Benchmarks and Standards for the Evaluation of Parallel Job Schedulers

Steve J. Chapin[1], Walfredo Cirne[2], Dror G. Feitelson[3], James Patton Jones[4], Scott T. Leutenegger[5], Uwe Schwiegelshohn[6], Warren Smith[7], and David Talby[3]

[1] Department of Electrical Engineering and Computer Science
Syracuse University, Syracuse, NY 13244-1240
chapin@cs.virginia.edu
[2] Computer Science and Engineering Department
University of California San Diego, La Jolla, CA 92093
walfredo@cs.ucsd.edu
[3] Institute of Computer Science
Hebrew University, 91904 Jerusalem, Israel
{feit,davidt}@cs.huji.ac.il
[4] MRJ Technology Solutions
NASA Ames Research Center, Moffett Field, CA 94035
jjones@nas.nasa.gov
[5] Mathematics and Computer Science Department
University of Denver, Denver, CO 80208
leut@cs.du.edu
[6] Computer Engineering Institute
University Dortmund, 44221 Dortmund, Germany
uwe@ds.e-technik.uni-dortmund.de
[7] Mathematics and Computer Science Division
Argonne National Laboratory, Argonne, IL 60439
wsmith@mcs.anl.gov

Abstract. The evaluation of parallel job schedulers hinges on the workloads used. It is suggested that this be standardized, in terms of both format and content, so as to ease the evaluation and comparison of different systems. The question remains whether this can encompass both traditional parallel systems and metacomputing systems.
This paper is based on a panel on this subject that was held at the workshop, and the ensuing discussion; its authors are both the panel members and participants from the audience. Naturally, not all of us agree with all the opinions expressed here...

1 Introduction

1.1 Motivation

The study and design of computer systems requires good models of the workload to which these systems are subjected, because the workload has a large effect

on the observed performance. This need was recognized long ago [26,1], and in several fields workload data was indeed collected, analyzed, and modeled. Well-known examples are address traces used to analyze processor cache performance [56,59], and records of file system activity used to motivate the use of file caching [49]. Recently we are witnessing a large increase in such activity, with data being collected relating to LAN traffic [45], web server loads [3], and video streams [43].

This new wave of collecting and analyzing data for use in evaluations is also present in the field of job scheduling on high-performance systems. Two approaches can be identified. One is to collect the data, describe it [21,60,37], and use it directly as input for future evaluations. This has the benefit of being considered completely realistic, but also suffers from various methodological concerns such as the danger that the data reflects local constraints rather than general principles [41,36]. The other approach is to use the data as a reference in designing workload models that are used to drive the evaluation. By selecting only invariants found in several data sets for inclusion in the model, the confidence in the model is improved [18,16].

A problem that remains is that too many workloads are now available, be they naive models based on guesswork, complex models based on measurements, or the measurements themselves. Faithful comparisons of different schemes require a representative set of workloads to be canonized as a benchmark, and used by all subsequent studies. The definition of a standard benchmark should include both the benchmark data (or a program to generate it), and its format, to enable efficient and easy use. Our goal in this paper is to explore the possibility of creating such a standard.

1.2 Scope

Application scheduling versus job scheduling Benchmarks are only useful if they sufficiently represent their target community. For instance, SPEC benchmarks have been carefully selected to cover a wide range of different applications. Similarly, benchmarks for the evaluation of parallel job schedulers must be based on the applications typically run on those parallel machines. Using a slightly simplified view we can distinguish two classes for these applications:

- *Rigid applications*[1] which are fine tuned for a specific parallel machine and configuration. The most common examples are programs written in the message passing paradigm, where all communication between the processors is carefully arranged to achieve a large degree of latency hiding. Such programs cannot cope with situations where the number of processors is reduced even by one during the execution, and there is also no benefit from assigning additional processors, as they will remain unused.

[1] This includes *moldable* applications [25] which are written so that they can run on different numbers of processors as chosen when the job *starts* execution; the point is that the job cannot change *during* execution, so there is no application scheduler.

– *Flexible applications*[2] which can be run on a variety of different machine configurations. Typically, a high degree of efficiency can only be achieved for these jobs if they are made adaptable to the actual configuration. Therefore, they frequently consist of a large number of interdependent modules for which a suitable schedule must be generated. A simple approach is to use a master-workers structure.

Based on these two applications classes it is also appropriate to distinguish two types of schedulers: machine schedulers and application schedulers. Machine schedulers for large parallel machines are, naturally, machine-centric. They typically do not look much inside a job. As input they receive characteristic data from a stream of independent jobs. Computing resources, like processors, memory, or I/O facilities, are allocated to these jobs with the goal of optimizing the value of the actual scheduling objective function. Therefore, machine schedulers try to keep the number of unassigned resources at a minimum while load balancing within a job is up to the owner of the job. Machine schedulers must deal with the on-line character of job submission and with a potential inaccuracy of job submission data, like the estimated execution time of a job. On the other hand they need not consider dependences between the submitted jobs. The performance of a machine scheduler may be highly dependent on the workload and possibly on the given objective function. Having a representative workload may therefore allow the administrator of a parallel machine to determine the scheduler best suited for him. Hence, those administrators can be assisted by a set of benchmarks that cover most workloads occurring in practice.

Application schedulers, on the other hand, arrange the modules of flexible applications to make best use of the currently available resources. They do not consider other independent jobs running concurrently on the same machine. Therefore, they are application-centric. Typically, it is their goal to minimize the overall execution time of their applications. To this end they must consider the dependences between the various modules of their applications. All modules are known to the schedulers up front. While quite a few different algorithms for application schedulers have been suggested it is not clear whether their performance varies significantly for different applications. It may therefore be possible to evaluate application schedulers with the help of a generic application model. In this case benchmarks for application schedulers are not needed.

But if application schedulers start to proliferate they may significantly influence the workload characteristics of parallel machines, changing it from being predominantly rigid to mostly flexible. It is also possible that machine schedulers and application schedulers may cooperate in the future to make best use of the available resources. The state of the art in workload benchmarking for rigid jobs, and questions about extending it to flexible jobs, are discussed in Section 2.

Scheduling for metacomputing and its requirements A recent area of research is how to collect resources from many organizations into entities called

[2] This is taken to include both *malleable* and *evolving* jobs in the terminology of [25].

metasystems or computational grids [28]. A metasystem consists of computers, networks, databases, instruments, visualization devices, and other types of resources owned by different organizations and located around the world. In addition to these resources, a metasystem contains software that people use to access it. There are several projects that provide such software [27,33,46,6] and, among many other things, this software supports meta schedulers: schedulers that help users select what resources to use for an application and help users to execute their application on those resources.

