resources intelligently. There is already an ongoing effort to hand tune applications and schedulers to specific architectures [54], but this effort may not be enough. Heterogeneous chip environments are challenging to schedule on, but also present many opportunities for a scheduler that can map tasks to different chip components appropriately.

Generalizing the Challenges

We noted that the introduction of parallel applications to the desktop workload challenges the commodity parallel scheduler with potentially conflicting requirements. Moreover, architectural changes also produce additional constraints for scheduling, such as heterogeneity/asymmetry, power management, and NUMA. But unlike classical parallel computers, the presence of various classes of applications in a single workload mix—including interactive and single-threaded applications—poses a significant additional challenge on top of the specific application requirements. Ignoring these scheduling constraints can lead to poor application performance because of lack of synchronization, as well as poor system-wide performance because of contention for resources [1,14,30,49]. Future research on parallel commodity computing must take these factors into account.

Most contemporary commodity schedulers are challenged at all levels of parallel execution, from the thread [6,49], through the SMP [1,16], the cluster [14,23], all the way to supercomputers [30]. In particular, parallel programs may suffer tremendously from lack of coscheduling[3] [14,30] as processes in parallel programs—as opposed to sequential and distributed programs—rely on frequent synchronization for their progress. Supercomputers, with a more uniform workload of parallel applications, typically operate in batch mode [15]. For commodity computers and workstations that host a multiuser, time-sharing system, this is not an acceptable solution [36]. We believe however that effective scheduling for a mixed workload is not only necessary, but also within reach, and could incorporate lessons learned from scheduling parallel jobs on large homogeneous systems.

Better scheduling is achieved when the OS has intimate understanding of the hardware's capabilities and the software's requirements. With regard to hardware, the OS should arbitrate between multiple and possibly heterogeneous resources, while considering cache and memory-pressure factors. With regard to applications, the OS needs to be cognizant of all levels of parallel execution: thread, process, and parallel program, in addition to sequential and interactive programs. Schedulers can manage these workloads by applying principles from such fields as parallel and multimedia scheduling. Particularly, cooperation, adaptivity, and classification can play a decisive role in achieving optimal user experience and utilization on next-generation computers.

[3] Coscheduling refers to scheduling all of a job's processes at the same time, to facilitate synchronization [37].

3 Grids

Most supercomputer workloads contain a large number of sequential applications or applications with little parallelism [38]. With few exceptions due to exhibitive memory consumption, most of these applications can run as well on common desktop or server systems not requiring a parallel machine with its expensive internal network. Therefore, executing these jobs on supercomputers is not efficient in general although a limited number of them is welcome as they do not affect the completion time of parallel jobs but increase utilization of the machine. The actual number depends on the characteristics of the workload. Many parallel applications can exploit different degrees of parallelism, that is, they are malleable or moldable [15]. In many of these cases, it is acceptable to forgo the maximum degree of parallelism if a system with fewer nodes is readily available. Therefore, it is often more efficient to acquire a Grid of machines with different numbers of processors instead of investing a significant larger amount of money into a big supercomputer with the same total number of processors. This is one reason for the increasing popularity of Grids [18]. Moreover, it is often difficult for user groups of a small enterprise to fully utilize a large parallel machine with their own applications. However, if many of those user groups share several parallel machines an improved average utilization can be achieved. As there are many different resource owners in a Grid, every such Grid represents a market in which different application owners compete for processors or other resources of different providers. On the one hand this is likely to lead to some form of bidding system [11]. On the other hand it increases the dynamics of machine availability as many resource owners may additionally have high priority local users.

Unfortunately, job scheduling on Grids is significantly more difficult than job scheduling on single parallel machines. The most obvious reason is the separation of the scheduling problem into two interdependent problems:

- machine allocation
- scheduling on the selected parallel processor

This problem separation is known from many parallel job scheduling problems [39] but it becomes more complicated in the presence of rigid parallel jobs without multisite scheduling [51]. In addition, some properties of Grids also influence job scheduling. We discuss these properties and their consequences to job scheduling in the following paragraphs.

Heterogeneity Manufacturers offer a large number of configurations for their supercomputers. This may include different types of nodes, like single processors or SMP nodes, differences in the network connection, or different amounts of memory. But in practice, most supercomputer installations have only a few nodes that are specially equipped, (for example, to execute server tasks), while almost all worker nodes are identical [28]. As the supercomputer has a single owner, the node equipment is governed by a single policy resulting in similar hardware and

mostly identical software on each node. Despite the rapid development in processor technology, few institutions abstain from mixing new and old processors within the same machine. Instead they rather invest in a new machine if the performance of the old one is no longer sufficient.

From the perspective of scheduling, supercomputers therefore exhibit little node heterogeneity. Hence, most job scheduling research on supercomputers assumes homogeneous nodes. But of course, heterogeneity exists among the various resources available at a single node, like processing power, memory, or network bandwidth [50].

In Grids, the situation changes completely: Since a Grid comprises different installations with different owners, there is a large amount of heterogeneity in Grids. The individual computers are typically not installed at the same time, resulting in the use of different processor technology. Moreover, the various machine owners in a Grid have different objectives when buying their computers, leading to different hardware and software equipment in the nodes of different machines in the Grid. Finally, the network performance within a machine is usually much better than the network performance between different machines. This characteristic of Grids particularly affects so called multisite jobs [9] that are executed on several machines in parallel. But as in practice, the performance of these multisite jobs is bad in comparison to single site execution [3] and they only occur rarely.

The heterogeneity within a Grid is one of the main advantages of Grid technology, since installations with many users (such as large compute centers in universities) can never satisfy all users when selecting the next machine to buy. Grid technology allows users to look within the Grid for the machines that are best suited to execute their jobs. This selection increases overall efficiency, since applications that perform poorly on local machines can be forwarded to other better-suited machines, possibly in exchange for other applications. On the one hand, there are system properties that are mandatory for the execution of an application, like the availability of a certain software. Clearly, the scheduler can easily consider these constraints. On the other hand, an application may run best on certain processors which are in high demand while other readily available processors will result in a reduced performance. In such situations, it is difficult for the scheduler to make an allocation decision, since critical job information like the execution time on the available machines may only be partially available.

