

**PARALLEL JOB SCHEDULING
ALGORITHMS AND INTERFACES**

Cynthia Bailey Lee
Department of Computer Science and Engineering
University of California, San Diego

clbailey@ucsd.edu

May 13, 2004

Abstract

High-performance computing continues to be dominated by the massively parallel processor (MPP) and cluster machine architectures, with 373 of the current Top 500 supercomputers in the world falling into this category. A single system typically costs millions or tens of millions of dollars, and supports a large and diverse user population, whose jobs each have unique resource requirements. How to best organize the running of these jobs is a mature but active research area. This paper reviews the methods, metrics and historical evolution of parallel job scheduling, in theory and in practice, and suggests future research directions.

1 Introduction

Early supercomputers most often had a single processor or a small number (e.g. 2 or 4) of very tightly coupled processors (symmetric multiprocessing, or SMP, architecture). So scheduling was done in a timesharing fashion, with many or all of the jobs running at the "same" time. On the massively parallel processor (MPP) and cluster architectures that currently dominate (see Figure 1), it is more practical to assign jobs subsets of the machine's processors, to which they have exclusive access for the duration of the job. This is known as space sharing.

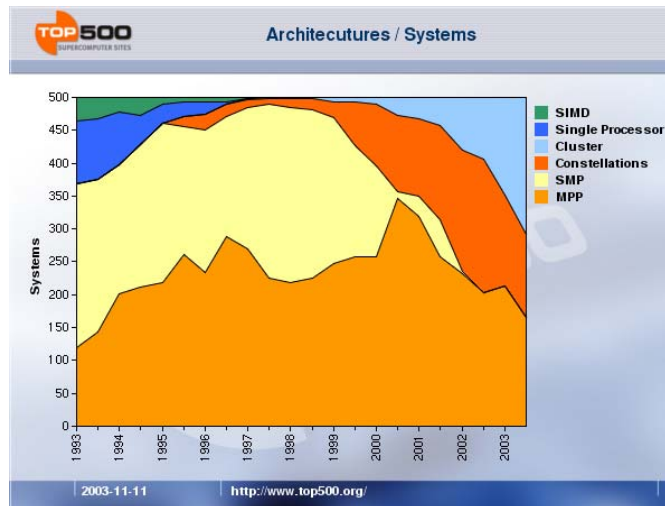


Figure 1: Proportion of different system architecture types of the "Top 500" supercomputers. (Image courtesy of <http://www.top500.org/>.)

In [FR96], Feitelson and Rudolph note that there are a variety of distinct sub-areas of research under the umbrella of parallel scheduling, and compare the (too often implicit) assumptions made in the work of each area. One conclusion of their work was to advocate that researchers in scheduling always explicitly state the problem situation they are proposing to solve, with its underlying assumptions. I do so now, making use of their suggested terminology in some cases.

The following assumptions distinguish the particular sub-area of scheduling research to be studied in this paper:

1. Purpose: The purpose of the system is to process a workload consisting of parallel batch jobs.
2. Processor Homogeneity: The machine has some number of homogenous processors that can be divided into arbitrary subsets to be assigned to jobs. In particular, the network topology is abstracted.
3. Job Specification: Jobs are defined in two dimensions: the number of processors required and the duration of time required on those processors (called *requested runtime*). Neither of these changes after the job has been submitted to the system. These parameters are user-specified and can have arbitrary values (up to the number of processors on the machine, and perhaps some upper time limit). Sometimes other information, e.g. priority, is also associated with the job.
4. Processor Exclusivity and Non-Preemption: Each processor is only assigned one job at a time. Once a job has begun running on a set of processors, it has exclusive access to those processors until it ends. Thus once the scheduler starts a job, that scheduling decision is irrevocable.
5. Online Scheduling: Jobs arrive in the system in a stochastic manner. There is no knowledge of future jobs. However, general workload knowledge, in other words typical distributions of job attributes and arrival rates, is often available when designing the scheduler.

6. **User Independence:** Users are assumed to be independent and self-interested, in that a user wants his or her job(s) to complete as soon as possible, and has no reason to value completion of other users' jobs. (Typically in the literature, this user independence is approximated as job independence; jobs are viewed anthropomorphically as competing with each other for earlier scheduling slots.) There is some accounting scheme that tracks users' resource consumption; most often users are allocated a limited amount of resource credits and/or charged for their use in some fashion.

Selected work that falls outside these parameters will be presented for contrast, however unless noted specifically otherwise, these assumptions will apply.

The question of how solutions to the above-defined problem are to be compared and evaluated is a complicated one. Assumptions 1 and 6 will provide the underlying motivation for how metrics are to be selected, but they do not dictate a single obvious choice. This problem will be discussed throughout the rest of the paper, and is addressed in detail in section 7.

2 Methodology

Even taking only the literature relevant to this particular sub-problem in parallel scheduling (as defined by the assumptions listed in Section 1), the literature is still so vast that it is useful to do a brief comparative examination of the methods employed. Here I define an ordered set of layers at which researchers engage in the study of scheduling systems. Many papers operate on only one or two of these layers at a time, but the literature includes works addressing all four. The layers are defined as:

Layer 1: *Complexity*. Research at this layer classifies by complexity various abstracted schedule optimization problems. In particular several variants of parallel processor-style scheduling have been shown to be NP-hard [GJ79].

Layer 2: *Systems and Implementation*. Research at this layer examines and compares implementations of scheduling algorithms as part of usable software systems. This research attends to details such as unexpected resource failure that are abstracted out of Layer 1 inquiries. The definition and evaluation of metrics describing the efficiency of a scheduler's handling of real workload traces is characteristic of Layer 2.

Layer 3: *Human-Computer Interface*. Layer 3 concerns how users feel about their experience of using a scheduling system. This work relates findings in Layers 1 and 2 to users' notions of acceptable scheduling system behavior. This work is characterized by metrics such as predictability (i.e. low variance in Layer 2 metrics). It addresses issues such as users' ability to understand the scheduler and provide correct input.

Layer 4: *Community*. While Layer 3 asks about an individual users' interaction with the scheduling system, research at Layer 4 asks how communities of users behave as a group. Issues of organizational politics and other external forces exerted on the scheduling system may also be addressed.