While there are many types of meta schedulers, they often have several common requirements. First, a user or meta scheduler has a larger and more diverse set of resources to pick from than those present in a single supercomputer. A meta scheduler therefore needs information about resources and applications to determine which resources to select for an application. A meta scheduler needs to know when resources are available, what they cost, which users have access to them, how an application performs on them, etc. Information on current availability of resources is easily available and there is ongoing work on predicting the future availability of network bandwidth [61] and when a scheduler will start applications [57,14]. Predictions of application performance on various sets of resources is also being investigated [6]. Even though this information is becoming available, an additional need is a common way to gain access to this information such as the Metacomputing Directory Service provided by the Globus [27] software.

In addition to the new types of information described above, many meta schedulers need resources from more than one source — similar to the idea of gang scheduling on parallel machines [22]. This requires mechanisms for gaining simultaneous access to resources. One such mechanism is reserving resources at some future time. Mechanisms for network quality of service [29] allow such reservation of networking resources and reservation mechanisms are currently being added to scheduling systems for parallel computers [54].

The issues of benchmarking the application schedulers for metacomputing are discussed in Section 3, and the relationship between scheduling on parallel systems and metasystems are examined in Section 4.

Possible inclusion of the objective function The measured performance of a system depends not only on the system and workload, but also on the metrics used to gauge performance. It is these metrics that serve as the objective function of the scheduler, whose goal is to optimize their value. For some objective functions, such as utilization and throughput, the goal is to maximize; for others, such as response time or slowdown, the goal is to minimize.

The problem is that measurement using different metrics may lead to conflicting results. For example, one of the papers in the workshop showed contradicting results for the comparison of two scheduling algorithms if response time or slowdown were used as a metric [31]. Another paper [42] specifically addressed the issue of deriving objective functions tailored to a set of owner defined policy rules. This paper also showed significant differences in the ranking of various

scheduling algorithms if applied to objective functions that only differ in the selection of a weight. It may therefore be appropriate to standardize the objective functions that are used, in order to enable a truthful comparison between different studies. However, this is only appropriate if a large number of different objective functions are used in practice and if machine schedulers produce significantly different results for those different objective functions. Currently, only a few standard objective functions — like the average response time or the machine utilization — can be found in almost all installations. However, it is not clear whether this small number is due to a missing concept for generating objective functions that are better tailored to the rules of the owners of parallel machines.

In this paper we do not discuss this issue further. We just note that further research into the relative merits of different metrics is needed [24].

2 Workload Benchmarks for Parallel Systems

A mere five years ago practically no real data about production workloads on parallel machines was available, so evaluations had to rely on guesswork. This situation has changed dramatically, and now practically all evaluations of parallel job schedulers rely on real data, at least to some degree. While more details can always be added, the time seems ripe to start talking about standardization of workload benchmark data.

2.1 State of the Art

A large amount of data on production parallel supercomputers has been collected in the Parallel Workloads Archive [19]. This includes both raw logs and derived models.

Workload logs Most parallel supercomputers maintain accounting logs for administrative use. These logs contain valuable information about all the activity on the machine, and in particular, about the attributes of each job that was executed. The format of the logs is typically an ASCII file with one line per job (although some systems maintain a much more detailed log). Analyzing such logs can lead to important insights into the workload. Such work has been done for some systems, including the NASA Ames iPSC/860 [21], the SDSC Paragon [60], the CTC SP2 [37], and the LANL CM-5 [17].

While most logs contain the same core data about each job (such as the submittal, start, and end times, the number of processors used, and the user ID), there are other less-standard fields as well. Some systems contain data about resource requests made before the job started. Some contain data about additional resources such as memory usage. Some contain internal data about the queue to which the job was submitted, and prioritization parameters used by the scheduler. Moreover, these fields appear in different orders and formats. The standard format suggested below attempts to accommodate all the important and useful fields, even if they do not appear in every log.

Workload models Workload models are based on some statistical analysis of workload logs, with the goal of elucidating their underlying principles. This then enables the creation of new workloads that are statistically similar to the observations, but can also be changed at will (e.g. to modify the system load) [16].

The most salient feature of workload models is that they include exactly what the modeler puts into them. This is both an advantage and a disadvantage. It is an advantage because the modeler knows about all the features of the model, and can control them. It is a disadvantage because real workloads may contain additional features that are unknown, and therefore not included in the models. As the effect of various workload features is typically not known in advance, it is prudent to at least include as many known workload features as possible.

Current workload models fall into two categories: those of rigid jobs, and those of flexible jobs. Rigid job models create a sequence of jobs with given arrival time, number of processors, and runtime (e.g. [18,39,47]). The task of the scheduler is then to pack these "rectangular" jobs onto the machine. Given the relative simplicity of rigid jobs, a number of rather advanced models have been designed. A statistical analysis [58] shows that the one proposed by Lublin [47] is relatively representative of multiple workloads.

Flexible job models attempt to describe how an application would perform with different resource allocations, and maybe even how it would perform if the resources are changed at runtime. One way to do this is to provide data about the total computation and the speedup function [55,13], instead of the required number of processors and runtime. This enables the scheduler to choose the number of processors that will be used, according to the current load conditions. Another approach is to provide an explicit model of the internal structure of the application [7,24]. This allows for a detailed simulation of the interactions between the scheduling and the application, leading to better evaluations at the cost of more complex simulation. While several models have been proposed, there is still insufficient data about the relative distribution of applications with different speedup characteristics and internal structures to allow for any statements regarding which is more representative.

2.2 Future Work

Workload models may be improved in three main ways: by including additional resources, such as memory and I/O, by including feedback, and by including the internal structure of parallel programs. In addition, the evaluation of schedulers will benefit from data about outages that schedulers have to deal with.

Including memory requirements and I/O Current workload models concentrate on one type of resource: computing power. However, in reality, jobs require other resources as well, and the interaction between the demands for different resources can have a large effect on possible schedules.

One resource that has received some attention is memory. Several papers acknowledge the importance of memory requirements and their effect on scheduling

[2,51,50]. However, there is only little data about actual memory usage patterns [17], and this has so far not been incorporated in any workload model. Moreover, it is necessary to model not only the total amount of memory that is used, but also the degree of locality with which it is accessed, as this has a great impact on the amount of memory that has to be allocated in practice [4].

Another important characteristic that has a significant impact on scheduling is I/O activity. The Charisma project has collected some data on the I/O behavior of parallel programs [48][3], but this has only been used for the design of parallel file system interfaces. We are only beginning to see considerations of I/O in scheduling work [44,53], but this is so far not based on much real data. As real applications obviously do perform I/O (and sometimes even a lot of it), this is a severe deficiency in current practice.

For both memory and I/O, we do not have enough data yet for contemplating a standard benchmark, at least not one that is known to be representative and is based on measurements.

