User Modeling of Parallel Workloads

Thesis submitted for the degree of

"Doctor of Philosophy"

by

David Talby

Submitted to the Senate of the Hebrew University

December 2006

User Modeling of Parallel Workloads

Thesis submitted for the degree of

"Doctor of Philosophy"

by

David Talby

Submitted to the Senate of the Hebrew University

December 2006

This work was carried out under the supervision of

Dr. Dror G. Feitelson

Prof. Adi Raveh

A B S T R A C T

The goal of workload modeling is to simulate the expected workload, accurately enough to enable making correct design and administrative decisions. Several statistical features of production parallel computer workloads, which are not embodied in current models, have been identified. Their practical importance is demonstrated by two new kinds of schedulers – a key component in determining the overall performance of a parallel computer. The first is adaptive scheduling, which takes advantages of the locality of sampling and known cycles in parallel workloads, and achieves an average improvement of 10% in performance and 35% in stability for the tested production workloads. The second is shortest-job-backfill-first scheduling, which relies on runtime prediction, done by analyzing user and session histories.

These schedulers cannot be correctly evaluated by existing workload models, and we argue that the correct approach for future workload models (as well as on-line algorithms) is user and session-based modeling, instead of modeling jobs directly as done today. As the basis for such a model, we use PCA to provide variable sets which explain over 80% of the variance between users and sessions, and clustering to identify five stable session clusters and four stable user clusters. We then model the distributions of the arrival and activity patterns of both users and sessions, including a complete analysis of their dependencies and temporal structure. The model is based on logs from seven different parallel supercomputers, spanning over 87 months, analyzed together to ensure that results are location and architecture-neutral.

C O N T E N T S

INTRODUCTION

1. Overview

Understanding the expected workload that a system will face is crucial to making the right decisions when designing and configuring it. Workload analysis for parallel computers has therefore attracted a large body of research, divided to two main flavors. The first is the construction of workload models (Downey, 1997; Jann et al., 1997; Feitelson and Jette, 1997; Cirne and Berman, 2001; Lublin and Feitelson, 2003), which are statistical models based on observations from real-world traces. These models can be used to create synthetic workloads, in order to compare resource management algorithms under different conditions (load or machine size, for example) or to gain general insights.

The second flavor exploits features of parallel workloads directly, either by designing heuristic algorithms that exploit discovered workload features (Schroeder and Harchol-Batler, 2000; Talby and Feitelson, 2005) or by designing adaptive or prediction-based algorithms that learn the workload as they go (Dinda et al., 1999; Yoo and Jette, 2001; Dinda, 2002; Vazhkudai et al., 2002). Adaptive, learning and prediction-based algorithms always include a lot of prior knowledge about the workload – which parameters should be adaptive? What cues are used to change them? Which variables should be used to learn from history? Many times, their main innovation is discovering a particular workload phenomenon and modeling it well.

In many areas, the basic resource management policies are well-known and understood – and the major performance advances in the past few years are the result of tuning the algorithms to exploit workload features found in many historic traces. Examples can be found in scheduling (Feitelson and Jette, 1997; Tsafrir et al., 2003; Talby and Feitelson, 2005), task allocation (Jarvis et al., 1997; Schroeder and Harchol-Batler, 2000), management of a computational grid (Jarvis et al., 1997), load balancing (Yu et al, 1997), soft real-time systems (Dinda et al., 1999), wide-area data replication (Vazhkudai et al., 2002) and others. All of these areas can potentially benefit from this work.

The goal of this research is to provide new information, discovered by means of sound statistical techniques, which can benefit both worlds – synthetic workload modeling, and on-line resource management algorithms. Although this study began with the identification of several statistical properties found in production workloads and missing from current workload models, the first stage was not to create a new model which incorporates these features, but rather to design new on-line algorithms which take advantage of them. These algorithms prove the practical importance of these workload features, thus motivating the modeling effort.

The investigated algorithmic problem is parallel scheduling, for two reasons. First, parallel schedulers are very sensitive to their given workload, and particularly to its temporal structure (Downey and Feitelson, 1999; Feitelson, 2003; Talby and Feitelson, 2005). Second, the scheduler is a key component in a parallel computer, and has a dramatic effect on its overall performance. It is a software-only, relatively independent module of a parallel computer's operating system, making its improvement a relatively low-cost and highly practical opportunity.

The second stage of this research is the construction of a user based parallel workload model. The model is motivated by both the new scheduling algorithm and a set of methodological problems regarding the current usage of workloads, which we have identified. We argue that a solution to these problems can be attained by building a new layered model: a user model, followed by a session model per user type, followed by a job model per session type. Such an approach has an extra benefit of providing new insight into the behavior of users of parallel machines. This is also the first parallel workload model to investigate several fundamental modeling questions, such as which variables should be modeled, and what the model's parameters should be.

This thesis is constructed as follows. The two following sections provide basic background; additional context-specific background is embedded in later sections to enhance readability. The methodology chapter begins by providing a high-level view of the research plan, followed by three sections describing key applied techniques. The results chapter starts with two sections describing new scheduling algorithms which exploit some of these features, and continues with three sections describing the construction of the user-based workload model. The discussion and conclusions chapter summarizes and proposes future research directions.

2. Parallel Schedulers

The parallel computers considered in this work are of the most widespread type today, which uses variable partitioning: A new job requires a certain number of processors upon its arrival, and these processors are dedicated to it once it starts running. In addition, each job provides an estimate for its runtime; this is an upper bound, since if a job exceeds it, it is killed. Users also have an incentive to provide low estimates, since this enables promoting jobs from the back of the queue to fill idle processors – an optimization known as backfilling.

The first backfilling scheduler was the EASY Scheduler (Skovira et al., 1996). It was used mainly in IBM SP2 machines since the mid ‘90s, but a recent survey (Etsion and Tsafrir, 2005) has found that its policies are still the default in the most popular schedulers deployed today. After starting all the jobs that can be in FCFS order, EASY makes a reservation for the first job left in the queue. The time at which the first job in the queue is going to run is called the shadow time; the idle

nodes after the first queued job starts running are called extra nodes. Subsequent jobs are backfilled (pass the first job in the queue and starts running immediately) if one of the following two conditions holds:

1. They require no more than the currently free nodes, and will terminate by the shadow time.
2. They require no more than the minimum of the currently free nodes and the extra nodes.

The algorithm's formal pseudo-code is given in Appendix A. EASY uses an aggressive backfilling strategy, in the sense that the above two conditions are only checked for the *first* (oldest) job in the queue – all other jobs can suffer unbounded delays. For example, consider the following scenario, in which job J1 is running, job J2 is waiting, and jobs J3, J4 and J5 arrive in this order. Job J3 cannot be backfilled, since this will delay the oldest waiting job (J2); however, J4 will be backfilled because it will terminate before the shadow time, and J5 will be backfilled since it requires less than the extra nodes. Job J5 would be backfilled even if it is very long and if J3 requires all processors – as long as J3 isn't the oldest waiting job, it can suffer an unbounded delay.

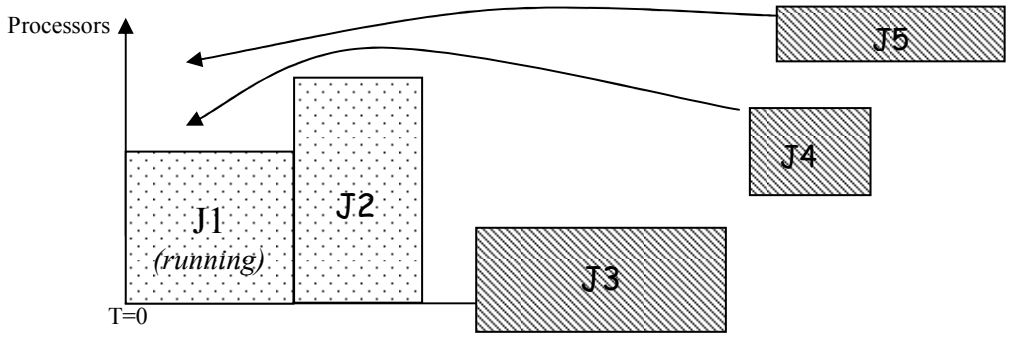


Figure 1. EASY Backfilling Example

As the example demonstrates, backfilling reduces fragmentation, and indeed, studies have shown that it improves utilization by about 15% (Jones and Nitzberg, 1999). The example also shows the current standard of fairness in parallel schedulers, which is composed of two rules:

- Attempt FCFS scheduling before backfilling.
- Backfilled jobs must not delay the first job in the queue. This prevents starvation, but also means that a job can be delayed for an unbounded amount of time (Mu'alem and Feitelson, 2001).

Other backfilling schedulers define different fairness criteria. For example, the Conservative Scheduler (Mu'alem and Feitelson, 2001) uses a different rule: A job can be backfilled only if does not delay *any* previous job in the queue. This enables runtime guarantees and decreases starvation on one hand, but may hamper utilization and responsiveness for small jobs on the other.

A third scheduler is Maui (Jackson et al, 2001). Maui is a high-end scheduling system, successfully deployed in many computing sites over the past few years, from IBM SP2 machines to Linux clusters (Bode et al., 2000; Jackson et al., 2001). The scheduler supports backfilling, and is highly configurable: In particular, the number of jobs that cannot be delayed and the order in which jobs are backfilled can be controlled. The default and recommended configuration (which is used in all simulations in this paper) is essentially EASY: Make reservations for the first job in the queue only, and backfill jobs by FCFS. When a new job arrives, it is not the only one that the scheduler tries to backfill. Instead, all waiting jobs are sorted in order of ascending arrival time, and are then backfilled in that order.

A fourth scheduler that we consider here is the Flex Scheduler (Talby and Feitelson, 1999). Flex takes a different approach from the above three algorithms, by trying to reach a global optimization of the entire queue, rather than just the head of the queue. This means that whenever a decision has to be made (job arrival, termination or cancellation events), all possible queues are compared, and the best one (according to a configurable criteria) is chosen. For example, if two jobs are queued and a third arrives, Flex will consider three alternative schedules: Running the new job first, in the middle, and last. Each possible schedule is graded, suffering a penalty for every job that must wait. To prevent starvation, Flex introduces the concept of slack: Each job is given a slack upon arrival, and it can never be delayed by *more than its slack*. This is safer than EASY and enables runtime guarantees, yet is more flexible than Conservative.

The above four scheduling algorithms have been studied and compared in depth (Talby and Feitelson, 1999; Mu'alem and Feitelson, 2001). The bottom-line reported results are as follows:

- Easy is generally better than Conservative under the response time metric, and sometimes also under bounded slowdown, mainly in high-load workloads.
- Flex and Maui offer a level of performance 10%-20% better than that of Easy, depending on the workload and metric.

Another scheduling strategy is to use Shortest-Job-First (SJF) (Gibbons, 1997) and neglect FCFS altogether; such a strategy is theoretically proven to improve response times, by favoring short jobs, and this was indeed verified by several empirical studies (Smith et al., 1999; Zotkin and Keleher, 1999; Chiang et al., 2002; Tsarir and Feitelson 2006b). However, this comes at the expense of fairness – resulting in starvation of long jobs – even if reservations of some kind are used (Chiang et al., 2002).

Recently, a new backfilling scheduler has been proposed which combines the fairness criteria of EASY with the improved performance of SJF, by maintaing the basic EASY policy but selecting

the shortest jobs to be backfilled first (Tsafrir et al., 2006). The scheduler is called Shortest-Job-Backfill-First (SJBF), and its pseudo-code is given in Appendix A. Determing the expected length of waiting and running jobs requires a runtime prediction scheme, and since a user's history provides the most accurate predictors, improving this scheduler's runtime prediction ability is a goal of this work.

3. Parallel Workload Modeling

A characterization of the workload a system will face is necessary in order to evaluate schedulers, processor allocators, and make many other design decisions. Two kinds of workloads are typically used: A trace of a real production workload, or a synthetic workload produced by a statistical model. For concreteness, we will consider traces and models of parallel supercomputer workloads in this paper.

Production logs have the advantage of being more realistic, as they are a direct recording of a workload that has occurred in practice. However, they may suffer from various problems. For example, the trace may contain errors or otherwise unreliable data: mysterious jobs that exceeded the system's limits, undocumented downtime, dedication of the system to certain users, and patterns of activity that are not generally representative (Koldinger et al., 1991; Windisch et al., 1996; Downey and Feitelson, 1999; Feitelson and Tsafrir, 2006). In addition, different workloads can be highly variable, and even the typical usage on the same machine can significantly change over time (Hotovy et al., 1996; Talby, Feitelson and Raveh, 1999). Such problems limit the degree to which we can draw conclusions from past workloads to predict future ones, or infer from one installation – one hardware configuration, user base, and scheduler – about other ones. It is therefore necessary to map invariants that are common to multiple workloads and can be relied upon to be representative.

The alternative to using production traces is to generate synthetic models (Ferrari, 1972; Calzarossa and Serazzi, 1993), and several such models have been proposed for parallel workloads. Such models are based on measurements of real workloads (Agrawala et al., 1976; Feitelson and Tsafrir, 2006). Models have the advantage over production logs of putting all the assumptions "on the table", and of being more flexible, by allowing their user to easily vary the model's parameters.

The currently available synthetic models are the following. The first model was proposed by (Feitelson, 1996). This model is based on observations from several workload logs. Its main features are the hand-tailored distribution of job sizes (i.e. the number of processors used by each job), which emphasizes small jobs and powers of two, a correlation between job size and running time, and the repetition of job executions. In principle such repetitions should reflect feedback from the

scheduler, as jobs are assumed to be re-submitted only after the previous execution terminates. Here we deal with a pure model, so we assume they run immediately and are resubmitted after their running time. The second model is a modification from '97 (Feitelson and Jette, 1997).

The model by Downey is based mainly on an analysis of the SDSC Paragon log (Downey, 1997 & 1997b). It uses a novel log-uniform distribution to model service times (that is, the total computation time across all nodes) and average parallelism. This is supposed to be used to derive the actual runtime based on the number of processors allocated by the scheduler. Again, as we are dealing with a pure model here, we instead use the average parallelism as the number of processors, and divide the service time by this number to derive the running time.

Jann's model is based on a careful analysis of the CTC SP2 workload (Jann et al., 1997). Both the running time and inter-arrival times are modeled using hyper Erlang distributions of common order. A separate distribution is used for different ranges of number of processors, with the parameters derived by matching the first three moments of the empirical distribution from the log.

The model by Lublin (Lublin and Feitelson, 2003) is based on a statistical analysis of four logs. It includes a model of the number of processors used which emphasizes powers of two, a model of running times that correlates with the number of processors used by each job, and a model of inter-arrival times. While superficially similar to the Feitelson models, Lublin based the choice of distributions and their parameters on better statistical procedures in order to achieve a better representation of the original data.

The last and most recent model is by (Cirne and Berman, 2001). This is a comprehensive model for generating moldable jobs, based on the analysis of four SP2 logs. It is composed of two parts: A model for generating a stream of rigid jobs; and a model for turning rigid jobs into moldable ones. The model also addresses the issues of requested times for a job, and the possibility of job cancellation. Here we test the basic features of the model – the generation of arrival time, runtime, and parallelism for each job. The model takes into account workday cycles, and its inter-arrivals pattern can be adjusted to match each of the logs used to build it.

The problem with models is that they need to be representative of real workloads. In (Talby, Feitelson and Raveh, 1999 and 2007) we compared eleven production workloads with the generated output of six statistical models, and tested how the models measure up to reality. All models were found to be reasonable in the sense that they span the same range of variable combinations as the real workloads. However, we also found numerous problems in current models, which make them wanting for tasks such as comparing schedulers. The next chapter explains these issues in depth.

METHODOLOGY

4. Methodological Issues in Workload usage

4.1. Parameterization

This chapter summarizes the evidence regarding statistical features which exist in production parallel workloads, but are missing from current workload models. The work is largely based on (Talby, Feitelson and Raveh, 1999 and 2007), and divided to two sections: This section focuses on methodological problems in the way current models are used, and the next focuses on the (mis-) representation of the temporal structure of workloads. These sections serve both to present the methodology of this research (i.e. some of the key analysis tools used), and to rationalize the research flow, which is presented in the final section.

The Co-Plot method was used to compare the available production workloads among themselves, and against the workload models. Co-Plot is a technique which analyzes observations and variables simultaneously, in contrast to classical multivariate analysis methods, such as cluster analysis and principal component analysis. This means that we are able to see, in the same analysis, clusters of observations (workloads in our case), clusters of variables, the relations between clusters (correlation between variables, for example), and a characterization of observations. Appendix B presents a more detailed overview of the Co-Plot algorithm.

Co-plot is especially suitable for tasks in which there are few observations and relatively many variables – as opposed to regression based techniques, in which the number of observations must be an order of magnitude larger than the number of variables. This is crucial in our case, in which there are few workloads (eleven production ones and six synthetic ones), and a similar number of variables. Co-plot's output is a visual display of its findings: It is based on two graphs that are superimposed on each other. The first graph maps the n observations into a two-dimensional space. This mapping, if it succeeds, conserves relative distance: observations that are close to each other in p dimensions are also close in two dimensions, and vice versa. The second graph consists of p arrows, representing the variables, and shows the direction of the gradient for each one.

Table 1 lists the production workloads in our dataset, which are freely available online from the Parallel Workloads Archive (Feitelson, 1999). Table 2 lists the values of the analysed variables. These variables include the median and 90% interval of job runtimes, actual and normalized (to a

128-nodes machine) number of processors, total used CPU time, and inter-arrival time between jobs, in addition to the number of jobs per day, number of distinct users and executables, and the machine's load (measured in two ways: CPU load measures actual CPU usage, and runtime load measure wall-clock runtime usage). The median and interval statistics were preferred since as shown in (Downey and Feitelson, 1999), the average and standard deviation of some of these fields are extremely unstable due to the very long tail of the involved distributions. Removing the top 0.1% jobs from a workload, for example, could change the average by 5% and the CV by 40%. These findings follow similar ones in (Lazowska, 1977).