Further, the above mentioned heterogeneity in supercomputers also exists in Grids: A Grid often comprises resources of different types like storage resources, computing installations and networks connecting the other resources. Therefore, Grids are, for instance, well suited for applications analyzing large amounts of data, like evaluations of experiments in particle physics. For reasons of cost and efficiency, experiment data are stored in large data centers that are specially equipped with hardware. An application requires the transfer of these data to an appropriate computing facility. Therefore, the execution of such an application consists of a workflow with several stages [19]. While most scheduling problems on parallel processors deal with independent jobs, Grid scheduling

uses precedence constraints and scheduling routes through different resources effectively transforming job scheduling problems into a kind of job shop scheduling problems [39]. This property of Grid scheduling problems directly influences the scheduling techniques on parallel processors: For instance, to consider the processing time of the data transfer, computer resources must be reserved in advance. Hence, simple batch job scheduling is not sufficient anymore, and most Grid schedulers support *advance reservation* [48].

Because of the existence of different resource owners in a Grid, the Grid scheduler is not run by these resource owners as in the supercomputer case, but rather by an independent broker. This Grid scheduler may then interact with the local schedulers that are run on each machine and also support local users that do not submit their jobs via the Grid [45].

Service Level Agreements A supercomputer typically has a single owner and is governed by a single policy that determines the rules and constraints of scheduling. The users must accept these rules unless they are able to manually submit their applications to other supercomputers. However, this alternative does not affect the scheduling process. In Grids, there are often several independent owners which have established different rules and restrictions for their resources. Note that this assertion may not be true for so-called Enterprise Grids [26], that belong to a single enterprise with many different locations and resource installations. But if the Grid comprises similar resources from different owners, users expect that the machine allocation decision considers the rules and policies of the various owners. Especially when being charged for the resource usage, users want the allocation and schedule properties of their applications to be guaranteed in form of so-called *service level agreements* (SLA) [34].

In addition to static components, like the level of security, those service level agreements typically contain various dynamic and job-specific properties, like the amount of available resources within a time frame, the start time of a time frame, and the cost of the resource per occupied time unit. The actual values of these SLA components are closely related to the actual schedule. They may depend on the amount and the type of job requests as well as on the amount and type of available resources. Dynamic parameters of an agreement are typically determined with the help of a negotiation process [58]. Therefore, Grid scheduling may also include a negotiation component [32]. Moreover, global scheduling objectives, like makespan, average utilization, average throughput, or average (weighted) response time have a different meaning and relevance in a scenario that involves independent resources providers and independent job owners. For instance, if the utilization of a specific machine in the Grid is low then the owner of this machine may decide to drop the resource price in order to attract more applications.

A Grid scheduling system that supports SLAs must include processes that automatically generate SLAs based on possibly complex directives of owners and users [52]. It must also be able to support complex objective functions in order to decide between different offers for a job request. Therefore, even if we ignore the machine allocation problem, the local machine scheduling becomes significantly

more complex than the scheduling of a single parallel processor. Assuming that in the future, many parallel processors will also be part of a Grid, the Grid poses new challenges even for job scheduling on parallel processors. Also if an independent broker runs the Grid scheduler he may define additional SLAs with resource providers and job owners.

Finally, there is always the possibility that an SLA cannot be satisfied. If such a problem is foreseeable and there is still enough time to react then some form of rescheduling [4] may provide some help. These considerations will again influence the Grid scheduler. However, if the violation of the SLA is only determined after the execution of the job, such as too few resources were provided within the promised time frame, then either the SLA contains some clause to handle this problem, or a mediator in the Grid is needed. In this case however, the Grid scheduler is not affected unless we speak of an SLA that covers the actual scheduling process.

Accounting and Billing Since traditional supercomputers typically have a single owner, their accounting and billing is rather simple. It is sufficient to log the job requests and the actual resource utilization of the jobs. As the rate is typically invariable, the cost can easily be determined. Because of the monopoly of the machine provider, a best-effort strategy usually suffices. Therefore, the user has few options if the resources are suddenly not available or other problems occur.

However, in a Grid environment, resource providers may be willing to provide guarantees that are marked down in an SLA. To verify whether the conditions of the SLA have been satisfied, the entire process from job request submission to the delivery of the results must be recorded [40]. Therefore, accounting becomes more complicated than for isolated parallel processors. Similarly, billing is not equivalent with multiplying a fixed rate with the actual resource consumption but requires the considerations of the SLAs including possible penalties for violating parts of the agreement. The Grid scheduling system is part of the above-mentioned process. Therefore, a scheduling system must be transparent to enable validating the correct execution of an agreement.

As already mentioned, a broker [53] in a Grid system may be bound by agreements with resource providers and job owners. This is especially true if several brokers compete with each other in a single Grid. Then the details of the scheduling process must be recorded to determine whether the guarantees of the SLA covering the scheduler have been satisfied.

Security There are significant security concerns in a Grid as an application of a user may run on a distant resource. Most resource policies require some form of user screening before a user is admitted to a resource. In case of a local compute center with a few carefully selected remote users, this policy can be enforced with relatively little effort. In Grids, this policy is not feasible and must be replaced by some form of trust delegation. This task is often handled by so-called virtual organizations (*VO*) [17]. Although security has a significant impact on the Grid infrastructure it does not affect the scheduling system to a large extent. But security concerns may prevent schedulers from providing

information about future schedules freely, and thus reduce the efficiency of Grid schedulers. This problem is particularly relevant for rearrangement tasks that try to save an SLA in case of unexpected problems.

Reliability and Fault Tolerance In large systems, occasional failures are unavoidable. If the system is subject to a best-effort policy such a failure is a nuisance to the users but has no other consequences. For important applications, users can try to secure a second resource if they accept the additional cost. In Grids, users may try to push the responsibility toward the resource providers by negotiating appropriate SLAs. In case of a failure, the resource provider typically attempts to use rescheduling in order to avoid or at least reduce the penalty costs. Therefore, reliability directly influences Grid scheduling as well [29].

Virtualization As already discussed user may benefit from the heterogeneity of Grid systems. However, this heterogeneity also comes with a disadvantage: Only few systems in a Grid may actually be able to execute a given application due to all the constraints involving application software, system software and hardware. This may lead to bottlenecks even in large Grids. Users can partially avoid this problem by *virtualization*, that is, by providing an execution environment together with their application, see Section 5. This concept receives increasing interest on the operating system level and is recently considered in Grids as well. As with security, virtualization has little direct influence on Grid scheduling. It opens new scheduling opportunities (as discussed in Section 5), but predictions of execution parameters become less reliable. However, virtualization directly affects scheduling on the operating system level.