Including feedback Another problem with current workload models is the lack of feedback. The observed workload on a production machine is *not* created by random sampling from a population of programs. Rather, it is the result of interleaving the sequences of activities performed by many human beings. Activities in such sequences are often dependent on each other: you first edit your program, then compile it, and then execute it; you change parameters and execute it again after observing the results of the previous execution. Thus the instant at which a job is submitted to the system may depend on the termination of a previous job. As the time of the previous termination depends on the system's performance, so does the next arrival. In a nutshell, there is a feedback effect from the system's performance to the workload.

The realization that such feedback exists is not new. In fact, feedback has been included explicitly in some queueing studies, especially those employing closed queueing networks with a delay center representing user think time in the feedback loop (see, e.g., [38]). However, this practice has so far not extended to performance analysis based on observed workloads, because it does not appear explicitly in the observations. Accounting logs do not include explicit information about feedback, so this effect is lost when a log is replayed and used in an evaluation. However, it is possible to make educated guesses in order to insert postulated dependencies into an existing log. The methodology is straight forward: we identify sequences of dependent jobs (e.g. all those submitted by the same user in rapid succession), and replace the absolute arrival times of jobs in the sequence with interarrival times relative to the previous job in the sequence.

Including the internal job structure The feedback noted above is between the system and the user, and may affect the arrival process. There is also a

[3] A historical note — the Charisma data actually triggered the first study of a production parallel workload in [21].

possibility of feedback between the system and the parallel job itself. Specifically, the synchronization and communication patterns of the application may have various performance implications, that depend on how the application's processes are scheduled to different processors [35,23].

For example, earlier work in the sigmetrics community compared space slicing with time slicing. Two orthogonal issues were allocation of processing power among jobs and support for interprocess synchronization (IPS). The space slicing work recognized the importance of processing power allocation and developed dynamic and/or adaptive algorithms. Some of the algorithms necessitated fairly complicated mechanisms to ensure processor allocations could be changed and not hurt interprocessor synchronization. If synchronization *is* frequent, then either gang scheduling or IPS cognizant space slicing mechanisms are needed, but if common IPS is coarse grained it may be unnecessary. Assuming it is necessary, it may still be possible that IPS is coarse grained enough when doing gang scheduling that alternates could be fragments rather than requiring complete gangs be coscheduled.

In last year's introductory paper we presented a strawman proposal of how the internal structure of a parallel application can be summarized by a small number of parameters [24]. The main parameters were the number of processors, the number of barriers, the granularity, and the variance of these attributes. While this cannot capture the full spectrum of possible parallel applications, it is expected to provide enough flexibility in order to create a varied workload that will exercise the interactions between applications and the scheduler in various ways.

The problem with including internal structure in the workload benchmark is the complete lack of knowledge about what parameter values to use. This information could be collected by augmenting a library providing synchronization facilities to trace this information (as was done in Charisma for the I/O library). This functionality already exists in PVM and Legion for example. If the library is a dynamic library then theoretically it would be easy to take someone's code and measure it. Such an undertaking has to be done at a large production site, provided it would not slow down users production level codes for measurement purposes.

An obvious alternative to modeling the internal structure is to use real applications [62,12]. However, the question remains of which applications to use, in what mixes, and how to create different sizes. This again boils down to the question of how to create a representative workload, and the lack of data about the relative popularity of different application types.

Including outage information While simulations and models are useful for comparing different algorithms, in the real world, there are many more variables that come into play than the few that are typically used in scheduling models. If the purpose of running a new scheduling algorithm through a simulator on a real workload is to measure how well that algorithm will work in production on

a similar workload, then it cannot possibly be accurate if it ignores all factors external to a scheduler's trace file.

Parallel systems have matured considerably over the past decade, but still are not as stable or reliable as traditional vector systems like the Cray C90. This instability should be taken into consideration when creating a scheduler simulator. Such factors as node failure, network interruption, disk failure, mean time between failure, and length of failures are important variables that a production scheduler has to cope with. In a distributed memory system like the IBM SP, it is possible for a node to drop offline, but the system continues to operate. Any job running on that node would have to be restarted, but it has no affect on any other running jobs. The system scheduler detects the failed nodes, and takes action to schedule around the failed hardware. This information however is not recorded in typical job trace files, and is therefore not taken into account during the analysis of the traces.

Another important aspect of system availability is the impact of human-generated outages. All production systems are taken down for scheduled maintenance and often for dedicated time. This outage information is often available to the job scheduler so that jobs can be scheduled around the outages, or such that the system is drained up to the outage. This information does not appear in the scheduler trace files, but is needed input for simulators. Most sites collect outage data, and many archive it for historical comparisons (like NAS). A standard format for outage data should be created to compliment the scheduling workload traces. The two datasets should be keyed to each other, and should contain the necessary information to accurately predict scheduler behavior in a real work environment.

As an initial start, we propose the following information should be collected and reported in a standard format, for every outage that removes any portion of a system from operation:

- Announced time of outage (e.g. when did the outage info become available to the scheduler — was it known in advance, or did the scheduler suddenly detect that there were fewer nodes available?)
- Start time of outage (when the outage actually occurred)
- End time of outage (when the affected resources were again schedulable)
- Type of outage (CPU failure, network failure, facility)
- Number of nodes affected (or perhaps percentage of machine affected — for example, a failed scratch file system may prevent only a few users from running, but the others can continue.)
- Specific affected components (which nodes went down, what part of the network failed)

2.3 A Standard Workload Format

The goal of the standard format is to help researchers using workloads, either real or synthetic. Its main advantages over what is currently available are:

- Ideas and tests regarding workload models could be easily applied to all available workloads. This is rarely done because of the need to write scripts to handle the different formats of workloads today.
- The file format is easy to parse and use: while it is a text file (to avoid problems with converting data files) all data is in integers (no character strings!), so there are no problems with parsing dates or other special entries. This provides simplicity and absolute standardization at the expense of generality and extensibility: you are guaranteed to be able to parse and understand every file abiding by the standard, because users cannot add their own new fields.
- Every datum must abide to strict consistency rules, that when checked ensure that the workload is always "clean".
- Data is in standard units. Moreover, users and executables are given by incremental numbers, which makes their parsing easier, makes grouping by users/executables easier, hides administrative issues, and hides sensitive information.

A major design goal was to be able to use the format for both real and synthetic workloads. This means that only some of the fields will usually be meaningful for any given workload — a synthetic workload may only include information about submit times, runtimes, and parallelism, while a real workload won't include any information about scheduler feedback. Therefore, unknown values are part of the standard. The fields were chosen so that all information from logs we have will be saved except very rare fields (that appeared in only one log, for example). For synthetic workloads, future research directions were also considered: For example, the format enables expressing the existence of scheduler feedback, which can be generated using a variety of models. The internal structure (I/O, barriers, and so forth) of jobs is still not included, since no logs and only one model address this issue and the right way of doing it is still unclear. Future version of the standard may include additional fields for this and other purposes.

