The Forgotten Factor: Facts on Performance Evaluation and its Dependence on Workloads

Dror G. Feitelson

School of Computer Science and Engineering
The Hebrew University, 91904 Jerusalem, Israel
feit@cs.huji.ac.il
http://www.cs.huji.ac.il/~feit

Abstract. The performance of a computer system depends not only on its design and implementation, but also on the workloads it has to handle. Indeed, in some cases the workload can sway performance evaluation results. It is therefore crucially important that representative workloads be used for performance evaluation. This can be done by analyzing and modeling existing workloads. However, as more sophisticated workload models become necessary, there is an increasing need for the collection of more detailed data about workloads. This has to be done with an eye for those features that are really important.

1 Introduction

The scientific method is based on the ability to reproduce and verify research results. But in practice, the research literature contains many conflicting accounts and contradictions — especially multiple conflicting claims to be better than the competition. This can often be traced to differences in the methodology or the conditions used in the evaluation. In this paper we focus on one important aspect of such differences, namely differences in the workloads being used. In particular, we will look into the characterization and modeling of workloads used for the evaluation of parallel systems.

The goal of performance evaluation is typically not to obtain absolute numbers, but rather to differentiate between alternatives. This can be done in the context of system design, where the better design is sought, or as part of a procurement decision, where the goal is to find the option that provides the best value for a given investment. In any case, an implicit assumption is that differences in the evaluation results reflect real differences in the systems under study. But this is not always the case. Evaluation results depend not only on the systems, but also on the metrics being used and on the workloads to which the systems are subjected.

To complicate matters further, there may be various interactions between the system, workload, and metric. Some of these interactions lead to problems, as described below. But some are perfectly benign. For example, an interaction

between the system and a metric may actually be a good thing. If systems are designed with different objectives in mind, metrics that measure these objectives should indeed rank them differently. In fact, such metrics are exactly what we need if we know which objective function we wish to emphasize. An interaction between the workload and the metric is also possible, and may be meaningless. For example, if one workload contains longer jobs than another, its average response time will also be higher. On the other hand, interactions between a system and a workload may be very important, as they may help identify system vulnerabilities.

But when the effects leading to performance evaluation results are unknown and not understood, this is a problem. Conflicting results cast a shadow of doubt on our confidence in all the results. A solid scientific and experimental methodology is required in order to prevent such situations.

2 Examples of the Importance of Workloads

To support the claim that workloads make a difference, this section presents three specific cases in some detail. These are all related to the scheduling of parallel jobs.

A simple model of parallel jobs considers them as rectangles in processors×time space: each job needs a certain number of processors for a certain interval of time. Scheduling is then the packing of these job-rectangles into a larger rectangle that represents the available resources. In an on-line setting, the time dimension may not be known in advance. Dealing with this using preemption means that the job rectangle is cut into several slices, representing the work done during each time slice.

2.1 Effect of Job-Size Distribution

The packing of jobs obviously depends on the distribution of job sizes. A good example is provided by the DHC scheme [12], in which a buddy system is used for processor allocation: each request is extended to the next power of two, and allocations are always done is power-of-two blocks of processors. This scheme was evaluated with three different distributions: a uniform distribution in which all sizes are equally likely, a harmonic distribution in which the probability of size s is proportional to $1/s$, and a uniform distribution on powers of two. Both analysis and simulations showed significant differences between the utilizations that could be obtained for the three distributions [12]. This corresponds to different degrees of fragmentation that are inherent to packing with these distributions. For example, with a uniform distribution, rounding each request size up to the next power of two leads to 25% loss to fragmentation — the average between no loss (if the request is an exact power of two) to nearly 50% loss (if the request is just above a power of two, and we round up to the next one). The DHC scheme recovers part of this lost space, so the figure is actually only 20% loss, as shown in Figure 1.