Workloads From supercomputers, we know that only few theoretical results provide benefits for job schedulers in real systems [44]. Instead, the development of new schedulers for parallel processors and the improvement of given schedulers is often based on discrete event simulations with recorded workloads as presented in numerous publications of previous JSSPP workshops, for instance [25,5]. Therefore, a lot of work in the domain of JSSPP has been devoted to workloads in recent years. As there is only a limited number of recorded traces, this work focuses on the characterization of these workloads and on scaling them so that an appropriate workload can be provided for a new installation [10].

Since there are few Grids running in production mode, only few real Grid workloads are available yet [35]. Nevertheless, there is some effort to record Grid workloads (see `http://gwa.ewi.tudelft.nl`), which may lead to a comprehensive archive in the future. However, it is likely that these workloads will depend to a large extend on the community running the Grid. Generally it is very difficult to optimize a Community Grid by using a workload from another Community Grid. Similarly, it is unclear how to scale traces as there may be strong dependencies between the machine composition in a Grid and the workload.

In general, simulation with workloads are only meaningful if the properties of the individual jobs remain invariant. To a large extend, this is true for job execution in a rigid and exclusive fashion on a parallel processor. If the job is

malleable or moldable a simulation requires the prediction of the processing time for a certain degree of parallelism from the recorded processing time using the original degree of parallelism. Theoretical studies often assume a divisible load characteristic which holds in practice only for bag-of-tasks of embarrassingly-parallel jobs, while jobs with extensive communication between processors show a different behavior [42]. In Grids, similar problems occur in connection with heterogeneity. It is difficult to predict the processing time of a job on certain processors if only the recorded processing time on other processors is available. The different speed model (Q_m) of scheduling theory generally does not apply to processors, see the results of the SPEC benchmarks (`http://www.spec.org/benchmarks.html`). Therefore, it is very difficult to optimize a Grid scheduler even on the same system that was already used to record the applied workload.

Metrics and Evaluation Since parallel computers are expensive, they are usually not available for extensive experiments to optimize system components. Instead, simulations are frequently applied on models that have a sufficiently close relationship to the real system. Then only some final tuning must be performed on the real system. As already discussed, this approach has been successfully executed on parallel computers. It requires a given metric that can be evaluated with the help of simulations. For parallel systems, the most common metrics are accepted utilization, average throughput, and average weighted response time [15]. All these metrics depend on the completion time of the jobs which is provided by the simulations.

In Grids, we must consider the various (dynamic) objectives of resource providers and application owners. Therefore, scheduling becomes a multi-objective problem since it is very difficult to combine these objectives into a single scalar metric [57]. Moreover, as the objectives are not static, it is not possible to simply evaluate another schedule unless the objective functions and the negotiation process are invariant and predictable. However, this assumption will not hold in many real situations. Hence, it is not realistic to assume that a real Grid scheduling system can be optimized with the help of simulations even if appropriate workloads and sufficient computing power is available. In Grid scheduling, we face a problem that is similar to the optimization of the performance of a stock broker. But while the Grid is some form of a market it is likely less volatile than the stock market. We may therefore try to optimize single parts of this complex scheduling system and assume that the rest remains unchanged. Once enough workloads and sufficient data on the dependencies between the various components are available, we may start to model and simulate a whole system.

Generalizing the Challenges

Grid scheduling is a very complex problem that uses common job schedulers for parallel processors as subcomponents. Even if job scheduling for parallel processors has reached some degree of maturity, many subproblems in Grid scheduling are not yet solved.

Grids typically consist of machines with different numbers of processors. As small machines with few processors cannot efficiently execute highly parallel jobs unless multisite scheduling with performance loss is supported, large machines with many processors should be reserved for those highly parallel jobs. On the other hand, load balancing may require to use those large machines also for sequential jobs or jobs with little parallelism. The machine allocation algorithm must find a suitable tradeoff between both objectives.

Heterogeneity transforms job scheduling problems into job shop problems. In addition, the comparison between different schedules may become rather difficult as the prediction of execution properties on other machines is subject to a significant amount of uncertainty.

Service level agreements introduce new dynamic objectives into the scheduling problems resulting in multi-objective problems. Moreover, the objective functions of the various job and resource owners may not be available for the evaluation of a scheduling system. In order to satisfy SLAs even in the case of machine failure, the scheduling system should support rescheduling which can be considered as a deterministic problem that must be solved within a rather short time frame.

Other properties of Grid systems, like security or virtualization, pose significant challenges to Grid infrastructures but have limited influence on the scheduling system.

Finally, there are not yet enough public workloads traces on Grid systems. It is also not clear how to use such workloads in new systems with different sizes or on systems which belong to a different community. With respect to the evaluation, the common metrics of parallel processors may not be applicable to Grids. But it is not clear how to determine an objective that can be used for evaluation and sufficiently represents the multi-objective character of the real Grid scheduling problem.

4 Web Services

Large-scale web services are one of the fastest-growing sectors of the computer industry since the mid 1990s. This growth is expressed not only in revenue and market share, but also in the scale of the problems solved and the infrastructure required to provide the solutions. Generally speaking, large-scale web services have three usage models with strong relevance to our field:

1. Online service–this is the part that is most visible to users, where they interact with the system through queries or requests. This aspect is latency-sensitive, and typically relies on parallelism to provide the shortest response time and the highest reliability. Much of the parallel logic behind these large-scale transactional systems is devoted to resilient resource management and load balancing, and less to computation. Using a search engine as an example, the online service represents the user query page, where a query is received, parsed, and distributed to query servers, and the results are aggregated, ranked, and presented to the user in HTML form.

