Pitfalls in Parallel Job Scheduling Evaluation

Eitan Frachtenberg[1] and Dror G. Feitelson[2]

[1] Modeling, Algorithms, and Informatics Group,
Los Alamos National Laboratory
eitanf@lanl.gov
[2] School of Computer Science and Engineering,
The Hebrew University, Jerusalem, Israel
feit@cs.huji.ac.il

Abstract. There are many choices to make when evaluating the performance of a complex system. In the context of parallel job scheduling, one must decide what workload to use and what measurements to take. These decisions sometimes have subtle implications that are easy to overlook. In this paper we document numerous pitfalls one may fall into, with the hope of providing at least some help in avoiding them. Along the way, we also identify topics that could benefit from additional research.

Keywords: parallel job scheduling, performance evaluation, experimental methodology, dynamic workload, static workload, simulation.

1 Introduction

Parallel job scheduling is a rich and active field of research that has seen much progress in the last decade [31]. Better scheduling algorithms and comparative studies continually appear in the literature as increasingly larger scale parallel systems are developed and used [70]. At the same time, parallel job scheduling continues to be a very challenging field of study, with many effects that are still poorly understood. The objective, scientific difficulties in evaluating parallel job schedulers are exacerbated by a lack of standard methodologies, benchmarks, and metrics for the evaluation [9,27].

Throughout the years, the authors have encountered (and sometimes committed) a variety of methodological leaps of faith and mistakes that are often recurring in many studies. In this paper we attempt to sum up the experience gleaned from ten years of the workshop on job scheduling strategies for parallel processing (JSSPP).[1] Our main goal is to expose these topics in a single document with the hope of helping future studies avoid some of these pitfalls.

To limit this paper to a reasonable size, we chose to exclude from the scope of this paper topics in static and DAG scheduling, which is a separate field that is not generally covered by the JSSPP workshop. Additionally, grid scheduling is not specifically targeted, although many of the topics that are covered bear relevance to grid scheduling.

[1] www.cs.huji.ac.il/~feit/parsched/

D. Feitelson et al. (Eds.): JSSPP 2005, LNCS 3834, pp. 257–282, 2005.

Naturally, not all pitfalls are relevant to all classes of evaluation or scheduling strategies (e.g., time slicing vs. space slicing). Others, such as those related to the choice of workload, affect almost any quantitative evaluation. As a notational convention, we marked each pitfall with one to three lightning bolts, representing our perception of the severity of each item.

Several of the pitfalls we list may not always represent a methodological mistake. Some choices may be correct in the right context, while others may represent necessary compromises. Some of these suggestions may even seem contradictory, depending on their context. Many topics and methodological choices remain open to debate and beg for further research. But it is necessary that we remain cognizant of the significance of different choices. We therefore recommend not to consider our suggestions as instructions but rather as guidelines and advisories, which are context-dependent.

We have made a deliberate choice not to point out what we perceive as mistakes in others' work. Instead, we described each pitfall in general terms and without pointing to the source of examples. When detailing suggestions, however, we attempted to include positive examples from past works.

The list of pitfalls we present in this paper is by no means exhaustive. The scope of this paper is limited to issues specific to our field, and does not cover methodological topics in general. For example, we exclude general problems of statistics, simulation techniques, or presentation. Similarly, issues that are are not very significant to parallel job scheduling evaluation and minor methodological problems were left out. Still, we wish to enumerate at this point the following general principles, that should be considered for any performance evaluation study:

- Provide enough details of your work to allow others to reproduce it.
- Explore the parameter space to establish generality of results and sensitivity to parameters.
- Measure things instead of assuming they are so, even if you are sure.

The designer of a parallel job scheduler is faced with myriad choices when reaching the evaluation stage. First, the researcher must choose a workload (or more than one) on which the scheduler is evaluated. This workload may reflect choices typical for a researcher's site or could try to capture the salient properties of many sites. Either way, the choice of a good workload structure entails many tricky details. Inexorably related to the workload structure are the workload applications that must next be chosen. The evaluation applications, whether modeled, simulated, or actually run, also have a large impact's on the evaluation's results, and must be chosen carefully. Next, experiments must be designed, and in particular, a researcher must choose the factors (input parameters) to be evaluated and the metrics against which the scheduler is measured. Here too, different choices can have radically different results. For example, scheduler A might have lower average response time than scheduler B, but also be more unfair. A different choice of input parameters could reverse this picture. A good choice of input parameters and metrics can typically not be done in isolation of the measurements, but rather, after careful examination of the effect on each of

the evaluated system. Lastly, the measurement process itself entails many traps for the unwary researcher, which could be averted with a systematic evaluation.

We group pitfalls into groups that loosely reflect the order of choices made for a typical job scheduling study. We start in Section 2 with pitfalls relating to workload structure. Section 3 delves into what comprises the workload, namely, the applications. The input parameters for the evaluation are considered in Section 4. Next, Sections 5 and 6 discuss methodological issues relating to measurement and metric choices. Finally, we conclude in Section 7.

2 Workload Structure

Workload issues span all methods of parallel job scheduling evaluation. The choices and assumptions represented in the tested workloads may even determine the outcome of the evaluation [21,30]. With experimental evaluations on real systems, our choice of workload is often limited by practical issues, such as time or machine availability constraints. Analysis may be limited by considerations of mathematical tractability. Simulation studies are typically less limited in their choice of workload [75].

As a general rule of thumb, we recommend using multiple different workloads, each of which is long enough to produce statistically meaningful results. Several workload traces and models can be obtained from the Parallel Workload Archive [54]. Whether using an actual trace or a workload model, care must be given to specific workload characteristics. For example, a workload dominated by power-of-two sized jobs will behave differently from one containing continuously sized jobs [47]. It is important to understand these workload characteristics and their effect on the evaluation's results, and if possible, choose different workload models and compare their effect on the evaluation [1,21].

This section lists the challenges that we believe should be considered when defining the workload structure. Many of these pitfalls can be summarized simply as "employing overly-simplistic workload models". Given that better data is often available, there is no justification to do so. Therefore, whenever possible we would suggest to use realistic workloads, and moreover, to use several different ones.

Pitfall 1 ϟϟϟ

Using invalid statistical models

Problem. Workload models are usually statistical models. Workload items are viewed as being sampled from a population. The population, in turn, is described using distributions of the various workload attributes.

A model can be invalid, that is, not representative of real workloads, in many different ways. The most obvious is using the wrong distributions. For example, many researchers use the exponential distribution for interarrival times, assuming that arrivals are a Poisson process. This ignores data about self similarity (pitfall 4), and also ignores irregularities and feedback effects, as well as job resubmittals [22,38]. Some studies even use uniform

distributions, e.g., for the parallelism (size) of jobs, which is entirely unrepresentative; a more representative distribution favors small jobs (e.g. the log-uniform), and moreover is modal, emphasizing powers of two [14,16,47].

To be fair, finding the "right" distribution is not always easy, and there are various methodological options that do not necessarily produce the same results. For example, even distributions that match several moments of the workload data may not match the shape of the distributions, especially the tail. The story does not end with distributions either: it is also important to model correlations between different attributes, such as job size, interarrival time, and length [14,20,47].

Suggestions. If a workload model is used, one must ensure that it is a good one. The alternative is to use a real trace. This has the advantage of including effects not known to modelers, but also disadvantages like abnormal data (pitfall 5).

Research. Workload modeling is still in its infancy. There is much more to learn and do, both in terms of modeling methodology and in terms of finding what is really important for reliable performance evaluations. Some specific examples are mentioned in the following pitfalls.