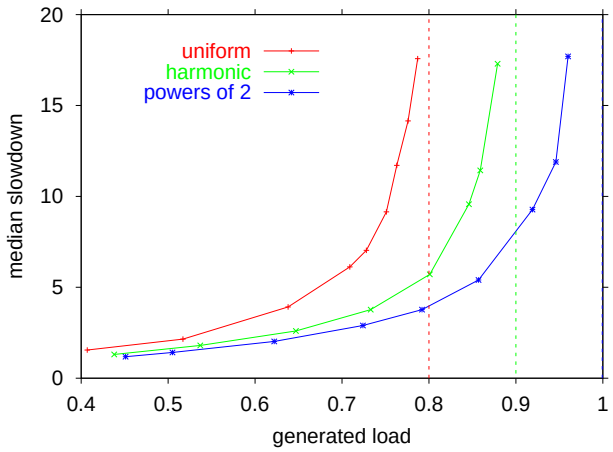

Fig. 1. *Simulation results showing normalized response time (slowdown) as a function of load for processor allocation using DHC, from [12]. The three curves are for exactly the same system — the only difference is in the statistics of the workload. The dashed lines are proven bounds on the achievable utilization for the three workloads.*

Note that this analysis tells us what to expect in terms of performance, provided we know the distribution of job sizes. But what is a typical distribution encountered in real systems in production use? Without such knowledge, the evaluation cannot provide a definitive answer.

2.2 Effect of Job Scaling Pattern

It is well-known that average response time is reduced by scheduling short jobs first. The problem is that the runtime is typically not known in advance. But in parallel systems scheduling according to job *size* may unintentionally also lead to scheduling by *duration*, if there is some statistical correlation between these two job attributes.

As it turns out, the question of whether such a correlation exists is not easy to settle. Three application scaling models have been proposed in the literature [30, 23]:

- *Fixed work.* This assumes that the work done by a job is fixed, and parallelism is used to solve the same problems faster. Therefore the runtime is assumed to be inversely proportional to the degree of parallelism (negative correlation). This model is the basis for Amdahl's law.
- *Fixed time.* Here it is assumed that parallelism is used to solve increasingly larger problems, under the constraint that the total runtime stays fixed. In this case, the runtime distribution is independent of the degree of parallelism (no correlation).
- *Memory bound.* If the problem size is increased to fill the available memory on the larger machine, the amount of productive work typically grows at

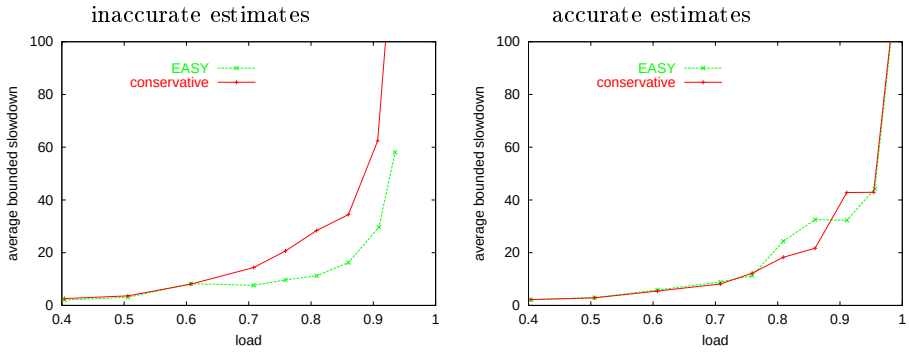

Fig. 2. *Comparison of EASY and conservative backfilling using the CTC workload, with inaccurate and accurate user runtime estimates.*

least linearly with the parallelism. The overheads associated with parallelism always grow superlinearly. Thus the total execution time actually increases with added parallelism (a positive correlation).

Evaluating job scheduling schemes with workloads that conform to the different models leads to drastically different results. Consider a workload that is composed of jobs the use power-of-two processors. In this case a reasonable scheduling algorithm is to cycle through the different sizes, because the jobs of each size pack well together [16]. This works well for negatively correlated and even uncorrelated workloads, but is bad for positively correlated workloads [16, 17]. The reason is that under a positive correlation the largest jobs dominate the machine for a long time, blocking out all others. As a result, the average response time of all other jobs grows considerably.

But which model actually reflects reality? Again, evaluation results depend on the selected model of scaling; without knowing which model is more realistic, we cannot use the performance evaluation results.