A common trait of all the layers is that investigations often involve examining the validity of assumptions made in the literatures of the preceding layers.¹ For example, a systems and implementations study may reject algorithms proposed in the complexity literature which run in time $O(n^a)$ with a $a > 1$ or 2.

3 Beginnings of MPP Scheduling

3.1 First-Come First-Serve

In the period before network topology of MPP's was abstracted, only certain subsets (e.g. contiguous subsets) of processors could be aggregated and assigned to a job. Thus to prevent fragmentation,

¹ In [G04], Goguen notes that a progression from technological, to individual-psychological, to group-sociological has been seen in many fields of study, in computer science and generally.

significant effort went into the selection of which nodes to run the next job on, or *allocation* of processors; and, at first, the selection of which job should be the next to run, or *scheduling*, was simply done first-come, first-serve (FCFS) [KLD94].

On an MPP system, FCFS works as follows. Let J_1 be the first job in a FCFS queue, and let I be the number of currently available processors. The number of processors required by job J_i is denoted $J_i.p$. If $J_i.p > I$, then J_i cannot begin running, and furthermore all the other jobs behind J_i must also wait (because the queue is only accessed at its head), and the I processors remain idle. Note that in this formulation, a FCFS scheduler does not require the user to specify the *requested runtime* of the job (see Assumption 3 above). Jobs are simply run until their natural completion, and then the next job is started when possible.

The idleness caused by this blocking approach can lead to significant wasted resources in the wake of wide (many-processor) jobs. The metric used to describe the relative amount of consumed to wasted resources on the machine is *utilization*:

$$Utilization = \frac{UsedProcessor\text{-}Hours}{TotalProcessor\text{-}Hours} \quad (2)$$

Where *Used Processor-Hours* is the number of processor-hours in the observed period that were assigned to jobs, and *Total Processor-Hours* is the number of processors on the machine times the number of hours in the observed period (usually in analyzing actual systems, failures and other downtimes that prevent processors from being assigned to jobs are subtracted from the total). Utilization only captures external fragmentation—there is an implicit assumption that jobs make good use of the processor-hours assigned to them.

In response to a paper showing that scheduling has more impact on utilization and *average response time* than allocation [KLD94], researchers began focusing more on scheduling. This gave rise to *backfilling* algorithms, which allow a job J_i , behind J_1 in the queue, to "skip ahead" if $J_i.p \leq I$. Selected backfilling algorithms will be discussed in detail in Section 4. Average response time is the average over all jobs of:

$$ResponseTime = EndTime - SubmitTime \quad (3)$$

The average response time metric is intended to measure the inconvenience placed on users by the presence of other users on the system, by measuring how long their jobs wait to begin running.

3.2 Tennis Court Scheduling

Some papers, in particular [MF01, FRSSW97], seem to imply that FCFS scheduling was a prevailing method of scheduling supercomputers of the type described in the assumptions given in this paper. Although it has been demonstrated in simulations (e.g. [SAWP95]) that FCFS results in poor utilization for typical workloads on a system conforming to the assumptions, evidence to be presented here, and even the evidence cited in [MF01, FRSSW97], indicates that actual use of FCFS on major production systems had declined by the time these types of supercomputers came into use.

One system that was said to have used FCFS (in [MF01]) was the Concurrent Supercomputing Consortium's Touchstone Delta, a 568-node Intel system that became operational in May 1991 [M93]. In his report on the first year of the project, Messina says flatly, "The Delta also lacks a batch scheduler." [M93] Of course, ambiguity remains about how this space-shared system was scheduled in the absence of any scheduling software.

According to Wan [W04], who wrote a scheduler for a contemporary system at the San Diego Supercomputer Center (that was later ported to the consortium's Delta) and Pfeiffer [P04], also at SDSC, the Delta employed a system known as "tennis-court" scheduling. The user would place a phone call to

the system's operators, and request a certain number of processors and length of time.² The operators were responsible for scheduling the phoned-in jobs. This is consistent with Messina's explanation that the lack of a batch scheduler "affects operations less than you might expect, since most system time is allocated to one or two jobs that run for four hours or more at a time" [M93], and thus only infrequent human intervention is required for much of the day.

For a moment, let us assume that the operators created an inviolable reservation for each job at the time it was called in. Then in a sense the schedule slots are available on a "first-come, first serve" basis. However, it seems difficult to imagine that the operators would not allow a job to begin ahead of a job that was called in earlier, as long as it fit into a "hole" not used by any of the previous reservations. In other words, it seems reasonable to believe the operators did some form of backfilling, and thus were not implementing the blocking FCFS algorithm, as it is understood in the literature. That Messina documents users being required to give a requested runtime is circumstantial evidence that backfilling occurred, since FCFS does not require known job end times. Finally, there is no evidence that our assumption that operators created an inviolable reservation is true. One could imagine an operator rearranging the schedule from time to time, for example if an urgent situation arose with an important user.

I claim that the tennis-court scheduling paradigm is not simply entertaining historical trivia, but can provide an important perspective from which to judge the software-based schedulers that came after it. With FCFS as a baseline, almost any alternative algorithm would be judged favorably. But scheduling software systems must have high aspirations indeed to compete with the creativity, flexibility and analysis of a human being, not just in terms of the bin-packing algorithmics, but also in terms of the total user interface. One can imagine a lengthy negotiation between user and operator over job parameters and schedule availability to achieve the most satisfying result. Humans can also make fine adjustments for the relative status of users, and the relative importance amongst a user's jobs.

3.3 Other Early Experiences

Tennis-court scheduling on CSC's Delta was replaced soon after Messina's report with a proper scheduler designed by Wan et al. of SDSC. The scheduler was developed for SDSC's Intel iPSC/860 system and is described in [WMKS96], as it was adapted for the subsequent Paragon system³ at SDSC (there is no documentation of the scheduler at the time of its original development for the iPSC/860).

Wan's scheduler is based on the well-known Network Queuing System (NQS) software package, and addresses several significant shortcomings of that software as it had been used at SDSC. In particular, it suffered from a blocking problem in which a low priority job could not run ahead of a higher priority job, causing starvation of lower priority jobs.