	Log	**Location**	**# of Nodes**	**# of Jobs**	**Machine Type**	**Period**
1	NASA	NASA Ames	128	42,264	iPSC/860	Oct 1993 – Dec 1993
2	PAR5	San Diego Supercomputing Center	416	67,846	Paragon	Dec 1994 – Dec 1995
3	PAR6	San Diego Supercomputing Center	416	38,702	Paragon	Dec 1995 – Dec 1996
4	BLUE	San Diego Supercomputing Center	1152	250,440	Blue Horizon	Apr 2000 – Jan 2003
5	SDSP	San Diego Supercomputing Center	128	73,496	IBM SP/2	Apr 1998 – Apr 2000
6	CTC	Cornell Theory Center	512	79,302	IBM SP/2	Jun 1996 – May 1997
7	KTH	Swedish Institute of Technology	100	28,490	IBM SP/2	Sep 1996 – Aug 1997
8	LACM	Los Alamos National Lab	1024	201,384	CM-5	Oct 1994 – Sep 1996
9	O2K	Los Alamos National Lab	2048	121,989	Origin 2000	Nov 1999 – Apr 2000
10	LLNL	Lawrence Livermore National Lab	256	21,323	Cray T3D	Jun 1996 – Sep 1996
11	OSC	Ohio Supercomputing Center	57	80,713	Linux Cluster	Jan 2000 – Nov 2001

Table 1: Parallel Computers in our data set

Variable:	Sign:	CTC 1	KTH 2	LACM5 3	O2K 4	LLNL 5	NASA 6	BLUE 7	PAR5 8	PAR6 9	SDSP2 10	OSC 11
Processors in Machine	TN	512	100	1024	2048	256	128	1152	416	416	128	57
Jobs per Day	JD	233.76	83.83	279.32	885.65	183.82	459.59	255.41	209.66	104.45	99.84	119.20
Runtime Load	RL	0.556	0.690	0.735	0.640	0.616	0.467	0.762	0.628	0.611	0.829	0.431
CPU Load	CL	0.464	0.690	0.478	0.391	0.616	0.467	0.627	0.667	0.685	0.734	0.590
Users per KJobs	UJ	8.56	7.51	1.06	2.76	7.18	1.63	1.87	1.28	1.55	5.95	2.63
Executables per KJobs	EJ	155.29	N/A	19.74	N/A	32.88	11.67	N/A	N/A	N/A	896.50	0.01
Runtime Median	Rm	946	847	68	569	36	19	210	25	174	318	383
Runtime Interval	Ri	57226	47861	9063	32243	9143	1170	21738	26535	29924	41583	64089
Processors Median	Pm	2	3	64	15	8	1	8	4	8	2	1
Processors Interval	Pi	37	31	224	127	62	31	128	63	63	48	5
Norm. Procs. Median	Nm	0.5	3.8	8.0	0.9	4.0	1.0	0.9	1.2	2.5	2.0	2.2
Norm. Procs. Interval	Ni	9.3	39.7	28.0	7.9	31.0	31.0	14.2	19.4	19.4	48.0	11.2
CPU Work Median	Cm	559	847	6	21	36	19	30	24	129	264	646
CPU Work Interval	Ci	54794	47861	3658	9624	9143	1170	15990	25788	29640	41079	92838
Inter-Arrival Median	Am	79	192	59	26	119	59	102	73	124	135	64
Inter-Arrival Interval	Ai	1487	3810	1356	357	1660	446	1256	1786	3838	3823	2496

Table 2: Data of production workloads

Running the Co-Plot algorithm on a given data set is done in several iterations. At first, all observations and variables are used. Then outlier observations and low-correlation variables are removed, and runs on different combinations of the remaining data are done as well. Each run produces a different plot, which enables the analyst to see which observations are stable, and which tend to change. The conclusions presented in this paper are only ones that proved stable.

Figure 2 includes all the observations – none of them is an extreme outlier. The variables used are those that had the highest correlations, which facilitates the creation of a slightly more accurate 2-D map of the observations (the full set of variables will be analyzed in the next section). The coefficient of alienation of this Co-Plot is 0.10, and the average correlation of variables is 0.85. These are generally considered as excellent goodness-of-fit values (Borg and Groenen, 1997).

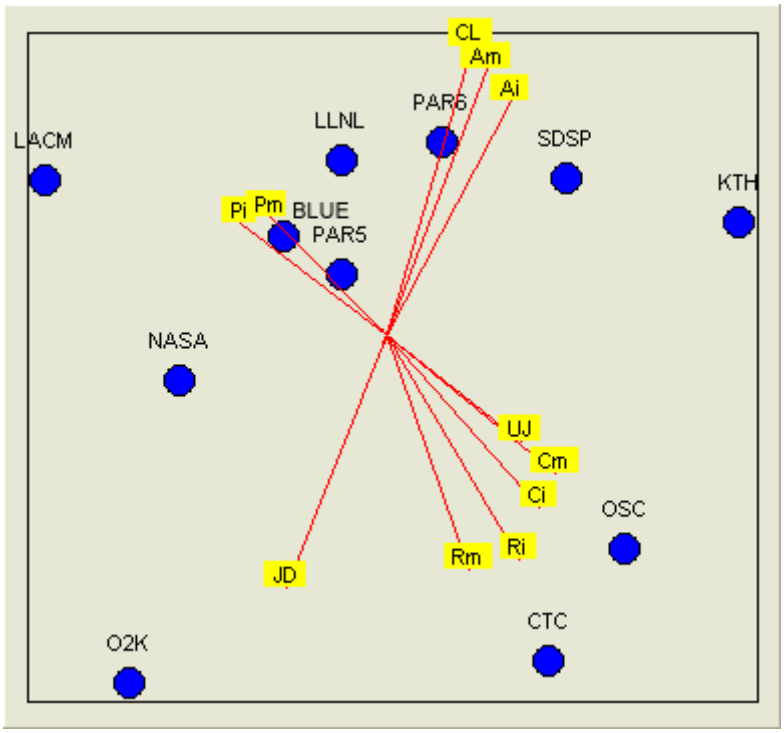


Figure 2: Co-plot output of all production workloads
See Table 2 for variable signs

The most obvious conclusion that can be drawn about the logs from this map is that they are not clustered. Architecture does not imply proximity – the CTC, KTH and SDSP logs are all IBM SP2 machines of similar size. Neither duration nor the year in which logs were recorded affects the distance between them. Even the two paragon logs are not as close as could be expected, which means that the workload changed over time. The location has a negligible effect as well – the SDSP, PAR5, PAR6 and BLUE logs are all from the San Diego Supercomputing Center, and are in the same vicinity; however, the Los Alamos CM5 and O2K log are far-off from one another. Also, the

San Diego logs range over six years (1995-2000 inclusive), during which many other factors could change as well.

Note that although we can see that the workloads are ‘far’ from each other, and use terms such as ‘high’ and ‘low’ values, this notion of distance is always relative to the other observations in this analysis. This happens because all variables in a Co-Plot analysis must be normalized – otherwise we can’t compare relations between them – and means that we should beware of attaching real distance to Co-plot’s output. We consider the workloads "far" because they are far enough so that the results of comparing schedulers will depend on the workload being used, which it does (Feitelson, 2003). In addition, this observation map, especially because we have no outliers, defines an intuitive notion of the “parallel workloads space”.

Since the workloads occupy all of this space, it is obvious that any single model – which will naturally occupy one point – can't represent all of them well within this space. The next section, with a Co-Plot that includes the available models as well, demonstrates this point. This also means that a parametric model is called for, and since we have successfully placed the data within a two-dimensional space (otherwise the coefficient of alienation was higher), looking for two parameters to represent the location of a workload on this space seems like the best choice.

4.2. Flurries

A flurry (Tsafrir and Feitelson, 2006) is a burst of very high activity by a single user. In contrast to normal active work, a flurry is an extreme case in which the load created by a user, over a short period of time, is orders of magnitude higher than usual, and affects the entire workload in a significant way. Sometimes the runtimes or CPU work used in a flurry is above the declared limits of the system, which makes them even more questionable. The following figure shows two examples of this phenomenon, both of which caused by an unusually high number of jobs by a user over a short period of time. On the right, from the SDSC SP2 log, a single user (out of 428 users) created a stream of jobs in one week, that is about seven times higher than the maximum number of jobs per week created by all other users, anytime in that log. On the left, from the LANL CM-5 log, 3 out of 213 users create similar short-term streams of a very large number of jobs – five to six times the maximum of all other users during the log.

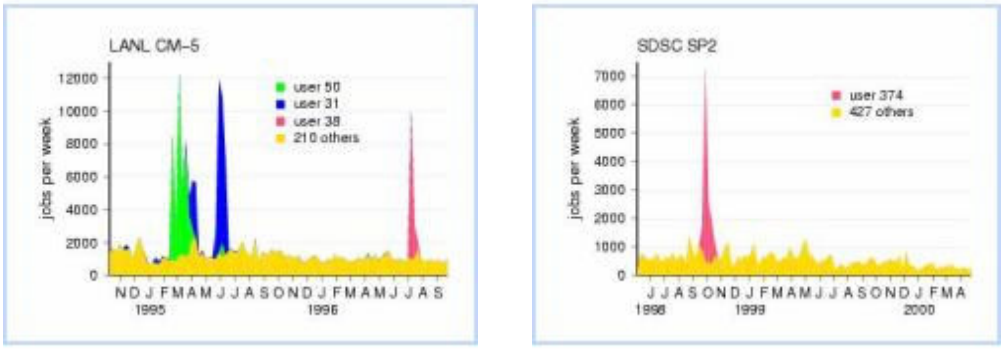


Figure 3: Jobs-per-week flurries in the LANL CM-5 and SDSC SP2 logs

As argued in (Feitelson and Tsafrir, 2006; Tsafrir and Feitelson, 2006), a workload model should in general not include flurries, since they are non-representative behavior, and should be modeled separately. Therefore, like any other case of outliers in a statistical data set, it is recommended to remove flurries from a data set before analyzing or modeling it. On the other hand, from the eleven inspected logs only four didn't have any flurries (KTH, LLNL, O2K and NASA). The NASA log did have a large number of system jobs, which were not user generated behavior (Feitelson and Nitzberg, 1995). Hence, flurries are almost to be expected when deploying a new supercomputing center, and when using workloads – to compare schedulers, for example – it would be highly advisable to test them both with and without flurries in the tested scenarios. Current workload models do not offer this feature, and cannot be easily extended to do so, since they do not recognize the notion of a job's user, or maintain a user's history and adjust the used distributions accordingly.

In order to test that conclusions drawn form Figure 2 are maintained when flurries are removed from the examined workloads, and also to compare the production logs against the workload models, we analysed them in a joint Co-Plot, shown in Figure 4. Note that in this case the CPU Load (CL), Users per thousand jobs (UJ) and Jobs per Day (JD) no longer appear: When adding the synthetic models to the analysis, the correlation of these variables significantly drops, so they were removed from the analysis. In addition, the Cirne and Berman model, which was analysed using three variants (its inter-arrival time distribution can be configured to match either of the KTH, CTC or SDSP2 logs), was also an outlier, and was subsequently removed to prevent it from distorting the other results.

Two insights can be drawn from Figure 4. The first is that cleaning the production logs (cleaned versions are marked with the letter *c*) didn't affect the results: The placement of workloads

Variable:	Sign:	CTC 1	OSC 2	LACM5 3	O2K 4	SDSP2 5	NASA 6	BLUE 7	PAR5 8	PAR6 9
Processors in Machine	TN	512	57	1024	2048	128	128	1152	416	416
Jobs per Day	JD	227.63	119.18	169.29	885.65	81.13	198.34	248.14	147.19	86.68
Runtime Load	RL	0.556	0.428	0.734	0.640	0.827	0.466	0.762	0.627	0.611
CPU Load	CL	0.464	0.583	0.477	0.391	0.732	0.466	0.627	0.666	0.685
Users per Kjobs	UJ	8.79	2.63	1.75	2.76	7.32	3.78	1.92	1.82	1.87
Executables per Kjobs	EJ	159.48	0.01	32.58	N/A	1103.21	27.03	N/A	N/A	N/A
Runtime Median	Rm	1114	382	414	569	237	86	219	38	207
Runtime Interval	Ri	57562	63952	11202	32243	47463	3716	22688	30478	31048
Processors Median	Pm	3	1	32	15	4	4	8	8	8
Processors Interval	Pi	39	5	480	127	64	63	128	63	63
Norm. Procs. Median	Nm	0.8	2.3	4.0	0.9	4.0	4.0	0.9	2.5	2.5
Norm. Procs. Interval	Ni	9.8	11.2	60.0	7.9	64.0	63.0	14.2	19.4	19.4
CPU Work Median	Cm	669	645	64	21	170	86	35	39	156
CPU Work Interval	Ci	55184	92580	7826	9624	47102	3716	16545	30305	30765
Inter-Arrival Median	Am	85	64	176	26	201	90	104	113	212
Inter-Arrival Interval	Ai	1527	2496	2048	357	4738	1217	1292	2467	4363

Table 3: Data of flurry-free production workloads

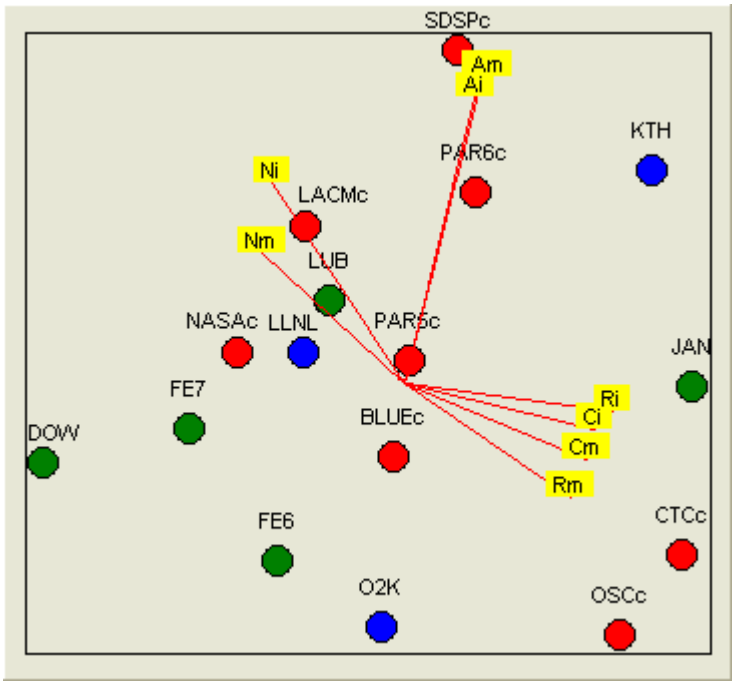


Figure 4. Co-Plot of cleaned logs versus models

in 2D space remains almost the same, and the direction of all the variable arrows remains the same as well. This result repeats with other observation and variable groups, and also when the original and cleaned workloads are analysed together (no shown here due to shortage of space). Second, with the exception of the Cirne and Berman model, the remaining models are not outliers, but naturally each of them occupies a single point on the plot, so none of them can represent all production workloads well.

4.3. Load Manipulation

Besides the need for a parametric model, which can optionally also model flurries, another serious problem resulting from figures 2 and 4 regards the current widespread methodology of manipulating the load of a given workload. Testing a new scheduling algorithm or comparing algorithms is often done by simulating how they schedule the same workload, over a range of loads. A graph of performance versus load is then drawn and analyzed. Such a test is usually carried out by taking a single workload, and manipulating its load (Majumdar et al., 1988; Lo et al., 1998).

There are three basic ways to raise a workload's load: Lowering the inter-arrival time, raising the runtimes, and raising the degree of parallelism. The most common (Majumdar et al., 1988; Lo et al., 1998) technique is to expand or condense the distribution of one of these three fields by a constant factor. Note that by doing so the median and interval (any interval) are also multiplied by the same factor. The choice of which field to alter in order to change a system's load should depend on the correlations between the runtime load and these three variables. We would choose lowering inter-arrival times if it were negatively correlated with load, and raising runtimes or parallelism if they were positively correlated with it. By doing so, we minimize the side effects of raising the load on other features of the workload.

Regrettably, this logical criterion does not seem to match the data. First, from Figure 2 it is clear that systems with a higher average load (CL) tend to have a higher inter-arrival time median (Am), not a lower one. Likewise, the median of jobs runtimes (Rm) is negatively correlated to the load, rather than being positively correlated. As for the degree of parallelism, while it is indeed correlated with the runtime load, this is only a weak correlation. In addition, raising parallelism is almost never possible, since it breaks the dominance of powers-of-2 requests, which completely alters the behavior of many algorithms for which the workloads are needed (schedulers are the most obvious example).

These correlations mean that a correct way to raise a system's load would end up with somewhat higher inter-arrival times, a somewhat higher degree of parallelism, and shorter runtimes. None of the three simplistic methods to alter the load satisfy these requirements; rather, they contradict them. This puts in question any results achieved by employing these methods, in particular results that discuss how a given algorithm will perform in extremely high loads. Such high loads are likely to be very different in practice from what the results were based on.

5. The Temporal Structure of Parallel Workloads

5.1. Locality of Sampling

The existing workload models draw job attributes from distributions, which means that each job is independent of all other jobs in the workload. Exceptions are the repetitions of identical jobs in (Feitelson, 1997) and a tailored multi-parametric distribution of inter-arrival times to simulate the daily cycle in (Lublin and Feitelson, 2003). This approach is in sharp contrast to the fact that four layers of temporal correlations in the arrival patterns of parallel jobs have been recently identified. The first is locality of sampling (Feitelson, 2002a; Feitelson, 2006).

Locality of Sampling is the name given to the phenomenon that although all job parameters tend to have a low diversity over short time frames (minutes to hours), they have a much higher diversity over long time scales (months to years). The distributions of all parameters of parallel jobs – arrival time, runtime, degree of parallelism and total CPU work – are typically continuous, possibly with a long tail (Downey and Feitelson, 1999; Feitelson, 2005; Feitelson and Tsafrir, 2006). On the other hand, when looking at a single day instead of a whole year, it is easily visible that the distributions of workload attributes tend to be modal. On a given day, usually just a few users are working on the machine, each one repeatedly executing the same job or similar jobs. It seems that the overall effect of continuous distributions and larger diversity is caused by the aggregation of these sessions, triggered by independent users. This is in line with the self-similarity of parallel workload attributes, since one of the mathematical ways to simulate it is exactly by such an aggregation (Beran, 1994).

(Feitelson, 2006) presents an overview of locality of sampling and provides the following. First, it shows that this phenomenon is distinct from other workload features, such as long-range dependence. Second, it provides a formal method of quantifying the effect in a given workload. Third, it shows that under this quantification the known production workloads exhibit significant locality – much more than what can be expected at random, as the current workload models work. And fourth, it suggests a simple way to model the phenomenon, by selecting jobs from a single global distribution, and then repeating them by a number of times distributed by a Zipf-like distribution.

A simple and effective tool to show that locality of sampling exists in production workloads but not in current models is auto-correlation. Auto-correlation is defined as the correlation between a time series, and itself under some shift. For example, if we define a time series J_t such that its

value in time *t* is the total number of jobs submitted in the *t*'th day of the workload, then by computing the auto-correlation of this series with a shift of one, we'll test the correlation of the number of jobs between adjacent days. To test for locality of sampling, we computed the auto-correlation with a shift of one for the workload aggregated at 15 minutes and at 150 minutes, for five workload attributes: Number of jobs, runtime, number of processors, total CPU work and inter-arrival time. The full results for both logs and models are given in a table in Appendix C; the average correlations for all production logs are as follows:

Runtime		**# of Processors**		**Total CPU Work**		**Inter-Arrival Time**		**# of Jobs**	
15m	150m	15m	150m	15m	150m	15m	150m	15m	150m
0.17	**0.18**	**0.36**	**0.32**	**0.07**	**0.06**	**0.14**	**0.03**	**0.51**	**0.52**

Table 4. Locality of Sampling: Average Auto-Correlations in Production Logs

Results show significant auto-correlation of job runtimes (0.17-0.18), number of processors (0.32-0.36), and the number of jobs (0.51-0.52); a significant auto-correlation (0.14) of inter-arrival times only in the 15-minutes time scale; and a negligible auto-correlation for the total CPU work parameter (0.06-0.07). This means that jobs tend to "cluster" in time near other jobs, which have similar runtimes and degree of parallelism as they do. This matches the intuition of users logged in a session, running the same or similar jobs over and over again. However, the inter-arrival time between jobs is not kept – users wait different times between successive runs of the same job – and the total CPU work of jobs changes as well.