2.3 Effect of User Runtime Estimates

Returning to the 2D packing metaphor, a simple optimization is to allow the insertion of small jobs into holes left in the schedule. This is called backfilling, because new jobs from the back of the queue are used to fill current idle resources. The two common variants of backfilling are conservative backfilling, which makes strict reservations for all queued jobs, and EASY backfilling, which only makes a reservation for the first queued job [19]. Both rely on users to provide estimates of how long each job will run — otherwise it is impossible to know whether a backfill job may conflict with an earlier reservation. Users are expected to be highly motivated to provide accurate estimates, as low estimates improve the chance for backfilling and significantly reduced waiting time, but underestimates will cause the job to be killed by the system.

It has been shown that in some cases performance evaluation results depend in non-trivial ways on the accuracy of the runtime estimates. An example is given in Figure 2, where EASY backfilling is found to have lower slowdown with inaccurate estimates, whereas conservative backfilling is better at least for some loads when the estimates are accurate. This contradiction is the result of the following [8]. When using accurate estimates, the schedule does not contain large holes. The EASY scheduler is not affected too much, as it only heeds the reservation for the first queued job; other jobs do not figure in backfilling decisions. The conservative scheduler, on the other hand, achieves less backfilling of long jobs that use few processors, because it takes all queued jobs into account. This is obviously detrimental to the performance of these long jobs, but turns out to be beneficial for short jobs that don't get delayed by these long jobs. As the slowdown metric is dominated by short jobs, it shows the conservative backfiller to be better when accurate estimates are used, but not when inaccurate estimates are used.

Once again, performance evaluation has characterized the situation but not provided an answer to the basic question: which is better, EASY or conservative backfilling? This depends on the workload, and specifically, on whether user runtime estimates are indeed accurate as we expect them to be.

3 Workload Analysis and Modeling

As shown above, workloads can have a big impact on performance evaluation results. And the mechanisms leading to such effects can be intricate and hard to understand. Thus it is crucially important that representative workloads be used, which are as close as possible to the real workloads that may be expected when the system is actually deployed. In particular, unbased assumptions about the workload are very dangerous, and should be avoided.

3.1 Data-Less Modeling

But how does one know what workload to expect? In some cases, when truly innovative systems are designed, it is indeed impossible to predict what workloads will evolve. The only recourse is then to try and predict the space of possible workloads, and thoroughly sample this space. In making such predictions, one should employ recurring patterns from known workloads as guidelines. For example, workloads are often bursty and self-similar, process or task runtimes are often heavy tailed, and object popularity is often captured by a Zipf distribution [4].

3.2 Data-Based Modeling

The more common case, however, is that new systems are an improvement or evolution of existing ones. In such cases, studying the workload on existing systems can provide significant data regarding what may be expected in the future.

The case of job scheduling on parallel systems is especially fortunate, because data is available in the form of accounting logs [22]. Such logs contain the details of all jobs run on the system, including their arrival, start, and end times, the number of processors they used, the amount of memory used, the user who ran the job, the executable file name, etc. By analyzing this data, a statistical model of the workload can be created [7, 9]. This should focus on recurrent features that appear in logs derived from different installations. At the same time, features that are inconsistent at different installations should also be identified, so that their importance can be verified.

A good example is the first such analysis, published in 1995, based on a log of three months of activity on the 128-node NASA Ames iPSC/860 hypercube supercomputer. This analysis provided the following data [11]:

- The distribution of job sizes (in number of nodes) for system jobs, and for user jobs classified according to when they ran: during the day, at night, or on the weekend.
- The distribution of total resource consumption (node seconds), for the same job classifications.
- The same two distributions, but classifying jobs according to their type: those that were submitted directly, batch jobs, and Unix utilities.
- The changes in system utilization throughout the day, for weekdays and weekends.
- The distribution of multiprogramming level seen during the day, at night, and on weekends. This also included the measured down time (a special case of 0 multiprogramming).
- The distribution of runtimes for system jobs, sequential jobs, and parallel jobs, and for jobs with different degrees of parallelism. This includes a connection between common runtimes and the queue time limits of the batch scheduling system.
- The correlation between resource usage and job size, for jobs that ran during the day, at night, and over the weekend.
- The arrival pattern of jobs during the day, on weekdays and weekends, and the distribution of interarrival times.
- The correlation between the time a job is submitted and its resource consumption.
- The activity of different users, in terms of number of jobs submitted, and how many of them were different.
- Profiles of application usage, including repeated runs by the same user and by different users, on the same or on different numbers of nodes.
- The dispersion of runtimes when the same application is executed many times.