Pitfall 2 ϟϟϟ

Using only static workloads

Problem. Static workloads are sets of jobs that are made available together at the beginning of the evaluation, and then executed with no additional jobs arriving later — akin to off-line models often assumed in theoretical analyses. This is significantly different from real workloads, where additional jobs continue to arrive all the time.

Static workloads are used for two reasons. One is that they are easier to create (there is no need to consider arrivals), and are much smaller (fewer jobs take less time to run). The other is that they are easier to analyze, and one can in fact achieve a full understanding of the interaction between the workload and the system (e.g. [33]). While this may be useful, it cannot replace a realistic analysis using a dynamic workload. This is important because when a static workload is used, the problematic jobs tend to lag behind the rest, and in the end they are left alone and enjoy a dedicated system.

Suggestions. It is imperative to also use dynamic workloads, whether from a trace or a workload model.

Research. An interesting question is whether there are any general principles regarding the relationship of static vs. dynamic workloads. For example, are static workloads always better or worse?

Pitfall 3 ϟϟ

Using too few different workloads

Problem. Workloads from different sites or different machines can be quite different from each other [68]. Consequently, results for one workload are not

necessarily valid for other workloads. For example, one study of backfilling based on three different workloads (one model and two traces) showed EASY and conservative to be similar [32], but a subsequent study found that this happens to be a nonrepresentative sample, as other workloads bring out differences between them [51].

Suggestions. Whenever possible, use *many* different workloads, and look for invariants across all of them; if results differ, this is a chance to learn something about either the system, the workload, or both [21].

Research. It is important to try to understand the interaction of workloads and performance results. *Why* are the results for different workloads different? This can be exploited in adaptive systems that learn about their workload.

Pitfall 4 

Ignoring burstiness and self-similarity

Problem. Job arrivals often exhibit a very bursty nature, and realistic workloads tend to have high variance of interarrival times [22,67]. This has a strong effect on the scheduler, as it sometimes has to handle high transient loads. Poisson models make the scheduler's life easier, as fluctuations tend to cancel out over relatively short time spans, but are not realistic.

Suggestions. Burstiness is present in workload traces, but typically not in models. Thus real traces have an advantage in this regard.

Research. One obvious research issue is how to incorporate burstiness and self-similarity in workload models. This has been done in network traffic models (e.g. [76]), but not yet in parallel job models.

Another issue is the effect of such burstiness on the evaluation. If a model creates bursts of activity randomly, some runs will lead to larger bursts than others. This in turn can lead to inconsistency in the results. The question then is how to characterize the performance concisely.

Pitfall 5 

Ignoring workload flurries and other polluted data

Scope. When using a real workload trace.

Problem. We want workload traces to be "realistic". What we mean is that they should be representative of what a scheduler may be expected to encounter. Regrettably, real traces often include subsets of data that cannot be considered representative in general. Examples include:

- Heavy activity by system administrators [24].
- Heavy activity by cleanup scripts at night.
- Heavy and unusual activity by a single user that dominates the workload for a limited span of time (a workload flurry) [73].

Suggestions. Data needs to be sanitized before it is used, in the sense of removing obvious outlier data points. This is standard practice in statistical analysis. Logs in the Parallel Workloads Archive have cleaned versions, which are recommended.

Research. The question of what to clean is not trivial. At present, this is done manually based on human judgment, making it open to debate; additional research regarding considerations and implications can enrich this debate. Another interesting question is the degree to which cleaning can be automated.

Pitfall 6

using oblivious open models with no feedback

Problem. The common way to use a workload model or trace is to "submit" the jobs to the evaluated scheduler as defined in the model or trace, and see how the scheduler handles them. This implicitly assumes that job submittals are independent of each other, which in fact they are not.
Real workloads have self-throttling. When users see the system is not responsive, they reduce the generation of new load. This may help spread the load more evenly.

Suggestions. Use a combination of open and closed model. A possible example is the repeated jobs in the Feitelson model where each repetition is only submitted after the previous one terminates [16].

Research. Introducing feedback explicitly into workload models is an open question. We don't know how to do it well, and we don't know what its effects will be.

Pitfall 7

Limiting machine usage assumptions

Problem. A related issue to pitfall 3 is the embedding of workload assumptions that are too specific to the workload's site typical usage. Some sites run the same applications (or class of applications) for months, opting to use the machine as a capability engine [36,58]. Others use far more heterogeneous workloads representing a machine running in capacity mode [47,54]. The application and workload characteristics of these two modes can be quite different, and not all evaluations can address both.

Suggestions. If a specific usage pattern is assumed, such as a site-specific or heterogeneous workload, it should be stated, preferably with an explanation or demonstration of where this model is valid [39]. Whenever possible, compare traces from different sites [10,15,47].

3 Applications

The application domain of parallel job schedulers is by definition composed of parallel jobs. This premise predicates that applications used for the evaluation of schedulers include some necessary aspects of parallel programs. An obvious minimum is that they use several processors concurrently. But there are others too.

Virtually all parallel applications rely on communication, but the communication pattern and granularity can vary significantly between applications. The degree of parallelism of an application also varies a lot (depending on application type) between sequential applications—with zero parallel speedup—to highly parallel and distributed applications—with near-linear speedup. Other parameters where parallel applications show high variability include services time, malleability, and resource requirements, such as memory size and network bandwidth. In addition, typical applications for a parallel environment differ from those of a grid environment, which in turn differ from distributed and peer-to-peer applications. This high variability and wide range of possible applications can translate to very dissimilar results for evaluations that differ only in the applications they evaluate. It is therefore vital to understand the different factors that applications imply on the evaluation.

Many job scheduling studies regard parallel jobs as rectangles in processors × time space: they use a fixed number of processors for a certain interval of time. This is justifiable when the discussion is limited to the workings of the scheduler proper, and jobs are assumed not to interact with each other or with the hardware platform. In reality, this is not always the case. Running the same jobs on different architectures can lead to very different run times, changing the structure of the workload [78]. Running applications side by side may lead to contention if their partitions share communication channels, as may happen for mesh architectures [45]. Contention effects are especially bad for systems using time slicing, as they may also suffer from cache and memory interference.

On the other hand, performing evaluations using detailed applications causes two serious difficulties. First, it requires much more detailed knowledge regarding what application behaviors are typical and representative [30], and suffers the danger of being relevant to only a small subset of all applications. Second, it requires much more detailed evaluations that require more time and effort. The use of detailed application models should therefore be carefully considered, including all the tradeoffs involved.

Pitfall 8 ϟϟϟ

Using black-box applications

Scope. When contention between jobs and interactions with the hardware platform are of importance.

Problem. An evaluation that models applications as using P processors for T time is oblivious of anything that happens in the system. This assumption is reasonable for jobs running in dedicated partitions that are well-isolated from each other. However, it does not hold in most systems, where communication links and I/O devices are shared. Most contemporary workload models do not include such detailed data [9], so the interaction between different applications with different properties and the job scheduler are virtually impossible to capture from the workload model alone.

In a simulation context that does not employ a detailed architecture simulator that runs real applications (often impractical when testing large parallel

machines), shortcut assumptions are regularly made. For example, assuming that applications are all bag-of-tasks.

If synthetic applications are measured or simulated, the benchmark designer is required to make many application-related choices, such as memory requirements and degree of locality, communication granularity and pattern, etc. [18]. Another example is the role of I/O in parallel applications, that is often ignored but may actually be relevant to a job scheduler's performance [42,81], and present opportunities for improved utilization [77].