To address this, Wan's scheduler introduces the concept of *aging*. Aging is the idea of sorting jobs by a continually recalculated *effective priority*, which is a function of the initial (assigned) priority and the amount of time the job has currently been waiting. Recall that the first assumption (from Section 1), was, "The purpose of the system is to process a workload consisting of parallel batch jobs." Wan's aging policy is an instance of this assumption affecting design—a low priority job should wait longer than a high priority one, but all jobs should eventually be processed, without excessive delay. To track average delay of jobs, administrators at SDSC used a runtime-weighted metric called *expansion factor* (also called average *slowdown*):

$$\text{Expansion Factor} = (\text{Runtime} + \text{WaitTime}) / \text{Runtime} \quad (4)$$

² The Delta system was in violation of several of the assumptions stated in the beginning of this paper. Specifically, the hypercube topology of the network was not abstracted, so processors had to be assigned in contiguous subsets, and only power-of-two job sizes were possible.

³ Unlike SDSC's iPSC/860 system and CSC's Delta, the Paragon did have an abstracted network topology. This allowed jobs to request an arbitrary number of processors, and processors to be assigned in arbitrary subsets. Processor allocation schemes were still used, but only as a network performance optimization.

4 Backfilling

4.1 Introduction to Backfilling

Backfilling is a policy of strategically allowing jobs to run out of order.⁴ Abstractly, we would like to say that even if one job "should" run before another, if the first is currently unable to make use of the resource—and the other can—it is only logical to allow the other to use the resource.

Experience has shown that backfilling improves utilization by about 20% on most systems and workloads, and greatly improves the response time for small jobs, which are most likely to be able to backfill. It is reported that over 90% of short, narrow (few processor) jobs are typically able to backfill, in particular the year-long CHPC workload trace shows over 90% of small jobs backfilling [JSC01, CHPC as cited in JSC01]. This is all while often providing moderate improvement for even the largest jobs. This is because increased utilization means that often all jobs run sooner than without backfill, not just those that were backfilled.

As a result, backfilling has been described as "something for nothing," a benefit without a tradeoff, made possible because of the previous inefficiency of the system [JSC01]. Of course there are some occasional minor drawbacks and several claimed guarantees of non-delay in the literature were subsequently shown to be false. These will be explored as each backfilling approach is examined in turn.

4.2 EASY

The first explicitly documented use of backfilling is in the Extensible Argonne Scheduling sYstem (EASY), developed by Lifka for Argonne National Laboratory's 128-node IBM SP system. Eventually integrated with IBM's LoadLeveler administration software, EASY was "used by many MPP sites throughout the world and is known for its efficient scheduling, simplicity, and robustness." [SCZL96].

The EASY scheduler orders jobs by arrival. Scheduling decisions are made when a new job arrives or a currently running job ends. If the job at the head of the queue can run on the available processors, it is started. Otherwise, EASY determines the earliest time at which the first job can begin running (according to the requested runtime for each running job) and scans the queue for a smaller job. The smaller job must fit both *width-wise*, on the available processors, and *length-wise*, i.e. its requested runtime is such that it would not delay the scheduled start time of the first job in the queue. The designers of EASY claim that it obeys the principle that "jobs in the queue are never delayed from running by jobs submitted to the queue after them." [SCZL96]

However, Mu'alem and Feitelson [MF01] showed that EASY backfilling could cause a job to be delayed by jobs that were submitted after it. This is because the delay check that a job must pass in order to backfill is only carried out against the job at the head of the queue. In some cases, even though backfilling a job does not delay the first job in the queue, it may delay jobs subsequent jobs, which are still ahead of the candidate backfilling job.

⁴ The queue order is often thought of by default as the order of arrival (FCFS), but can be an arbitrary priority ordering. Note the distinction between algorithms that operate on a FCFS *queue* (e.g. backfilling schedulers), and the actual FCFS *algorithm*.

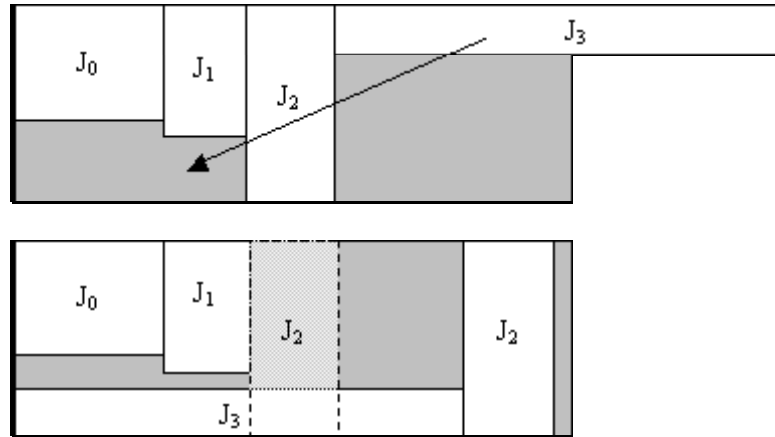


Figure 2: Delay in EASY

Figure 2 shows an example of the situation described in [MF01]. The y-axis represents the machine's processors, and the x-axis represents time, with the current time marked by a heavy black line (at far left). Job J_0 is already running, and jobs J_1 , J_2 and J_3 are queued (in that order). Job J_1 , the head of the queue, cannot begin until J_0 ends, due to insufficient available processors. Then job J_2 is the first candidate for backfilling, but J_2 cannot run either. So job J_3 is backfilled, because sufficient processors are available and it will not delay the start of J_1 . But if we had not backfilled J_3 , J_2 could begin running much sooner. This is an undesirable situation because J_2 is ahead of J_3 in the queue (though perhaps not an unreasonable tradeoff for efficiency).

4.3 Conservative Backfilling

Mu'alem and Feitelson [MF01] compare EASY to an approach they aptly term *conservative backfilling*, in which candidate backfill jobs are checked against *every* job ahead of them in the queue to see if the backfilling action will delay them. This is achieved by giving each job a reservation at the time it is submitted. New jobs may backfill as long as they fit in the cracks between the existing reservations.