The data fields Standard workload files contain one line per job, that contains a list of space separated integers. Missing values are denoted by -1, and all other values are non-negative. Lines beginning with a semicolon are treated as comments and ignored. The beginning of every file contains several such lines that describe the workload in general. The jobs are numbered consecutively in the file. Job IDs from workloads that are converted to the standard format are discarded, since they are not always integers and not always unique (if they combine data from several years). Each line in the file has these fields, in this order:

1. Job Number — a counter field, starting from 1.
2. Submit Time — in seconds. The earliest time the log refers to is zero, and is the submittal time the of the first job. The lines in the log are sorted by ascending submittal times.

3. Wait Time — in seconds. The difference between the job's submit time and the time at which it actually began to run. Naturally, this is only relevant to real logs, not to models.
4. Run Time — in seconds. The wall clock time the job was running (end time minus start time).
   We decided to use "wait time" and "run time" instead of the equivalent "start time" and "end time" because they are directly attributable to the scheduler and application, and are more suitable for models where only the run time is relevant.
5. Number of Allocated Processors — an integer. In most cases this is also the number of processors the job uses; if the job does not use all of them, we typically don't know about it.
6. Average CPU Time Used — both user and system, in seconds. This is the average over all processors of the CPU time used, and may therefore be smaller than the wall clock runtime. If a log contains the total CPU time used by all the processors, it is divided by the number of allocated processors to derive the average.
7. Used Memory — in kilobytes. This is again the average per processor.
8. Requested Number of Processors.
9. Requested Time. This can be either runtime (measured in wallclock seconds), or average CPU time per processor (also in seconds) — the exact meaning is determined by a header comment. If a log contains a request for total CPU time, it is divided by the number of requested processors.
10. Requested Memory (again kilobytes per processor).
11. Completed? 1 if the job was completed, 0 if it was killed. This is meaningless for models, so would be -1.

if a log contains information about checkpoints and swapping out of jobs, a job can have multiple lines in the log. In fact, we propose that the job information appear *twice*. First, there will be one line that summarizes the whole job: its submit time is the submit time of the job, its runtime is the sum of all partial runtimes, and its code is 0 or 1 according to the completion status of the whole job. In addition, there will be separate lines for each instance of partial execution between being swapped out. All these lines have the same job ID and appear consecutively in the log. Only the first has a submit time; the rest only have a wait time since the previous burst. The completed code for all these lines is 2, meaning "to be continued"; the completion code for the *last* such line is 3 or 4, corresponding to completion or being killed. It should be noted that such details are only useful for studying the behavior of the logged system, and are not a feature of the workload. Such studies should ignore lines with completion codes of 0 and 1, and only use lines with 2, 3, and 4. For workload studies, only the single-line summary of the job should be used, as identified by a code of 0 or 1.

12. User ID — a natural number, between one and the number of different users.
13. Group ID — a natural number, between one and the number of different groups. Some systems control resource usage by groups rather than by individual users.

14. Executable (Application) Number — a natural number, between one and the number of different applications appearing in the workload. in some logs, this might represent a script file used to run jobs rather than the executable directly; this should be noted in a header comment.
15. Queue Number — a natural number, between one and the number of different queues in the system. The nature of the system's queues should be explained in a header comment. This field is where batch and interactive jobs should be differentiated: we suggest the convention of denoting interactive jobs by 0.
16. Partition Number — a natural number, between one and the number of different partitions in the systems. The nature of the system's partitions should be explained in a header comment. For example, it is possible to use partition numbers to identify which machine in a cluster was used.
17. Preceding Job Number — this is the number of a previous job in the workload, such that the current job can only start after the termination of this preceding job. Together with the next field, this allows the workload to include feedback as described in Section 2.2.
18. Think Time from Preceding Job — this is the number of seconds that should elapse between the termination of the preceding job and the submittal of this one.

The last two fields work as follows. Suppose we know that `a.out`, job number 123, should start ten seconds after the termination of `gcc x.c`, which is job number 120. We could give job number 123 a submittal time that is 10 seconds after the submittal time plus run time of job 120, but this wouldn't be right — changing the scheduler might change the wait time of job 120 and spoil the connection. The solution is to use fields 17 and 18 to save such relationships between jobs explicitly. In this example, for job number 123 we'll put 120 in its preceding job number field, and 10 in its think time from preceding job field.

Header Comments The first lines of the log may be of the format `;Label: Value1, Value2, ...`. These are special header comments with a fixed format, used to define global aspects of the workload. Predefined labels are:

`Computer` : Brand and model of computer
`Installation` : Location of installation and machine name
`Acknowledge` : Name of person(s) to acknowledge for creating/collecting the workload.
`Information` : Web site or email that contain more information about the workload or installation.
`Conversion` : Name and email of whoever converted the log to the standard format.
`Version` : Version number of the standard format the file uses. The format described here is version 2.
`StartTime` : In human readable form, in this standard format: Tuesday, 1 Dec 1998, 22:00:00

`EndTime` : In the same format as StartTime.

`MaxNodes` : Integer, number of nodes in the computer (describe the sizes of partitions in parentheses).

`MaxRuntime` : Integer, in seconds. This is the maximum that the system allowed, and may be larger than any specific job's runtime in the workload.

`MaxMemory` : Integer, in kilobytes. Again, this is the maximum the system allowed.

`AllowOveruse` : Boolean. 'Yes' if a job may use more than it requested for any resource, 'No' if it can't.

`Queues` : A verbal description of the system's queues. Should explain the queue number field (if it has known values). As a minimum it should be explained how to tell between a batch and interactive job.

`Partitions` : A verbal description of the system's partitions, to explain the partition number field. For example, partitions can be distinct parallel machines in a cluster, or sets of nodes with different attributes (memory configuration, number of CPUs, special attached devices), especially if this is known to the scheduler.

`Note` : There may be several notes, describing special features of the log. For example, "The runtime is until the last node was freed; jobs may have freed some of their nodes earlier".

3 Workload Benchmarks for Metacomputing

Most of the resources of a conventional parallel computer are used by batch jobs. Therefore, job schedulers are typically not required to provide compute resources at a specific time. However, this has changed with the appearance of metacomputers. Many metasystems are based on the concept of a single virtual machine which can also be used to run large parallel jobs. But this requires the availability of compute resources on different machines at the same time. In addition network resources may be needed as well. This can only be achieved if the schedulers that control the participating parallel machines accept reservations. Unfortunately, it is not clear how to include resource reservation into present scheduling algorithms. A simple approach may be an extension of backfilling. In the workshop some participants reported promising results with this concept. However, this assumes that the best time instant for such a resource reservation is already known. In any case, the widespread use of a parallel computer as part of a metasystem will certainly affect the workload and may therefore require new benchmarks.