Suggestions. Offer descriptions or analysis of relevant application properties, like network usage [3,42], parallelism [79], I/O [42,53], or memory usage [17,57,61]. If approximating the applications with synthetic benchmark programs or in a simulator, application traces can be used [79]. Based on these traces, a workload space can be created where desired parameters are varied to test the sensitivity of the scheduler to those parameters [81]. In the absence of traces, stochastic models of the relevant applications can be used [17,35,53,57].

Research. Current knowledge on application characteristics and correlations between them is rudimentary. More good data and models are required.

Pitfall 9 ///

Measuring biased, unrepresentative, or overly homogeneous applications

Problem. Many evaluations use applications that are site-specific or not very demanding in their resource requirements. Even benchmarks suites such as the NAS parallel benchmarks (NPB) [6] or ESP [78] can be representative mostly of the site that produced them (NASA and NERSC respectively for these examples), or ignore important dynamic effects, such as realistic job interarrival time [18].

Using "toy" and benchmark applications can be useful for understanding specific system properties and for conducting a sensitivity analysis. However, writing and experimenting with toy applications that have no relevant properties such as specific memory, communication, or parallelization requirements (e.g., a parallel Fibonnacci computation) helps little in the evaluation of a parallel job scheduler. Actual evaluations of scientific applications with representative datasets often take prohibitively long time. This problem is exacerbated when evaluating longer workloads or conducting parameter space explorations [8]. In some cases, researchers prefer to use application kernels instead of actual applications, but care must be taken to differentiate those from the real applications [5].

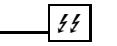
The complexity of evaluating actual applications often leads to compromises, such as shortening workloads, using less representative (but quicker to process) datasets, and selecting against longer-running applications, resulting in a more homogeneous workload that is biased toward shorter applications. On the other hand, a researcher from a real-site installation may prefer to use applications that have more impact on the site [36,39]. This

choice results in an analysis that may be more meaningful to the researcher's site than to the general case.

Suggestions. If possible, evaluate more applications. If choice of applications is limited, qualify the discussion to those limits. Identify (and demonstrate) the important and unimportant aspects of chosen applications. When possible, use longer evaluations or use more than one architecture [39].

Conversely, some situations call for using simple, even synthetic applications in experiments. This is the case for example when we wish to isolate a select number of application characteristics for evaluation, without the clutter of irrelevant and unstudied parameters, or when an agreed benchmark for the studied characteristics is unavailable [7]. To make the workload more heterogeneous, while still maintaining reasonable evaluation delays, a researcher may opt to selectively shorten the run time of only a portion of the applications (for example, by choosing smaller problem sizes).

Research. Research on appropriate benchmarks is never ending.

Pitfall 10 

Ignoring or oversimplifying communication

Problem. Communication is one of the most essential properties that differentiates parallel jobs from sequential (and to some extent, distributed) applications. Using a single simplistic communication model, such as uniform messaging, can hide significant contention and resource utilization problems. Assuming that communication takes a constant proportion of the computation time is often unrealistic, since real applications can be latency-sensitive, bandwidth-sensitive, or topology-sensitive. In addition, contention over network resources with other applications can dramatically change the amount of time an application spends communicating.

In a simulation or analysis context where communication is modeled, one should consider that communication's effect on application performance can vary significantly by the degree of parallelism and interaction with other applications in a dynamic workload. This is only important when communication is a factor, e.g., in coscheduling methods.

Suggestions. Whenever possible, use real and representative applications. If communication is a factor, evaluate the effect of different assumptions and models on the measured metrics [42,64], before possibly neglecting factors that don't matter. If a parameter space exploration is not feasible, model the communication and detail all the assumptions made [26,60], preferably basing the model on real workloads or measurements. Confidence intervals can also be used to qualify the results [69].

Research. How to select representative communication patterns? Can we make do with just a few? A negative answer can lead to an explosion of the parameter space. Some preliminary data about communication patterns in real applications is available [12], but much more is needed to be able to identify common patterns and how often each one occurs.

Pitfall 11

Using Coarse-grained applications

Scope. Time slicing systems.

Problem. Fine-grained application are the most sensitive to scheduling [25]. Omitting them from an evaluation of a time slicing scheduler will probably not produce credible results. This pitfall may not be relevant for schedulers where resources are dedicated to the job, such as space slicing methods.

Suggestions. Make sure that fine-grained applications are part of the workload. Alternately, conduct a sensitivity analysis to measure how the granularity of the constituent applications affects the scheduler.

Research. What are representative granularities? What is the distribution? Again, real data is sorely needed.

Pitfall 12

Oversimplifying the memory model

Problem. Unless measuring actual applications, an evaluation must account for applications' memory usage. For example, allowing the size of malleable jobs to go down to 1 may be great for the job scheduler in terms of packing and speedup, but real parallel applications often require more physical memory than is available on one node [58] (see pitfall 16). Time slicing evaluations must take particular care to address memory requirements, since the performance effects of thrashing due to the increased memory pressure can easily offset any performance gains from the scheduling technique. A persistent difficulty with simulating and analyzing memory usage in scientific applications is that it does not lend itself to easy modeling, and in particular, does not offer a direct relation between parallelism and memory usage [17].

Suggestions. State memory assumptions e.g., that a certain multiprogramming level (MPL) is always enough to accommodate all applications. Perform a parameter-space exploration, or use data from actual traces [17,43]. Malleable jobs may require that an assumption be made and stated on the minimal partition size for each job so that it still fits in memory.

Research. Collecting data on memory usage, and using it to find good memory-usage models.

Pitfall 13

Assuming malleable or moldable jobs and/or linear speedup

Problem. Malleable or moldable jobs can run on an arbitrary number of processors dynamically or at launch time, respectively. While many jobs are indeed moldable, some jobs have strict size requirements and can only take a limited range of sizes, corresponding to network topology (e.g., torus, mesh, hypercube), or application constraints (e.g., NPB's applications [6]). Another unlikely assumption for most MPI jobs is that they can change their size dynamically (*malleable* or *evolving* jobs [29]).

For cases where malleability *is* assumed (e.g. for the evaluation of dynamic scheduling) linear speedup is sometimes assumed as well. This assumption is wrong. Real workloads have more complex speedup behavior. This also affects the offered load, as using a different number of processors leads to a different efficiency [30].

Suggestions. If job malleability is assumed, state it [10], and model speedup and efficiency realistically [13,52,66]. If real applications are simulated based on actual measurements, use an enumeration of application speedup for all different partition sizes.

Pitfall 14

Ignoring interactive jobs

Problem. Interactive jobs are a large subset of many parallel workloads. Interactive jobs have a significant effect on scheduling results [55] that needs to be accounted for if they are mixed with the parallel jobs. Some machines however have separate partitions for interactive jobs, so they don't mix with the parallel workload.

Suggestions. An evaluator needs to be aware of the special role of interactive jobs and make a decision if they are to be incorporated in the evaluation or not. To incorporate them, one can use workload traces or models that include them (e.g., [48]). If choosing not to incorporate them, the decision should be stated and explained. A model or a trace can then be used from a machine with a noninteractive partition [60]. It should be noted that the presence (or lack) of interactive jobs should also affect the choice of metrics used. More specifically, response time (or flow) is of lesser importance for batch jobs that it is for interactive ones. Moreover, interactive jobs account for many of the short jobs in a workload, and removing those will have a marked effect on the performance of the chosen scheduling algorithm and metric [33].

Pitfall 15

Using actual runtime as a user estimate

Scope. Evaluating backfilling schedulers that require user estimates of runtime.

Problem. User estimates are rarely accurate predictions of program run times [41,51]. If an evaluation uses actual run times from the trace as user estimates, the evaluation results may differ significantly from a comparable evaluation with the actual user estimates [21].