One concern would be that following this more restrictive policy would give less efficient scheduling than the more aggressive EASY. But their results show that for many workloads, conservative backfilling does not result in a loss in performance compared with the EASY algorithm. Performance was measured in terms of utilization and average *bounded slowdown*:

$$\text{BoundedSlowdown} = \begin{cases} (\text{WaitTime} + \text{RunTime})/\text{RunTime} & \text{if RunTime} > 10 \text{ seconds} \\ (\text{WaitTime} + \text{RunTime})/10 & \text{otherwise} \end{cases} \quad (5)$$

Average response time attempts to quantify the inconvenience of running on a shared system, by averaging the *wait time* for each job, weighted by the (wall-clock) *run time* of the job. The value is bounded at 10 seconds, to prevent very small jobs, whose absolute wait time is small, though not relatively so, from unduly skewing the average.

Performance being equal, conservative backfilling is preferred because users are able to know their (worst-case) start time at the time the job is submitted. Then, like the EASY creators they corrected, the authors make the claim that conservative backfilling "guarantees that future arrivals do not delay previously queued jobs." [MF01] But the authors of the Maui scheduler [JSC01] in turn proved this claim wrong as well.

Although the reservation system in conservative backfilling guarantees that a job will never start later than its originally reserved time, backfilling jobs ahead of it may still cause it to start later than it might have otherwise. This delay-by-stolen-opportunity is called *pseudo-delay*; perhaps not the best name because the effects of pseudo-delay are quite real. This apparently contradictory situation arises because

jobs may not, and most often do not, end up using the full time they request. So when the scheduler checks that a backfill candidate job will finish before or at the same time as other critical-path jobs, it is relying on information that is most often not correct (the problem of unreliable user input will be discussed in more detail in Section 6).

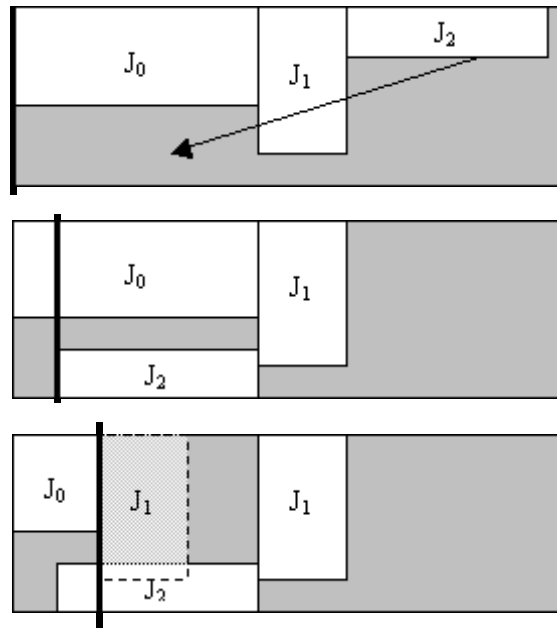


Figure 3. Pseudo-Delay

Figure 3 shows an example scenario, with jobs J_0 , J_1 and J_2 , queued in that order. The y-axis represents the machine's processors, and the x-axis represents time, with the current time marked by a heavy black line. J_0 is already running, and there are not enough free processors to start J_1 . There are enough free processors to start J_2 , and J_2 is scheduled to complete soon enough that backfilling J_2 will not interfere with J_1 's reservation. Then, according to conservative backfilling algorithm, J_2 is started. But a moment later, J_0 ends before its full requested runtime has elapsed. Had we not started J_2 , then J_1 could have started as soon as J_0 ended. But according to our non-preemption assumption, the decision to start J_2 is irrevocable now that J_2 has begun running. Although J_1 has not been delayed past its scheduled start time, it has still been prevented from running as soon as it could have, by a job that was behind it in the queue.

5 Priority

"All animals are equal, but some animals are more equal than others."

—George Orwell, *Animal Farm*

5.1 Introduction to Priority

As MPP supercomputer systems grew in size and number of users, researchers increasingly focused on the reality that a real-world solution requires not only high performance in terms of utilization, but support for a variety of policy goals of the organization owning the system. These goals are influenced by funding agency pressures, users' expectations from previous experiences, site administrators' preference for some projects over others, more immediate demands for some data (e.g. weather forecast), users' research paper deadlines, and so on.⁵ To codify these goals into policies, schedulers implement priority.

⁵ An apt priority-related anecdote from very early supercomputer history is the following: "IBM had been very generous to MIT in the fifties and sixties, donating its biggest scientific computers. When a new top of the line 36-

It is important to note that priority was certainly not invented at this stage. The notion of priority appears throughout many areas in the field of Computer Science. Within supercomputer scheduling, advanced priority options were available on many systems that predate the MPP systems discussed here. For example, jobs on the single-processor, time-sharing Cray XMP 48 at the San Diego Supercomputer Center (SDSC) were assigned a floating-point priority value between 0 and 2 by their owners. Jobs received shares of the processor according to their priority. If too many jobs were active, those with the lowest priority would be held until their owners increased their priority, or the system became less impacted. Users could change a job's priority at any time and as often as they pleased, whether the job was being held or in the process of running. According to Pfeiffer [P04], many users spent much of their time monitoring the current list of jobs like a stock market ticker tape, constantly adjusting their jobs' priorities to achieve the best performance for the least expense. Some users even wrote scripts to parse the list and manage priorities, to try to game the system. The prevailing priority at a given time became a congestion price—peaking during business hours and dropping over nights and weekends (for a related discussion of congestion pricing of computer network resources, see [GLS01]).

Such flexibility and user engagement are not seen in contemporary systems, even in scheduling systems with (relatively) advanced features like those to be discussed below. Much like the "primitive" tennis-court scheduling, which is still a lofty user-interface goal, the old Cray priority system remains an instructive example.

5.2 The Maui Scheduler

The Maui project [JSC01] originally had a single goal: maximize utilization. Developers soon realized that support for complex administrative and policy objectives would be required. So in addition to efficiency-enhancing backfilling, Maui features advanced job prioritization and *fairshare*.