Testing the models (by generating over 100,000 jobs from each one) suggests that the models do not generally represent these phenomena correctly. Downey's model does not capture locality of sampling, and in contrast exhibits a significant negative auto-correlation of inter-arrival times. Jann's models capture about half of the real observed auto-correlation of the runtimes, parallelism and number of jobs, and share Downey's model negative auto-correlation of inter-arrival times for the 150-minutes time scale. Feitelson's models are the only ones to capture the locality of the runtime and parallelism. This is achieved in the models by directly repeating some of the simulated jobs, using the same runtime and parallelism. On the other hand, these models show no locality in the number of jobs, and exhibit undesirable auto-correlations in the total CPU work and inter-arrival time parameters. Lublin's model shows no significant sign of locality in any of the parameters. Cirne's models capture the 150-minutes real-world results very well, except for the inter-arrival times, but are misguided in the 15-minutes time scale for all except the total CPU work parameter.

5.2. The Daily Cycle

The next layer of temporal correlations in parallel workloads is the daily cycle. It is caused by the simple fact that people generally work during the day and sleep at night. As Table 5 shows, the daily cycle is the dominant factor in determining temporal correlations between jobs in the 12-hour and 24-hour time scales. The numbers are auto-correlations (this is the classic statistical tool to measure cycles in time series) of the workloads aggregated at a six-hour scale, shifted by 2 and by 4 to compute the 12-hour and 24-hour auto-correlation figures respectively. The full data table for both logs and models in given in Appendix C.

Runtime		**# of Processors**		**Total CPU Work**		**Inter-Arrival Time**		**# of Jobs**	
12h	24h	12h	24h	12h	24h	12h	24h	12h	24h
-0.03	**0.26**	**0.05**	**0.21**	**-0.06**	**0.15**	**0.01**	**0.02**	**0.11**	**0.33**

Table 5. Daily Cycle: Average Correlations in Production Logs

The 24-hour figures for the production logs are significantly positive for all except the inter-arrival time parameter, showing that a daily cycle indeed exists. As with locality of sampling, the auto-correlation is higher in the number of jobs parameter, and lower in the total CPU work.

The 12-hour figures, however, are near-zero for all parameters, except the number of jobs. One possible explanation is that this is caused by the combination of two opposing forces that cancel each other out. The first is locality of sampling, which pushes the 12-hour auto-correlation to be positive. The second is the daily cycle, in which the night workload is negatively correlated to the daytime workload. The combined result is that the 12-hour correlations are near-zero, except for the number of jobs metric, for which – as shown in the previous section – the locality of sampling correlation is stronger.

Whether this hypothesis is accurate or not, the 12-hour figures prove one thing for sure: that the 24-hour auto-correlations are not caused by locality of sampling, but by some other phenomenon. This is because the effect of locality diminishes around the 12-hour time scale. The only candidate to explain the "other factor" is the daily cycle.

The results about the models are as follows. Downey's model is quite close to the average production logs in many of the parameters, except for the 24-hour cycle of runtimes and total CPU work. Jann's models show no significant auto-correlations at both the 12-hour and 24-hour range, except for a small positive auto-correlation in the number of jobs parameter. Feitelson's models show no significant auto-correlations in any of the parameters. Lublin's model shows positive auto-

correlations in the runtime and total CPU work parameters, and no auto-correlations in the other parameters. However, in the two parameters where correlation exists, in exists in both the 12-hour and the 24-hour scales. Cirne's models show very strong auto-correlations, sometimes positive and sometimes negative, in all parameters except for the total CPU work (in which these models are close to the production logs' average). In all other cases, the models' auto-correlations are quite far from the observed real-world averages.

5.3. The Weekly Cycle

So far we have seen that locality of sampling is the dominant factor in determining the temporal correlations between jobs at the minutes-to-few-hours time scale, and that the daily cycle is the dominant factor around the 24-hour time scale. The weekly cycle is the dominant factor around the 7-day time scale. "Around" means that the borders are vague – just like the daily cycle dominates somewhere between the 12 hour to 48 hour range, the weekly cycle usually dominates between the 3 days to 20 days range. Figure 5 is an auto-correlation plot of a 1-day aggregation of the BLUE log: Note the peak every seven days due to the weekly cycle, in addition to the peek at Day 1 due to the daily cycle.

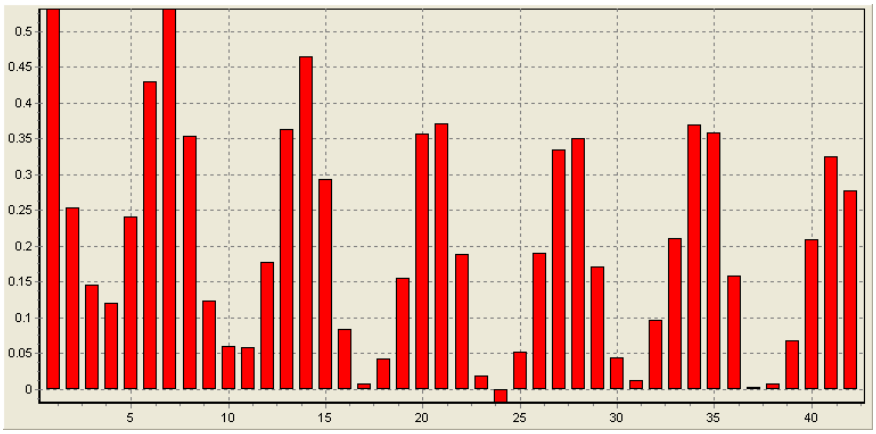


Figure 5. Number of Jobs per Day Auto-Correlation Plot for the SDSC Blue Horizon Log

Table 6 was created by doing a 1-day aggregation of the workloads and then computing the auto-correlation of each series at a shift of 7 – one week. As in the previous sections, the production logs show significant positive auto-correlations in all parameters, except the inter-arrival time. As for the models, none of them was designed with the weekly cycle in mind, so as in the previous sections – some of the models get it right for some of the parameters, but none of them is consistent.

Runtime	# of Processors	Total CPU Work	Inter-Arrival Time	# of Jobs
7 days	7 days	7 days	7 days	7 days
0.30	**0.20**	**0.17**	**0.00**	**0.28**

Table 6. Weekly Cycle: Average Correlations in Production Logs

5.4. Self-Similarity

Studies of traces of Ethernet network traffic (Leland et al., 1994), web server traffic (Crovella and Bestavros, 1996) and file system requests (Gribble et al., 1998) have revealed an unexpected property of these traces, namely that they are self-similar in nature. Intuitively, self-similar stochastic processes look similar and bursty across all time scales. Physical limitations, such as the finite bandwidth and lifetime of any network or server, inhibit true self-similar behavior, but the presence of self-similarity over considerably long amounts of time (months, in the case of parallel computers), makes this phenomenon of practical importance.

Understanding self-similarity requires a short mathematical preface. Given a time series $\{X_i\}_{i=1..n}$ we look at it at various aggregation levels: $X^{(m)}$ is a new time series defined by summing m elements of the original series into one element of the new one. For example, $X^{(3)} = \{ X_1+X_2+X_3, X_4+X_5+X_6, \ldots \}$. The auto-correlation function r(k) of a time series measures how "similar" the series is to itself shifted by k time points. A self-similar time series is one that is "similar to itself" over many time scales, and so the official definition is:

$$\forall k \;\; r^{(m)}(k) = r(k) \qquad \text{(exactly)} \qquad \textbf{(1a)}$$

$$\forall k \; r^{(m)}(k) \rightarrow r(k) \qquad \text{(asymptotically)} \qquad \textbf{(1b)}$$

This manifests itself in a number of mathematically equivalent ways:

Slowly Decaying Variances: The variances of the aggregated series decay hyperbolically and not exponentially as in most familiar distributions:

$$\mathrm{Var}(X^{(m)}) \propto m^{-\beta} \quad \text{for some } 0 < \beta < 2 \text{ and all } m \geq 1. \qquad \textbf{(2)}$$

This also means that the original series X has infinite variance.

Long Range Dependence: The auto-correlation functions decreases very slowly:

$$r(k) \approx k^{-\beta} L(t) \quad \text{for some } 0 < \beta < 2 \text{ as } k \rightarrow \infty \qquad \textbf{(3)}$$

Where L(t) is a slowly changing functions (asymptotically constant). This implies that each X_i affects future X_i's for a very long time into the future. Long-Range Dependence means that

the distribution of the original time series is Heavy-Tailed: Values that are extremely far from the mean will appear quite often.

The Spectral Density obeys a power law (near the origin). The spectral density measures cycles in a time series. The formal definition is:

$$f(w) = \frac{2}{N} \times \left[\left(\sum_{k=1}^{N} X_i \cos(wk) \right)^2 + \left(\sum_{k=1}^{N} X_i \sin(wk) \right)^2 \right] \qquad (4)$$

Where ϖ is a frequency. The Periodogram is a graph that is computed by plotting $f(\varpi_i)$ against the frequencies $\varpi_i = 2\Pi i / N$ for i = 1..N.

If the original time series is self-similar, then the following property will hold:

$f(\varpi) \approx c_0 \varpi^{-(1-\beta)}$ for some $0 < \beta < 2$ as $\varpi \to 0$ (**5**)

The $0 < \beta < 2$ parameter in the above equations has the same value in all of them, and it is a measure of how strong the self-similarity in the original times series is. However, for historical reasons we don't use β but instead use the Hurst Parameter, simply defined as:

$$H = 1 - \frac{\beta}{2} \qquad (6)$$

It is named after Hurst, who first discovered it in the late '50s. If H = ½ the process is not self-similar (it is a 'random walk'), and when ½ < H < 1 it is self-similar with positive drift (most self-similar data to date are positive). Hurst worked on many time series from nature, and measured self similarity using the Rescaled Adjusted Range (R/S) Statistic. For a time series X having average $\overline{X}(n)$ and sample variance $S^2(n)$ this statistic is given by:

$R(n) / S(n) = [\ 1 / S(n)\ ] \times [\ \max(0, W_1, W_2, \ldots, W_k) - \min(0, W_1, W_2, \ldots, W_k)\ ]$

Where:

$W_k = (X_1 + X_2 + \ldots + X_k) - k\,\overline{X}(n) \qquad (k \geq 1)$

Short-range dependent observations seem to satisfy $E[\ R(n) / S(n)\ ] \approx c_0 n^{0.5}$, while long-range dependent data, such as self-similar processes, are observed to follow:

$E\ [\ R(n) / S(n)\ ] \approx c_0 n^H \quad (0 < H < 1)$ (**7**)

This is known as The Hurst Effect, and can be used to differentiate self-similar from non-self-similar processes.

The above equations give rise to three independent methods for testing for self-similarity and estimating the Hurst parameter of a given time series:

Variance-Time Plot: Take a logarithm out of both sides of equation (2):

$$\log [\mathrm{Var}(X^{(m)})] = c_1 - \beta \log(m) \qquad (\mathbf{8})$$

In words: by plotting $\mathrm{Var}(X^{(m)})$ against log(m) on a log-log plot, we should get a straight line, and its slope should give us an estimate of β, from which we have an estimate of H.

R/S Analysis or Pox Plot: Here we call on the Hurst Effect for help. Taking a logarithm out of both sides of equation (7) gives:

$$\log [R(n) / S(n)] = c_2 + H \log(n) \qquad (\mathbf{9})$$

So if we plot R/S(n) against *n* on a log-log plot, the slope will be an estimate of H. This method is claimed to be the most robust to slight changes in the distribution.

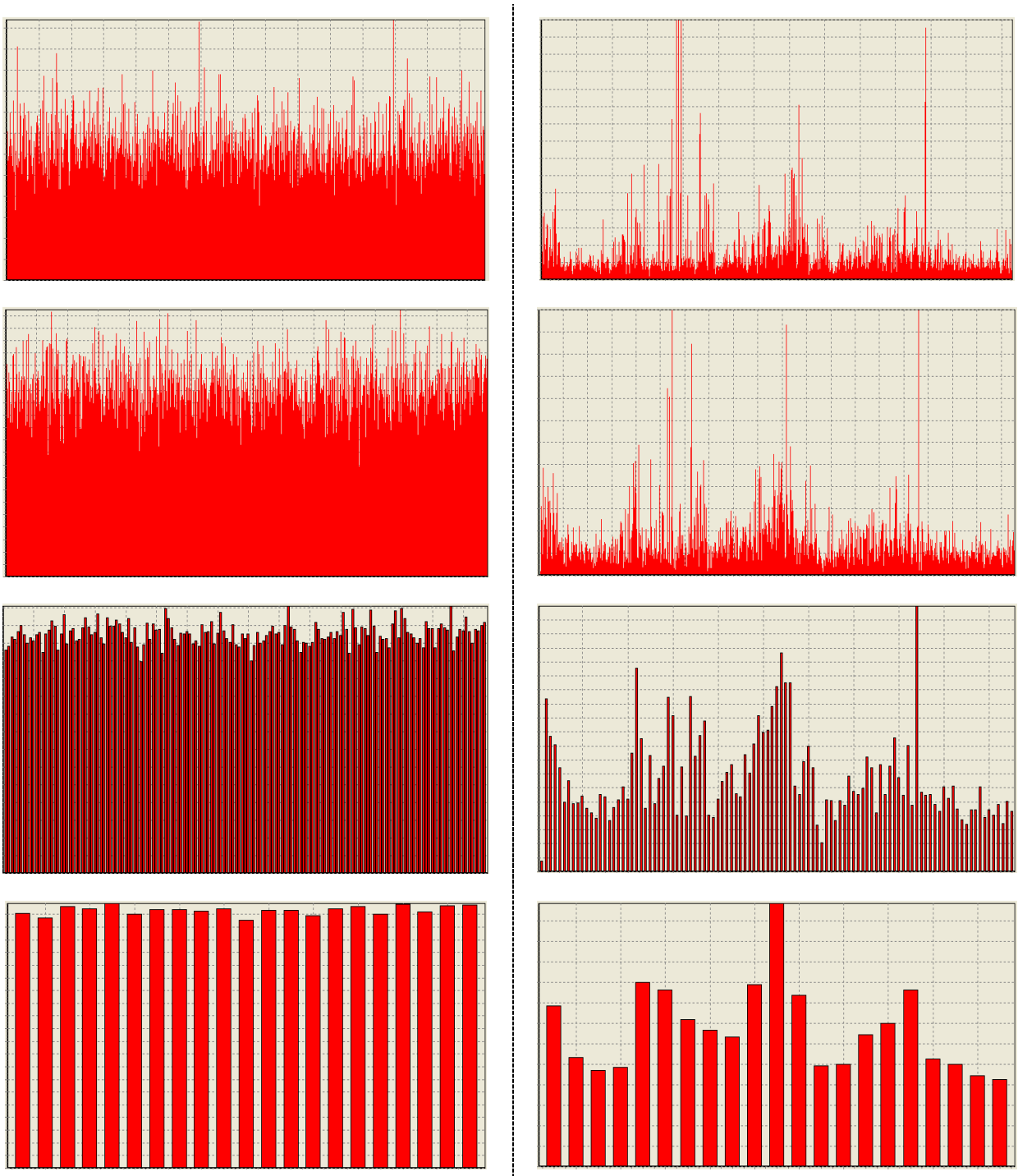
Periodogram: Here we take the logarithm from equation 5, and do the same:

$$\log [f(\varpi_i)] = c_0 - (1- \beta) \log(\varpi_i) \qquad (\mathbf{10})$$

for a more comprehensive description of self-similarity see (Beran, 1994). Figure 6 presents a more intuitive explanation of self-similarity and its meaning. To the right are four plots presenting the number of used processors over time in the SDSP2 log; to the left are similar plots, for the SDSP2 variant of the Cirne and Berman model. From top to bottom, the aggregation level of the four plots are 100 minutes, 1000 minutes, 10000 minutes (≈1 week), and 50000 minutes (≥1 month). The log is self-similar in the used processors parameter, with an average estimated Hurst parameter of 0.83, while the model is not – it has an average estimated Hurst parameter of 0.50.

The end result is that the model's burstiness at short time scales decays quickly, and at a weekly or monthly scale, the variance in its number of processors is negligible. On the other hand, the log demonstrates high variance even across weeks and months – this is "slowly decaying variances" in practice. This also means that there is a long-term dependence between jobs – to create month-long peaks, a job's used processors count must be able to affect jobs that are weeks away.

Table 6 summarizes our self-similarity results as follows. For each log and model, we computed the three tests of Variance-Time Plot, Pox Plot and Periodogram. Each test produced an estimate of the Hurst Parameter, between 0.50 and 1.0. Due to space constraints, table 6 contains only the average of these numbers for each log and parameter.



Cirne-SD Model, Used Processors | *SDSC-SP2 Log, Used Processors*

Figure 6. Visual Demonstration of Self-Similarity

Runtime	# of Processors	Total CPU Work	Inter-Arrival Time	# of Jobs
Average Hurst parameter over all logs:				
0.72	**0.75**	**0.69**	**0.67**	**0.77**
Average Hurst parameter over logs that are at least one year long:				
0.75	**0.75**	**0.72**	**0.77**	**0.84**

Table 7. Self-Similarity: Average Hurst Parameter Estimates in Production Logs

The main result is that the production logs are self-similar in all the tested parameters. Note that we computed two averages: the first is over all logs, and the second is for the six logs that were at least one year long. The second average is higher and also more representative, since shorter logs are limited by definition by the time scale in which self-similarity can exist. The models, on the other hand, do not exhibit self-similarity (the full data table for all logs and models in given in Appendix C) . The only model to show significant Hurst parameter estimates is the Feitelson '97 model, probably due to its notion of repeating jobs.

6. Towards a User-Based Parallel Workload Model

The last two sections presented some of the techniques by which workload analysis is done, and have also provided us with a list of current limitations of existing workload models:

Methodological Limitations:	*Temporal Structure Limitations:*
Need for a parametric model	Locality of Sampling
Inability to model flurries	The Daily Cycle
Lack of a correct way to manipulate load	The Weekly Cycle
Need for direct user modeling	Self Similarity

Methodological limitations hinder a researchers' ability to correctly evaluate new algorithms and policies for parallel computers (whether they use synthetic or production workloads), and temporal structure limitations hinder the ability to rely on synthetic models for such tasks. This is especially true for evaluating resource management policies (Feitelson, 2002 and 2003). The first two sections of the results chapter present two types of scheduling algorithms which specifically rely on the temporal structure of workloads, as well as on direct user modeling (i.e. the algorithm relies on knowing which user submitted each job). This provides motivation for developing the new parallel workload model which will address the above issues.

We argue that building a user-based workload model, instead of directly modeling parallel jobs as done today, provides an elegant way to address all of the above issues.

These problems have been identified from other angles (Feitelson, 2002a; Frachtenberg and Feitelson, 2005) before and during this research. (Jann et al, 1997), (Cirne and Berman, 2001) and (Lublin and Feitelson, 2003) manipulate parametric distributions to model the daily cycle. (Feitelson and Jette, 1997) create locality of sampling using exact job repetitions, and (Song,

Ernemann, and Yahyapour, 2004) create it by modeling runtime and parallelism using Markov chains. (Karatza and Hilzer, 2003) present a model which tackles the long-range dependency in the distributions of runtime and parallelism. (Song, Ernemann and Yahyapour, 2005) create a user based workload model based on four clusters of users, but model only runtime and parallelism, and do not relate to the temporal structure issues, which as we show are crucial to the user clustering process as well. None of these approaches scales well to a solution which covers the other limitations.