Suggestions. Evaluators should strive to use actual user estimates, when available in the trace (these are currently available in 11 of the 16 traces in the workload archive). Lacking user estimates, the evaluator can opt to use a model for generating user estimates [72]. Another alternative is to test a range of estimates (e.g., by a parametrized model) and either describe their effect on the result, or demonstrate that no significant effect exists. Simply multiplying the run times with different factors to obtain overestimation [33,51] can lead to artificial, unrepresentative performance improvements [72].

4 Evaluation Parameters

Even after meticulously choosing workloads and applications, an evaluation is only as representative as the input parameters that are used in it. Different scheduling algorithms can be very sensitive to parameters such as input load, multiprogramming level, and machine size. It is therefore important to understand the effect of these parameters and the reasonable ranges they can assume in representative workloads.

Pitfall 16 ⚡⚡⚡

Unrealistic multiprogramming levels

Problem. Increasing the multiprogramming level in time slicing schedulers also increases the memory pressure. High MPLs have value when studying the effect of the MPL itself on scheduling algorithms, and as a limiting optimal case. But for actual evaluations, high MPLs are unrealistic for many parallel workloads, especially in capability mode, where jobs could potentially use as much memory as they can possibly allocate. For many time slicing studies it is not even required to assume very high MPLs: several results show that increasing the MPL above a certain (relatively low) value offers little additional benefit, and can in fact degrade performance [33,50,65].

Since MPL can be interpreted as the allowed degree of resource oversubscribing, with space slicing scheduling a higher MPL translates to more jobs waiting in the scheduler's queues. In this case, the MPL does not have an effect on memory pressure, but could potentially increase the computation time of the space allocation algorithm.

Suggestions. Ideally, a comprehensive model of application memory requirements can be incorporated into the scheduler, but this remains an open research topic. For time slicing algorithms, bound the MPL to a relatively low value [80], say 2–4. Alternately, use a technique such as admission control to dynamically bound the MPL based on memory resources [7] or load [79]. Another option is to incorporate a memory-conscious mechanism with the scheduler, such as swapping [2] or block paging [75].

Research. More hard data on memory usage in parallel supercomputers is required. Desirable data includes not only total memory usage, but also the possible correlation with runtime, and the questions of locality, working sets, and changes across phases of the computation.

Pitfall 17 ⚡⚡

Scaling traces to different machine sizes

Problem. The parameters that comprise workloads and traces do not scale linearly with machine size. Additionally, scaling down workloads by trimming away the jobs that have more processors than required can distort the workload properties and affect scheduling metrics [39].

Suggestions. If using simulation or analysis, adjust simulated machine size to the one given in trace [10]. If running an experimental evaluation or for experimental reasons the workload's machine size needs to be fixed (e.g., to compare different workloads on the same machine size), use a scaling model to change the workload size [15]. If possible, verify that the scaling preserves the metrics being measured. Alternatively, use a reliable workload model to generate a synthetic workload for the desired machine size. For example, Lublin postulated a piecewise log-uniform distribution of job sizes, and specified the parameters of the distribution as a function of the machine size [48].

Research. No perfect models for workload scaling exist yet. Such models should be checked against various real workloads. Specifically, what happens with very large systems? does it depend on the machine's usage? e.g., do capability machines have more large jobs than general usage machines?

Pitfall 18 

Changing a single parameter to modify load

Problem. It is often desired to repeat an experiment with various offered loads, in order to evaluate the sensitivity and saturation of a job scheduler. This is often done by expanding or condensing the distribution of one of these parameters: job interarrival time, job run time, and degree of parallelism [47,68]. In general however, the following problems arise:

- changing P (job size) causes severe packing problems that dominate the load modification, especially since workloads tend to have many powers of 2, and machines tend to be powers of 2.
- changing T (job runtime) causes a correlation of load and response time.
- changing I (job interarrivals) changes the relative size of jobs and the daily cycle; in extreme cases jobs may span the whole night.

In addition, changing any of these parameters alone can distort the correlations between these parameters and incoming load [68].

Suggestions. One way to avoid this problem is to use model-derived workloads instead of trace data. Ideally, a workload model should be able to produce a representative workload for any desired load. However, if we wish to use an actual trace for added realism or comparison reasons, we cannot suggest a bulletproof method to vary load. To the best of our knowledge, the question of adjusting traces load in a representative manner is still open, so we may have to compromise on changing a single parameter, and advise the reader of the possible caveat.

If an evaluation nevertheless requires a choice of a single-value parameter change to vary load, changing interarrivals is in our opinion the least objectionable of these.

Research. How to correctly modify the load is an open research question.

Pitfall 19 

Using FCFS queuing as the basis of comparison

Problem. First-come-first-serve with no queue management makes no sense for most dynamic workloads, since many proven backfilling techniques exist and offer better performance [31]. It only makes sense when the workload is homogeneous with large jobs, since backfilling is mostly beneficial when the workload has variety.

Suggestions. Employ any reasonable backfilling method, such as EASY or conservative [44,51].

Pitfall 20 

Using wall-clock user estimates for a time-slicing backfilling scheduler

Problem. Backfilling requires an estimate of run times to make reservations for jobs. These run times cannot be guaranteed however in a time-slicing scheduler, even with perfect user estimates, because run times change with the dynamic MPL.

Suggestions. One heuristic to give an upper bound for reservation times is to multiply user estimates by the maximum MPL [33].

Research. Come up with more precise heuristics for estimating reservation times under time slicing.

5 Metrics

Different metrics are appropriate for different system models [30]. Makespan is suitable for off-line scheduling. Throughput is a good metric for closed systems. When considering on-line open systems, which are the closest model to how a real system operates, the metrics of choice are response time and slowdown.

Pitfall 21 

Using irrelevant/wrong/biased metrics

Problem. Some metrics do not describe a real measured value, such as utilization (see pitfall 22). Other metrics may not mean the same thing in different contexts (e.g., slowdown in a time slicing environment vs. non-time-slicing [21,82], or makespan for open vs. closed workloads).

Suggestions. Metrics should be used in an appropriate context of workload and applications [30]. Even within the context of a workload, there is room to measure metrics separately (or use different metrics) for different classes of applications, e.g., based on their type or resource requirements [33,75].

Pitfall 22 

Measuring utilization, throughput, or makespan for an open model

Problem. This is a special case of the previous pitfall, but occurs frequently enough to merit its own pitfall.

"Open models" correspond to systems that operate in an on-line mode [23]. This means that jobs arrive in a continuous but paced manner. In this context, the most significant problem with utilization and throughput as metrics is that they are a measure of the offered load more than of any effect the scheduler may have [30]. In fact, utilization should *equal* the offered load unless the system is saturated (pitfall 25); thus measuring utilization is useful maily as a sanity check. Makespan is also irrelevant for an open model, as it is largely determined by the length of the workload: how many jobs are simulated or measured.

Suggestions. The relevant metric in an open system is not the utilization, but the saturation point: the maximum utilization that can be achieved. In practice, this is reflected by the "knee" of the response-time or slowdown curve [59]. However, identification of the knee is somewhat subjective. A more precise definition is the limiting value of the asymptote, which can also be identified by the departure from the diagonal of the utilization curve, where the measured load becomes lower than the offered load. Somtimes it can be found analytically based on the distribution of job sizes [28].

For open systems, other metrics can and should also be used, and are relevant for all loads up to the saturation point. These include slowdown, response time, and wait time. But note that measuring these metrics beyond (and even close to) the saturation point leads to meaningless results that reflect only the size of the workload. Measuring utilization is may only be relevant in the context of addmission controls (pitfall 25).