Now widely deployed, the Maui scheduler represents a mature software product. The authors' claim in the paper is still true, that Maui is "currently in use on...systems throughout the world including a high percentage of the largest and most cutting edge research sites." [JSC01]

Maui's backfill algorithm generalizes the existing EASY and conservative backfilling approaches into a parameterized algorithm. Recall that conservative backfilling permits backfilling only when no job ahead of the backfilling job will have its reservation disrupted, while EASY permits backfilling when the first job in the queue's reservation is not disrupted. Maui has a parameter RESERVATIONDEPTH, by default set to 1 to indicate EASY-style backfilling, which controls the number of jobs for which reservations should be enforced.

As described above, the core of Maui's scheduling employs a queue of jobs. But unlike the EASY scheduler, which ordered the queue based solely on arrival time, Maui's queue is ordered according to a complex set of interconnected factors, configurable for each system. This enables system administrators to enact a wide variety of policy goals. Table 1 shows the job, user and system conditions that may comprise the final priority number for a job (table taken from [JSC01]).

Priority Component	Evaluation Metrics	Use
Service	Current queue time and expansion factor.	Allows favoring jobs with lowest current scheduling performance (promotes balanced delivery of job queue time and expansion)

big scientific machine came out, MIT expected to get one. In the early sixties, the deal was that MIT got one 8-hour shift, all the other New England colleges and universities got a shift, and the third shift was available to IBM for its own use. One use IBM made of it was yacht handicapping: the president of IBM raced big yachts on Long Island Sound, and these boats were assigned handicap points by a complicated formula. There was a special job deck kept at the MIT Computation Center, and if a request came in to run it, operators were to stop whatever was running on the machine and do the yacht handicapping job immediately." [V95]

		factor).
Requested Resources	Requested processors, memory, swap, local disk, nodes and processor-equivalents.	Allows favoring of jobs which meet various requested resource constraints (e.g. favoring large processor jobs counters backfills proclivity for smaller jobs and improves overall system utilization).
Fairshare	User, group, account, QoS, and class fairshare utilization.	Allows favoring jobs based on historical usage associated with their credentials.
Direct Priority Specficiation	User, group, account, QoS, and Class administrator specified priorities.	Allows political priorities to be assigned to various groups.
Target	Current delta between measure and target queue time and expansion factor values.	Allows ability to specify service targets and enable non-linear priority growth to enable a job to reach this service target.
Bypass	Job bypass count.	Allows favoring of jobs bypassed by backfill to prevent backfill based job starvation.

Table 1: Maui Priority Components

Fairshare is perhaps the most unique feature of Maui; it allows administrators to specify what percentage of the machine should go to different groups. Here, percentage of the machine is defined as the percentage only of the "actual delivered utilization, not configured or available" resources. The fairshare calculation includes not the number of processor-hours, but arbitrary other resources as well (e.g. memory and disk), by using configurable exchange factors, to convert any resource amount unit to processor hours. The authors advise that fairshare may not always be enforceable by the scheduler; for example, if a group does not submit enough jobs to use their share.

5.3 A Microeconomic Scheduler

A scheduling algorithm that takes the feature of customizable priorities to an even further level is the Microeconomic Scheduler (MS) proposed by Stoica, Abdel-Wahab and Pothen [SAWP95]. MS establishes a marketplace wherein users compete for access to processors, using a currency allowance allocated to them by the system. The design philosophy of MS encompassed two main goals, "to ensure fairness in resource allocation among the users ... [and] to give the user flexibility in controlling the relative share of resources allocated among his jobs."

MS requires users establish individual expense accounts for each of their jobs. The expense accounts are configured to draw currency from the main account at customizable rates, thereby allowing users to differentiate with fine granularity the relative importance of their various jobs. More importantly, users are describing the relative importance of their jobs, compared with the jobs of other users, using a common measure (the currency).

Some novel features of the system include a policy wherein users are charged not only for the processor time their job consumes while running, but for the processor time spent idle before their job runs, while the job is waiting for more processors to become available. By explicitly charging users for the schedule holes created in their jobs' wake, the system discourages user practices that tend to cause fragmentation. MS, unlike other schedulers, does not necessarily kill a job when its requested runtime has expired, instead allowing it to continue running as long as it continues to outbid other jobs for the processors it occupies. This policy benefits users while still holding them accountable to the system.

These examples show the diverse and unconventionally high-level policy objectives that can be achieved with the flexible and powerful microeconomic mechanism. The authors believe "that the microeconomic paradigm may serve as a unifying theme for multiprocessor scheduling" [SAWP95]. Feitelson and Rudolph share this view, claiming that the diverse sub-research-areas of scheduling (e.g.

gang scheduling, dynamic partitioning) can converge under a framework with similar philosophy, by associating cost functions with various user behaviors and job characteristics [FR96].

For example, the 2-dimensional, rectangular job specification assumed in this paper does not reflect the computational requirements of many jobs, which may have naturally alternating serial and parallel phases. Other scheduling disciplines can reduce the effects of this internal fragmentation by allowing jobs to formally express their plans to occupy different numbers of processors at different times (called *evolving jobs*), so unused processors can be assigned to other jobs. The two disciplines can be fused through a cost function that provides an incentive for users to spend the extra effort to rewrite their code in a manner that provides this additional information and functionality.

6 Reliability of User-Provided Inputs

In classic and recent papers on parallel job scheduling, a perennial problem that is mentioned is the inaccuracy of user estimates of their jobs' runtime. This estimate, the requested runtime, is a parameter users provide, along with the number of processors required, as part of job submission. The scheduler uses it to plan a schedule for the jobs in its queues. However, due to inherent difficulty in predicting execution times of codes on computers, especially parallel computers, as well as unforeseen crashes and the like, jobs almost never use the full requested amount of time.

Particularly in backfilling and related schemes, the scheduler critically relies on the accuracy of the requested runtime to determine if a job will finish in time to not delay jobs it skips ahead of. Hence almost all systems enforce the requested runtime as an upper bound by killing jobs that exceed their time.

Many papers, including [CB01, MF01, SKSS02, CADV02, LS02] have characterized the distribution of the error in requested runtimes in various real workload traces. Cirne and Berman [CB01] found that in four different traces, 50-60% of jobs use less than 20% of their requested time. The consensus is that the estimates are similarly highly inaccurate in all the workloads analyzed.