Our modeling framework is multi-scale (Calzarossa and Serazzi, 1994; Menascé et al, 2003) and is based on users and sessions. From the modeling point of view, this means that we don't model distributions of jobs, but instead model distributions of user classes. For each user class, we model distributions of session classes; and for each session class, we model jobs. To make things simple, the session classes are identical for all user types. A key part of the model construction process will be to uncover the session classes independently of user types – so the only difference between user types is the frequency of each session class.

At the bottom line, such a model is used to generate a stream of synthetic jobs, like the existing models reviewed earlier. But the data for each job includes, in addition to its arrival time, runtime and number of processors, also an identifier of the user who submitted it, and an identifier of the session it belongs to. Sessions, although a natural and well-known aspect of human use of computers (Arlitt, 2000), have not been analyzed in the context of parallel workloads to date. The user identifier enables using this model when evaluating algorithms that rely on users directly, for example to predict a new job's runtime based on its user's history. This is the first advantage of a user-session-based model.

Second, such a model is expected to be self similar. There are two ways to synthesize a self-similar time series: directly or by aggregation. The direct method produces signals based on a self-similar distribution, such as fractional Brownian motion or fractional ARIMA processes. These processes allow good control over the Hurst parameter, but they make it harder to control other desirable properties of the workload. The second method to produce self similarity is by aggregating the signals generated by a set of independent sources – intuitively, users – under the following conditions: users must switch between active and inactive periods – intuitively, sessions – and the duration of the "on" and "off" times must be from a heavy-tailed distribution. A distribution is called "heavy-tailed" if it has infinite variance; informally, this means that there is "significant"

probability for seeing values that are extremely far from the mean. Therefore, it seems likely that basing the workload model on an aggregation of users, each exhibiting on on/off behavior (sessions), will result in a self-similar model, as long as heavy-tailed distributions are used.

Third, a user/session model provides a simple way to model time-dependent features, such as locality of sampling and cycles. Locality is very hard to model using a distribution on all jobs, since while the local diversity is very small, the overall diversity is very high. Users and sessions make it simple, by capturing that high diversity and separating the intra-session low diversity model in a natural way. Cycles are also easy to model, due to a similar kind of separation of concerns: the inter-arrival time of jobs within a session (which doesn't have cycles) is modeled separately from the inter-arrival time between session, which obeys the daily and week cycles. We have also included locality and the known cycles as features used to define user and session classes, to ensure that our model will capture the differences between sessions that happen at different times (day versus night sessions, for example). These features were indeed found to be of major importance.

Fourth, a user/session model captures flurries, as outlier sessions. Flurries were identified immediately in our analysis as outliers, and a full model can be configured to either include them, as a special kind of session, or not at all. This gives researchers the choice, which is vital (Tsafrir and Feitelson, 2006) since on one hand flurries are not characteristic of common workload, but on the other hand most long production traces contain at least one.

Fifth, a user/session model provides a more natural way to manipulate load correctly. Beyond making the model parametric (which is not made easier or harder by building a user-based model), the load will be (optionally) one of the parameters. As we've seen, the problem with load manipulation of current production logs and models is that all three simple ways to manipulate their load are problematic. A user-based model opens new possibilities, such as having a higher load result in more active users, more sessions per user, or a higher proportion of intensive sessions. This provides a more natural model of how load increases are caused in real life, integrates easily into our modeling framework, and is more likely to maintain other statistical properties of the workload.

And sixth, a user/session model is beneficial to understanding and exploiting workload features because it works: We succeeded in identifying the core variables that explain most of the variance between users and sessions, and then identified a small number of user and session classes that cover their entire spectrum. This proves the whole approach to be viable in practice.

RESULTS

7. Adaptive Parallel Scheduling

7.1. Introduction

This study began with the observation that widespread scheduling systems in use today, such as Maui (Jackson et al., 2001) and EASY (Skovira et al., 1996), sometimes exhibit inconsistent and unstable behavior. As the next section will show, on a typical year these schedulers have several "good" months, a few "bad" ones, and one or two "major blunders". In very bad months, the computer's response time is several times slower than usual, making it far less usable.

Another observation that we make is that another scheduling algorithm, Flex (Talby and Feitelson, 1999), offers comparable performance to Maui, with an important difference: Although it also has "good" and "bad" months, they are different than those of Maui in every case we examined. This raises an opportunity: If we could predict when each scheduler should be used, we can enjoy the best of both worlds.

This section describes two algorithms that use different strategies to predict when each scheduler is most appropriate. One chooses by recent past performance, and the other by the recent degree of parallelism, which is shown to be correlated to algorithmic superiority. Simulation results for the algorithms on production workloads are analyzed, and illustrate unique features of the chaotic temporal structure of parallel workloads. Specifically, the adaptive schedulers rely on locality of sampling, the daily cycle and the weekly cycle.

We provide best parameter configurations for each algorithm, which both achieve average improvements of 10% in performance and 35% in stability for the tested workloads. This is a useful and highly practical improvement, as it refers to the overall performance of the entire computer.

7.2. The Inconsistent Performance Problem

The study compares four backfilling parallel schedulers: EASY, Conservative, Flex and Maui, which have been studied and compared before (Talby and Feitelson, 1999; Mu'alem and Feitelson, 2001). We have repeated these experiments by simulation on logs from three parallel IBM SP2 computers – the SDSC Blue Horizon (BLUE), the SDSC IBM SP2 (SDSP2), and the KTH IBM SP2 (KTH). Together, these three logs constitute almost six years of real user activity.

	Wait Time				Bounded Slowdown			
	Cons	Easy	Flex	Maui	Cons	Easy	Flex	Maui
BLUE	1878	2083	1661	1535	9.1	11.0	8.0	10.2
SDSP2	6379	6033	5500	5612	28.4	29.2	25.7	17.6
KTH	7302	6806	6250	6731	89.2	88.9	70.5	63.3

Table 8. Backfilling Schedulers Performance for Entire Logs

Table 8 uses both the average wait time and average bounded slowdown to compare the schedulers. The first metric is equal towards all jobs, while the second one penalizes causing short jobs to wait, encouraging better interactive behavior. This table confirms past results: Flex is the best scheduler in wait-time in two out of three cases, and in slowdown in one of the three; Maui is the best scheduler in the other cases, and Conservative and Easy typically have lower performance. However, this is not the whole picture.

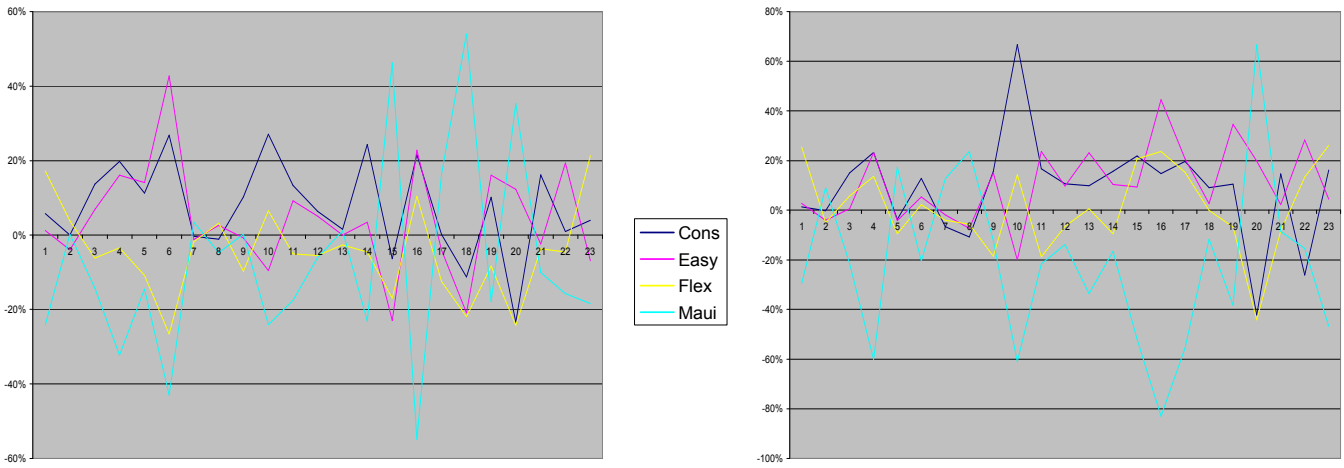
In order to gain a better understanding of the algorithms than previously done, we re-ran the simulations and gathered per-month results. This is significant since as shown in (Talby, Feitelson and Raveh, 1999) the usage pattern in production logs may change radically over time, and also because this gives us a total dataset of 67 unique workloads, instead of just three. This monthly breakdown uncovered an unexpected result, shown in Table 9.

The table counts how many months, in absolute and relative terms, each scheduler "won" by providing the best performance among the four schedulers. As the data shows, Maui has a clear lead and "wins" about 70% of the months, under both metrics, consistently across all logs. When only Flex and Maui are compared, Maui "wins" about 75% of the months, and Flex wins the other 25%.

This is surprising because as Table 8 shows, Maui's overall yearly statistics are not better than Flex's. This can only happen if Flex wins by very high margins when it does, and this is indeed what we found: about one month per year, Maui shows catastrophic performance, by presenting average wait times that are 2-4 hours longer than those of Flex. Such a wait time is about three times the yearly average on each of the machines we study, and is enough to make Flex the more responsive overall scheduler in two out of three cases.

	Wait Time				Bounded Slowdown			
	Cons	Easy	Flex	Maui	Cons	Easy	Flex	Maui
BLUE	0	0	8	23	2	0	7	22
	0%	0%	26%	74%	6%	0%	23%	71%
SDSP2	0	2	6	15	3	0	4	16
	0%	9%	26%	65%	13%	0%	17%	70%
KTH	0	1	2	8	0	0	3	8
	0%	9%	18%	73%	0%	0%	27%	73%

Table 9. Backfilling Schedulers Performance for Entire Logs



Wait Time *Bounded Slowdown*

Figure 7. Monthly deviations from the average performance in the SDSP2 log

	Months won by Flex		Months won by Maui	
	Average Gap	Maximal Gap	Average Gap	Maximal Gap
BLUE	46%	251%	43%	189%
SDSP2	44%	97%	29%	145%
KTH	56%	158%	29%	53%
BLUE cln	46%	144%	33%	147%
SDSP2 cln	26%	95%	25%	93%

Table 10. Average and Maximal Wait Time Gaps between Flex and Maui

Flex shows considerable improvement over Maui in the months in which it leads, but this works the other way as well: Maui often outperforms Flex significantly. Table 10 shows both the average and maximum monthly gap in average wait time between these two schedulers; the gap is shown in percentage to make it possible to compare across logs. The numbers are consistently very high, which is an important unexpected fact, on which we base the second part of this chapter.

Figure 7 shows this phenomenon visually. It shows the monthly deviation of each log from the average performance of all logs that month. The deviations are shown for the response time and bounded slowdown metrics, in the SDSP2 log. Similar results repeat for all other logs and metrics.

The last two rows of table 10 prove that these results are largely not the result of outliers in the logs. Following the work on flurries in (Tsafrir and Feitelson, 2006), we re-ran the simulations on "cleaned" versions of the logs – such versions exist for the BLUE and SDSP2 logs, since no flurries where found in the KTH log. The results show that although the removals of flurries (which exist in real-life workloads) reduce some of the figures, most of the performance gap remains unexplained.

Months in which the four schedulers greatly differ also do not stand out statistically as outliers. On the contrary – the statistics of workload attributes such as the inter-arrival time, runtime

and parallelism is often similar across months with very different scheduling performance. Therefore, we suspect that the cause of the performance gaps is the temporal structure of the workload, or the correlations between adjacent jobs. More specifically, it is likely that self-similarity – shown in (Talby, Feitelson and Raveh, 1999) to exist in parallel workloads – plays a role here, since it exhibits itself in long-term correlations between jobs, that cause long-term (i.e. over months) load patterns that change chaotically over time. However, the focus of this section is not the theoretical explanation of the performance gap phenomenon, but rather a practical utilization of it, by using the most practical tool to handle chaos in dynamic systems: adaptability.

7.3. Performance Based Adaptive Scheduling

7.3.1. Rationale

The practical opportunity that the performance gap presents is clear: we can theoretically improve performance by 30-40%, if we could predict in advance which scheduler will win each month. In practice, we can't predict the future but can afford mistakes, since a 10% gain will be a significant and very useful achievement as well.

An adaptive algorithm makes sense only if we can make one very basic assumption – that past behavior predicts future behavior. Using the terms we defined in the previous chapter, we are heavily relying here on locality of sampling. We will also need to rely on the daily and weekly cycle, since it's not enough to know that the past predicts the future: We also need to know what is the right time scale to look at the past with. For example, the daily cycle suggests that using the last 24 hours to predict the next 24 hours is smarter than using the last 17 hours to predict the next 17 hours. However, we will not assume this in advance when designing the algorithm, but rather test it using a wide range of time frames, and see if the 24-hour and 7-day time frames result in superior performance. Such a result will be evidence that exploiting these cycles is useful and practical.

7.3.2. Algorithm

The performance-based adaptive scheduler (PBAS), summarized in figure 4, assumes that the scheduler that performed best in the recent past is likely to perform best in the near future. A nice feature of adaptive algorithms is simplicity: Run a set of candidate schedulers in the background, at each point in time use only one of them as the active one; at regular intervals, review the performance of all schedulers, and for the next time frame activate the best-performing one.

Performance-Based Adaptive Scheduler Algorithm

- *Initialization*
  - Initialize each of the candidate schedulers.
  - Select one of them arbitrarily as the *active* scheduler.
- *On Job Arrival Event*
  - Notify each of the candidate schedulers of the arrival.
  - Start running only the jobs that the active scheduler decided to run.
- *On Job Termination or Cancellation Event*
  - Notify each of the candidate schedulers of the event.
  - Only start running jobs the active scheduler decided to run.
- *On Switch Schedulers Event*
  - Measure the performance of each of the candidate schedulers during the last measurement period.
  - Set the active scheduler to be the one which performed the best according to a preferred metric.
  - Synchronize all schedulers to the current real state, which may be different than the state they reached (since they don't "know" that their output is not used to drive the actual system).
  - Set another 'switch' event to happen after another time interval.

Figure 8. Performance-Based Adaptive Scheduler Algorithm

7.3.3. Empirical Results

The PBAS algorithm can be configured by four parameters: The candidate schedulers set, the time frame between switching events, the metric used to compare schedulers' performance, and the history time frame to consider when comparing performance. In our runs, the history time frame was always identical to the switching time frame, because using a longer history period has an unclear semantics when switching happens during that period, and we did not want to add to the algorithm's complexity.

The six graphs of Figure 9 summarize the PBAS results for different metrics and switching time frames. Results are given in terms of both average wait time and bounded slowdown for each log. The candidate schedulers set includes only Flex and Maui, for a reason that will be explained later. The time scale is logarithmic, since we tested over five orders of magnitude of time – from one minute to fifty days. The metrics whose results are presented are response time (PBAS-Resp),

bounded slowdown (PBAS-BSld), and utilization (PBAS-Util). The wait time and slowdowns metrics were tested as well, and result in similar or worse performance; they were left out of the graphs only to improve their readability.

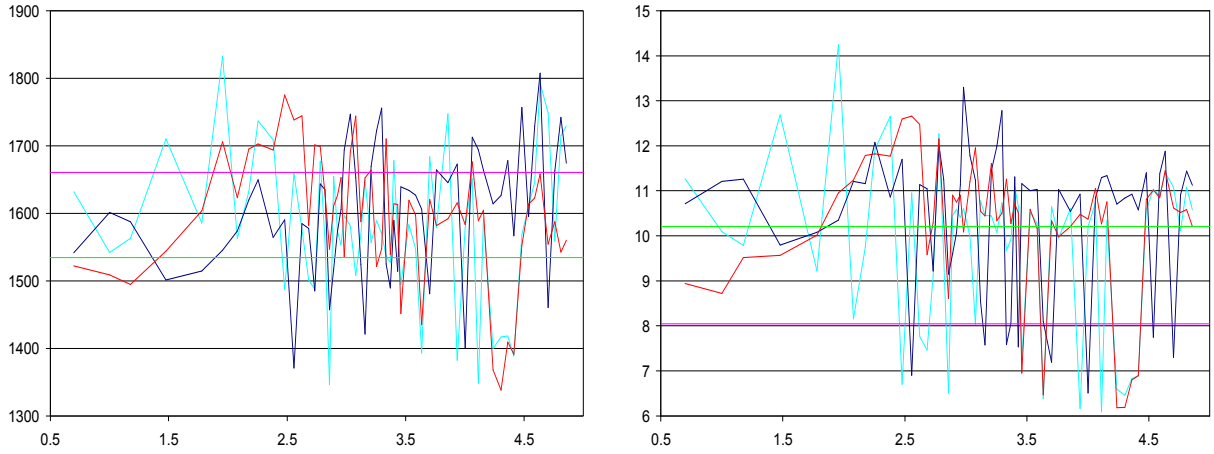
Perhaps the most surprising result of Figure 9 is the fact that performance is highly non-continuous: it seems to be very sensitive to the time frame, and does not lend itself to any elegant explanation. There are several unique points which are consistent across logs, but most of the time, behavior can only be predicted by experimentation. The same happens across metrics: using different metrics for the same time frames results in inconsistent and non-continuous performance. Also, with a few rare exceptions, using a certain metric does not guarantee good results in that metric.

Maui and Flex are shown as straight lines in the plots, since they are not adaptive. The visuals show that most PBAS runs are between the Maui and Flex results for that log, which is expected – after all, the adaptive scheduler can only run one of them at any given time. It is also visible that the wrong selection of parameters can lead to performance that is worse than both Easy and Flex, but on the other hand – the right selection can result in outperforming both. It is expected and doesn't matter than most parameter combinations don't work, as long as one or more combination consistently improves performance. In our case, there are two consistently winning combinations, presented in Table 11. The numbers show the percent of improvement over Maui, both in average wait time (performance), and in standard deviation between wait times of different months (stability).

The average performance gain of the Utilization-7 Days configuration is 10%, and its average stability gain is 36%. The numbers for the Response Time-12 Hours configuration are 9% and 25%. Another good configuration was Bounded Slowdown-1 Hour, which worked very well for KTH and SDSP2, but caused a small decline in BLUE's performance.