In a closed system, throughput is the most important metric. For static workloads, where all the jobs are assumed to arrive at the same time, makespan can be used [30]. In this case, makespan and utilization provide the same information.

Usage note. Yet another difficulty with utilization is that there are some variations in its definition. Basically, utilization is that fraction of the available resources that is actually used. One question is then what are the available resources, and specifically, whether or not to take machine inavailability into account [59]; we would suggest to do so, but availability data is not always available.

Another issue is that "actually used" can be interpreted in different ways. A commonly used option is to consider all nodes that are *allocated* to running jobs. However, some parallel machines allow processor allocation only in fixed quanta, or specific dimensions, thereby forcing the allocation of more processors than the job actually requires. For example, BlueGene/L allocates processors in units of 512 [40], and the Cray T3D allocates power-of-two processors, starting from two [22]. The question then arises, how to measure utilization in light of the fact that many jobs in parallel workloads actually have a very low degree of parallelism [43,48], or just different sizes than those of the machine's allocated sizes, and thus necessarily have unused processors allocated to them.

The above leads to the alternative of only counting nodes that are actually *used*, thus explicitly accounting for effects such as the internal fragmentation

cited above. An even more extreme definition only considers actual CPU utilization, in an attempt to factor out effects such as heavy paging that reduces CPU utilization [29]. However, the required data is typically not available.

When referring to utilization, one should therefore be specific about what exactly is measured.

Pitfall 23 ⚡⚡

Using the mean for asymmetrically distributed (skewed) results

Problem. Some metrics are asymmetrically distributed, sometimes even heavy tailed. One example is slowdown, where short jobs have disproportionately longer slowdowns than the rest of the workload. Averaging these values yields a distorted picture, since the mean is disproportionately affected by the tail [11,33].

Suggestions. Try to describe the distribution instead of using a mean. Alternative metrics can often be devised to bypass this problem, such as bounded slowdown [19] and weighted response time [60]. Other statistical tools can be used to describe the measurements more accurately, such as median, geometric mean, or box plot. Another alternative is to divide the results into bins (e.g., short/long narrow/wide jobs [33,61]) and analyze each bin separately.

Pitfall 24 ⚡⚡

Inferring scalability trends from O*(1) nodes*

Problem. Performance results rarely scale linearly [5]. Measuring a near-constant growth of a metric of a small number of nodes and inferring scalability of the property is risky. This generalization naturally is even more applicable for simulation and analysis based studies, that hide an assumption about the scalability of the underlying hardware or mechanisms.

Suggestions. Barring actual measurement, use detailed models or qualified estimates. In case of simulation, the most reasonable approach is to simulate larger machines.

Research. Indeed, how can we learn about scalability with minimal effort? what can we do if we cannot get a 1000+ node machine to run on?

6 Measurement Methodology

In both simulations and actual measurements, performance is evaluated by having the scheduler schedule a sequence of jobs. Previous sections have listed pitfalls related to the workload itself, i.e. which jobs should appear in this sequence, and to the metrics used to measure performance. This final section is about the context of the measurements.

The considerations involved in these pitfalls are well-known in the simulation literature, and can be summarized as ensuring that we are simulating (and measuring) the system in its steady state. The problem is that with traces it may

be hard to achieve a steady state, or easy to overlook the fact that the state is not steady.

Pitfall 25 ///

Measuring saturated workloads

Problem. This is one of the most common errors committed in the evaluation of parallel job scheduling schemes. In queuing theory terms, a system is stable only if the arrival rate is lower than the service rate ($\lambda < \mu$, or alternatively $\rho = \frac{\lambda}{\mu} < 1$). If this condition is violated the system is unstable — it has no steady state.

The saturation point is the maximal load that the system can handle. Keep in mind that many parallel workloads and machines do not even get close to 100% utilization [39,59], due to loss of resources to fragmentation (pitfall 26). Evaluating the system for loads beyond the saturation point is unrealistic and typically leads to meaningless results.

Measuring the behavior of a parallel system with an offered load that is higher than the saturation point would yield infinitely-growing queues on a real system, and metrics such as average wait time and slowdown will grow to infinity. Thus the results of the measurement would *depend on the length of the measurement* — the longer your workload, the worse it gets. However, it is easy to miss this situation in a real measurement, because all the evaluations we perform are finite, and finite workloads always converge in the end. But results we measure on such workloads are actually invalid.

Suggestions. Identify the saturation point by comparing the offered load to the achieved utilization (see pitfall 22) and discard data points from saturated experiments, effectively limiting the experimental results to the relevant domain [22,33].

Transient saturation for a limited time should be allowed, as it is a real phenomenon. However, if this happens toward the end of a simulation/measurement, it may indicate a saturated experiment.

The only case where measurement with an offered load higher than the saturation point are relevant is when we are considering admission policies. This means that the system is designed to deal with overload, and does so by discarding part of its input (i.e. some jobs are simply not serviced). In this case relevant metrics are the fraction of jobs that are serviced and the achieved utilization.

Pitfall 26 //

Ignoring internal fragmentation

Problem. Even an optimal scheduler that eliminates external fragmentation entirely (i.e., all processors are always allocated) might suffer pitiful response times and throughput if processor efficiency is not taken into account. Most applications scale sublinearly with processors and/or include inherent internal fragmentation (e.g., due to the "memory wall"). When using such

applications, measuring system-centric metrics only (such as machine utilization in terms of processor allocation, without considering efficiency) can produce results that indeed favor the system view, while specific applications suffer from poor response times.

Suggestions. To the extent that a scheduler's designer can alleviate these phenomena, reducing internal fragmentation should be given the same consideration as reducing external fragmentation. For example, adaptive and dynamic partitioning can increase application efficiency by matching the partition size to the degree of parallelism of applications, although this poses certain requirements on the scheduler and applications [52,66]. If possible, include a dynamic coscheduling scheme [4,34,63,65] in the evaluation, as these tend to reduce internal fragmentation.

Whether using any of these methods or not, an experimental evaluator should remain cognizant of internal fragmentation, and address its effect on the evaluation.

Pitfall 27 

Using exceedingly short workloads

Problem. Some phenomena, including finding the saturation point (pitfall 22) or fragmentation of resources (such as contiguous processors or disk space in a file system [62]), only appear with long enough workloads. Thus we need long workloads not only to achieve a steady state, but even more so to see the realistic conditions that a real system faces.

Suggestions. For an analysis or simulation study, use thousands of jobs, e.g. use a rule of thumb of 30 batches of 5000 jobs each (but note that even this may not suffice for some skewed distributions) [49]. This is more difficult for an experimental evaluation. In this case, use as many jobs as practical and/or use a higher load to pack more jobs into the same amount of time (but note pitfall 18).

Research. Immediate research issues are related to aging. How to do aging quickly? How to quantify appropriate aging?

At a broader level, these phenomena are related to new research on software rejuvenation [71] — rebooting parts of the system to restore a clean state. In real systems this may also happen, as systems typically don't stay up for extended periods. As each reboot causes the effects of aging to be erased, it may be that short workloads are actually more representative! This would also imply that transient conditions that exist when the system is starting up or shuting down should be explicitly studied rather than being avoided as suggested in pitfalls 28 and 29.

Pitfall 28 

Not discarding warm-up

Problem. A special case of pitfall 27 that is important enough to be noted separately.

Initial conditions are different in a consistent manner, so even averaging over runs will be biased. Thus it is wrong to include the initial part of a measurement or simulation in the results, as it is not representative of steady state conditions.