3.1 Scheduling in a Metacomputing Environment

In the metacomputing scenario, there are many schedulers simultaneously acting over the system. Some of these schedulers control the resources they schedule over and thus constitute the access point to such resources (i.e., one has to submit a request to the scheduler in order to use the resources it controls). On the

other hand, there are schedulers that do not actually control the resources they use. Instead they communicate with multiple lower-level schedulers and decide which of them should be used, and which part of the parallel computation each of them should carry out. Requests to the appropriate low-level schedulers are then created and submitted on behalf of the user.

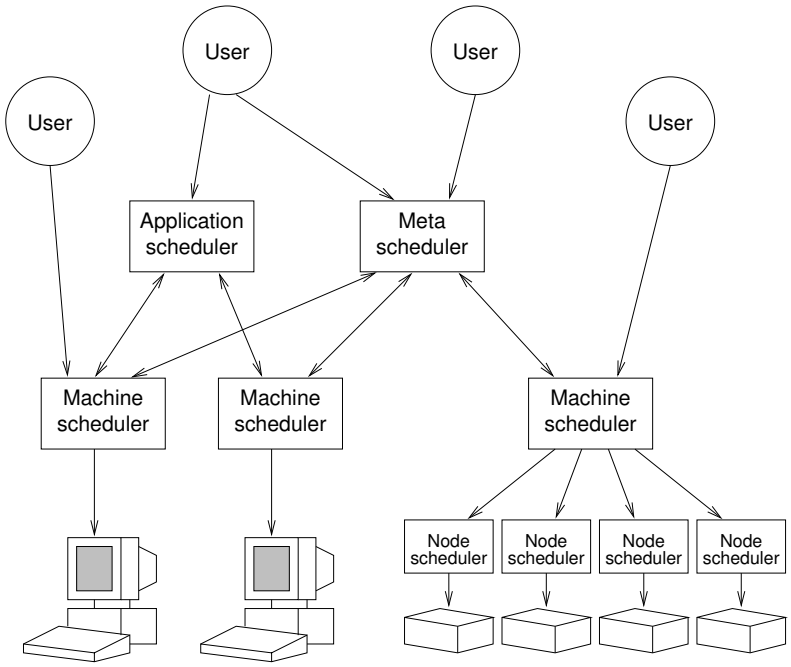

Fig. 1. *Entities involved in scheduling in a metacomputing environment.*

In order to keep the discussion focused, we suggest the following terminology and definitions (which are summarized graphically in Figure 1). We call the scheduler that controls a certain machine a *machine scheduler*. this is typically the OS scheduler on this machine, especially on desktop machines. On a parallel supercomputer, this may be the parallel operating environment scheduler running on the front end, or a batch queueing system such as NQS or PBS used to access the machine. Parallel machines may also have *node schedulers*, which control individual nodes, usually according to the directions of the machine scheduler (e.g. to implement gang scheduling). These are internal to the parallel machine implementation and therefore not relevant in a discussion of external workloads. Finally, there are *meta-schedulers* that interact with several machine schedulers in order to find usable resources and use them to schedule metacomputing applications. A special case of meta schedulers are *application schedulers*, that are developed in conjunction with a specific application, and use

application-specific knowledge to optimize its execution.

In order to decide which machine schedulers to use (and what each of them should do), the meta-scheduler needs to know how long a given request will take to be processed on a given machine scheduler, under the current system load. That is, in order to make reasonable decisions, the meta-scheduler needs information on how the machines schedulers are going to deal with its requests. Although some have proposed mechanisms to promote effective communication among the different schedulers in the system [11,8], the machine schedulers currently in use have not been designed with this need in mind. Therefore, researchers in metacomputing have developed tools that monitor and forecast how long a request is going to take to run over a particular set of resources (e.g., [61]).

Today there is no such tool for space-sliced parallel supercomputers. Since jobs run on a dedicated set of nodes in these machines, the information meta-schedulers can expect to obtain regards the queue waiting time. In principle, work on supercomputer queue time prediction [15,57,32] could be used to provide this information. However, the results obtained for queue time predictions are still relatively inaccurate, making them inadequate for many metacomputing applications, notably those that perform co-allocation (i.e., that spread across multiple machine schedulers). This has prompted the metacomputing community to ask for the enhancement of supercomputer schedulers by the introduction of reservations [29] or guaranteed computing power [30,52]. Reservations consist of a guarantee that a certain amount of resources is going to be available continuously starting at a pre-determined future time. Computing power guarantees consist of guarantees that a certain amount of computing power will be available over time, e.g. 25% of the time on 16 processors. However, there is still the question of how the meta-scheduler decides what is the right reservation to ask for. The very first efforts towards answering this question are now under way [10].

3.2 Components of a Benchmark Suite

One of the challenges in building a benchmark suite is determining the application space to be covered, and assembling a set of applications which cover the space (the analog of a basis set in linear algebra). The obstacle to doing this is that we lack two fundamental pieces of information: what a real metasystem workload looks like, and what the appropriate axes of the application space should be. While we have experience running one or two applications simultaneously, we do not have experience running truly large-scale systems (thousands to millions of nodes with hundreds to thousands of simultaneous users). We are therefore required to take an evolutionary approach. We will build a benchmark suite based on the "tools at hand", and will refine it over time as we learn more about metasystem computation.

A good first step will be to use accepted practice and generate micro-benchmarks: individual programs which stress one particular aspect of the system. For example, we can create a compute-intensive meta-application that can use all the cycles from all the machines it can get, a communication-intensive meta application that requires extensive data transfers between its parts, or a meta-application

that requires a specific set of devices from different locations. To test metacomputing schedulers, we can generate workloads consisting of large numbers of applications of a single type, and also mixed-mode workloads composed of diverse meta-applications.

As a second step, we can add real-world applications which we already run on metasystems. These applications will be components of an overall metasystem workload, and can help us to understand the interactions of complex applications in a metasystem environment. Using this benchmark suite, we can attempt to determine how well particular schedulers work, both alone and in competition.

3.3 Logging Scheduling Events in a Metacomputer

The two traditional methods of analyzing the performance of scheduling algorithms are to simulate synthetic workloads or simulate trace data recorded from parallel computers. Even though synthetic workloads do not explicitly require trace data, a synthetic workload that is useful must approximate actual workloads and therefore the characteristics of actual workloads must be known.