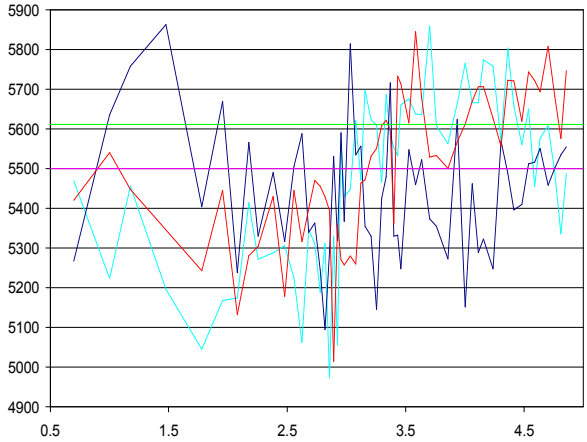
	Performance Gain			Stability Gain		
	KTH	SDSP2	BLUE	KTH	SDSP2	BLUE
PBAS Utilization 7 Days	13%	8%	9%	63%	29%	17%
PBAS Response Time 12 Hours	5%	11%	12%	24%	27%	24%

Table 11. Best parameter combinations for PBAS, and overall improvement over Maui

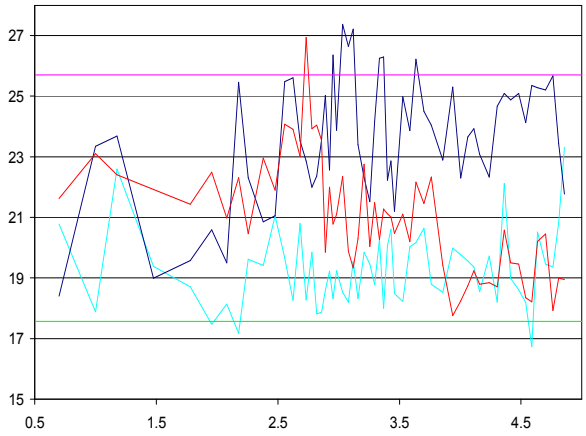


BLUE, Wait Time

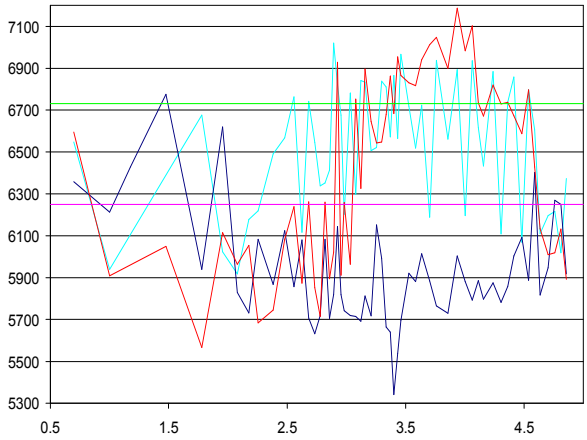
BLUE, Bounded Slowdown



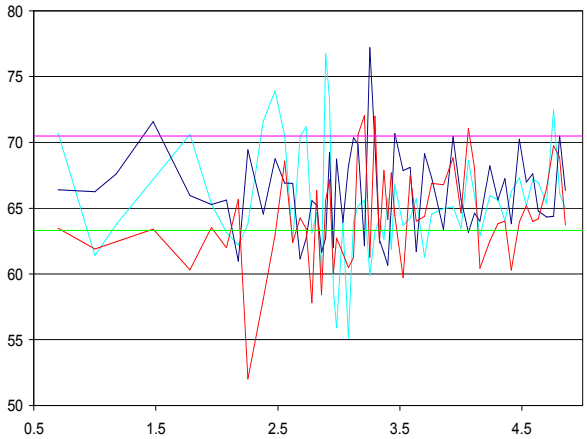
SDSP2, Wait Time



SDSP2, Bounded Slowdown



KTH, Wait Time



KTH, Bounded Slowdown

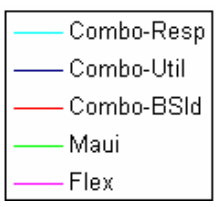


Figure 9. Performance-Based Adaptive Backfilling by Time frame and metric.

X axis is log(minutes), for example: log(5 minutes) ≈ 0.5, log(1 hour) ≈ 1.8, log(1 day) ≈ 3.15, log(7 days) ≈ 4

7.3.4. The Performance Gap Revisited

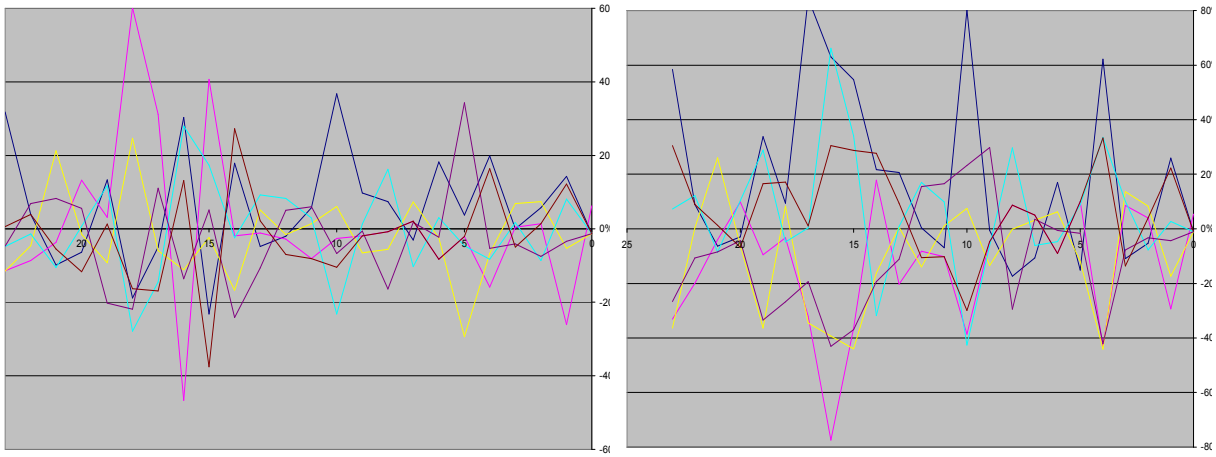
In light of our experience with Maui and Flex, it is required to examine the performance of PBAS on a monthly basis as well. This would show us whether stability has indeed improved, and may again lead to opportunities for further optimization. Figure 10 presents the same information given in Figure 7 – monthly deviations from the average wait time and bounded slowdown in the SDSP2 log – for four PBAS configurations. The chosen configurations are the best-performing representatives of four orders of magnitude of time: five minutes, one hour, twelve hours and a week.

Flex and Maui are included in the graphs to visually demonstrate that they are the most fluctuating schedulers – the PBAS algorithms show considerable variations among months, but generally succeed in avoiding the extremes that Flex and Maui reach. Data from the other logs supports this conclusion.

On the other hand, Figure 10 does show that there are considerable per-month differences between different PBAS configurations. These four schedulers perform at the same overall level, but each one has "strong" and "weak" months. This leads back to the idea we started from: Why not build an adaptive scheduler – that will dynamically switch between different PBAS candidate configurations at runtime? This algorithm is called Indirect Performance-Based Adaptive Scheduling, after the extra level of delegation, and its formal description is identical to that of PBAS (Figure 8). Since both algorithms use the candidate schedulers as black boxes, they are in fact the same algorithm, with a different configuration.

Indirect PBAS was tested on a large variety of configurations, and for the most part, results were disappointing. Most results were random and highly inconsistent across logs. The best configuration that we found was the 7 Days-Utilization Indirect PBAS, which internally used the two top PBAS configurations listed in table 5. Its average performance improvement over Maui was 6%, and the average stability gain was 20% – less than each of the two schedulers it relied on.

Our hypothesis, as for why adaptability doesn't work in this case, is that it requires that the core schedulers being used be different enough from one another. For example, Flex and Maui use a completely different "mindset" when approaching the scheduling problem. On the other hand, different PBAS configurations are much more likely to make the same decisions – for better or worse – since the different metrics are correlated, and high-impact workload phenomena influence over long durations. As a general principle in adaptive algorithm design (Eiben et al., 2003), each candidate must posses some unique added value – special cases with which it can cope well – and we believe that this principle worked here.



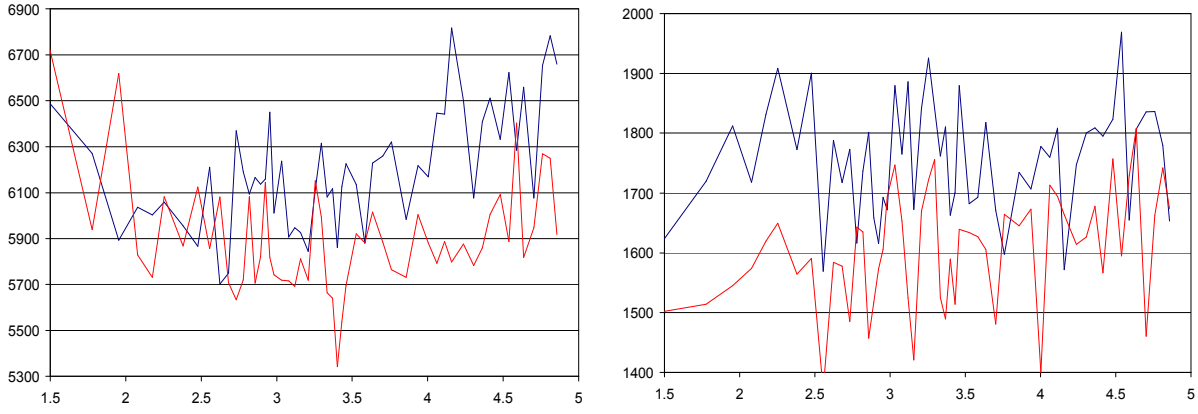
Wait Time *Bounded Slowdown*

Figure 10. Monthly deviations from the average performance in the SDSP2 log, for Flex, Maui and four PBAS configurations listed to the left

7.3.5. Candidate Schedulers

All the PBAS experiments analyzed so far use only Maui and Flex, for a simple reason – using any other candidate set, and in particular the set of all four schedulers, provides significantly worse results. Figure 11 demonstrates this visually, for specific logs and PBAS configurations – but the results repeat for other configurations and logs as well.

The explanation is as follows. Since the adaptive scheduler makes local decisions, there are occasions when Easy or Cons will be chosen. Since these algorithms are generally weaker than Flex and Maui, the probability that their past success indicates future success is smaller than for Flex and Maui. This means that erroneous choices are made more often, and overall performance suffers.



KTH *BLUE*

Figure 11. Comparison of 4-Schedulers PBAS (blue) and 2-Schedulers PBAS (red) results are wait times, switching metric is utilization, switch times are 30 min. to 50 days

Table 12 illustrates this from a different point of view, by listing the portion of time that PBAS uses each scheduler. The five first rows are results for the BLUE log, using utilization as the switching metric, and all four schedulers as the candidate set. As the table shows, Cons and Easy are used together between 21% and 47% of the time, depending on the time frame. This means that during this portion of the time, PBAS uses an inferior scheduler – and as Figure 11 shows, this is apparent in its performance. To borrow an analogy from sports: If an adaptive algorithm is a team's coach, it should prefer to use a strong player on a bad day, than a mediocre player on a winning streak.

To complete the presentation of Performance-Based Adaptive Scheduling, Table 12 also includes the absolute and relative number of switch points in which switches actually happened. Note that the portion of time spent using each scheduler is relatively constant across time scales – evidence that locality is a dominant factor here.

		% Flex	% Maui	% Easy	% Cons	# Taken Switches	% Taken Switches
4 Schedulers	30 min	27%	52%	17%	4%	2295	5%
	4 hours	22%	47%	23%	8%	627	30%
	24 hours	21%	52%	19%	9%	278	28%
	7 days	11%	43%	40%	7%	44	31%
	50 days	17%	44%	33%	6%	10	51%
2 Schedulers	30 min	41%	59%	-	-	1761	4%
	4 hours	37%	63%	-	-	678	12%
	24 hours	36%	64%	-	-	214	22%
	7 days	46%	54%	-	-	32	23%
	50 days	41%	59%	-	-	6	31%

Table 12. Switching Behavior for PBAS with the utilization metric, on the BLUE log

7.4. Workload Based Adaptive Scheduling

7.4.1. Rationale

Performance-Based Adaptive Scheduling takes a greedy approach to profit from the performance gap. The ability to bypass the need to understand the root cause of a phenomenon and still benefit from it is the key advantage of adaptive algorithms – but yet, if such an understanding can be reached, then the rewards in terms of performance are usually high (Eiben et al., 2003). On the other hand, some dynamic environments are too complex to fully "understand", and then an adaptive heuristic behavior is the best practical choice.

Workload-Based Adaptive Scheduling is an intermediate path between these extremes. Instead of blindly following the best-performing candidate scheduler, we will examine which local features of the workload cause a given scheduler to be preferred. Afterwards, our on-line algorithm will work by monitoring that feature, and switching schedulers accordingly. This should enable us to benefit by identifying and reacting to a new trend in the workload as its first jobs enter the queue – instead of when the first jobs terminates, when performance statistics are first available to PBAS.

The workload variables that were tested are the number of jobs and users, the load, and the medians and intervals of the runtime, parallelism, total CPU work, inter-arrival time, and number of jobs of the workload. The average and standard deviation metrics were not used, since they are known to be unreliable in parallel workloads due to the long tails of some of the distributions (Downey and Feitelson, 1999). These variables were compared against two measures of relative algorithmic superiority: the absolute difference between the wait times of Flex and Maui, and the relative difference (in percent) between the wait times of the two algorithms. The first measure tests correlation to "major blunders" since large absolute values dominate the computation, while the second (relative) regards small and large wait times equally.

Log	Jobs Count	Runtime Load	Users /TJobs	Runtime Median	Runtime Interval	Procs Med.	Procs Int.	CPU Med.	CPU Int.	Inter-Arrival Med.	Inter-Arrival Int.
Correlations to relative Flex/Maui difference:											
KTH	0.44	-0.35	0.13	0.83	-0.38	-0.05	-0.60	N/A	N/A	-0.46	0.64
SDSP2	0.01	0.11	-0.05	-0.04	0.28	-0.30	-0.19	-0.02	0.29	0.02	-0.11
BLUE	0.20	-0.05	-0.09	0.14	-0.02	-0.33	-0.24	0.00	0.09	-0.21	-0.13
Correlations to absolute Flex-Maui difference:											
KTH	0.39	-0.03	0.11	0.32	-0.28	-0.14	-0.72	N/A	N/A	-0.52	-0.14
SDSP2	0.10	0.08	-0.16	-0.24	0.16	-0.19	-0.21	-0.16	0.19	-0.11	-0.28
BLUE	-0.01	-0.12	0.07	0.02	0.13	-0.09	-0.02	0.01	0.17	-0.11	0.08

Table 13. Correlations between workload attributes and relative Flex-Maui performance

In Table 13, positive correlations mean that large values of that workload variable indicate better performance of Maui over Flex. Negative values indicate that large values correlate with better Flex performance. The correlations were computed between the statistics of each month of a log, to the difference between Flex and Maui's performance for that month. As the table shows, most variables do not consistently indicate a preference towards a certain algorithm, and this dataset is too small to assume that a majority means anything. There are two exceptions: the number of processors (both median and interval) for both the absolute and relative performance measures, and the median of inter-arrival times for the absolute measure only.

7.4.2. Algorithm

The Workload-Based Adaptive Scheduler (WBAS) uses the number of processors as an indicator of scheduler preference. It was chosen because it works for both the relative and absolute measures, and because its correlations were the most consistent across logs.

Workload-Based Adaptive Scheduler Algorithm

- *Initialization*
  - Select one of the candidate schedulers arbitrarily as the *active* scheduler.
- *On Job Arrival Event*
  - Update the moving average of parallelism AP using the number of processors requested by the new job P_{new} and this formula:

$$AP \leftarrow (1 - MAF^{-1}) \text{ x } AP + MAF^{-1} \text{ x } P_{new}$$

  - Start running only the jobs that the active scheduler decided to run.
- *On Job Termination or Cancellation Event*
  - Only start running jobs the active scheduler decided to run.
- *On Switch Schedulers Event*
  - Measure the average level of parallelism of jobs that arrived during the last time frame: AP_{last}.
  - If $AP < AP_{last}$, then activate Flex, else activate Maui.
  - If the active scheduler changed, synchronize it to the current situation.
  - Set another 'switch' event to happen after another time interval.

Figure 12. Workload-Based Adaptive Scheduler Algorithm

The algorithm is adaptive and works in the following manner. When a switching event occurs, it checks if the number of processors requested in the last time frame is greater than the long-term average parallelism of the workload – which indicates that Flex should be activated – or not, which means that Maui should be chosen. The long-term average parallelism of the workload is updated each time a job arrives, using an exponential moving average, to enable the global average to slowly adjust even after the scheduler has been running for a long time.

7.4.3. Empirical Results

The WBAS can be configured by two parameters: The switching time frame, and the moving average factor (MAF) which determines the length of the "long-term memory" of the algorithm. The candidate set and history time frame, in the sense used in PBAS, are irrelevant here. Since both

parameters are continuous, we could not test all possible combinations in order to find the best one, and therefore took the following approach to find the best combination. In Figure 13, all the performance-per-switch-time graphs use MAF=5000, and all the performance-per-MAF graphs use one day as the switch time. These values were chosen after preliminary test runs, which indicated that these values both perform well and are relatively stable to small changes. The performance measure in Figure 13 is wait time in all graphs; the scales for both MAF and switch times are logarithmic, since as usual we tested values from several orders of magnitude.

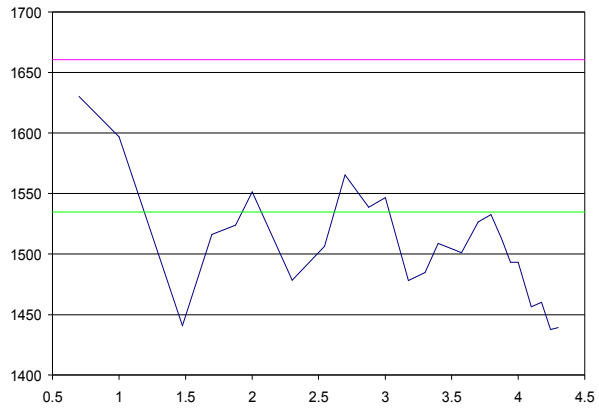
The first question to answer, before analyzing patterns in the above graphs, is whether WBAS improves overall performance. The answer is a clear yes, and table 8 shows the best configurations that do so. As was done in table 5, the results are the percentage of improvement compared to Maui. The performance gain is measured by the average wait time, and the stability gain is measured by the standard deviation of monthly wait times.

The average performance gain of the first two combinations is 10%, and their average stability gains are 35% and 22%. The third combination's average improvement is 9%, and its average stability gain is 33%. These averages are similar to the improvements achieved using PBAS, with the distinction that PBAS achieved much better gains for the BLUE log, while WBAS achieved better gains for the KTH log. The specific difference between the KTH and BLUE logs that is responsible for this result is unknown at this time.

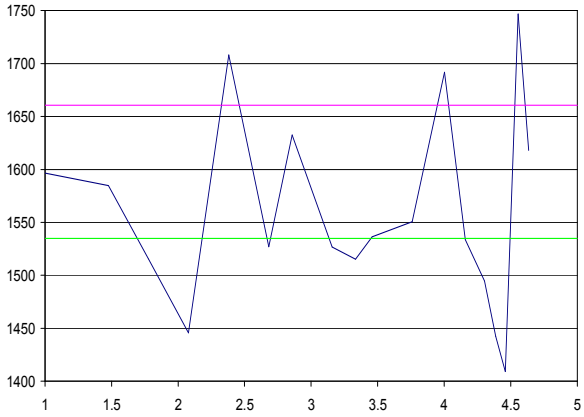
Another difference between PBAS and WBAS, made obvious by visually comparing Figure 9 and Figure 13, is that the performance of WBAS is less sensitive to parameter changes. This is not to say that the WBAS results are "well-behaved": we still cannot see continuous and predictable lines, and the best way to know the results for a given parameter combination is to run it in a simulation. But there is still an improvement compared to PBAS: For example, changing a MAF of 5000 by several hundreds has a moderate effect, and performance changes are usually visible only when crossing the borders to a higher or lower order of magnitude. The switching time frame is more sensitive than that, but we can still observe some common features in the three logs' graphs.