Suggestions. The first few data points can be discarded to account for warmup time. There are various statistical methods to decide how much to discard, based on estimates of whether the system has entered a steady state [56]. As an approximation, a running average of the performance metric can be drawn to see when it stabilizes (which will not necessarily be easy to identify because of diverging instantaneous values).

Research. Which statistical methods are specifically suitable to identify the steady state of parallel workloads? And is there really a steady state with real workloads? Alternatively, is rebooting common enough that it is actually important to study the initial tranzient conditions?

Pitfall 29 

Not avoiding cool-down

Problem. Another special case of pitfall 27.

Towards the end of a measurement/simulation run, problematic jobs remain and enjoy a dedicated system with less competition. This also causes the measured load to deviate from the intended one. Therefore a good indication of this problem is that the system appears to be saturated (pitfall 25).

Suggestions. Stop the measurements at the last arrival at the latest. Count terminations to decide when enough jobs have been measured, not arrivals. Then, check jobs left in queue, to see if there are systematic biases.

Pitfall 30 

Neglecting overhead

Problem. Overhead is hard to quantify. Overhead and operation costs come in many forms, not all of them even known in advance. Some overhead effects are indirect, such as cache and paging effects, or contention over shared resources.

Suggestions. Model the distribution of overhead [66], incorporate data from traces or actual machines [37], or use a range of overhead values [50].

Research. What are overheads of real systems? not much data on overhead is available, and it continuously changes with technology.

Pitfall 31 

Measuring all jobs

Problem. One aspect of this pitfall has already been covered as pitfall 5: that some jobs are unrepresentative, and the data should be cleaned. But there is more to this.

As loads fluctuate, many of the jobs actually see an empty or lightly loaded system. In these cases the scheduler has no effect, and including them just

dilutes the actual measurements. Additionally, different jobs may experience very different conditions, and it is questionable whether it is meaningful to average all of them together.

Another aspect of this pitfall is the distinction between jobs that arrive during the day, at night, and over the weekend. Many sites have policies that mandate different levels of service for different classes of jobs at different times [46,74]. Obviously evaluating such policies should take the characteristics of the different job classes into account, and not bundle them all together.

Suggestions. Partition jobs into classes according to conditions, and look at performance of each class separately. For example, only look at jobs that are high-priority, prime-time, interactive, or belong to a certain user. The emphasis here is not on presenting results for all possible categories, but rather, on identifying the important conditions or categories that need to be taken into account, and then presenting the results accordingly.

Research. To the best of our knowledge, nobody does this yet. Many new pitfalls may be expected in doing it right.

Pitfall 32

Comparing analysis to simulations

Problem. Comparisons are important and welcome, but we must be certain that we are validating the correct properties: Both simulation and analysis could embody the same underlying hidden assumptions, especially if developed by the same researchers. Experimental evaluations tend to expose unaccounted-for factors.

Suggestions. Compare analysis and/or simulation to experimental data [1].

7 Conclusion

As the field of parallel job scheduling matures, it still involves many poorly understood and often complex factors. Despite this situation, and perhaps because of it, we need to approach the study of this field in a scientific, reproducible manner. This paper concentrates on 32 of the more common pitfalls in parallel job scheduling evaluation, as well as a few of the more subtle ones. It is unlikely that any study in this field will be able to follow all the complex (and sometimes contradicting) methodological suggestions offered in this paper. Nor is it likely that any such study will be immune to other methodological critique. Like other fields in systems research, parallel job scheduling entails compromises and tradeoffs. Nevertheless, we should remain circumspect of these choices and make them knowingly. It is the authors' hope that by focusing on these topics in a single document, researchers might be more aware of the subtleties of their evaluation. Ideally, disputable assumptions that are taken in the course of a study can be justified or at least addressed by the researcher, rather than remain undocumented. By promoting more critical evaluations and finer attention to methodological issues, we hope that some of the "black magic" in the field will be replaced by reliable and reproducible reasoning.

References

1. K. Aida, H. Kasahara, and S. Narita. Job scheduling scheme for pure space sharing among rigid jobs. In *Fourth Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1459 of *Lecture Notes in Computer Science*, pp. 98–121. Springer-Verlag, 1998.
2. G. Alverson, S. Kahan, R. Korry, C. McCann, and B. Smith. Scheduling on the Tera MTA. In *First Workshop on Job Scheduling Strategies for Parallel Processing*, volume 949 of *Lecture Notes in Computer Science*, pp. 19–44. Springer-Verlag, 1995.
3. C. D. Antonopoulos, D. S. Nikolopoulos, and T. S. Papatheodorou. Informing algorithms for efficient scheduling of synchronizing threads on multiprogrammed SMPs. In *30th International Conference on Parallel Processing (ICPP)*, pp. 123–130, Valencia, Spain, September 2001.
4. A. C. Arpaci-Dusseau. Implicit Coscheduling: Coordinated scheduling with implicit information in distributed systems. *ACM Transactions on Computer Systems*, 19(3):283–331, August 2001.
5. D. H. Bailey. Misleading performance in the supercomputing field. In *IEEE/ACM Supercomputing*, pp. 155–158, Minneapolis, MN, November 1992.
6. D. H. Bailey, L. Dagum, E. Barszcz, and H. D. Simon. NAS parallel benchmark results. In *IEEE/ACM Supercomputing*, pp. 386–393, Minneapolis, MN, November 1992.
7. A. Batat and D. G. Feitelson. Gang scheduling with memory considerations. In *14th International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 109–114, May 2000.
8. T. B. Brecht. An experimental evaluation of processor pool-based scheduling for shared-memory NUMA multiprocessors. In *Third Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1291 of *Lecture Notes in Computer Science*, pp. 139–165. Springer-Verlag, 1997.
9. S. J. Chapin, W. Cirne, D. G. Feitelson, J. P. Jones, S. T. Leutenegger, U. Schwiegelshohn, W. Smith, and D. Talby. Benchmarks and standards for the evaluation of parallel job schedulers. In *Fifth Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1659 of *Lecture Notes in Computer Science*, pp. 67–90. Springer-Verlag, 1999.
10. W. Cirne and F. Berman. Adaptive selection of partition size for supercomputer requests. In *Sixth Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1911 of *Lecture Notes in Computer Science*, pp. 187–207. Springer-Verlag, 2000.
11. M. E. Crovella. Performance evaluation with heavy tailed distributions. In *Seventh Workshop on Job Scheduling Strategies for Parallel Processing*, volume 2221 of *Lecture Notes in Computer Science*, pp. 1–10. Springer Verlag, 2001.
12. R. Cypher, A. Ho, S. Konstantinidou, and P. Messina. Architectural requirements of parallel scientific applications with explicit communication. In *20th International Symposium on Computer Architecture (ISCA)*, pp. 2–13, May 1993.
13. A. B. Downey. A model for speedup of parallel programs. Technical Report UCB/CSD-97-933, University of California, Berkeley, CA, January 1997.
14. A. B. Downey and D. G. Feitelson. The elusive goal of workload characterization. *Performance Evaluation Review*, 26(4):14–29, March 1999.
15. C. Ernemann, B. Song, and R. Yahyapour. Scaling of workload traces. In *Ninth Workshop on Job Scheduling Strategies for Parallel Processing*, volume 2862 of *Lecture Notes in Computer Science*, pp. 166–182. Springer-Verlag, 2003.