Practically all of this empirical data was unprecedented at the time. Since then, several other datasets have been studied, typically emphasizing job sizes and runtimes [27, 14, 15, 6, 2, 1, 18]. However, some new attributes have also been considered, such as speedup characteristics, memory usage, user estimates of runtime, and the probability that a job be cancelled [20, 10, 19, 2].

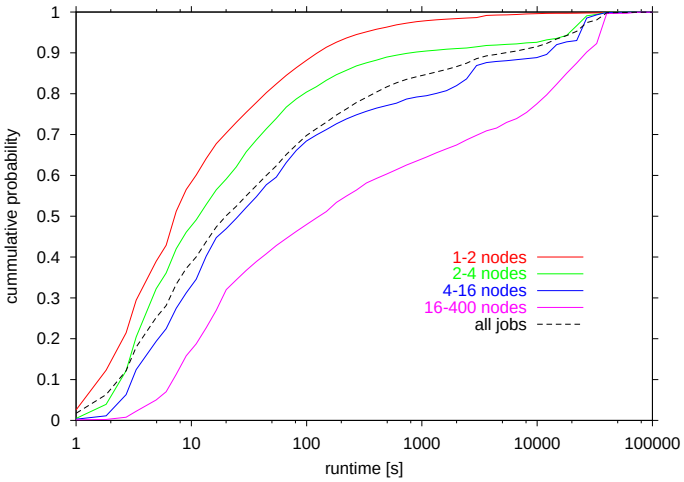

Fig. 3. *The cumulative distribution functions of runtimes of jobs with different sizes, from the SDSC Paragon.*

3.3 Some Answers and More Questions

Based on such analyses, we can give answers to the questions raised in the previous section. All three are rather surprising.

The distribution of job sizes has often been assumed to be bimodal: small jobs that are used for debugging, and large jobs that use the full power of the parallel machine for production runs. In fact, there are very many small jobs and rather few large jobs, and large systems often do not have *any* jobs that use the full machine. especially surprising is the high fraction of serial jobs, which is typically in the range of 20–30%. Another prominent feature is the emphasis on power-of-two job sizes, which typically account for over 80% of the jobs. This has been claimed to be an artifact of the use of such size limits in the queues of batch scheduling system, or the result of inertia in system where such limits were removed; the claim is supported by direct user data [3]. Nevertheless, the fact remains that users continue to prefer powers of two. The question for workload modeling is then whether to use the "real" distribution or the empirical distribution in models.

It is hard to obtain direct evidence regarding application scaling from accounting logs, because they typically do not contain runs of the same applications using different numbers of nodes, and even if they did, we do not know whether these runs were aimed at solving the same problem. However, we can compare the runtime statistics of jobs that use different numbers of nodes. the result is that there is little if any correlation in the statistical sense. However, the distributions of runtimes for small and large jobs do tend to be different, with large jobs often having longer runtimes [7] (Figure 3). This favors the memory bound or fixed time scaling models, and contradicts the fixed work model. There is also some evidence that larger jobs use more memory [10]. Thus, within a sin-

gle machine, parallelism is in general not used for speedup but for solving larger problems.

Direct evidence regarding user runtime estimates is available in the logs of machines that use backfilling. This data reveals that users typically overestimate job runtime by a large factor [19]. This indicates that the expectations about how users behave are wrong: users are more worried about preventing the system from killing their job than about giving the system reliable data to work with. This leads to the question of how to model user runtime estimates. In addition, the effect of the overestimating is not yet fully understood. One of the surprising results is that overestimating seems to lead to better overall performance than using accurate estimates [19].