	Performance Gain			Stability Gain		
	KTH	SDSP2	BLUE	KTH	SDSP2	BLUE
WBAS 7500 MAF 1 Day	20%	10%	1%	60%	32%	14%
WBAS 7500 MAF 2 Hours	15%	7%	8%	28%	22%	17%
WBAS 750 MAF 1 Day	17%	11%	0%	62%	31%	5%

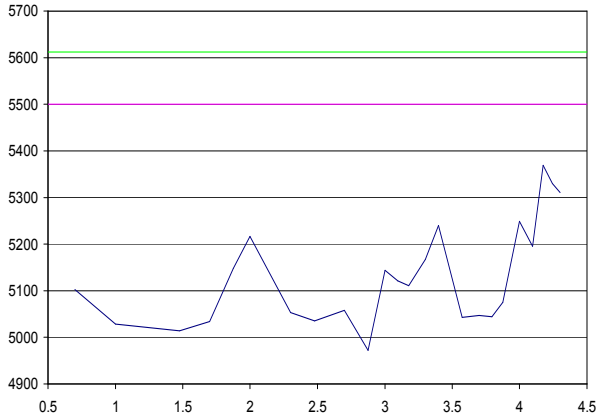
Table 14. Best parameter combination for WBAS, and overall improvement over Maui



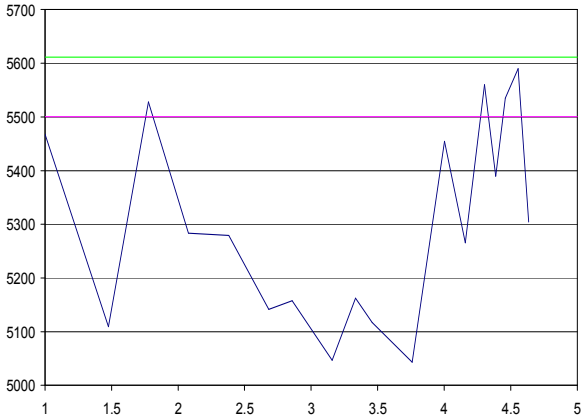
BLUE, by MAF



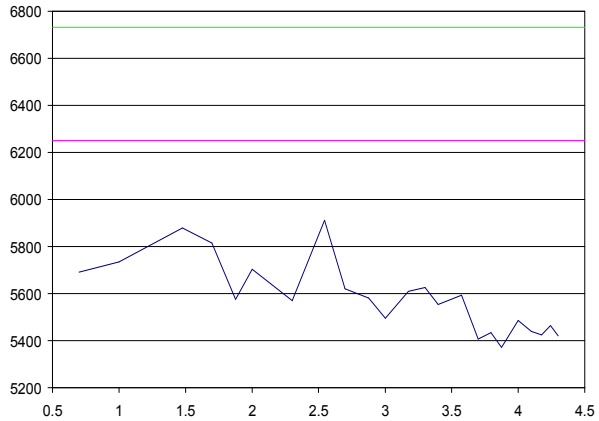
BLUE, by Switch Time



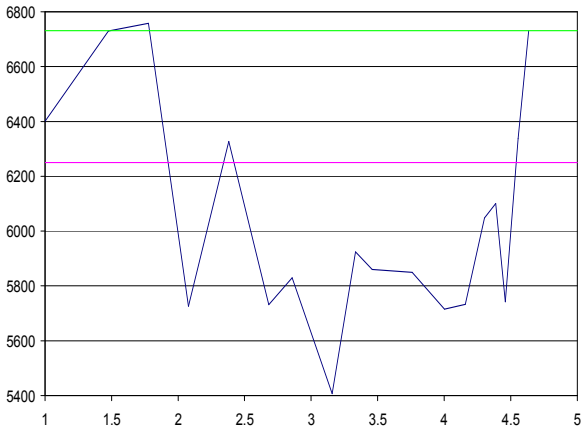
SDSP2, by MAF



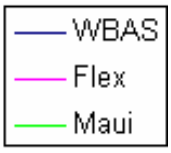
SDSP2, by Switch Time



KTH, by MAF



KTH, by Switch Time



X axis on left is **log-MAF**

X axis on right is **log-Minutes**

Y Axis on both is **Wait Time**

Figure 13. Workload-Based Adaptive Backfilling by Time Frame and MAF

A month-by-month analysis of WBAS reconfirms the conclusions that WBAS contributes to stability, and that its parameters are more robust to change than in PBAS. Parameter combinations that result in very different per-month results can be demonstrated, but as discussed in section 4.3.4, this is not a real opportunity for further improvements.

Since both PBAS and WBAS show the same average overall performance, the selection between them can be based on secondary considerations. Most of these are in favor of WBAS: Its parameters are more robust to minor errors, it is simpler to implement since it doesn't require simulating both schedulers in memory at runtime, and it is slightly more efficient since only one scheduler needs to run at any given time.

A final word is due about the algorithms' performance. The runtime of adaptive algorithms is the sum of runtimes of the candidate algorithms, plus the time required to synchronize between them. This means that the performance of an adaptive scheduler is the same, in the Big-O sense, as that of its slowest candidate. In practice, both algorithms run an entire log simulation in several dozen seconds on a strong PC, which equals millisecond-range time per scheduling event. Moreover, switching events are done asynchronous to user requests, so users experience the same service times that Flex and Maui provide alone.

7.5. Summary

This study makes three contributions. First, it identifies a practical problem – the performance gap. The problem is shared by all the logs we examined, has a dire affect on a system's usability, and cannot be gracefully handled once it starts. Since our study focuses on the most widely used schedulers today, deployed on several platforms, we suspect that this problem is widespread and often ignored.

Second, we propose a practical solution, in the form of adaptability. The combination of the different approaches of Maui and Flex, with the recommended parameter sets that we found, significantly improves the stability and predictability of the tested systems. The average improvement for our dataset was 35% using the recommended configurations.

Moreover, the new algorithms enable a 10% gain in overall performance over existing algorithms, which makes them useful for any large parallel computer. Compared to other alternatives for boosting performance, improving the scheduler is easy: It is a software-only, relatively independent module of the operating system; implementing the algorithms described here requires writing several thousand lines of code.

What adaptive algorithms don't explain is why they do or don't work. Our results provide many hints, but the overall picture is still a complex and highly chaotic one. Future research is required to further investigate this issue, which will require an understanding of the temporal structure of parallel workloads, the dynamics of backfilling and the specific candidate schedulers used here, and the dynamics of the switching decisions of the adaptive scheduler.

The third contribution of this study is by showing that locality of sampling, the daily cycle and the weekly cycle are not just statistical features – they are of practical importance. Further, it is clear that models which do not exhibit these phenomena cannot be used to evaluate such schedulers: They are likely to provide answers that will contradict what will really happen. This provides motivation for the upcoming modeling effort of these features.

8. Runtime Predictors for Backfilling Schedulers

This section presents joint work with Dan Tsafrir and Zviki Goldberg.

8.1. Introduction

The scheduler is implemented in most systems today using FCFS with backfilling (Etsion and Tsafrir, 2005). Recent findings about backfilling (Tsafrir and Feitelson, 2006b) suggest that past work, showing that inaccurate runtime estimates improve its performance, have actually traded off fairness in exchange for this improvement. In addition, a Shortest-Job-Backfill-First strategy can achieve superior performance when equipped with an accurate runtime prediction algorithm, while maintaining the originally intended fairness criteria (Tsafrir et al., 2006).

This scheduler's effectiveness relies on providing it with accurate runtime predictions for incoming jobs. This motivates two research goals: To find new prediction algorithms that outperform current ones, both in accuracy and in the resulting bottom-line performance; and to gain new understandings about the value of different strategies and types of information in the context of parallel runtime prediction.

To achieve these goals, we define a framework for comparing on-line runtime prediction algorithms, including metrics for accuracy and for isolating the predictor's effect on performance. We apply our framework to existing predictors, and then define and analyze a set of new prediction algorithms, comparing a large variety of potential strategies: Session-based versus user-based partitioning of history, recent versus far history, exact matching of job attributes, and use of the user

estimate. This provides a quantifiable view of the value of information of each such concept in the context of runtime prediction.

Recommended predictors, in terms of both performance and accuracy, are given for the general case, the case of unavailable user estimate, and the case of no available information. The two later cases are likely to be encountered in grid environments, where jobs migrating between different sites do not generally travel with their entire user's history.

The most successful runtime predictors rely heavily on the user who submitted each job and on locality of sampling – specifically within the list of jobs of a single user. In addition, we investigate the effectiveness of directly taking advantage of the concept of sessions: Can we benefit by identifying a set of jobs as belonging to the same session, and assuming that the user's next job within the same session is likely to be similar?

8.2. The Need for Accurate User Estimates

8.2.1. Accuracy of User Estimates

When submitting a job to any backfilling parallel scheduler, users are required to submit an estimate of how long the job will run. This estimate is in fact an upper bound, since jobs that exceed it are killed by the system. On the other hand, users also have an incentive not to overestimate, since this undermines a job's chance to backfill (and run much sooner than originally queued), and may also cost the user more 'CPU seconds' under some accounting systems.

The popularity of the EASY scheduler has enabled empirical studies of how it works in practice, based on accounting logs from multiple installations (Feitelson, 1999). These studies showed that user estimates are generally inaccurate (Mu'alem and Feitelson, 2001), and are reproduced in Figure 14. There seem to be three types of jobs: Jobs that used only a fraction of their estimate, and presumably failed on startup (20-30% of jobs, shown in green); jobs that exceeded their runtime estimates and were killed by the system (10-20% of jobs, shown in red), and jobs that terminated regularly (in grey), for which the histogram is quite flat – meaning that for such jobs, any level of accuracy is almost equally likely to happen.

The conclusion is that user estimates are actually rather poor, probably since users find the motivation to overestimate, so that jobs will not be killed, much stronger than the motivation to provide accurate estimates and potentially enable the scheduler to perform better packing. However, a recent study indicates that users are actually quite confident of their estimates, and most probably would not be able to provide much better estimates (Lee et al., 2004).

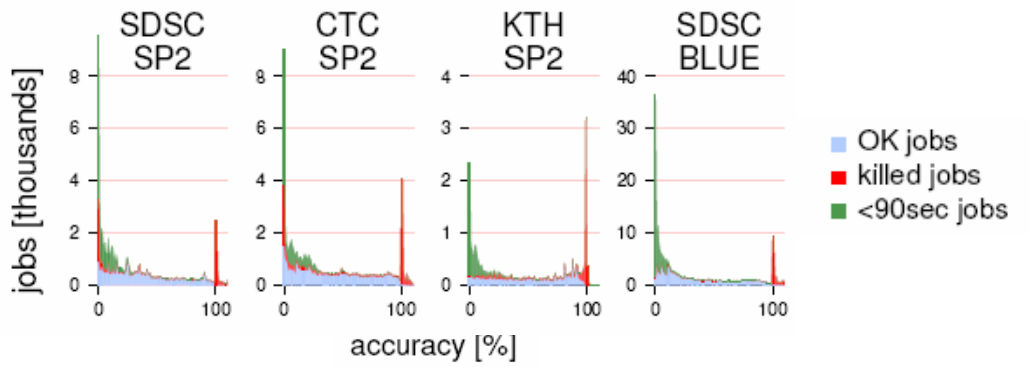


Figure 14. Relative Accuracy of User Estimates

8.2.2. Using Accuracy to Improve Performance

The search for accurate runtime prediction of parallel jobs attracted little attention in recent years, because of two reasons: Findings which suggested that accuracy hinders performance, instead of improving it; and lack of other needs for accuracy. Both of these reasons have been recently refuted.

First, using the EASY scheduler produced a surprising yet consistent result: Backfilling schedulers actually perform better with less accurate estimates, in particular large over-estimations. Several studies (Zotkin and Keleher, 1999; Mu'alem and Feitelson, 2001) suggested *doubling* the user estimates, and investigated the effect of multiplying it by higher factors as well. Some of these results are reproduced in Figure 15, and show that indeed doubling the estimates improves overall performance. This effect is observed even when the actual runtimes are used as estimates (perfect accuracy), allegedly showing that accuracy has a marginal role in improving performance.

These results are in contrast to the initial intuition, that more accurate predictions should lead to better packing opportunities, and thus to better utilization and performance. However, as a recent study (Tsafrir and Feitelson, 2006) discovered, this intuition is not faulty. Instead, the cause of this

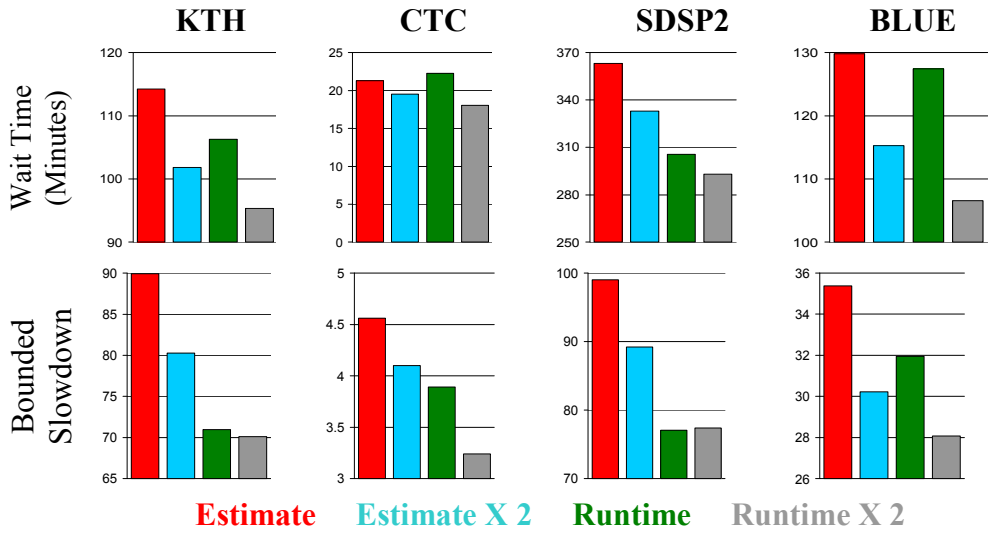


Figure 15. Effect of Doubling Estimates on Performance

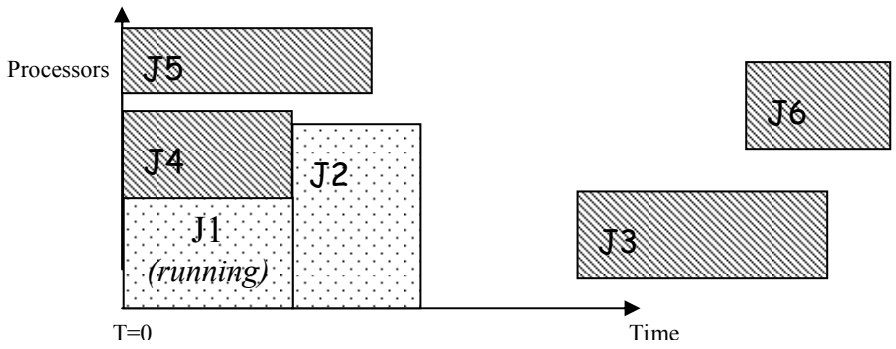


Figure 16. EASY Backfilling Example Revisited

behavior is a much stronger trend, which affects in the opposite direction: Exchanging performance for fairness, by weakening the reservation guarantee for the first job, and thus allowing more short jobs to be backfilled.

This happens as follows. The only effect of the estimate is on the computation of the shadow time and its use. Therefore, if a running job will actually terminate within one hour, but has an estimated termination time of two hours, then the reservation guarantee for the first waiting job is two hours away. This means that short jobs (ones that are estimated to terminate within two hours) can be backfilled, and start running. This improves utilization and response time, but at the direct expense of the first waiting job. This is because even through the first job will terminate ahead of its estimated time, the waiting job may not be able to start running then, because of the backfilled jobs. This happens because once a job starts running, it cannot be preempted or killed by the scheduler.

A visual example is given in Figure 16, which is a continuation of Figure 1, after J4 and J5 have been backfilled. Assume that all estimates have been doubled. A few seconds later, J1 terminates – way ahead of its estimate, which has been doubled. Because of J4, J2 can't exploit this early termination and start running: It cannot kill or preempt it. Note that J3 still can't start running as well – its estimate has been doubled as well – but new, short jobs can and will be backfilled. For example, a job such as J6, whose doubled estimated is less than that of J4, could start running. Since J4 will terminate ahead of time as well (after all, it is reserved twice the upper bound of its runtime), J2 could later be delayed by J6, and later by other shorter jobs as well. Figure 17 illustrates the outcome of this process in the logs: When doubled estimates are used, a larger fraction of jobs are backfilled.

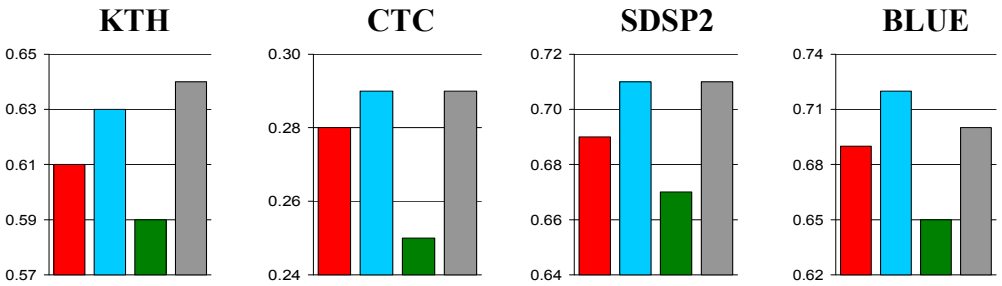


Figure 17. Percent of Backfilled Jobs under the Estimation Methods from Figure 15

To conclude, the real tradeoff of doubling estimates is not performance versus accuracy, but performance versus fairness. This is very similar (in both concept and empirical performance) to the Flex scheduler (Talby and Feitelson, 1999), which introduces the notion of reservation slack explicitly. This recent understanding also led to a scheduling algorithm (Tsafrir and Feitelson, 2006b) that improves performance and accuracy but maintains the fairness guarantees inspired by EASY. This algorithm, to be presented in the next section, improves in direct relation with the quality of runtime predictions, thus raising the importance of this research direction.

8.2.3. Other Needs for Accuracy

In addition to performance, two other recent developments raise the requirement for accurate runtime predictions of parallel jobs. The first is advanced reservation for grid co-allocation, shown to considerably benefit from better accuracy (Smith et al., 2000). The second is batch scheduling of moldable jobs – Jobs that can run on any number of processors (Downey, 1997c; Smith et al., 1999). Since the goal of the scheduler is to minimize response time, it must decide whether it is preferable to start running a job on the processors that are available now, or wait until more processors are available. A good prediction of how long current jobs are expected to run is obviously crucial to making the correct decisions.

8.3. Prediction Algorithms Comparison Framework

8.3.1. Predictors Defined

This section is devoted to comparing a set of runtime prediction algorithms, and analyzing the root causes for their empirical performance and accuracy. We thus need a framework for comparing prediction algorithms: The exact scope of a prediction algorithm, how to measure accuracy and how to measure performance are all non-trivial issues. This section defines them.