16. D. G. Feitelson. Packing schemes for gang scheduling. In *Second Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1162 of *Lecture Notes in Computer Science*, pp. 89–110. Springer-Verlag, 1996.
17. D. G. Feitelson. Memory usage in the LANL CM-5 workload. In *Third Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1291 of *Lecture Notes in Computer Science*, pp. 78–94. Springer-Verlag, 1997.
18. D. G. Feitelson. A critique of ESP. In *Sixth Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1911 of *Lecture Notes in Computer Science*, pp. 68–73. Springer-Verlag, 2000.
19. D. G. Feitelson. Metrics for parallel job scheduling and their convergence. In *Seventh Workshop on Job Scheduling Strategies for Parallel Processing*, volume 2221 of *Lecture Notes in Computer Science*, pp. 188–1205. Springer Verlag, 2001.
20. D. G. Feitelson. Workload modeling for performance evaluation. In *Performance Evaluation of Complex Systems: Techniques and Tools*, volume 2459 of *Lecture Notes in Computer Science*, pp. 114–141. Springer-Verlag, September 2002.
21. D. G. Feitelson. Metric and workload effects on computer systems evaluation. *Computer*, 36(9):18–25, September 2003.
22. D. G. Feitelson and M. A. Jette. Improved utilization and responsiveness with gang scheduling. In *Third Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1291 of *Lecture Notes in Computer Science*, pp. 238–261. Springer-Verlag, 1997.
23. D. G. Feitelson and A. W. Mu'alem. On the definition of "on-line" in job scheduling problems. *ACM SIGACT News*, 36(1):122–131, March 2005.
24. D. G. Feitelson and B. Nitzberg. Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860. In *First Workshop on Job Scheduling Strategies for Parallel Processing*, volume 949 of *Lecture Notes in Computer Science*, pp. 337–360. Springer-Verlag, 1995.
25. D. G. Feitelson and L. Rudolph. Gang scheduling performance benefits for fine-grain synchronization. *Journal of Parallel and Distributed Computing*, 16(4):306–318, December 1992.
26. D. G. Feitelson and L. Rudolph. Coscheduling based on run-time identification of activity working sets. *International Journal of Parallel Programming*, 23(2):136–160, April 1995.
27. D. G. Feitelson and L. Rudolph. Parallel job scheduling: Issues and approaches. In *First Workshop on Job Scheduling Strategies for Parallel Processing*, volume 949 of *Lecture Notes in Computer Science*, pp. 1–18. Springer-Verlag, 1995.
28. D. G. Feitelson and L. Rudolph. Evaluation of design choices for gang scheduling using distributed hierarchical control. *Journal of Parallel and Distributed Computing*, 35(1):18–34, May 1996.
29. D. G. Feitelson and L. Rudolph. Toward convergence in job schedulers for parallel supercomputers. In *Second Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1162 of *Lecture Notes in Computer Science*, pp. 1–26. Springer-Verlag, 1996.
30. D. G. Feitelson and L. Rudolph. Metrics and benchmarking for parallel job scheduling. In *Fourth Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1459 of *Lecture Notes in Computer Science*, pp. 1–24. Springer-Verlag, 1998.
31. D. G. Feitelson, L. Rudolph, and U. Schwigelshohn. Parallel job scheduling – A status report. In *Tenth Workshop on Job Scheduling Strategies for Parallel Processing*, volume 3277 of *Lecture Notes in Computer Science*, pp. 1–16. Springer-Verlag, 2004.

32. D. G. Feitelson and A. M. Weil. Utilization and predictability in scheduling the IBM SP2 with backfilling. In *12th International Parallel Processing Symposium (IPPS)*, pp. 542–546, April 1998.
33. E. Frachtenberg, D. G. Feitelson, J. Fernandez-Peinador, and F. Petrini. Parallel job scheduling under dynamic workloads. In *Ninth Workshop on Job Scheduling Strategies for Parallel Processing*, volume 2862 of *Lecture Notes in Computer Science*, pp. 208–227. Springer-Verlag, 2003.
34. E. Frachtenberg, D. G. Feitelson, F. Petrini, and J. Fernandez. Adaptive parallel job scheduling with flexible coscheduling. *IEEE Transactions on Parallel and Distributed Systems*, To appear.
35. A. Gupta, A. Tucker, and S. Urushibara. The impact of operating system scheduling policies and synchronization methods on the performance of parallel applications. In *SIGMETRICS Measurement & Modeling of Computer Systems*, pp. 120–32, San Diego, CA, May 1991.
36. G. Holt. Time-critical scheduling on a well utilised HPC system at ECMWF using LoadLeveler with resource reservation. In *Tenth Workshop on Job Scheduling Strategies for Parallel Processing*, volume 3277 of *Lecture Notes in Computer Science*, pp. 102–124. Springer-Verlag, 2004.
37. A. Hori, H. Tezuka, and Y. Ishikawa. Overhead analysis of preemptive gang scheduling. In *Fourth Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1459 of *Lecture Notes in Computer Science*, pp. 217–230. Springer-Verlag, 1998.
38. J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira, and J. Riodan. Modeling of workload in MPPs. In *Third Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1291 of *Lecture Notes in Computer Science*, pp. 95–116. Springer-Verlag, 1997.
39. J. P. Jones and B. Nitzberg. Scheduling for parallel supercomputing: A historical perspective of achievable utilization. In *Fifth Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1659 of *Lecture Notes in Computer Science*, pp. 1–16. Springer-Verlag, 1999.
40. E. Krevat, J. G. Castaños, and J. E. Moreira. Job scheduling for the BlueGene/L system. In *Eighth Workshop on Job Scheduling Strategies for Parallel Processing*, volume 2537 of *Lecture Notes in Computer Science*, pp. 38–54. Springer Verlag, 2002.
41. C. B. Lee, Y. Schwartzman, J. Hardy, and A. Snavely. Are user runtime estimates inherently inaccurate? In *Tenth Workshop on Job Scheduling Strategies for Parallel Processing*, volume 3277 of *Lecture Notes in Computer Science*, pp. 253–263. Springer-Verlag, 2004.
42. W. Lee, M. Frank, V. Lee, K. Mackenzie, and L. Rudolph. Implications of I/O for gang scheduled workloads. In *Third Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1291 of *Lecture Notes in Computer Science*, pp. 215–237. Springer-Verlag, 1997.
43. H. Li, D. Groep, and L. Wolters. Workload characteristics of a multi-cluster supercomputer. In *Tenth Workshop on Job Scheduling Strategies for Parallel Processing*, volume 3277 of *Lecture Notes in Computer Science*, pp. 176–193. Springer-Verlag, 2004.
44. D. Lifka. The ANL/IBM SP scheduling system. In *First Workshop on Job Scheduling Strategies for Parallel Processing*, volume 949 of *Lecture Notes in Computer Science*, pp. 295–303. Springer-Verlag, 1995.