On systems where jobs are killed when the requested time expires, Feitelson hypothesizes [MF01] that users have less incentive to provide tight estimates, which may result in their jobs being able to squeeze into backfill slots, than they have to provide over estimates, which will avoid having their jobs killed early.

While some scheduling algorithms are unaffected by inaccurate runtime estimates [MF01], many schedulers experience significant performance degradation as a result of this phenomenon [SKSS02, CADV02, LS02]. Intuitively, the scheduler is making optimization decisions based on imprecise information from users—the lack of precise information causes waste.

I claim that calling the time users provide when they submit their job an "estimate" of its runtime is incorrect. Some users may have a true estimate in mind, but add generous padding to it, because the chance of having their job killed is so objectionable. An interesting question is, how accurate could users be, given the proper incentives and without the threat of death for their job? Colleagues at the San Diego Supercomputer Center and myself have recently undertaken such a study, using various consumer electronics items as gifts to reward user accuracy, and assuring users that their jobs would not be killed when their estimated time had elapsed [L04]. Results show that while nearly half of users provided an improved estimate, cutting on average 35% from their previous requested runtime, the resulting new estimates were not substantially more accurate overall. This study confirms that while users' behavior is responsive to incentive stimuli, it is unlikely that as a group they could ever produce reliably accurate runtime estimates (at least in their current circumstances).

7 Measurement and Evaluation

7.1 Introduction and Common Metric Pitfalls

Careful metric selection clearly plays a key role in scheduling research. Every metric hides some implicit assumptions. Utilization, one of the earliest and most commonly reported metrics, assumes that jobs make productive use of the processors they are assigned. Administrators at NAS noted that on their Intel iPSC/860 system (1990-1994), the constraint on processor partition sizes (required to be power of

two) caused internal fragmentation when jobs actually only used, for example, 2^n+1 processors [JN99]. Additionally, although many scheduling mechanisms and policies explicitly encourage users to run on high processor counts (often for political reasons), Amdahl's Law dictates that many parallel jobs would be more efficient if run on fewer processors [G88, B91].

Another property of common metrics is that they measure the workload as much as the measure the scheduler. For example, many systems with periods of low utilization suffer not from poor scheduling, but from low volume of jobs. In [JN99], different periods of particularly low utilization are explained variously as users leaving the system "to avoid the frustrations of trying to use a system in flux," and code migration and debugging on a newly installed system. Metrics such as response time and expansion factor are also clearly affected by job volume. Subtler workload characteristics that affect scheduling metrics include: percentage of jobs requesting power-of-two processor counts, (in)accuracy of user-provided requested runtimes, relationship between job length and width, and presence of rigidly scheduled reservations (for example planned maintenance outages) [MF01, FN95].

7.2 Performance Distribution Across Jobs

In addition to the classic metrics already discussed throughout this paper, many other metrics appear in the literature. Some of the more useful contributions are those that attempt to express the distribution of the scheduler's effects on all jobs.

The pitfalls focusing exclusively on a single collapsed value are evident in [ZK99, ZKP00]. After noting that "the typical metric used to evaluate the performance of job schedulers, average slowdown, is heavily influenced by short jobs," the authors suggest, apparently without irony, that the solution is to run all jobs in order of size, smallest to largest. Again, a more constructive solution is to use more complete evaluation techniques.

Chiang et al. [CADV02] not only report performance in terms of average wait time, but also 95th percentile wait time, and maximum wait time. Thus while trying to optimize average wait time, they do not lose sight of the effects on the most heavily impacted jobs. An indirect way to keep track the poorest-performing jobs is used by Ernemann et al. [EHY02], who in their average give more emphasis to wide (many-processor) jobs, which are by their nature often harder to fit into a schedule. The metric employed is *average weighted response time (AWRT)*, which is the average over all jobs of:

$$WRT = (RunTime + WaitTime) * Processors * RequestedRunTime \quad (6)$$

Srinivasan et al. [SKSS02] explicitly consider the scheduler's effects on jobs of different types, by presenting their results itemized by job shape, as shown in Table 2:

	≤ 8 Processors	> 8 Processors
≤ 1 Hour	SN	SW
> 1 Hour	LN	LW

Table 2: Categories of Job Shapes

Comparing results for individual size categories prevents a false appearance of good scheduler performance that can happen with a single average, which can obscure highly detrimental effects suffered by some minority class of jobs. Another example of more complete evaluation is Cirne and Berman [CB03], who simply present CDFs of performance measures, showing the results for every individual job.

7.3 High-Level Definitions of Performance

Response time, bounded slowdown and expansion factor are all metrics that are designed to capture the users' desires to not be kept waiting. User-adjustable priorities are one crude mechanism for communication between the user and scheduler, describing the relative urgency of a job. But the interplay

between users and the completion of their jobs involves a richer context than is captured with these numbers—users have their own daily schedules and preferences, externally imposed schedules and preferences (e.g. conference paper deadlines), and so on. Virtually all supercomputer workload traces show diurnal patterns, with more small (presumably debugging) jobs during the day, and only very infrequent new job submissions at night. If a scheduler sacrifices utilization in favor of very good response time for a particular job at 2 a.m., the effort is most likely wasted—the user will not even check the job until morning—but such a tradeoff could be highly desirable during peak business hours. Feitelson et al. present an example scenario to use as a starting point for discussion about the true needs and desires of users.

Assume that a job i needs approximately 3 hours of computation time. If the user submits the job in the morning (9am) he may expect to receive the results after lunch. It probably does not matter to him whether the job is started immediately or delayed for an hour as long as it is done by 1pm. Any delay beyond 1pm may cause annoyance and thus reduce user satisfaction, i.e. increase costs. This corresponds to tardiness scheduling. However, if the job is not completed before 5pm it may be sufficient if the user gets his results early next morning. Moreover, he may be able to deal with the situation easily if he is informed at the time of submittal that execution of the job by 5pm cannot be expected. Also, if the user is charged for the use of system resources, he may be willing to postpone execution of his job until nighttime when the charge is reduced. [FRSSW97]

The *utility function* $u(t)$, or the change in user satisfaction over time, implied by Feitelson's scenario is not a simple linear function, as is implicitly assumed in metrics such as response time. There are discontinuities (e.g. at 1 p.m.) and periods where the slope is zero (e.g. between 5 p.m. and the following morning). Furthermore, it is likely that every (user, job) pair will have a function with a different pattern.