To begin with, we need a standard definition for a prediction algorithm. A prediction algorithm is an event-based program, used internally by a scheduler. It is activated by the scheduler in four events, specified in Listing 1, and in each of these events, the algorithm may return a list of updated predictions for waiting or running jobs. The only constraints are that a prediction must be given for a job on its arrival; must be given when a job has missed its deadline (meaning that it is still running, and its runtime is greater than its current prediction); and must not be smaller than the job's actual runtime, if it already began to run.

```
struct Prediction { Job job, int prediction }
list<Prediction> OnJobArrival ( Job job )
list<Prediction> OnJobStart ( Job job )
list<Prediction> OnJobTermination ( Job job )
list<Prediction> OnJobDeadlineMissed ( Job job )
```

Listing 1. A Predictor's Interface.

8.3.2. Measuring Accuracy

Defining and measuring the accuracy of predictions is non-trivial. First, since a prediction may be lower than a job's actual runtime, it may have to be changed during its execution – so one job can have several different predictions during its lifetime. Second, since most prediction algorithms rely on history, they must rely on the list of terminated and running jobs – which means that their accuracy also depends on the scheduler being used, in addition to the workload. Third, different metrics may be sensitive to different aspects of accuracy, as is the case in performance metrics. For example, the wait time and bounded slowdown metrics don't always agree, since wait time is dominated by long jobs, while slowdown is more sensitive to changes to short jobs. This is why more than one metric is required in many situations.

We define and measure two accuracy metrics. The first is Absolute Error, defined as the absolute difference between the prediction and the actual runtime. It is analogous to the wait time metric used to measure performance, and similarly desired to be as small as possible. The second is the Relative Accuracy metric, defined as the relation between the prediction and the actual runtime, where the smaller of them is the numerator. It is inspired by the slowdown metric, and is always between 0% and 100%. When analyzing a full log we will consider the averages of these metrics, meaning that Absolute Accuracy will be dominated by large prediction mistakes in long jobs, while Relative Accuracy gives similar weight to short and long jobs. Note that a bounded variant of Relative Accuracy is not needed, since its denominator is always greater than the numerator. Equation 11 defines the two metrics; R stands for Runtime and P for Prediction.

$$AA = |R - P|$$

Equation 11a. Absolute Accuracy.

$$RA = \begin{cases} 1 & if \quad R = P \\ R/P & if \quad R < P \\ P/R & if \quad R > P \end{cases}$$

Equation 11b. Relative Accuracy.

These definitions only work for jobs that have a single prediction throughout their lifetime. In the frequent case of under-predictions (and in general in other cases as well), a job's prediction may have to be changed several times during its lifetime. For such cases, we define its accuracy (of both kinds) to be the weighted average of its accuracies, where the weights are given by durations in which each prediction was active. Formally, if T0 and TN are a job's submission and termination time, and we denote by Ai its accuracy (again, either absolute or relative) from time Ti-1 to Ti (where Ti-1 ≤ Ti), then its accuracy is:

$$A = \frac{1}{(T_N - T_0)} \cdot \sum_{i=1}^{N} A_i \cdot (T_i - T_{i-1})$$

Equation 12. Accuracy (Absolute or Relative) Under Multiple Active Predictions.

8.3.3. Measuring Performance: The SJBF Scheduler

The effect of a prediction algorithm on system performance is derived from and highly dependant on the scheduler which activates the predictor. This chosen scheduler should satisfy all of the following criteria:

1. More accurate predictions should result in higher performance (as measured by wait time and bounded slowdown). This means that EASY is not an option – as shown in section 8.2.2, more accurate predictions *degrade* EASY's performance.
2. The scheduler should provide the level of fairness required for the original EASY, or better. This means that Shortest-Job-First (SJF), which may lead to starvation, is not a legitimate option, although its performance improves with accuracy.
3. The scheduler should perform better than EASY, when a good predictor is used. The performance gain comes in addition to the improved accuracy and fairness.
4. The scheduler should be practical – easy to implement and integrate in current systems.

The scheduler which we will use in this section, which satisfies all of the above criteria, is the Shortest Job Backfill First (SJBF) algorithm proposed by (Tsafrir et al. 2006). The algorithm is identical to EASY except for three places:

1. Use predictions instead of user estimates to compute the shadow time.
2. Use predictions instead of estimates to test if a job terminates before the shadow time.
3. Backfill jobs in order of ascending predictions (shortest job first), instead of ascending arrival time (first come first serve).

The algorithm's pseudo-code is given in Appendix A, where the three words that need to be replaced in EASY to turn it into SJBF are underlined. The SJBF algorithm has been shown to improve performance with improved accuracy, and do so without sacrificing fairness. It is also very easy to implement in practice, particularly in existing EASY or Maui based installations.

The scheduler's improved performance stems from prioritizing shorter jobs. Most of the studies dealing with predictions and accuracy indicate that improved performance due to increased accuracy is most evident when shorter jobs are favored (Smith et al., 1999; Zotkin and Keleher, 1999; Chiang et al., 2002; Tsarir and Feitelson 2006b). Again, this means that the problem with applying improved predictions to improve performance is not with the accuracy of predictions, but with the scheduler which uses them. As explained in section 8.2.2, this is also the real reason behind the performance boost caused by doubled estimation: more short jobs are able to backfill, at the expense of longer ones. The SJBF algorithm does this explicitly. Thus, as the empirical results will show in later sections, it is able to enjoy similar performance benefits.

On the other hand, SJBF maintains and even improves the fairness guarantees defined by the original EASY. Note that the algorithm only changes the order in which the backfilling optimization is attempted: Jobs are still scheduled by FCFS before activating backfilling, and the reservation is still made for the oldest waiting jobs (not the shortest one). In addition, SJBF does not reduce the reservation guarantees for the first job by artificially raising the estimate of running jobs – as done by EASY with doubled estimates. In contrast, since the algorithm works with accurate predictions, it actually provides *tighter* reservation guarantees than the original EASY. This happens because, as the empirical results will show, most of the analyzed predictors are more accurate than user estimates.

The dataset used in this study consists of the following four logs from Table 1: SDSP2, CTC, KTH and BLUE. Together, these logs constitute over 400,000 jobs and six years of real user activity, from three different sites. These four logs have been chosen since they have all been extensively cleaned and studied before, and in particular include user estimates which are widely considered to be representative and of good quality (Tsafrir et al., 2005).

8.4. Simple Prediction Algorithms

Now that our framework for comparing predictors is in place, we can explore the search space for prediction algorithms, starting with three very simple predictors. The first is the perfect predictor, which guesses the actual runtime. This is a theoretic predictor, since the actual runtime is not known in advance, but will provide a reference point for what can be reached with full information. The second predictor is the constant predictor: predict the same, constant runtime for all jobs. Again, this predictor is only meant as a reference to what can be achieved with no information. The third predictor is the estimate predictor: use the user estimate as the predictor. Listing 2 specifies these simple predictors formally.

PERFECT Predictor:	OnJobArrival (Job job):	return <job, job.runtime>
CONSTANT Predictor:	OnJobArrival (Job job):	return <job, *Constant*>
ESTIMATE Predictor:	OnJobArrival (Job job):	return <job, job.estimate>

Listing 2. Reference Predictors

Note that SJBF with the Estimate Predictor is different than the original EASY/Maui, in two aspects. First, it backfills jobs by order of ascending estimates, instead of ascending arrival time (FCFS). Second, it updates its prediction in deadlines misses – events in which the job's runtime exceeds its prediction. This can happen if a job's runtime exceeds its estimate (which happens in practice in rare cases), and is handled, as suggested in (Tsafrir et al., 2006), by gradually increasing the prediction by predefined increasing values. This is also the strategy used in the Constant Predictor, if the runtime exceeds the constant. All other predictors in this paper use the following strategy: If a deadline is missed and the current prediction is smaller than the user estimate, then raise it to be equal to the user estimate; else, raise it by the predefined gradual increments.

The fourth predictor we will compare ourselves to is the Recent User History Predictor, presented in (Tsafrir et al., 2006) and named EASY++ there. To the best of our knowledge, it is the highest performing scheduler/predictor combination to date abiding our fairness criteria. It is based on SJBF as well, as predicts runtimes based on a very simple rule: A job's prediction is the median of the runtimes of the three last jobs of the same user.

```
OnJobArrival (Job job):
    if there exist at least three terminated jobs of job's user then
            return [job, median of last three terminated jobs of job's user
    else
            return [job, job.estimate];
```

Listing 3. Recent User History (RUH) Predictor

Predictor:	Improvement over EASY, Average over all logs:			
	Wait Time	B.Slowdown	Abs. Error	Rel. Acc.
Perfect	22%	47%	100%	176%
Estimate	11%	22%	0%	0%
Constant 1 second	16%	13%	41%	37%
[1] RUH without Propagation	18%	32%	40%	69%
[1] RUH with Propagation	17%	32%	41%	71%

Table 15. Simple and RUH Prediction Algorithms and their Improvement over EASY

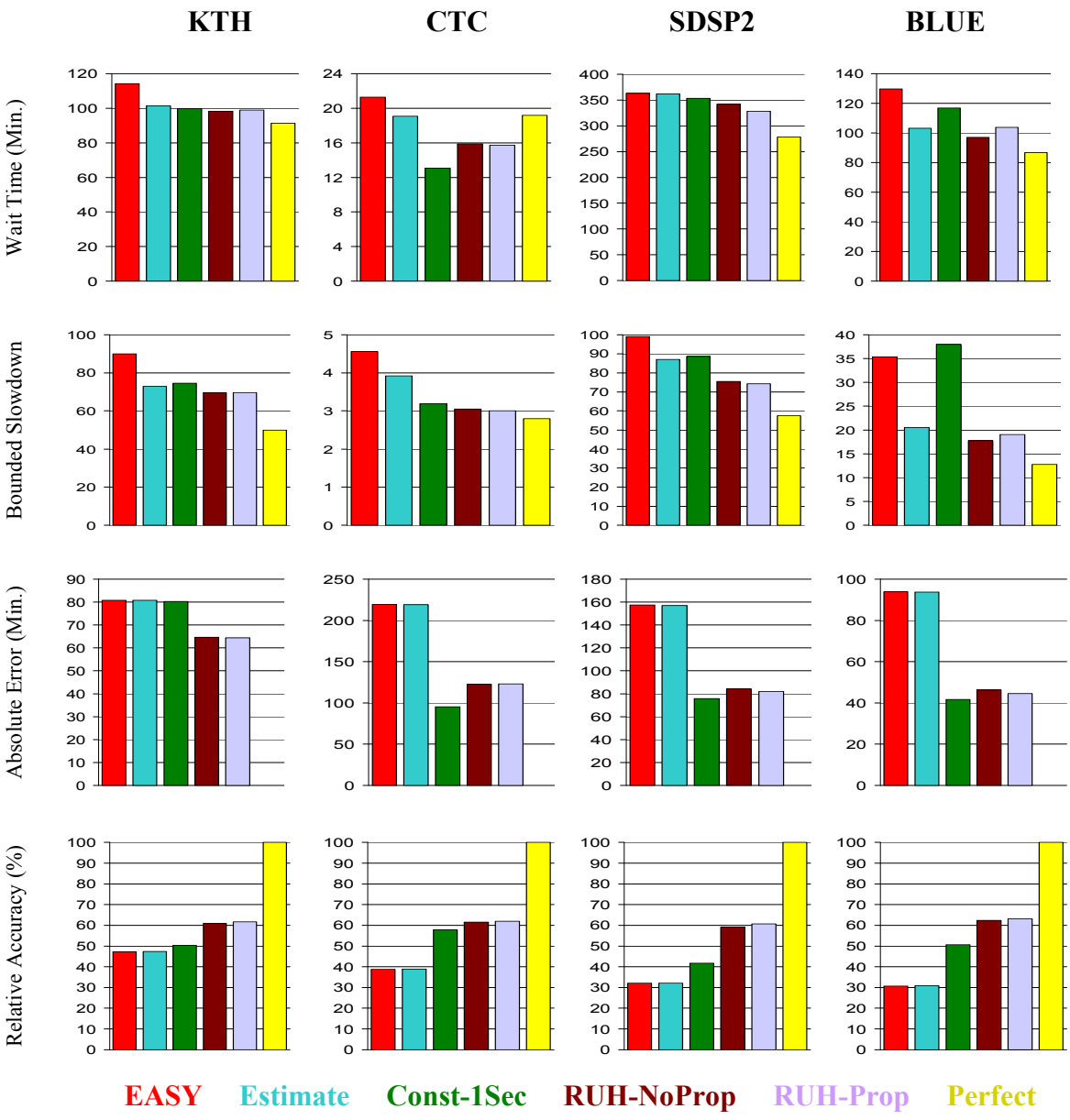


Figure 18. Performance and Accuracy Comparison of Reference Predictors

[1] This comparison was made against the best RUH configuration known at the time of this study. An improved version was later developed and reported in (Tsafrir et al., 2006).

The RUH Predictor can be configured by several parameters, underlined in the above pseudo-code: how many past jobs to consider, which metric to use on them, should running or only terminated jobs be considered, and how to handle the first jobs of a new user. We have simulated it under its best published parameter configuration[1], in two variants – with and without propagation. Propagation is an optional optimization, which forwards new information on a job – specifically, its actual runtime when it terminates or its updated prediction on a deadline miss event – to the rest of the waiting and running jobs of the same user. The predictions of these jobs are recomputed, based on the updated information.

Table 15 and Figure 18 compare the above predictors. As expected, EASY is substantially inferior to SJBF, even under the Estimate predictor. RUH behaves almost the same with and without propagation, and improves both performance and accuracy further, up to about two thirds of the potential gain, as assessed by the results of the Perfect predictor.

The Constant predictor, using a constant of one second (higher values produce far worse results), provides a very surprising result. Although it uses no information to make predictions – not the job's user estimate, not past jobs, not other attributes of the job – it succeeds to match the performance of RUH in the wait time and absolute error metrics, and significantly outperforms EASY in the two other metrics. Note that the accuracy is achieved here by first guessing one second, and gradually increasing the prediction as the job runs longer. The initial low guess enables to backfill any job, regardless of its attributes – and the algorithm indeed backfills on average 22% more jobs than EASY. However, this is not done at the expense of long jobs (since all jobs have an equal chance to backfill). Also, the gradual ascent in prediction during runtime maintains a small gap between a job's runtime and its reserved shadow time, thus maintaining a high level of fairness as well.

8.5. Session-Based Prediction

8.5.1. Rationale

It is well known that human users typically work in sessions – periods of intense, repetitive work. A recent study (Zilber, Amit and Talby, 2005 – see Section 9 below) formalized this notion, and identified five stable clusters of sessions in parallel workloads. It was also found that in four of the session clusters, consisting of over 95% of the observed sessions, the diversity between jobs in the session is very small – a median of less than two unique runtimes and levels of parallelisms within a session. This explains why the Recent User History Predictor is successful, but also suggests that basing a predictor explicitly on sessions may be even better.

CTC, 20 Sep 1996, User #289					BLUE, 21 Apr 2001, User #315				
Arrival	Procs	Estimate	Exec #	Runtime	Arrival	Procs	Estimate	Exec #	Runtime
19:49:20	16	300	3012	91	04:29:56	8	30600	N/A	31
19:52:32	12	300	3012	67	04:30:14	8	30600	N/A	28
19:57:45	12	300	3012	348	04:33:16	8	30600	N/A	31
19:58:33	16	300	3012	342	04:36:26	8	30600	N/A	28
20:05:25	16	300	3012	105	04:37:09	8	30600	N/A	43
20:08:17	16	300	3012	87	04:40:33	8	30600	N/A	33
20:10:07	200	300	3032	332	04:43:29	8	30600	N/A	29
20:15:54	100	300	3033	314	04:46:34	8	30600	N/A	29
20:22:15	200	300	3033	389	04:49:40	8	30600	N/A	28
20:31:34	200	1800	3033	168	04:54:37	8	30600	N/A	28
20:31:45	200	1800	3033	352	04:58:04	8	30600	N/A	30
20:33:25	200	1800	3033	348	05:06:57	8	30600	N/A	28

Table 16. Two Sample Sessions

Table 16 shows two stereotypical sessions. On the right is a session from the BLUE log, in which the user repeatedly ran the same job; note how runtimes are easy to predict based on history, but on the other hand have no correlation to the user estimate. On the left is a session from the CTC log, which shows how sometimes the user hints about changes in runtime by changing the number of processors, the estimate or the executable. Based on these observations, we designed the session-based predictor to use both proximity in time and similarity in job attributes to decide on which jobs a new prediction should be based.

8.5.2. Algorithm

The Session-Based History (SBH) Predictor works as follows. It maintains each user's past jobs partitioned by sessions, where two jobs are defined to be in the same session if the think time between them (the time between the termination of the first one and the arrival of the next) is smaller than twenty minutes. This threshold is taken from (Zilber, Amit and Talby, 2005), where sensitivity analysis showed that it is stable to changes within the same order of magnitude. We have ran simulations using different thresholds, including ones far different than 20 minutes, and concluded that other values sometimes show comparable performance, but cannot be used to obtain consistent superior performance or accuracy. These results will not be presented here due to lack of space. All simulation results presented here use the 20-minutes value.

In addition, the SBH Predictor requires an ordered list of similarity criteria, by which jobs in a session are matched to the job for which a prediction is required. Each criterion defines whether the number of processors (P), the user estimate (E) and the executable (X) should match; the algorithm only uses jobs that match the given criterion to generate a prediction. For example, if the criteria list is [PEX,PX,EX], then the algorithm will first look only for jobs that match the new job

```
OnJobArrival (Job job):
    for each criterion in the similarity criteria list
        for each of the last three sessions of job's user, by descending start time
            if there exists at least one terminated job in the session,
                which matches the current similarity criterion, then
                    return <job, median of all jobs in session that match criterion>
    // (the following line is only reached if no matching job was found)
      return <job, job.estimate>
```

Listing 4. Session Based History (SBH) Predictor

in parallelism, estimate and executable; if there is no such job in the current session, it will look for jobs that match in parallelism and executable; and if there is no such job as well, it will look for jobs that match in estimate and executable. If no matching job is found at all, then the algorithm repeats the search in the previous sessions, in descending order. If no matching job is found in any session (for example, in the first job of a new user, or when a user starts working with a new executable), the algorithm resorts to using the user estimate as the prediction.

The algorithm can be configured in several ways, which are underlined in the above pseudo-code. We found two parameters to be the most influential. The first is the similarity criteria list, since it defines the balance between predicting by exact matches, and not being able to predict at all (when the criteria are too strict). The second is the order of the algorithm's two loops. The algorithm as explained above and defined in Listing 4 uses "Depth-First Search" (DFS): Its first priority is to find an exact match to the current criterion, at the expense of relying on the further past. For example, if the setting is [PEX,PX,EX], then this algorithm will prefer to predict based on a job that matches in parallelism, estimate and executable three sessions ago, then predict based on a job that matches only in parallelism and executable, from the current session. An alternative strategy would be "Breadth-First Search", in which the two loops are exchanged, and the algorithm first looks at the current session for any match to the similarity criteria list, and only search past sessions if no match to any criterion is found.

8.5.3. Performance and Accuracy

The SBH Predictor can be configured in a vast number of ways, but most parameters and combinations have a marginal effect. Table 17 and Figure 19 compare the performance and accuracy of RUH with the two best SBH configurations we have found.