2. Offline processing–this is the part that gathers and processes the data that is used in the online service. It is typically less sensitive to latency and more sensitive to throughput, not unlike the Grids mentioned in Section 3. Load balancing and fault tolerance play a larger role in the economics of the service than in the online service. Additionally, the offline processing can potentially be significantly more reliant on computing and I/O resources than the online service. These differences translate to different scheduling and resource-management requirements between the online and offline parts. In the search engine example, this part represents the crawling, indexing, reversing the search engine index as well merging, and distributing it.
3. Research and Development (R&D)–large web service companies are always looking for ways to improve and expand their services. Developing newer services and features often requires similar resources to those that are already used by the production services, whether online or offline, and for large companies, the scale of the resources required for R&D approximates the scale of the production systems. Unlike the production systems though, resource management can be more lax on the one hand (neither latency or throughput is as critical as on production systems), and more strained on the other (more users are competing for the same resources in a less-predictable environment). Going back to the search engine environment, this part represents the ongoing work on improving crawling algorithms, ranking, database/index representations, and performance tuning to mention just a few aspects.

The following paragraphs give a breakdown of some of the main resource management challenges in large-scale web services.

Economy Because of the large scale of some web serving farms and the business nature of the companies that run them, economical issues become a primary consideration in web server resource management. For example, choices such as how requests are distributed and balanced across a cluster, the degree of redundancy in request execution, and which nodes to route the request to in a heterogeneous cluster, have an effect not only on the cost per request, but also on the server's reliability and responsiveness, themselves being part of the company's business model. Thus, the algorithms used for balancing and managing these resources can have a significant impact on the company's bottom line. Resource managers have to respond to temporal cycles in load, as well as peak and average load, while aiming to minimize overall costs of hardware, power, and human maintenance [7]. Complicating the economical models further are the potential large differences between different services and the resources they need to manage, making a generalized solution hard to develop.

Power management The rising concern about power consumption in microprocessors (Sec. 2) has permeated virtually all systems where microprocessors are used. Multiply the power consumption of a single microprocessor by the thousands of microprocessors that typically comprise a large web serving farm, and you get an expensive power bill and significant excessive heat that further taxes the reliability and economic balance of the farm. The larger the

scale of the server, the worse the problem becomes, as is demonstrated by Google's move to a more efficient power supply component of their own design (see `http://services.google.com/blog_resources/PSU_white_paper.pdf`). Scheduling, however, can play a significant role in increasing the power efficiency of a large-scale farm [7]. For example, outside of peak hours, jobs can be all scheduled on a subset of nodes, while suspending the remaining load until peak increases. Or jobs can be distributed across multi-core nodes so that some cores remain in low-power idle mode until requested.

Resource sharing Despite the occasional peaks in request loads to web services, most load exhibits cycles and peaks, based on diurnal cycles, weekly cycles, external events, etc. [59]. Web server loads can be hard to predict, and there is still plenty of room for research in characterizing and modeling their workloads. Still, even with a complete understanding of the workloads, it is reasonable to assume that most servers will operate below capacity some of the time. It is desirable to manage the shared resources of the online system with the R&D activities, so that lightly-loaded production machines can run R&D tasks, while still being highly available if load suddenly increases.[4] Oversubscribing online resources to handle R&D tasks touches many interesting and familiar topics in parallel job scheduling such as load balancing, managing priorities, service guarantees, predictability and modeling of load, and dynamic management of responsiveness.

Resilience and fault tolerance If there is one working assumption that holds true for large data stores and web servers it is "Everything Fails. Everything!"[5]. Many thousands of commodity components typically comprise a large web service, and as their number increase, so does the probability of a component failure at any given time. Resource management algorithms therefore cannot have the luxury of dedicated supercomputer middleware that often assumes a reasonably reliable hardware. Redundancy, fault-tolerance, and graceful degradation under load and failures must be built into the resource manager. One example is the offline work distribution algorithm MapReduce [8], that can transparently and scalably replicate tasks and re-execute them if required. Online resource management requires different scheduling for resilience, since service latencies are more important than throughput. As servers grow even larger and the services grow more complex, the importance of fault tolerance will similarly grow and require novel resource management solutions.

Heterogeneity One of the distinguishing characteristics of large-scale web servers, as opposed to most supercomputer and to a lesser extent, Grid environments,

[4] We assume that offline production systems have a more predictable load and operate at near capacity most of the time.

[5] Quoted Sivasubramanian and Vogels' talk "Challenges in Building an Infinitely Scalable Datastore" in the 2007 Google Scalability Conference `http://www.google.com/events/scalability_seattle/`.

is that server farms are rarely acquired at once to serve a predetermined capacity. Instead, servers are expected to constantly grow as the required capacity increases over time. Web servers grow by adding nodes that correspond to the optimal balance between price and required performance at the time, and because of the dynamic nature of the industry, these nodes are likely to differ from the previous acquisition or the next one. Consequently, a large-scale web server consists of at least a few different types of nodes–possibly with varying degrees of performance, memory, storage space, I/O capabilities, or all at once. Dynamically allocating tasks to these servers has to take into account this heterogeneity in order to meet the expected performance requirements. The evolving nature of the cluster, as well as the constant changes in configuration resulting from node failures, suggest that a very dynamic approach to resource management is needed, as opposed to most supercomputers and Grids. Although here too some initial studies have addressed these issues [7], there is still much room for research in the area.

Workload characterization The workloads of the online and offline environments are typically quite different from each other, as well as from traditional supercomputer workloads. Perhaps the most significant difference from supercomputer workloads is that Web-server workloads tend to be embarrassingly parallel: loosely-coupled, with little or no synchronization between parallel tasks. This removes an important constraint that facilitates the development of efficient scheduling. On the other hand, other workload characteristics make scheduling more challenging than with supercomputer workloads. For example, both the offline and online systems are often data-bound and require access to information that is distributed across the compute nodes. Scheduling tasks to compute close to the data they operate on reaps a significant performance benefit in these cases. Other idiosyncratic workload characteristics include the dynamic nature of offered load on the online system, which is largely determined by uncontrolled agents outside of the system (the users). Devising better scheduling for the environments requires that we understand and characterize workloads for the online and offline parts of the web servers, as we do for more traditional parallel environments.[6]

Generalizing the Challenges

Probably the single most challenging and representative factor in large-scale web service scheduling is the unprecedented huge scale of most aspects involved with it: the size of the clusters; the number of users; the rate of transactions; the amount of data, files, and I/O required to service these requests; and the network resources used internally and externally. Scheduling and resource management are further complicated by the dynamic nature of the underlying infrastructure, with heterogeneous resources being added and removed constantly. Because of

[6] The workload of the R&D environment usually consists of a mix of online and offline applications, and is probably even harder to generalize.

these factors, as well as the different workloads, scheduling for web services might require some different approaches, compared to Grids or supercomputers. For example, the already mentioned MapReduce algorithm introduced by Google [8] does a good job in managing dynamic resources for the offline aspect of Google's search engine, but would perform poorly with the typical fine-grain, tightly coupled supercomputer workload. Nevertheless, many scheduling principles, as well as work on metrics, workloads, and methodological issues, have much in common with other parallel environments.