It is very difficult to collect data to form a workload of the events that occur in a metasystem. The problems are the distributed ownership of the constituents of the metasystem, the many points of access to it, and its sheer size. First, the metasystem consists of a diverse set of resources owned by dozens of organizations. These organizations are fully autonomous and cannot be forced to record the events on their local resources and provide them for a metasystem workload. Also, collecting events in a large distributed system is not a trivial task. Clock synchronization and causal order techniques can help, but the size and geographic dispersion of the metasystem makes it a hard problem. Second, each user may have their own application scheduler and thus there may be a large number of different application schedulers. We cannot force these schedulers to record events or to provide these events for a metasystem workload. Third, even if we could record all of these events and form them into a workload, the system would probably be too large to simulate conveniently.

There are some steps we can take toward recording a metasystem workload. First, events can be recorded on a subset of the metasystem. Small sets of sites tend to be closely aligned with each other and willing to share data with each other. One problem with this technique is that the resources used by users may not lie entirely within or without the subset we are recording. If programs use resources from across a sub-system boundary, important application information will not be recorded. Second, machine scheduling systems typically already have recording mechanisms to record events. Third, the current metacomputing software [27,33] each provide a common interface to machine schedulers and events can be recorded in this interface. Such trace data may provide enough data to extract information on which requests are co-allocation requests and are part of the same application. Note, however, that recording metacomputing applications alone would miss applications submitted directly to the local scheduler.

3.4 Evaluating Matacomputing Scheduling

Another problem we have not discussed is how do we evaluate the performance of schedulers in metacomputing environments? First we need to recognize that there will be many meta schedulers with different goals. Some schedulers will try to run applications on single parallel computers as soon as possible, some will try to co-allocate resources, others will try to run many serial applications, and others will try to have their applications complete as soon as possible by adapting to resource availability. The metrics used will vary for each meta scheduler and will include metrics such as wait-time, throughput, and turn-around time.

Even though we cannot record a complete metasystem workload, we can use synthetic data to evaluate scheduling algorithms. We have the advantage that we may be able to construct a synthetic workload by expanding on trace data from part of the metasystem and we can at least use the currently available trace data from parallel computers to form synthetic trace data for machine scheduling systems. In essence, this means that sampling is used to solve size problem, as has also been done with address traces [40]. More research is required to establish the methodological basis and limitations of this approach.

4 Convergence

4.1 A Comparison

Scheduling for parallel systems has been studied for a long time, and many schemes have been proposed and evaluated [20]. Scheduling in metasystems is relatively new, and the evaluation methodology still needs to be developed. A relevant question is therefore the degree to which ideas and techniques developed for parallel systems can be carried over to metacomputing systems.

The main difference that is usually mentioned in comparisons of parallel systems and metacomputing is that metacomputing deals with heterogeneity, whereas parallel systems are homogeneous [5]. This is in fact not so. Heterogeneity comes in three flavors: *architectural* heterogeneity, where nodes have a different architecture, *configuration* heterogeneity, where nodes are configured with different amounts of resources (e.g. different amounts of memory, or different processors from the same family), and *load* heterogeneity, which means that the available resources are different due to current load conditions. While parallel systems usually do not contain architectural heterogeneity, they certainly do encounter configuration and load heterogeneities. Therefore their schedulers need to deal with nodes that have different amount of resources available, just as in metacomputing. They need to make decisions based on estimates of when resources will become available, just as in metacomputing. They need to employ models of application behavior to estimate how sensitive the application is to heterogeneity, just as in metacomputing. They need to deal with requests for specific resources (such as extra memory, a certain device, or use of a specific license), just as in metacomputing.

The difference between parallel systems and metacomputing is therefore not a clear cut absence of certain problems, but their degree of severity. Some of the above issues could be ignored by parallel schedulers, at the cost of some inefficiency. This has been a common practice, and is one of the reasons for the limited utilization observed on many parallel systems. At the present time, these issues are beginning to be addressed. This is happening concurrently with the emergence of metacomputing, where these issues cannot be ignored, and have to be handled from the outset.

4.2 Integration of Parallel Systems and Metacomputing

In a metasystem environment, there is interaction between scheduling at the local level and scheduling at the meta level. An obvious example is that meta schedulers send applications to local schedulers. Another example is that the local schedulers can dictate what resources are available to meta applications by limiting the number of nodes made available to meta applications or by the scheduling policy used when scheduling meta applications versus locally submitted applications. A third example is that meta applications my ask for simultaneous access to resources from several local schedulers. This requires local mechanisms such as reservation of resources and these reservations affect the performance of local scheduling algorithms.

One major question is how much interaction is there and can we evaluate local and meta schedulers independently or using a simple model of the other type of scheduler? For example, mechanisms for combining queuing scheduling with reservation in a local scheduler can be evaluated using a synthetic workload of reservation requests or a recording of reservation requests. This requires little to no knowledge of meta-scheduling algorithms.

Another example is that meta schedulers can be evaluated using simple models of local schedulers if we assume that meta schedulers will not interfere with each other. A simple model of a local scheduler would just model the wait time of applications submitted to it, the error of wait time predictions, when reservations can be made, etc. We can assume meta schedulers will not interfere with each other if there are relatively few metasystem users when compared to the number of resources available. If meta schedulers can interfere with each other, we will have to simulate other meta schedulers using recorded or synthetic data.

We must take care when designing our metrics. In the past, supercomputer centers have focused on low-level, system-centric metrics such as percent utilization. Metaschedulers, on the other hand, are more focused on high-level, user-centric metrics such as turnaround time and cost. We believe that these apparently contradictory metrics can be unified through a proper economic model. Utilization metrics are frequently used to justify the past or future purchase of a machine ("Look, the machine is busy, it must've been worth the money we spent!" or "The machine is swamped! We need to buy a new one!"), but in the end, all they really tell us is that the machine is busy, not how much effective work is being done. With an economic model, the suppliers (supercomputer centers, et al.) can control utilization by altering the cost charged per unit time.

Users can employ personal schedulers to optimize their important criteria. In the end, this step has to be taken if metasystems are to become a reality, so we should make it work for us.

4.3 An Evaluation Environment

As noted earlier, it will be nearly impossible to run real benchmark suites across large-scale metasystems. Therefore, we opt for simulation to evaluate schedulers. A proposed evaluation environment for schedulers is the WARMstones project (WARM = Wide-Area Resource Management, and stones is from the traditional naming of "stones" for benchmark suites). This is somewhat of a misnomer, as WARMstones will encompass a simulation and evaluation environment in addition to a benchmark suite, and part of the WARMstones environment will simulate and evaluate scheduling for local systems.