4 A Workloads RFI[1]

There is only so much data that can be obtained from accounting logs that are collected anyway. To get a more detailed picture, active data collection is required. When studying the performance of parallel systems, we need high-resolution data about the behavior of applications, as this affects the way they interact with each other and with the system, and influences the eventual performance measures.

4.1 Internal Structure of Applications

Workload models based on job accounting logs tend to regard parallel jobs as rigid: they require a certain number of processors for a given time. But runtime may depend on the system. For example, runs of the ESP system-level benchmark revealed that executions of the same set of jobs on two different architectures led to completely different job durations [28]. The reason is that different applications make different use of the system in terms of memory, communication, and I/O. Thus an application that requires a lot of fine-grain communication may be relatively slow on a system that does not provide adequate support, but relatively fast on a system with an overpowered communication network.

In order to evaluate advanced schedulers that take multiple resources into account we therefore need more detailed workload models. It is not enough to model a job as a rectangle in processors×time space. We need to know about its internal structure, and model that as well. Such a model can then form the basis for an estimation of the speedup a job will display on a given system, when provided with a certain set of resources.

A simple proposal was given in [13]. The idea is to model a parallel application as a set of tasks, which are either independent of each other, or need to synchronize repeatedly using barriers. The number of tasks, number of barriers, and granularity are all parameters of the model. While this is a step in

[1] Request for Information

the right direction, the modeling of communication is minimal, and interactions with other system resources are still missing. Moreover, representative values for the model parameters are unknown.

There has been some work on characterizing the communication behavior of parallel applications [5, 25]. This has confirmed the use of barrier-like collective communications, but also identified the use of synchronization-avoiding non-blocking communication. The granularity issue has remained open: both very small and very big intervals between communication events have been measured, but the small ones are probably due to multiple messages being sent one after the other in the same communication phase. The granularity of computation phases that come between communication phases is unclear. Moreover, the analysis was done for a small set of applications in isolation; what we really want to know is the distribution of granularities in a complete workload.

More detailed work was done on I/O behavior [21, 24]. Like communication, I/O is repetitive and bursty. But again, the granularity at which it occurs (or rather, the distribution of granularities in a workload) is unknown. An interesting point is that interleaved access from multiple processes to the same file may lead to synchronization that is required in order to use the disks efficiently, even if the application semantics do not dictate any strict synchronization.

Very little work has been done on the memory behavior of parallel applications. The conventional wisdom is that large-scale scientific applications require a lot of memory, and use all of it all the time without any significant locality. Still, it would be nice to root this in actually observations, especially since it is at odds with reports of the different working set sizes of SPLASH applications [29]. Somewhat disturbing also is a single paper that investigated the paging patterns of different processes in the same job, and unexpectedly found them to be very dissimilar [26]. More work is required to verify or refute the generality of this result.

4.2 User Behavior

Workload models typically treat job arrivals as coming from some independent external source. Their statistics are therefore independent of the system behavior. While this makes the evaluation easier, it is unrealistic. In reality, the user population is finite and often quite small; when the users perceive the system as not responsive, they tend to reduce their use (Figure 4). This form of negative feedback actually fosters system stability and may prevent overload conditions.

Another important aspect of user behavior is that users tend to submit the same job over and over again. Thus the workload a system has to handle may be rather homogeneous and predictable. This is very different from a random sampling from a statistical distribution. In fact, it can be called "localized sampling": while over large stretches of time, e.g. a whole year, the whole distribution is sampled, in any given week only a small part of it is sampled.

In terms of performance evaluation, two important research issues may be identified in this regard. One is how to perform such localized sampling, or in other words, how to characterize, model, and mimic the short-range locality