45. W. Liu, V. Lo, K. Windisch, and B. Nitzberg. Non-contiguous processor allocation algorithms for distributed memory multicomputers. In *IEEE/ACM Supercomputing*, pp. 227–236, November 1994.
46. V. Lo and J. Mache. Job scheduling for prime time vs. non-prime time. In *Fourth Proceedings of the IEEE International Conference on Cluster Computing*, pp. 488–493, September 2002.
47. V. Lo, J. Mache, and K. Windisch. A comparative study of real workload traces and synthetic workload models for parallel job scheduling. In *Fourth Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1459 of *Lecture Notes in Computer Science*, pp. 25–46. Springer-Verlag, 1998.
48. U. Lublin and D. G. Feitelson. The workload on parallel supercomputers: Modeling the characteristics of rigid jobs. *Journal of Parallel and Distributed Computing*, 63(11):1105–1122, November 2003.
49. M. H. MacDougall. *Simulating Computer Systems: Techniques and Tools*. MIT Press, 1987.
50. J. E. Moreira, W. Chan, L. L. Fong, H. Franke, and M. A. Jette. An infrastructure for efficient parallel job execution in terascale computing environments. In *IEEE/ACM Supercomputing*, Orlando, FL, November 1998.
51. A. W. Mu'alem and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Transactions on Parallel and Distributed Systems*, 12(6):529–543, June 2001.
52. T. D. Nguyen, R. Vaswani, and J. Zahorjan. Parallel application characterization for multiprocessor scheduling policy design. In *Second Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1162 of *Lecture Notes in Computer Science*, pp. 175–199. Springer-Verlag, 1996.
53. N. Nieuwejaar, D. Kotz, A. Purakayastha, C. S. Ellis, and M. L. Best. File-access characteristics of parallel scientific workloads. *IEEE Transactions on Parallel and Distributed Systems*, 7(10):1075–1089, October 1996.
54. Parallel workload archive. `www.cs.huji.ac.il/labs/parallel/workload`.
55. E. W. Parsons and K. C. Sevcik. Multiprocessor scheduling for high-variability service time distributions. In *First Workshop on Job Scheduling Strategies for Parallel Processing*, volume 949 of *Lecture Notes in Computer Science*, pp. 127–145. Springer-Verlag, 1995.
56. K. Pawlikowski. Steady-state simulation of queueing processes: A survey of problems and solutions. *ACM Computing Surveys*, 22(2):123–170, June 1990.
57. V. G. J. Peris, M. S. Squillante, and V. K. Naik. Analysis of the impact of memory in distributed parallel processing systems. In *SIGMETRICS Measurement & Modeling of Computer Systems*, pp. 5–18, Nashville, TN, May 1994.
58. A. program. ASCI technology prospectus: Simulation and computational science. Technical Report DOE/DP/ASC-ATP-001, National Nuclear Security Agency, July 2001.
59. L. Rudolph and P. Smith. Valuation of ultra-scale computing systems. In *Sixth Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1911 of *Lecture Notes in Computer Science*, pp. 39–55. Springer-Verlag, 2000.
60. U. Schwiegelshohn and R. Yahyapour. Improving first-come-first-serve job scheduling by gang scheduling. In *Fourth Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1459 of *Lecture Notes in Computer Science*, pp. 180–198. Springer-Verlag, 1998.

61. S. K. Setia. The interaction between memory allocation and adaptive partitioning in message-passing multicomputers. In *First Workshop on Job Scheduling Strategies for Parallel Processing*, volume 949 of *Lecture Notes in Computer Science*, pp. 146–164. Springer-Verlag, 1995.
62. K. A. Smith and M. I. Seltzer. File system aging—Increasing the relevance of file system benchmarks. In *SIGMETRICS Measurement & Modeling of Computer Systems*, pp. 203–213, June 1997.
63. P. G. Sobalvarro and W. E. Weihl. Demand-based coscheduling of parallel jobs on multiprogrammed multiprocessors. In *First Workshop on Job Scheduling Strategies for Parallel Processing*, volume 949 of *Lecture Notes in Computer Science*, pp. 106–126. Springer-Verlag, 1995.
64. A. C. Sodan and L. Lan. LOMARC—Lookahead matchmaking for multi-resource coscheduling. In *Tenth Workshop on Job Scheduling Strategies for Parallel Processing*, volume 3277 of *Lecture Notes in Computer Science*, pp. 288–315. Springer-Verlag, 2004.
65. M. Squillante, Y. Zhang, S. Sivasubramaniam, N. Gautam, H. Franke, and J. Moreira. Modeling and analysis of dynamic coscheduling in parallel and distributed environments. In *SIGMETRICS Measurement & Modeling of Computer Systems*, pp. 43–54, Marina Del Rey, CA, June 2002.
66. M. S. Squillante. On the benefits and limitations of dynamic partitioning in parallel computer systems. In *First Workshop on Job Scheduling Strategies for Parallel Processing*, volume 949 of *Lecture Notes in Computer Science*, pp. 219–238. Springer-Verlag, 1995.
67. M. S. Squillante, D. D. Yao, and L. Zhang. Analysis of job arrival patterns and parallel scheduling performance. *Performance Evaluation*, 36–37:137–163, 1999.
68. D. Talby, D. G. Feitelson, and A. Raveh. Comparing logs and models of parallel workloads using the Co-Plot method. In *Fifth Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1659 of *Lecture Notes in Computer Science*, pp. 43–66. Springer-Verlag, 1999.
69. S. Tongsima, C. Chantrapornchai, and E. H.-M. Sha. Probabilistic loop scheduling considering communication overhead. In *Fourth Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1459 of *Lecture Notes in Computer Science*, pp. 158–179. Springer-Verlag, 1998.
70. Top 500 supercomputers. `www.top500.org`.
71. K. S. Trivedi and K. Vaidyanathan. Software reliability and rejuvenation: Modeling and analysis. In *Performance Evaluation of Complex Systems: Techniques and Tools*, volume 2459 of *Lecture Notes in Computer Science*, pp. 318–345. Springer-Verlag, 2002.
72. D. Tsafrir, Y. Etsion, and D. G. Feitelson. Modeling user runtime estimates. In *11th Workshop on Job Scheduling Strategies for Parallel Processing*. 2005.
73. D. Tsafrir and D. G. Feitelson. Workload flurries. Technical Report 2003-85, Hebrew University, November 2003.
74. M. Wan, R. Moore, G. Kremenek, and K. Steube. A batch scheduler for the Intel Paragon with a non-contiguous node allocation algorithm. In *Second Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1162 of *Lecture Notes in Computer Science*, pp. 48–64. Springer-Verlag, 1996.
75. F. Wang, M. Papaefthymiou, and M. Squillante. Performance evaluation of gang scheduling for parallel and distributed multiprogramming. In *Third Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1291 of *Lecture Notes in Computer Science*, pp. 277–298. Springer-Verlag, 1997.

76. W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson. Self-similarity through high-variability: Statistical analysis of Ethernet LAN traffic at the source level. In *ACM SIGCOMM*, pp. 100–113, 1995.
77. Y. Wiseman and D. G. Feitelson. Paired gang scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 14(6):581–592, June 2003.
78. A. T. Wong, L. Oliker, W. T. C. Kramer, T. L. Kaltz, and D. H. Bailey. ESP: A system utilization benchmark. In *IEEE/ACM Supercomputing*, pp. 52–52, Dallas, TX, November 2000.
79. K. K. Yue and D. J. Lilja. Loop-level process control: An effective processor allocation policy for multiprogrammed shared-memory multiprocessors. In *First Workshop on Job Scheduling Strategies for Parallel Processing*, volume 949 of *Lecture Notes in Computer Science*, pp. 182–199. Springer-Verlag, 1995.
80. Y. Zhang, H. Franke, J. E. Moreira, and A. Sivasubramaniam. An integrated approach to parallel scheduling using gang-scheduling, backfilling, and migration. In *Seventh Workshop on Job Scheduling Strategies for Parallel Processing*, volume 2221 of *Lecture Notes in Computer Science*, pp. 133–158. Springer Verlag, 2001.
81. Y. Zhang, A. Yang, A. Sivasubramaniam, and J. Moreira. Gang scheduling extensions for I/O intensive workloads. In *Ninth Workshop on Job Scheduling Strategies for Parallel Processing*, volume 2862 of *Lecture Notes in Computer Science*, pp. 166–182. Springer-Verlag, 2003.
82. B. B. Zhou, R. P. Brent, D. Walsh, and K. Suzaki. Job scheduling strategies for networks of workstations. In *Fourth Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1459 of *Lecture Notes in Computer Science*, pp. 143–157. Springer-Verlag, 1998.