The primary components of WARMstones include a benchmark suite, an implementation toolkit for schedulers, a canonical representation of metasystems, and a simulation engine to evaluate execution of a suite of applications on a metasystem using a particular scheduler. As we have already described, the benchmark suite will initially comprise combinations of micro-benchmarks and existing applications. Rather than executing these applications directly, we will represent them using annotated graphs, and simulate the execution by interpreting the graphs. Legion program graphs [34] are well-suited to this purpose. Users will also be able to produce representations of their own applications.

The implementation toolkit will allow users to implement particular scheduling algorithms for simulation and evaluation. Again, we draw on earlier experience, and plan to use a system much like that in the MESSIAHS distributed scheduling system [9].

To evaluate a scheduler, we will first run the scheduler on the benchmark suite to produce mappings of programs (graphs) to resources, and then run the simulator using the resultant mapping and a system configuration (in canonical form) as input. The representation will encapsulate both the local infrastructure (workstations, clusters, supercomputers) and the overall structure of the metasystem. The system will also employ multiple levels of detail in the simulation. For example, depending on how much precision is required and how much time and computational resources are available, we could simulate every packet being transmitted across a network, or we can simply assume a simple model and estimate the communication time.

This evaluation system will enable evaluations of multiple scenarios and factors, e.g.:

- I have devised a new scheduling algorithm. I want to evaluate it using the benchmark suite and a range of "standard" machine representations, so that I can make "apples-to-apples" comparisons to other schedulers.
- I have an application I want to run, and I know the target system environment. I can use the evaluation system to help me select among several candidate scheduling algorithms.

- I want to enable run-time selection of "good" scheduling algorithms. I can make off-line runs iterating across the benchmark suite, the set of available schedulers, and a number of "standard" system configurations. I can store these results in a table, and at run time I can look up the closest matches on application structure and system configuration to find a scheduler which should work well for me.
- I have the choice of purchasing machine A or machine B for my system. I can generate program graphs for my top five applications and test them using an implementation of my current scheduler on system configurations including both machine choices.

5 Conclusions

Standardization and benchmarking are important for progress because without them research is harder to perform and results are harder to compare. While there is always place for improvements and additions, it is also necessary to draw the line and decide to standardize now. It seems that we can immediately do so for parallel systems, as enough data is available, at the same time leaving the door open for changes as more data becomes available in the future. The definitive definition and updates will be posted as part of the Parallel Workloads Archive [19].

Benchmarking for meta-scheduling is harder, because even less data is available, and the environment is more complex. It therefore seems that the best current course of action is to try and reduce the complexity by partitioning the problem into sub-problems, and trying to deal with each one individually. Thus application schedulers will be evaluated using simplified models of resource availability provided by separate machine schedulers, and machine schedulers will be evaluated using rudimentary models of the requests generated by application schedulers. As larger implementation materialize and data is accumulated, integrated evaluations may become possible.

Acknowledgements

Cirne is supported in part by CAPES (Grant DBE2428/95-4), NSF National Challenge Nile Project (Grant PHY-9318151), DoD Modernization Program (Contract #9720733-00), and NSF (Grant ASC-9701333). Feitelson is supported by the Ministry of Science, Israel. Leutenegger is supported by the NSF grant ACI-9733658.

References

1. A. K. Agrawala, J. M. Mohr, and R. M. Bryant, "*An approach to the workload characterization problem*". *Computer* **9(6)**, pp. 18–32, Jun 1976.

2. G. Alverson, S. Kahan, R. Korry, C. McCann, and B. Smith, "*Scheduling on the Tera MTA*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 19–44, Springer-Verlag, 1995. Lect. Notes Comput. Sci. vol. 949.
3. P. Barford and M. Crovella, "*Generating representative web workloads for network and server performance evaluation*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 151–160, Jun 1998.
4. A. Batat. Master's thesis, Hebrew University, 1999. (in preparation).
5. F. Berman, "*High-performance schedulers*". In *The Grid: Blueprint for a New Computing Infrastructure*, I. Foster and C. Kesselman (eds.), pp. 279–309, Morgan Kaufmann, 1999.
6. F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao, "*Application-level scheduling on distributed heterogeneous networks*". In *Supercomputing '96*, 1996.
7. M. Calzarossa, G. Haring, G. Kotsis, A. Merlo, and D. Tessera, "*A hierarchical approach to workload characterization for parallel systems*". In *High-Performance Computing and Networking*, pp. 102–109, Springer-Verlag, May 1995. Lect. Notes Comput. Sci. vol. 919.
8. S. J. Chapin, "*Distributed scheduling support in the presence of autonomy*". In *Proc. 4th Heterogeneous Computing Workshop*, pp. 22–29, Apr 1995. Santa Barbara, CA.
9. S. J. Chapin and E. H. Spafford, "*Support for implementing scheduling algorithms using MESSIAHS*". *Scientific Programming* **3(4)**, pp. 325–340, Winter 1994.
10. W. Cirne and F. Berman, "*S3: a metacomputing-friendly parallel scheduler*". Manuscript, UCSD, In preparation.
11. W. Cirne and K. Marzullo, "*The computational co-op: gathering clusters into a metacomputer*". In *Second Merged Symposium IPPS/SPDP 1999, 13th International Parallel Processing Symposium & 10th Symposium on Parallel and Distributed Processing*, pp. 160–166, April 1999.
12. D. E. Culler, J. P. Singh, and A. Gupta, *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann, 1999.
13. A. B. Downey, "*A parallel workload model and its implications for processor allocation*". In 6th *Intl. Symp. High Performance Distributed Comput.*, Aug 1997.
14. A. B. Downey, "*Predicting queue times on space-sharing parallel computers*". In 11th *Intl. Parallel Processing Symp.*, pp. 209–218, Apr 1997.
15. A. B. Downey, "*Using queue time predictions for processor allocation*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 35–57, Springer Verlag, 1997. Lect. Notes Comput. Sci. vol. 1291.
16. A. B. Downey and D. G. Feitelson, "*The elusive goal of workload characterization*". *Perf. Eval. Rev.* **26(4)**, pp. 14–29, Mar 1999.
17. 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.
18. D. G. Feitelson, "*Packing schemes for gang scheduling*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 89–110, Springer-Verlag, 1996. Lect. Notes Comput. Sci. vol. 1162.
19. D. G. Feitelson, "*Parallel workloads archive*". URL http://www.cs.huji.ac.il/labs/parallel/workload/.
20. D. G. Feitelson, *A Survey of Scheduling in Multiprogrammed Parallel Systems*. Research Report RC 19790 (87657), IBM T. J. Watson Research Center, Oct 1994.