5 Virtualization

Virtualization in this context refers to running multiple operating system environments in one or more nodes concurrently. Typically, a node would have a host OS that can run a "guest OS" as an application, often by emulating a complete hardware environment for each guest OS. Although the commoditized virtualization technology is relatively new, it is quickly becoming widespread, and new software and hardware is quickly being developed to support more features and provide better virtualization performance. [7]

One of the lingering challenges in managing virtualized environments efficiently is scheduling: since the host and guest OSs often operate with no coordination and knowledge of each other's scheduling, mis-scheduling issues continue to crop up. For example, an interactive application in a guest OS might be scheduled correctly by the guest OS, but since the host OS is unaware of the application's requirements, the guest OS (and by extension, the application) could be mistakenly scheduled as a noninteractive program.

The area of scheduling research for virtualization is still in its infancy, and it may be too early to explore well-developed scheduling issues with virtualization. Nevertheless, we identify the following topics where job scheduling research can benefit virtualized environments:

- Scheduling inside the guest OS: Currently, a guest OS schedules its processes oblivious of any host OS constraints, as demonstrated in the previous example with interactive applications. Scheduling research can address this class of problems by (1) characterizing the scheduling needs of different processes; (2) characterizing the scheduling constraints and services that the host OS can guarantee; and (3) communicating and matching these requirements and constraints to create an acceptable schedule in the guest OS within the host environment.
- Similarly, the guest OS is oblivious of any other guest OSs or processes running on the same host, creating more scheduling mismatches. The problem can thus be generalized to creating a schedule within the host OS that takes into account the requirements of all host processes and guest OS processes.

[7] Refer for example to Intel's new hardware support for virtualization in its latest architecture (code-named Penryn).

– Looking at a larger scale still, virtualization is often used in Grids and multi-host environments to provide dynamic allocation of customized computing images in the form of virtual images[8]. This extension creates additional meta-scheduling issues for the virtualized environment, not unlike those discussed in Section 3, but with additional consideration for the moldable and malleable nature of virtualized resources.

Generalizing the Challenges

Research into the implications of virtualization has only yet begun. We believe that scheduling will play an increasingly important role in virtualized environments where performance and utilization matter. One of the keys to the successful scheduling of such heterogeneous workloads and execution environments is the ability to characterize clearly the scheduling requirements of different processes, and scheduling them accordingly. The literature already contains examples of process characterizations for several scheduling domains, such as multimedia and parallel processing [2,12,24,55]. We think the time is ripe to create and generalize additional characterizations that would fit the virtualized environments as well. Such characterizations need to take into account the entire process stack, from the multi-threaded guest process at the one end to the host OS or meta-scheduler at the other. Eventually, these characterizations may even help define a standard of communication of scheduling information between host and guest OSs.

6 Overarching Considerations

There are several considerations from those listed above that span most or all of the new scheduling challenges. In this section, we will briefly generalize these considerations.

Workload Virtually all scheduling and performance evaluations start with a workload, and all use cases described above require good workloads for research progress. While a good workload often depends on the use case in general, useful workloads for research contain long enough traces (or model-derived data) for metrics to stabilize, and are detailed enough to allow multiple factor analyses. For classical supercomputers, a handful of such workloads has been collected and maintained by Feitelson [38]. An effort to collect Grid workloads is also underway, but it is in its early phases (see `http://gwa.ewi.tudelft.nl`). For other use cases, however, we are not aware of any centralized effort to collect such workloads. The workload challenge does not end with collection, but merely starts: To be useful for researchers and to enable performance evaluation, workloads need to be analyzed, characterized, and possibly classified and modeled. By

[8] for example, Amazon's *Elastic Compute Cloud* (`http://www.amazon.com/gp/browse.html?node=201590011`)

understanding and generalizing data from multiple workloads, we can develop better scheduling schemes, as well as a better understanding of the similarities between scheduling scenarios in Grids, web servers, supercomputers, and the like.

Heterogeneity Unlike traditional supercomputers, most contemporary parallel architectures offer some degree of heterogeneity: from the single chip level with parallel execution modules and simultaneous multithreading, through the single node and its heterogeneous components such as accelerators, through the cluster and the Grid with their heterogeneous nodes. Scheduling for all levels now needs to take into account unequal resources to manage which complicates both the optimization problem and the metrics themselves being optimized.

Scheduling for power Power constraints now appear in resource management schemes at virtually all levels, from keeping the temperatures on the surface of a multi-core chip controlled and equally distributed, to lowering the power and cooling bills of large web servers and Grid farms. In some instances, power consumption is the single most influential resource management constraint of a parallel installation, and every percent saved translates to significant cost and emission savings. Scheduling can play an important role in power saving by incorporating power considerations into the bigger resource management question. Although work on this incorporation started several years ago [47,33], scheduling for power saving is still a relatively unexplored research topic.

Security Security is a fast-growing concern in today's computing environments. Most of the scenarios described above involve multiple users, or at least multiple applications. Protecting the data and resources of one user or application is vital for the successful deployment of parallel and shared computing resources [56]. To some extent, security considerations affect scheduling. For example, some applications may request to run without sharing any memory or network resources with other applications. Security considerations can also conflict with scheduling considerations, such as the "black-box" approach of virtualized images, that discourages shared resource-management and scheduling decisions. A large research gap exists in the area of scheduling with/for security considerations in these new domains.

Economy Just like security considerations, economy considerations affect, and sometimes even govern, the shared use of compute resources. Large data bases may not be available for free if they are useful for commercial activities, like weather forecast and traffic data for logistics. Moreover, an increasing number of users and user groups need information systems which will become more complex and more expensive in the future, for instance, due to power consumption. This may lead to a shortage of resources and result in user priorities based on the price a user is willing to pay for the provided information service. It may not be the task of scheduling systems to determine market prices of information resources but scheduling systems certainly need to convert a given policy into the distribution of resources.