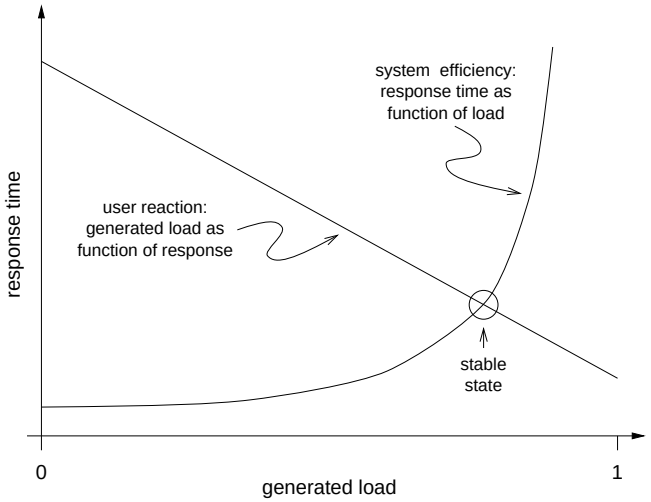

Fig. 4. *The workload placed on a system may be affected by the system performance, due to a feedback loop through the users.*

of real workloads. the other is to figure out what effect this has on system performance, and under what conditions.

5 The Rocky Road Ahead

Basing performance evaluation on facts rather than on assumptions is important. But it shouldn't turn into an end in itself. As Henri Poincaré said,

> Science is built up with facts, as a house is with stones. But a collection of facts is no more a science than a heap of stones is a house.

The systems we now build are complex enough to require scientific methodology to study their behavior. This must be based on observation and measurement. But knowing what to measure, and how to connect the dots, is not easy.

Realistic and detailed workload models carry with them two dangers. One is clutter and obfuscation — with more details, more parameters, and more options, there are more variations to check and measure. Many of these are probably unimportant, and serve only to hide the important ones. The other danger is the substitution of numbers for understanding. With more detailed models, it becomes harder to really understand the fundamental effects that are taking place, as opposed to merely describing them. This is important if we want to learn anything that will be useful for other problems except the one at hand. These two dangers lead to a quest for Einstein's equilibrium:

> Everything should be made as simple as possible, but not simpler.

The challenge is to identify the important issues, focus on them, and get them right. Unbased assumptions are not good, but excessive detail and clutter is probably not better.

Acknowledgement

This research was supported by the Israel Science Foundation (grant no. 219/99).

References

1. S-H. Chiang and M. K. Vernon, "*Characteristics of a large shared memory production workload*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 159–187, Springer Verlag, 2001. Lect. Notes Comput. Sci. vol. 2221.
2. W. Cirne and F. Berman, "*A comprehensive model of the supercomputer workload*". In 4th *Workshop on Workload Characterization*, Dec 2001.
3. W. Cirne and F. Berman, "*A model for moldable supercomputer jobs*". In 15th *Intl. Parallel & Distributed Processing Symp.*, Apr 2001.
4. M. E. Crovella, "*Performance evaluation with heavy tailed distributions*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 1–10, Springer Verlag, 2001. Lect. Notes Comput. Sci. vol. 2221.
5. R. Cypher, A. Ho, S. Konstantinidou, and P. Messina, "*A quantitative study of parallel scientific applications with explicit communication*". *J. Supercomput.* **10(1)**, pp. 5–24, 1996.
6. A. B. Downey, "*A parallel workload model and its implications for processor allocation*". In 6th *Intl. Symp. High Performance Distributed Comput.*, Aug 1997.
7. A. B. Downey and D. G. Feitelson, "*The elusive goal of workload characterization*". *Performance Evaluation Rev.* **26(4)**, pp. 14–29, Mar 1999.
8. D. G. Feitelson, *Analyzing the Root Causes of Performance Evaluation Results*. Technical Report 2002–4, School of Computer Science and Engineering, Hebrew University, Mar 2002.
9. D. G. Feitelson, "*The effect of workloads on performance evaluation*". In *Performance Evaluation of Complex Systems: Techniques and Tools*, M. Calzarossa (ed.), Springer-Verlag, Sep 2002. Lect. Notes Comput. Sci. Tutorials.
10. D. G. Feitelson, "*Memory usage in the LANL CM-5 workload*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 78–94, Springer Verlag, 1997. Lect. Notes Comput. Sci. vol. 1291.
11. D. G. Feitelson and B. Nitzberg, "*Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 337–360, Springer-Verlag, 1995. Lect. Notes Comput. Sci. vol. 949.
12. D. G. Feitelson and L. Rudolph, "*Evaluation of design choices for gang scheduling using distributed hierarchical control*". *J. Parallel & Distributed Comput.* **35(1)**, pp. 18–34, May 1996.
13. D. G. Feitelson and L. Rudolph, "*Metrics and benchmarking for parallel job scheduling*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 1–24, Springer-Verlag, 1998. Lect. Notes Comput. Sci. vol. 1459.