A utility function-gathering survey of users of an IBM Power3 system at the San Diego Supercomputer Center indicates a wide range of satisfaction levels for every proposed length of wait time [L04]. In connection with this survey, a scheduler that explicitly optimizes for these utility functions was proposed. This *delay-cost* scheduler, similar to the time-sharing delay-cost scheduler proposed by Franaszek and Nelson [FrN95], charges users according to the loss in value that the running of their job inflicts on other jobs through delay. Calculating this optimization based on user utility functions is a mixed-integer program (a linear programming problem with integer constraints on some variables), which is NP-hard. Notwithstanding, commercial solver software packages can achieve good solutions in very reasonable timeframes (on the order of seconds), on problem sizes that reflect most current systems and workloads. [L04]

7.4 Evaluation Beyond Performance

For a variety of applications, among them appointment alerts and reminders [DA00], delivery of messages and negotiation of message urgency [HS00] and communication [LI00], it is commonplace to use as a measuring stick the example of what a human would do in place of the proposed software system. For the example of message delivery, a human executive assistant employs complex and nuanced analysis of both the incoming messages for the executive, and the executive's current state, to determine which calls and messages should go through and which can wait or be ignored. For example, a phone call from a sick child warrants interrupting a meeting with a customer, whereas a phone call from the same child regarding dinner plans would best be returned at a later time.

As perhaps occurred with tennis court scheduling, human schedulers can perform many similarly advanced and nuanced functions to the human executive assistant—functions that are not implemented by existing or even proposed scheduling interfaces. Even very simple offerings, nothing more than providing an estimate of when the scheduler will start a currently queued job, are not offered on many systems (SDSC's Catalina scheduler [Y04] does have this feature). A visualization of the scheduler's currently planned layout for upcoming jobs (similar to the block diagrams of Figures 2 and 3 in this paper) was recently implemented for the Blue Horizon system and SDSC [L04]. Such features that educate users on

the scheduler's operation can make an enormous difference in the overall user satisfaction with the system; it is unfortunate they are so commonly overlooked.

8 Future Work

User time estimate inaccuracy is just one example of a detrimental effect that can arise due to poor mapping between the behaviors and desires of people, and assumptions made implicitly or explicitly by scheduler designers. Especially with emerging grid computing environments linking a multitude of diverse resources and users into virtual societies, such distinctively human behaviors are likely to cause even more problems for naïvely designed scheduling systems.

I plan to contribute to the development of a scheduling system whose function is to not just tolerate but harness the human tendencies of its users and direct them in a productive way. A model example of a large-scale system of this sort is a capitalistic economy. I anticipate that my research will draw much inspiration from principles of Economics. The work of Wolski [WPBB01] and others [SAWP95] also suggests market-based systems as the most rational way of dealing with complex, multi-party environment of scheduling parallel and grid resources.

A major contribution I plan to make is the identification and/or development of specific and well-defined metrics for these and other appropriate criteria. The classic type of metrics exemplified by utilization and response time will not sufficiently capture the design requirements of this next generation of scheduling systems.

I also hope to reflect insights from solutions to the, in many ways harder, problem of grid scheduling, back to the traditional setting of a single-machine parallel job scheduler. Especially in the early phases of development of the grid infrastructure, which are already taking place, the dominant architecture seems to be a hybrid of locally-scheduled traditional parallel machines, and networked sharing of resources.

9 Acknowledgements

The author wishes to acknowledge the editorial advice of her advisor Allan Snively on a draft of this paper. The author also wishes to thank those colleagues who participated in a reading group that reviewed some of the papers discussed here. The group included, at various times, Allan Snively and Larry Carter, of UCSD and SDSC, and Kenneth Yoshimoto, Jennifer Hardy and Yael Schwartzman, all of SDSC (log of reading group activity available on request). Wayne Pfeiffer and Michael Wan, of SDSC, graciously provided personal interviews covering historical practices of parallel job scheduling that were missing from the literature. David H. Bailey, of Lawrence Berkeley Laboratory, provided verification of some background information, in particular historical scheduling practices and typical current costs of supercomputer systems. Larry Carter, Julie Conner, Allan Snively and Keith Marzullo provided general advice and encouragement on how to present a successful research exam.

References

- [B91] Bailey, David. "Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers," RNR Technical Report, RNR-90-020, NASA Ames Research Center, 1991.
- [CADV02] Chiang, Su-Hui, Andrea Arpaci-Dusseau and Mary Vernon. "The Impact of More Accurate Requested Runtimes on Production Job Scheduling Performance," *Proc. of the 8th Workshop on Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, eds. July 2002.
- [CB01] Cirne, Walfredo and Francine Berman. "A comprehensive model of the supercomputer workload," *Proceedings of IEEE 4th Annual Workshop on Job Scheduling Strategies for Parallel Processing*. Cambridge, MA. 2001.
- [CB03] Cirne, Walfredo and Francine Berman. "When the Herd Is Smart: Aggregate Behavior in the Selection of Job Request." *IEEE Transactions on Parallel and Distributed Systems* 14(2): 181-192. 2003.