Metrics Properly using meaningful metrics is an inseparable part of performance evaluation. Although scheduler performance evaluations typically use well-publicized metrics such as average response time and throughput, these metrics are not always used correctly or do not describe the performance picture adequately [13,22]. Moreover, metrics will have to be somewhat adjusted to account for some of the newer use cases described above. For example, Grid users may care more about fairness than response time [43]. Scheduling on heterogeneous architectures requires different treatment of run time and resource utilization for performance than for billing, since not all resources are equal. There is therefore a need to extend and unify current scheduling evaluation metrics in order to be useful and meaningful for the actual use cases.

7 Conclusion

Scheduling for traditional multiprocessors is still a hard problem that is actively researched. The recent introduction of several architectures with different types of parallelism and a wide spectrum of uses, workloads, requirements, and hardware, poses an even harder challenge to the scheduling community. If parallel job scheduling can be viewed as a multi-dimensional optimization problem, these new architectures now add several more dimensions to the problem.

Nevertheless, this challenge is also a great opportunity. The scheduling community can evolve and incorporate lessons learned over many years of research (much of which has been published in JSSPP), and advance the state of the art in the new emerging fields. Despite the various architectures, there are many shared issues between the different scheduling domains: workloads, requirement characterization, resource management, meaningful metrics, power consumption, and others. All these topics are inter-related and are studied by contributors to JSSPP. The time is ripe now for this community to generalize these scheduling characterizations to the emerging domains in parallel job scheduling.

References

1. Christos D. Antonopoulos, Dimitrios S. Nikolopoulos, and Theodore S. Papatheodorou. Scheduling algorithms with bus bandwidth considerations for SMPs. In *32nd International Conference on Parallel Processing (ICPP)*, Kaohsiung, Taiwan, October 2003. Available from `www.cs.wm.edu/~dsn/papers/icpp03.pdf`.
2. Scott A. Banachowski and Scott A. Brandt. The BEST Scheduler for Integrated Processing of Best-Effort and Soft Real-Time Processes. In *Multimedia Computing and Networking (MMCN)*, San Jose, CA, January 2002. Available from `www.cse.ucsc.edu/~sbanacho/papers/banachowski-mmcn02.ps`.
3. Daniel Becker, Felix Wolf, Wolfgang Frings, Markus Geimer, Brian J.N. Wylie, and Bernd Mohr. Automatic trace-based performance analysis of metacomputing applications. In *21st International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE Computer Society, March 2007.
4. Francine Berman, Henri Casanova, Andrew Chien, Keith Cooper, Holly Dail, Anshuman Dasgupta, W. Deng, Jack Dongarra, Lennart Johnsson, Ken Kennedy,

Charly Koelbel, B. Liu, Xin Liu, Anirban Mandal, Gerald Marin, Mark Mazina, John Mellor-Crummey, Celso Mendes, Alex Olugbile, Jignesh M. Patel, Daniel Reed, Zhiao Shi, Otto Sievert, Huaxia Xia, and Asim YarKhan. New grid scheduling and rescheduling methods in the grads project. *International Journal of Parallel Programming*, 33(2):209–229, 2005.

5. A.I.D. Bucur and Dick Epema. Scheduling policies for processor co-allocation in multicluster systems. *IEEE Transactions on Parallel and Distributed Systems*, 18:958–972, 2007.
6. James R. Bulpin and Ian A. Pratt. Multiprogramming performance of the Pentium 4 with hyper-threading. In *Second Annual Workshop on Duplicating, Deconstruction and Debunking (WDDD)*, pages 53–62, Munchen, Germany, June 2004. Available from `www.ece.wisc.edu/~wddd/2004/06_bulpin.pdf`.
7. Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin M. Vahdat, and Ronald P. Doyle. Managing energy and server resources in hosting centers. *SIGOPS Operating Systems Review*, 35(5):103–116, 2001.
8. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. In *Symposium on Operating Systems Design and Implementation (OSDI)*, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.
9. Carsten Ernemann, Volker Hamscher, Uwe Schwiegelshohn, Achim Streit, and Ramin Yahyapour. Enhanced algorithms for multi-site scheduling. In *Proceedings of the Third International Workshop on Grid Computing (Grid'02)*, volume 2536 of *Lecture Notes in Computer Science*, pages 219–231. Springer, November 2002.
10. Carsten Ernemann, Baiyi Song, and Ramin Yahyapour. Scaling of workload traces. In Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, *Ninth Workshop on Job Scheduling Strategies for Parallel Processing*, volume 2862 of *Lecture Notes in Computer Science*, pages 166–182. Springer-Verlag, 2003. Available from `www.cs.huji.ac.il/~feit/parsched/`.
11. Carsten Ernemann and Ramin Yahyapour. Applying economic scheduling methods to grid environments. In J. Nabrzyski, J.M. Schopf, and J Weglarz, editors, *Grid Resource Management - State of the Art and Future Trends*, pages 491–506. Kluwer, 2003.
12. Yoav Etsion, Dan Tsafrir, and Dror G. Feitelson. Desktop scheduling: How can we know what the user wants? In *14th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, pages 110–115, County Cork, Ireland, June 2004. Available from `www.cs.huji.ac.il/~feit/papers/HuCpri04NOSSDAV.pdf`.
13. Dror G. Feitelson. Metrics for parallel job scheduling and their convergence. In Dror G. Feitelson and Larry Rudolph, editors, *Seventh Workshop on Job Scheduling Strategies for Parallel Processing*, volume 2221 of *Lecture Notes in Computer Science*, pages 188–1205. Springer Verlag, 2001. Available from `www.cs.huji.ac.il/~feit/parsched/`.
14. Dror G. Feitelson and Larry Rudolph. Gang scheduling performance benefits for fine-grain synchronization. *Journal of Parallel and Distributed Computing*, 16(4):306–318, December 1992. Available from `www.cs.huji.ac.il/~feit/papers/GangPerf92JPDC.ps.gz`.
15. Dror G. Feitelson, Larry Rudolph, and Uwe Schwigelshohn. Parallel job scheduling – A status report. In Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, *Tenth Workshop on Job Scheduling Strategies for Parallel Processing*, volume 3277 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, 2004. Available from `www.cs.huji.ac.il/~feit/parsched/`.