14. S. Hotovy, "*Workload evolution on the Cornell Theory Center IBM SP2*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 27–40, Springer-Verlag, 1996. Lect. Notes Comput. Sci. vol. 1162.
15. J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira, and J. Riodan, "*Modeling of workload in MPPs*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 95–116, Springer Verlag, 1997. Lect. Notes Comput. Sci. vol. 1291.
16. P. Krueger, T-H. Lai, and V. A. Dixit-Radiya, "*Job scheduling is more important than processor allocation for hypercube computers*". *IEEE Trans. Parallel & Distributed Syst.* **5(5)**, pp. 488–497, May 1994.
17. V. Lo, J. Mache, and K. Windisch, "*A comparative study of real workload traces and synthetic workload models for parallel job scheduling*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 25–46, Springer Verlag, 1998. Lect. Notes Comput. Sci. vol. 1459.
18. U. Lublin and D. G. Feitelson, *The Workload on Parallel Supercomputers: Modeling the Characteristics of Rigid Jobs.* Technical Report 2001-12, Hebrew University, Oct 2001.
19. A. W. Mu'alem and D. G. Feitelson, "*Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling*". *IEEE Trans. Parallel & Distributed Syst.* **12(6)**, pp. 529–543, Jun 2001.
20. T. D. Nguyen, R. Vaswani, and J. Zahorjan, "*Parallel application characterization for multiprocessor scheduling policy design*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 175–199, Springer-Verlag, 1996. Lect. Notes Comput. Sci. vol. 1162.
21. N. Nieuwejaar, D. Kotz, A. Purakayastha, C. S. Ellis, and M. L. Best, "*File-access characteristics of parallel scientific workloads*". *IEEE Trans. Parallel & Distributed Syst.* **7(10)**, pp. 1075–1089, Oct 1996.
22. *Parallel workloads archive.* URL http://www.cs.huji.ac.il/labs/parallel/workload/.
23. J. P. Singh, J. L. Hennessy, and A. Gupta, "*Scaling parallel programs for multiprocessors: methodology and examples*". *Computer* **26(7)**, pp. 42–50, Jul 1993.
24. E. Smirni and D. A. Reed, "*Workload characterization of input/output intensive parallel applications*". In 9th *Intl. Conf. Comput. Performance Evaluation*, pp. 169–180, Springer-Verlag, Jun 1997. Lect. Notes Comput. Sci. vol. 1245.
25. J. S. Vetter and F. Mueller, "*Communication characteristics of large-scale scientific applications for contemporary cluster architectures*". In 16th *Intl. Parallel & Distributed Processing Symp.*, May 2002.
26. K. Y. Wang and D. C. Marinescu, "*Correlation of the paging activity of individual node programs in the SPMD execution model*". In 28th *Hawaii Intl. Conf. System Sciences*, vol. I, pp. 61–71, Jan 1995.
27. K. Windisch, V. Lo, R. Moore, D. Feitelson, and B. Nitzberg, "*A comparison of workload traces from two production parallel machines*". In 6th *Symp. Frontiers Massively Parallel Comput.*, pp. 319–326, Oct 1996.
28. A. Wong, L. Oliker, W. Kramer, T. Kaltz, and D. Bailey, "*System utilization benchmark on the Cray T3E and IBM SP2*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 56–67, Springer Verlag, 2000. Lect. Notes Comput. Sci. vol. 1911.
29. S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "*The SPLASH-2 programs: characterization and methodological considerations*". In 22nd *Ann. Intl. Symp. Computer Architecture Conf. Proc.*, pp. 24–36, Jun 1995.
30. P. H. Worley, "*The effect of time constraints on scaled speedup*". *SIAM J. Sci. Statist. Comput.* **11(5)**, pp. 838–858, Sep 1990.