21. 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.
22. D. G. Feitelson and L. Rudolph, "*Distributed hierarchical control for parallel processing*". *Computer* **23(5)**, pp. 65–77, May 1990.
23. D. G. Feitelson and L. Rudolph, "*Gang scheduling performance benefits for fine-grain synchronization*". *J. Parallel & Distributed Comput.* **16(4)**, pp. 306–318, Dec 1992.
24. 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.
25. D. G. Feitelson and L. Rudolph, "*Toward convergence in job schedulers for parallel supercomputers*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 1–26, Springer-Verlag, 1996. Lect. Notes Comput. Sci. vol. 1162.
26. D. Ferrari, "*Workload characterization and selection in computer performance measurement*". *Computer* **5(4)**, pp. 18–24, Jul/Aug 1972.
27. I. Foster and C. Kesselman, "*Globus: a metacomputing infrastructure toolkit*". *International Journal of Supercomputing Applications* **11(2)**, pp. 115–128, 1997.
28. I. Foster and C. Kesselman (eds.), *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.
29. I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy, "*A distributed resource management architecture that supports advance reservations and co-allocation*". In *International Workshop on Quality of Service*, 1999.
30. H. Franke, P. Pattnaik, and L. Rudolph, "*Gang scheduling for highly efficient distributed multiprocessor systems*". In 6th *Symp. Frontiers Massively Parallel Comput.*, pp. 1–9, Oct 1996.
31. G. Ghare and S. T. Leutenegger, "*The effect of correlating quantum allocation and job size for gang scheduling*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), Springer Verlag, 1999. Lect. Notes Comput. Sci. vol. 1659.
32. R. Gibbons, "*A historical application profiler for use by parallel schedulers*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 58–77, Springer Verlag, 1997. Lect. Notes Comput. Sci. vol. 1291.
33. A. S. Grimshaw, J. B. Weissman, E. A. West, and E. C. Loyot, Jr., "*Metasystems: an approach combining parallel processing and heterogeneous distributed computing systems*". *J. Parallel & Distributed Comput.* **21(3)**, pp. 257–270, Jun 1994.
34. A. S. Grimshaw, W. A. Wulf, and the Legion team, "*The Legion vision of a worldwide virtual computer*". *Comm. ACM* **40(1)**, pp. 39–45, Jan 1997.
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 Conf. Measurement & Modeling of Comput. Syst.*, pp. 120–132, May 1991.
36. M. A. Holliday and C. S. Ellis, "*Accuracy of memory reference traces of parallel computations in trace-driven simulation*". *IEEE Trans. Parallel & Distributed Syst.* **3(1)**, pp. 97–109, Jan 1992.

37. 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.
38. R. Jain, *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, 1991.
39. 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.
40. R. E. Kessler, M. D. Hill, and D. A. Wood, "*A comparison of trace-sampling techniques for multi-megabyte caches*". *IEEE Trans. Comput.* **43(6)**, pp. 664–675, Jun 1994.
41. E. J. Koldinger, S. J. Eggers, and H. M. Levy, "*On the validity of trace-driven simulation for multiprocessors*". In 18th *Ann. Intl. Symp. Computer Architecture Conf. Proc.*, pp. 244–253, May 1991.
42. J. Krallmann, U. Schwiegelshohn, and R. Yahyapour, "*On the design and evaluation of job scheduling systems*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), Springer-Verlag, 1999. Lect. Notes Comput. Sci. vol. 1659.
43. M. Krunz and S. K. Tripathi, "*On the characterization of VBR MPEG streams*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 192–202, Jun 1997.
44. W. Lee, M. Frank, V. Lee, K. Mackenzie, and L. Rudolph, "*Implications of I/O for gang scheduled workloads*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 215–237, Springer Verlag, 1997. Lect. Notes Comput. Sci. vol. 1291.
45. W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "*On the self-similar nature of Ethernet traffic*". *IEEE/ACM Trans. Networking* **2(1)**, pp. 1–15, Feb 1994.
46. M. J. Litzkow, M. Livny, and M. W. Mutka, "*Condor - a hunter of idle workstations*". In 8th *Intl. Conf. Distributed Comput. Syst.*, pp. 104–111, Jun 1988.
47. U. Lublin, *A Workload Model for Parallel Computer Systems*. Master's thesis, Hebrew University, 1999. (In Hebrew).
48. 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.
49. J. K. Ousterhout, H. Da Costa, D. Harrison, J. A. Kunze, M. Kupfer, and J. G. Thompson, "*A trace-driven analysis of the UNIX 4.2 BSD file system*". In 10th *Symp. Operating Systems Principles*, pp. 15–24, Dec 1985.
50. E. W. Parsons and K. C. Sevcik, "*Coordinated allocation of memory and processors in multiprocessors*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 57–67, May 1996.
51. V. G. J. Peris, M. S. Squillante, and V. K. Naik, "*Analysis of the impact of memory in distributed parallel processing systems*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 5–18, May 1994.
52. A. Polze, M. Werner, and G. Fohler, "*Predictable network computing*". In 17th *Intl. Conf. Distributed Comput. Syst.*, pp. 423–431, May 1997.
53. E. Rosti, G. Serazzi, E. Smirni, and M. S. Squillante, "*The impact of I/O on program behavior and parallel scheduling*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 56–65, Jun 1998.

54. U. Schwiegelshohn and R. Yahyapour, "*Resource allocation and scheduling in metasystems*". In *Proc. Distributed Computing & Metacomputing Workshop at HPCN Europe*, P. Sloot, M. Bibak, A. Hoekstra, and B. Hertzberger (eds.), pp. 851–860, Springer-Verlag, Apr 1999. Lect. Notes in Comput. Sci. vol. 1593.
55. K. C. Sevcik, "*Application scheduling and processor allocation in multiprogrammed parallel processing systems*". *Performance Evaluation* **19(2-3)**, pp. 107–140, Mar 1994.
56. R. L. Sites and A. Agarwal, "*Multiprocessor cache analysis using ATUM*". In 15th *Ann. Intl. Symp. Computer Architecture Conf. Proc.*, pp. 186–195, 1988.
57. W. Smith, V. Taylor, and I. Foster, "*Using run-time predictions to estimate queue wait times and improve scheduler performance*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), Springer Verlag, 1999. Lect. Notes Comput. Sci. vol. 1659.
58. D. Talby, D. G. Feitelson, and A. Raveh, "*Comparing logs and models of parallel workloads using the co-plot method*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), Springer Verlag, 1999. Lect. Notes Comput. Sci.
59. D. Thiébaut, J. L. Wolf, and H. S. Stone, "*Synthetic traces for trace-driven simulation of cache memories*". *IEEE Trans. Comput.* **41(4)**, pp. 388–410, Apr 1992. (Corrected in *IEEE Trans. Comput.* **42(5)** p. 635, May 1993).
60. 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.
61. R. Wolski, N. T. Spring, and J. Hayes, "*The network weather service: a distributed resource performance forecasting service for metacomputing*". *Journal of Future Generation Computing Systems*, 1999.
62. 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.