16. Krisztián Flautner, Rich Uhlig, Steve Reinhardt, and Trevor Mudge. Thread-level parallelism and interactive performance of desktop applications. In *Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 129–138, November 2000. Available from `www.eecs.umich.edu/~tnm/papers/asplos00.pdf`.
17. Ian T. Foster. The anatomy of the grid: Enabling scalable virtual organizations. In *Proceedings of the Seventh Euro-Par(Euro-Par'01)*, pages 1–4, London, UK, 2001. Springer-Verlag.
18. Ian T. Foster and Carl Kesselman, editors. *The GRID: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.
19. Geoffrey C. Fox and Dennis Gannon. Special issue: Workflow in grid systems: Editorials. *Concurrency and Computation: Practice and Experience*, 18(10):1009–1019, 2006.
20. Eitan Frachtenberg. Process Scheduling for the Parallel Desktop. In *Proceedings of the International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN'05)*, Las Vegas, NV, December 2005.
21. Eitan Frachtenberg and Yoav Etsion. Hardware parallelism: Are operating systems ready? (case studies in mis-scheduling). In *Second Workshop on the Interaction between Operating Systems and Computer Architecture (WIOSCA'06), In conjunction with ISCA-33*, Boston, MA, June 2006.
22. Eitan Frachtenberg and Dror G. Feitelson. Pitfalls in parallel job scheduling evaluation. In Dror G. Feitelson, Eitan Frachtenberg, Larry Rudolph, and Uwe Schwiegelshon, editors, *11th Workshop on Job Scheduling Strategies for Parallel Processing*, volume 3834 of *Lecture Notes in Computer Science*, pages 257–282. Springer-Verlag, Boston, MA, June 2005. Available from `www.cs.huji.ac.il/~etcs/pubs/`.
23. Eitan Frachtenberg, Dror G. Feitelson, Fabrizio Petrini, and Juan Fernandez. Flexible CoScheduling: Mitigating load imbalance and improving utilization of heterogeneous resources. In *17th International Parallel and Distributed Processing Symposium (IPDPS)*, Nice, France, April 2003. Available from `www.cs.huji.ac.il/~etcs/pubs/`.
24. Eitan Frachtenberg, Dror G. Feitelson, Fabrizio Petrini, and Juan Fernandez. Adaptive parallel job scheduling with flexible coscheduling. *IEEE Transactions on Parallel and Distributed Systems*, 16(11):1066–1077, November 2005. Available from `www.cs.huji.ac.il/~etcs/pubs/`.
25. Carsten Franke, Joachim Lepping, and Uwe Schwiegelshohn. On advantages of scheduling using genetic fuzzy systems. In *12th Workshop on Job Scheduling Strategies for Parallel Processing*, volume 4376 of *Lecture Notes in Computer Science*, pages 68–93. Springer, June 2006.
26. Brajesh Goyal and Shilpa Lawande. *Grid Revolution: An Introduction to Enterprise Grid Computing*. The McGraw-Hill Companies, 2006.
27. H. Peter Hofstee. Power efficient processor architecture and the Cell processor. In *11th International Symposium on High-Performance Computer Architecture*, San Francisco, CA, February 2005. Available from `www.hpcaconf.org/hpca11/papers/25_hofstee-cellprocessor_final.pdf`.
28. Steven Hotovy. Workload evolution on the cornell theory center IBM SP2. In Dror G. Feitelson and Larry Rudolph, editors, *First Workshop on Job Scheduling Strategies for Parallel Processing*, number 1162 in Lecture Notes in Computer Science, pages 27–40, 1996.

29. Eduardo Huedo, Rubén S. Montero, and Ignacio Martín Llorente. Evaluating the reliability of computational grids from the end user's point of view. *Journal of Systems Architecture*, 52(12):727–736, 2006.
30. Terry Jones, William Tuel, Larry Brenner, Jeff Fier, Patrick Caffrey, Shawn Dawson, Rob Neely, Robert Blackmore, Brian Maskell, Paul Tomlinson, and Mark Roberts. Improving the scalability of parallel jobs by adding parallel awareness to the operating system. In *15th IEEE/ACM Supercomputing*, Phoenix, AZ, November 2003. ACM Press and IEEE Computer Society Press. Available from `www.sc-conference.org/sc2003/paperpdfs/pap136.pdf`.
31. Michael Kanellos. Designer puts 96 cores on single chip. `news.com.com/2100-1006_3-5399128.html`, October 2004.
32. Jiadao Li and Ramin Yahyapour. Learning-based negotiation strategies for grid scheduling. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2006), Singapore*, pages 567–583. IEEE Press, 2006.
33. Yung-Hsiang Lu, Luca Benini, and Giovanni De Micheli. Low-power task scheduling for multiple devices. In *Proceedings of the Eighth International Workshop on Hardware/software codesign (CODES'00)*, pages 39–43, New York, NY, USA, 2000. ACM.
34. Jon MacLaren, Rizos Sakellariou, Jon Garibaldi, Djamila Ouelhadj, and Krish T. Krishnakumar. Towards service level agreement based scheduling on the grid. In *Proceedings of the Workshop on Planning and Scheduling for Web and Grid Services*, pages 100–102, Whistler, BC, Canada, July 2004.
35. Emmanuel Medernach. Workload analysis of a cluster in a grid environment. In Dror G. Feitelson, Eitan Frachtenberg, Larry Rudolph, and Uwe Schwiegelshon, editors, *11th Workshop on Job Scheduling Strategies for Parallel Processing*, volume 3834 of *Lecture Notes in Computer Science*, pages 36–61. Springer-Verlag, Boston, MA, June 2005. Available from `www.cs.huji.ac.il/~feit/parsched/`.
36. Jason Nieh, James G. Hanko, J. Duane Northcutt, and Gerard A. Wall. SVR4 UNIX scheduler unacceptable for multimedia applications. In *Fourth ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, November 1993. Available from `citeseer.ist.psu.edu/443381.html`.
37. John. K. Ousterhout. Scheduling techniques for concurrent systems. In *Third International Conference on Distributed Computing Systems*, pages 22–30, Miami, FL, October 1982.
38. Parallel workload archive. `www.cs.huji.ac.il/labs/parallel/workload`.
39. Michael Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Prentice-Hall, New Jersey, second edition, 2002.
40. Rosario M. Piro, Andrea Guarise, and Albert Werbrouck. An economy-based accounting infrastructure for the datagrid. In *Proceedings of the Fourth International Workshop on Grid Computing (GRID'03)*, page 202, Washington, DC, USA, 2003. IEEE Computer Society.
41. James Reinders. *Intel Threading Building Blocks*. O'Reilly and Associates, July 2007.
42. Thomas G. Robertazzi and Dantong Yu. Multi-Source Grid Scheduling for Divisible Loads. In *Proceedings of the 40th Annual Conference on Information Sciences and Systems*, pages 188–191, March 2006.
43. Gerald Sabin and P. Sadayappan. Unfairness metrics for space-sharing parallel job schedulers. In Dror G. Feitelson, Eitan Frachtenberg, Larry Rudolph, and Uwe Schwiegelshon, editors, *11th Workshop on Job Scheduling Strategies for Parallel*