- [CHPC] Jackson, David. HPC workload repository. <http://www.supercluster.org/research/traces>. Technical report.
- [DA00] Dey, Anind K. and Gregory D. Abowd. "CybreMinder: A Context-Aware System for Supporting Reminders." *Proc. of the 2nd International Symposium on Handheld and Ubiquitous Computing (HUC2K)*, Bristol, UK, September 25-27, 2000.
- [EHY02] Ernemann, Carsten, Volker Hamscher and Ramin Yahyapour. "Economic Scheduling in Grid Computing," *Proc. of the 8th Workshop on Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, eds. July 2002.
- [FN95] Feitelson, Dror G. and Bill Nitzberg. "Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860," *Proc. of the 1st Workshop on Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph, eds. April 1995.
- [FR96] Feitelson, Dror G. and Larry Rudolph. "Toward convergence in job schedulers in parallel supercomputers," *Proc. of the 2nd Workshop on Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph, eds. April 1996.
- [FrN95] Franaszek, Peter A. and Randolph D. Nelson. "Properties of delay-cost scheduling in time-sharing systems," *Journal of Research and Development*, 39(3). 1995.
- [FRSSW97] Feitelson, Dror G., Larry Rudolph, Uwe Schwiegelsohn, Kenneth C. Sevcik and Parkson Wong. "Theory and Practice in Parallel Job Scheduling." *Proc. of the 3rd Workshop on Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph, eds. April 1997.
- [G88] Gustafson, John L., "Reevaluating Amdahl's Law." *CACM*. 31(5), 1988.
- [G04] Goguen, Joseph. Course notes: *User Interface Design: Social and Technical Issues*. <http://www.cs.ucsd.edu/users/goguen/courses/271/1.html>.
- [GJ79] Gary, Micheal R. and David S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York. 1979.
- [GLS01] Ganesh, Ayalvadi, Koenraad Laevens and Richard Steinberg. "Congestion pricing and user adaptation," *IEEE Infocom*, April 2001.
- [H96] Hotovy, Steven. "Workload evolution on the Cornell Theory Center IBM SP2," *Proc. of the 2nd Workshop on Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph, eds. April 1996.
- [HS00] Hollan, James and Scott Stornetta. "Asynchronous Negotiated Access." *Proceedings of Human Computer Interaction*, 2000.
- [JN99] Jones, James Patton and Bill Nitzberg. "Scheduling for Parallel Supercomputing: A Historical Perspective of Achievable Utilization," *Proc. of the 5th Workshop on Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph, eds. April 1999.
- [JSC01] Jackson, David, Quinn Snell and Mark Clement. "Core Algorithms of the Maui Scheduler," *Proc. of the 7th Workshop on Job Scheduling Strategies for Parallel Processing*, Dror G. Feitelson, Larry Rudolph, eds. June 2001.
- [KLDR94] Krueger, Phillip, Ten-Hwang Lai and Vibha A. Dixit-Radiya. "Job Scheduling Is More Important than Processor Allocation for Hypercube Computers," *IEEE Transactions on Parallel and Distributed Systems*, 5(5). May 1994.
- [L04] Lee, Cynthia, Allan Snavely, Robert H. Leary, Laura Carrington, Henri Casanova, Roger Bohn, Richard Carson, Jennifer Hardy and Yael Schwartzman. "Towards High-Order Performance Objectives for HPC System Scheduling." San Diego Supercomputer Center Technical Report. March 9, 2004.
- [LI00] Liechti, Oliver and Tadao Ichikawa. "A Digital Photography Framework Enabling Affective Awareness." *Home Communication, Personal Technologies*, 4(1), 2000.
- [LS02] Lawson, Barry and Evgenia Smirni. "Multiple-Queue Backfilling Scheduling with Priorities and Reservations for Parallel Systems," *Proc. of the 8th Workshop on Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, eds. July 2002.
- [M93] Messina, Paul C., "The Concurrent Supercomputing Consortium: year 1," *Parallel & Distributed Technology*, 1(1). February 1993..

- [MF01] Mu'alem, Ahuva W. and Feitelson, Dror G. "Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling," *IEEE Trans. Parallel & Distributed Systems*, 12(6). June 2001.
- [O46] Orwell, George. *Animal Farm*. Harcourt, Brace and Company. New York. 1946.
- [P04] Pfeiffer, Wayne. Personal Interview. San Diego Supercomputer Center, at the University of California, San Diego. La Jolla, CA. April 9, 2004.
- [SAWP95] Stoica, Ion, Hussein Abdel-Wahab and Alex Pothen. "A Microeconomic Scheduler for Parallel Computers," *Proc. of the 1st Workshop on Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph, eds. April 1995.
- [SCZL96] Joseph Skovira, Waiman Chan, Honbo Zhou, and David Lifka. "The EASY --- LoadLeveler API Project," *Proc. of 2nd Workshop on Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph, eds. April 1996.
- [SKSS02] Srinivasan, Srividya, Rajkumar Kettimuthu, Vijay Subramani and P. Sadayappan. "Characterization of Backfilling Strategies for Parallel Job Scheduling," *Proceedings of 2002 International Workshops on Parallel Processing*, August 2002.
- [STT95] Saphir, William, Leigh Ann Tanner and Bernard Traversat. "Job Management Requirements for NAS Parallel Systems and Clusters." NAS Technical Report NAS-95-006. NAS Scientific Computing Branch, NASA Ames Research Center. February 1995.
- [V95] Van Vleck, Tom. "The IBM 7094 and CTSS." *www.Multicians.com* (online history of Multics). <http://www.multicians.org/thvv/7094.html>
- [W04] Wan, Michael. Personal Interview. San Diego Supercomputer Center at the University of California, San Diego. La Jolla, CA. April 9, 2004.
- [WMKS96] Wan, Michael, Reagan Moore, George Kremenek and Ken Steube. "A Batch Scheduler for the Intel Paragon with a Non-contiguous Node Allocation Algorithm." *10th International Parallel Processing Symposium*. April 1996.
- [WPBB01] Wolski, Rich, James Plank, John Brevik and Todd Bryan. "Analyzing Market-based Resource Allocation Strategies for the Computational Grid." *The International Journal of High Performance Computing Applications*, Vol. 15: 3. 2001.
- [Y04] Yoshimoto, Kenneth. Catalina scheduler. <http://www.sdsc.edu/catalina/>
- [ZK99] Zotkin, Dmitry and Peter J. Keleher. "Job-length estimation and performance in backfilling schedulers." *8th High Performance Distributed Computing Conference*. IEEE. 1999.
- [ZKP00] Zotkin, Dmitry, Peter J. Keleher. and Dejan Perkovic. "Attacking the Bottlenecks of Backfilling Schedulers." *Cluster Computing: The Journal of Networks, Software Tools and Applications*, 3(4): 2000.