Processing, volume 3834 of *Lecture Notes in Computer Science*, pages 238–256. Springer-Verlag, Boston, MA, June 2005. Available from `www.cs.huji.ac.il/~feit/parsched/`.

44. Uwe Schwiegelshohn and Ramin Yahyapour. Fairness in parallel job scheduling. *Journal of Scheduling*, 3(5):297–320, 2000.
45. Uwe Schwiegelshohn and Ramin Yahyapour. Attributes for communication between grid scheduling instances. In J. Nabrzyski, J.M. Schopf, and J. Weglarz, editors, *Grid Resource Management – State of the Art and Future Trends*, pages 41–52. Kluwer Academic, 2003.
46. Stephen Shankland. Azul's first-generation Java servers go on sale. `news.com.com/2100-1010_3-5673193.html?tag=nl`, April 2005.
47. Youngsoo Shin and Kiyoung Choi. Power conscious fixed priority scheduling for hard real-time systems. In *Proceedings of the 36th ACM/IEEE conference on Design automation (DAC'99)*, pages 134–139, New York, NY, USA, 1999. ACM.
48. Mumtaz Siddiqui, Alex Villazón, and Thomas Fahringer. Grid capacity planning with negotiation-based advance reservation for optimized qos. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing (SC'06)*, page 103, 2006.
49. Allan Snavely and Dean M. Tullsen. Symbiotic jobscheduling for a simultaneous multithreading processor. In *Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 234–244, Cambridge, MA, November 2000. Available from `citeseer.ist.psu.edu/338334.html`.
50. Angela C. Sodan and Lei Lan. LOMARC—Lookahead matchmaking for multi-resource coscheduling. In Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, *Tenth Workshop on Job Scheduling Strategies for Parallel Processing*, volume 3277 of *Lecture Notes in Computer Science*, pages 288–315. Springer-Verlag, 2004. Available from `www.cs.huji.ac.il/~feit/parsched/`.
51. Andrei Tchernykh, Juna Manuel Ramírez, Arutyun Avetisyan, Nicolai Kuzjurin, Dimitri Grushin, and Sergey Zhuk. Two level job-scheduling strategies for a computational grid. In *Proceedings of the Second Grid Resource Management Workshop (GRMW'05) in conjunction with the Sixth International Conference on Parallel Processing and Applied Mathematics (PPAM'05)", EDITOR = "R. Wyrzykowski and J. Dongarra and N. Meyer and J. Wasniewski*, number 3911 in Lecture Notes in Computer Science, pages 774–781, September 2005.
52. Gabor Terstyanszky, Tamas Kiss, Thierry Delaitre, Stephen Winter, and Peter Kacsuk. Service-oriented production grids and user support. In D. Gannon and R. M. Badia, editors, *Proceedings of the Seventh IEEE/ACM international conference on Grid computing. Barcelona, 2006.*, pages 323–324, 2006.
53. Srikumar Venugopal, Rajkumar Buyya, and Lyle Winton. A grid service broker for scheduling distributed data-oriented applications on global grids. In *Proceedings of the Second Workshop on Middleware for grid computing (MGC'04)*, pages 75–80, New York, NY, USA, 2004. ACM.
54. Samuel Williams, John Shalf, Leonid Oliker, Shoaib Kamil, Parry Husbands, and Katherine Yelick. The potential of the cell processor for scientific computing. In *Proceedings of the Third Conference on Computing frontiers (CF'06)*, pages 9–20, New York, NY, USA, 2006. ACM.
55. Yair Wiseman and Dror G. Feitelson. Paired gang scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 14(6):581–592, June 2003.
56. Tao Xie and Xiao Qin. Enhancing security of real-time applications on grids through dynamic scheduling. In Dror G. Feitelson, Eitan Frachtenberg, Larry

Rudolph, and Uwe Schwiegelshon, editors, *11th Workshop on Job Scheduling Strategies for Parallel Processing*, volume 3834 of *Lecture Notes in Computer Science*, pages 219–237. Springer-Verlag, Boston, MA, June 2005. Available from `www.cs.huji.ac.il/~feit/parsched/`.

57. Guangchang Ye, Ruonan Rao, and Minglu Li. A multiobjective resources scheduling approach based on genetic algorithms in grid environment. In *Proceedings of the Fifth International Conference on Grid and Cooperative Computing Workshops (GCCW'06)*, pages 504–509, Washington, DC, USA, 2006. IEEE Computer Society.
58. Kenneth Yoshimoto, Patricia Kovatch, and Phil Andrews. Co-scheduling with user-settable reservations. In Dror G. Feitelson, Eitan Frachtenberg, Larry Rudolph, and Uwe Schwiegelshon, editors, *11th Workshop on Job Scheduling Strategies for Parallel Processing*, volume 3834 of *Lecture Notes in Computer Science*, pages 146–156. Springer-Verlag, Boston, MA, June 2005. Available from `www.cs.huji.ac.il/~feit/parsched/`.
59. Dayi Zhou and Virginia Lo. Wave scheduler: Scheduling for faster turnaround time in peer-based desktop grid systems. In Dror G. Feitelson, Eitan Frachtenberg, Larry Rudolph, and Uwe Schwiegelshon, editors, *11th Workshop on Job Scheduling Strategies for Parallel Processing*, volume 3834 of *Lecture Notes in Computer Science*, pages 194–218. Springer-Verlag, Boston, MA, June 2005. Available from `www.cs.huji.ac.il/~feit/parsched/`.