Configuration:	Improvement over RUH, Average over all logs:			
	Wait Time	B.Slowdown	Abs. Error	Rel. Acc.
Unlimited DFS, [PE,P,E,*], Propagation	5%	4%	5%	2%
Unlimited DFS, [E,P,X], Propagation	4%	8%	9%	3%

Table 17. Best SBH Configurations and Their Performance Gain over RUH

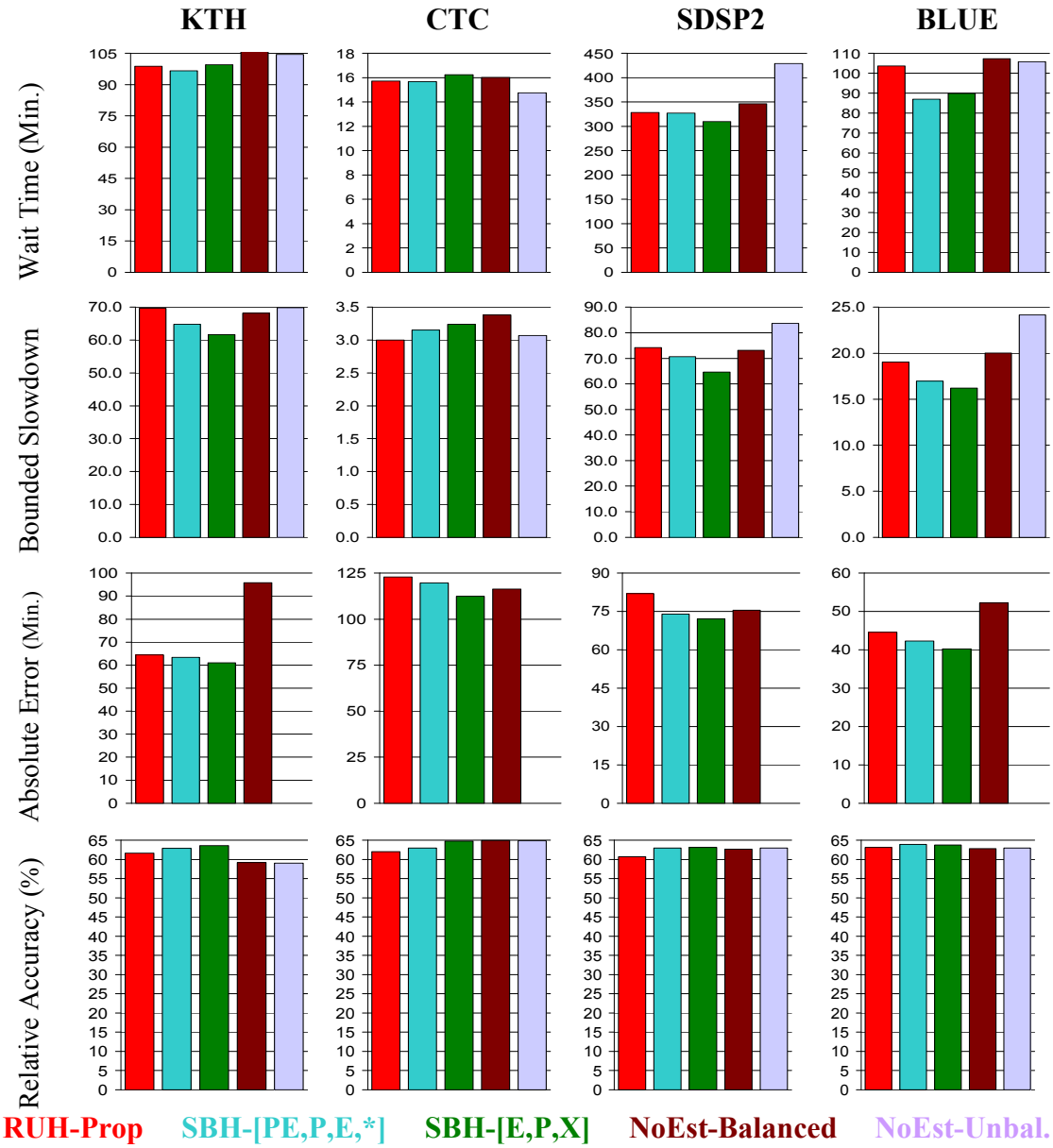


Figure 19. Performance and Accuracy Comparison of RUH, SBH and No-Estimate Predictors

After thousands of simulations, we have reached the following conclusions:

- *Depth-First Search is significantly better than Breadth-First Search*. Over a large set of simulations, DFS performed better than the equivalent BFS configuration in over 90% of the cases. The average gap in favor of DFS, over all four logs, was 5% in average wait time and 17% in average absolute error. Note that this happens even though we use DFS to unlimited depth (not just 3 sessions back). This means that exact similarity is more important than proximity in time for runtime prediction.

- *Session-Base Prediction based on proximity in time alone in not enough.* Using similarity criteria such as [*] (match any job) or [PEX,*] resulted in performance that is even worse than that of RUH. It seems that separating sessions by means of the think time between jobs alone is not enough to distinguish between different activities of the user, at the level of precision that is required for accurate runtime prediction.
- *Propagation is generally better than non-propagation.* Enabling propagation means taking advantage new information sooner, and indeed using this optimization was preferable in 75% of the cases, resulting in an average improvement (over all four logs) of 10% in average wait time and 6% in average absolute error.
- *Searching in sessions far back in history is useful.* In contract to the RUH Predictor, SBH prediction requires users' entire history to work optimally. Limiting the algorithm to look for less than 10 sessions back deteriorates performance, and surprisingly, even limiting the algorithm to 30 sessions results in performance that is 2%-3% worse, in all logs, than that of the unlimited configuration. Only a small fraction of a system's users have that many sessions, so the effect must be caused by a better fit of these few extremely active users.

8.6. Predicting Without User Estimates

Another interesting research question is whether our predictions algorithms are good enough to rid users of the annoying need to specify estimates (which they do not do well anyway). As shown in section 8.4, the Constant-1-Second Predictor doesn't use estimates and outperforms EASY, but this is not enough, since we have also shown that estimates can be used by other predictors to achieve even better results. So even given the entire history of all users, the value of information in a new job's estimate is not nil.

On the other hand, Session-Based Predictors enable us to build the highest performing predictors to date that don't use estimates, since SBH relies on estimates less than previous algorithms. By measuring the performance and accuracy loss caused by discarding estimates, we enable making an informed decision. Building a No-Estimate SBH Predictor requires replacing the estimates in the three places where they are potentially used:

1. As a default prediction in case no matching past job is found (the last line in Listing 4 is reached). This happens in slightly less than 1% of the jobs in the predictors given in section 8.5.3. The estimate can be replaced here by always predicting 1 second.
2. As a strategy for dealing with missed deadlines, as long as the job's current runtime is smaller than its estimate. The way to replace the estimate here (as we found empirically) is by multiplying the current prediction by a factor of 10 on every miss.

3. As a similarity criterion for matching past jobs to the predicted job. Obviously, Estimates cannot be used in similarity criteria lists if they are not available.

Table 18 and Figure 19 present two high-performing SBH variants that do not use estimates. Both variants don't use propagation, and using it has a negligible effect. The "Balanced" variant stops the exponential growth of predictions after it hits a certain (high) threshold, while the "Unbalanced" variant does not. Not stopping the exponential growth results in better performance in three of the four logs. However, this comes at the expense of an out-of-bounds absolute error, and also at some cost to fairness, since the high over-estimations that are inevitably created enable more backfilling of short jobs (that is, predicted-to-be-short jobs), at the expense of long ones.

Configuration:	**Comparison with RUH, Average over all logs:**			
	Wait Time	B.Slowdown	Abs. Error	Rel. Acc.
Balanced, DFS, [PX,P,X,*]	-5%	-4%	-13%	1%
Unbalanced, DFS, [PX,P,X,*]	-8%	-11%	N/A	1%

Table 18. Best No-Estimate SBH Configurations, Compared with the Best RUH Configuration
Negative values mean that SBH is inferior, positive values indicate improvement.

8.7. Summary

This study benefits both our scheduling and workload modeling goals. With respect to scheduling, it defines a standard framework for comparing the performance and accuracy of predictions algorithms, under common fairness criteria, based on recent new insights about the underlying interactions between the scheduler, predictor, workload and the framework's metrics. It then analyzes a set of new predictions algorithms – the Constant Predictor, the Session-Based Predictor, and the No-Estimate Predictors based on it – and summarizes conclusions drawn from a thorough empirical study of their many possible configurations. These conclusions also provide an intuitive sense of the value of each piece of information that can be used to design future predictors: Recent versus far history, user versus session partitioning of history, exact similarity, a given job's attributes, and in particular its user estimate. An interesting future research direction would be to quantify fairness, in a similar manner as done with accuracy in this paper, and compare scheduling and prediction algorithms quantitatively along the fairness axis as well.

With respect to workload modeling, a first and obvious conclusion from this study is the need for a workload model that models users directly, i.e. provides a user identifier for each generated job. Moreover, the better predictors analyzed here strongly rely on locality of sampling, and in a particular way: locality within the same user. We have also shown that directly applying the concept of sessions to runtime prediction can be beneficial – which reinforces the idea of a workload model which directly models users and sessions.

9. User and Session Analysis of Parallel Workloads

This section presents joint work with Julia Zilber and Ofer Amit, done while guiding their final project at Hadassah College. This section is largely based on (Zilber, Amit and Talby,2005).

9.1. Introduction

Having established the motivation for a user and session based workload model, we now begin the model construction phase. We begin by answer two very basic questions, which haven't been previously addressed.

The first is: **Which variables should be modeled?** Or in an algorithm-designer's words: which features of parallel workloads are important enough to affect performance? As the next section shows, first-generation models focused on modeling the most visible workload attributes, such as runtime and parallelism, but neglected other vital features such as the temporal structure of the workload, user behavior and so forth.

But the question is deeper: after we found these several previously unmodeled features, how do we know that this is "enough"? In other words, how do we know that a given set of variables explains most of the variance found in parallel workloads? To measure exactly that, we use Principal Components Analysis (PCA): a statistical technique used to find the important differentiating variables in a given dataset and measure the proportion of information they represent out of the total variance in the data. We also provide a concrete set of variables, which explains most of the variance between users and sessions in parallel workloads.

The second question we address is: **Which user and session classes exist?** This is required to build the multi-class hierarchical model we aim for, and also to gain first insights about who are the users of typical parallel computers, and what do their work habits look like. To do this, we analyze a set of production workload traces, and use clustering to identify four classes of users and five classes of sessions in parallel workloads. Since we mixed several traces before clustering, this classification is both architecture- and location-neutral – and as can be observed, is mostly focused on universal human traits, such as work in sessions and the daily cycle. In addition, we provide the observed distributions of both user classes and session classes – so that together with the concrete set of variables to model given by the PCA analysis, this study provides a major step towards a complete user-based workload model.

This study is based on the following seven production logs: SDSC Paragon, KTH IBM SP2, CTC IBM SP2, SDSC IBM SP2, LANL CM-5, LANL Origin 2000 and SDSC Blue Horizon. Together these logs contain more than 700,000 jobs that span over 87 months, and come from four different locations and five architectures. While all seven logs come from typical high-performance

computing centers, there are obvious differences caused by architectures, user population, and technical and administrative policies. In order to provide insights that are location- and architecture-neutral, which is our major goal, we analyzed **all logs together**. We extracted session and user statistics from each log, but then combined all sessions to a single list for the PCA and clustering analyses. All users were combined to a single list in the same manner. This ensures that log-specific features will disappear in the cumulative lists, and the main features left will be those that are universal to users of massive parallel computers.

At the end of our clustering analysis, we remapped the clusters – of both users and sessions – to the original logs, to verify that we haven't clustered according to the logs. The full data is not given here due to lack of space, but the bottom-line results are that the session and user classes that we identified exist clearly in all logs, and are indeed unrelated to particular locations or architectures.

9.2. Principal Components of Sessions

9.2.1. Sessions Defined

A session is intuitively a period of continuous work of one user. This does not mean that jobs of the user are active 100% of the session's time – a user may run a job to completion, think about the result, and run another job, all within the same session. The time between the completion of the previous job and the submission of the current job is called the think time of the current job. Intuitively, jobs are considered to be within a single session if there is a short think time between them. There is no other formal definition of a session, and no widely accepted think time that is considered as a session boundary (Arlitt, 2000). In order to decide what the boundary should be, we checked the cumulative distribution of think times in the logs.

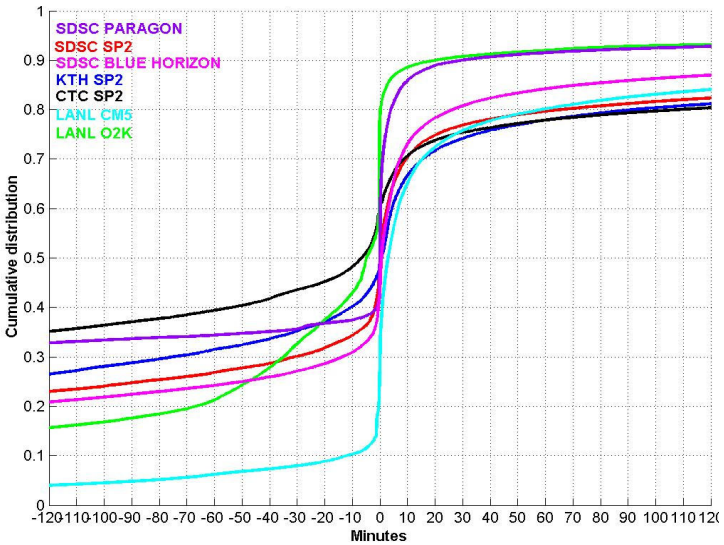


Figure 20. CDF of Think Times

The first observation from Figure 20 is that the different logs are strikingly similar. This is the main reason that enabled us to analyze all logs together and draw meaningful conclusions. The second observation is that most work is indeed done in sessions: while some think times are very high (not within the same session) or way below zero (next job started before previous one was finished), most of the jobs have think times around zero – within the same session.

Following Figure 20, we defined the session boundary to be twenty minutes. This is inspired by the fact that for all logs, around that time the CDF stops its steep climb and settle on a slow, steady rise – meaning that the number of jobs started after 25, 55 or 85 minutes after their previous jobs is about the same. We took this as a cue that above this imaginary boundary the dominating distribution is that of session inter-arrivals, and not the intra-session one.

Twenty minutes is obviously not the only possible choice, but we are confident in it for two reasons. First, we've done a sensitivity analysis, by repeating the following analyses using 15-minutes and 30-minutes boundary values, and received virtually the same results. Second, from a practical point of view, this choice works: Using this definition we received stable and consistent PCA and clustering results, which are useful for future modeling and algorithmic research.

9.2.2. Variables Set

The variables that we used in our analysis are divided to traditional ones, focused on the size of the incoming workload, and newer ones, measuring aspects of the workload's temporal structure. The traditional variables are summarized in Table 19. The median and interval were preferred over the average and standard deviation, as explained in section 4.1.

Symbol	Description
J	Number of Jobs
D	Duration
Rm	Median of Runtime
Ri	90% Interval of Runtime
Pm	Median of Parallelism
Pi	90% Interval of Parallelism
Im	Median of Inter-Arrival Time
Ii	Interval of Inter-Arrival Time
Tm	Median of Think Time
Ti	90% Interval of Think Time

Table 19. Workload Variable Definitions

There are three kinds of temporal structure aspects to represent: locality of sampling, daily cycle and weekly cycle. Note that self-similarity cannot be measured within one session: it is by definition a long-range phenomenon. Locality of a session is represented by two simple variables: the number of unique job sizes (UP) of jobs in that session, and the number of unique runtimes (UR) of jobs in that session, where jobs are considered unique if there is a 5% difference. For most sessions these numbers are very small, which implies locality: they use only a fraction of the overall parallelism and runtime distributions.

The daily cycle is represented by two variables: a binary variable which equals 1 if the session started during the day and 0 otherwise (?D), and a continuous variable measuring the percent of the session that occurred during daytime (%D). The definition of daytime has been derived from the Figure 21, which shows the distribution of job arrivals during the day (corrected for time zone shifts), for all logs.

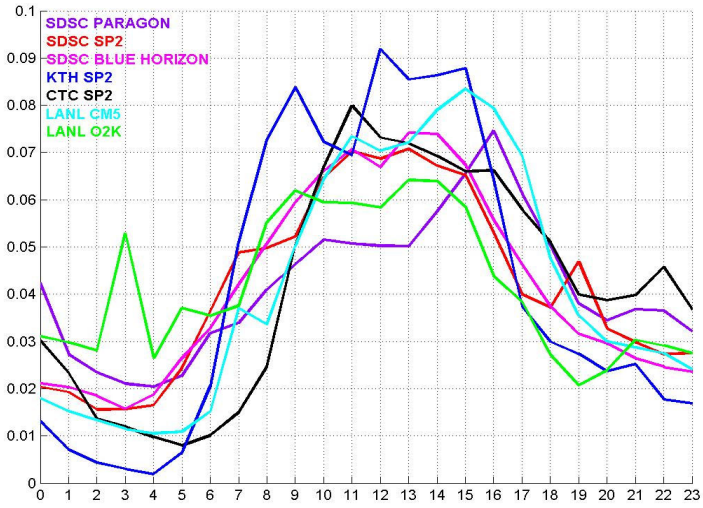


Figure 21. Job Arrivals Across Hours of the Day

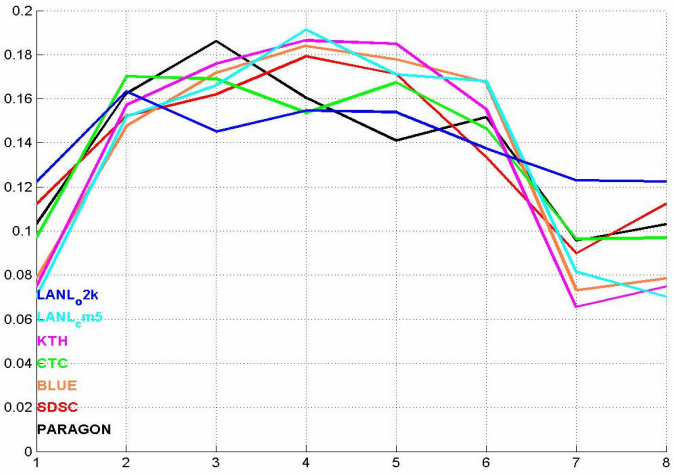


Figure 22. Job Arrivals Across Days of the Week

The results are surprisingly similar for all logs – probably because the daily cycle is a universal human trait, and not a technical one. Based on Figure 21, we defined daytime to be between 7:30 and 17:30.

The weekly cycle is represented by two similar variables: a binary variable that equals 1 if and only if the session started during a weekday (?W), and another one that measures the percentage of the session done during weekdays (%W). According to Figure 22 (in which day number 1 is Sunday), we defined workdays to be Monday to Friday.

9.2.3. Principal Components Analysis

Principal component analysis (PCA) is a statistical procedure that transforms a number of (possibly) correlated variables into a (smaller) number of uncorrelated variables called principal components. The first principal component accounts for as much of the variability in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible. The objectives of PCA are to identify patterns in data of high dimensionality, and to discover or reduce its dimensionality.

For a full presentation of PCA, see (Giudici, 2003); here we'll provide a short summary. Given a matrix of observations – for example, a row for each session – we normalize it, compute the covariance matrix, and calculate the eigenvectors and eigenvalues of the covariance matrix. We then sort the eigenvectors by decreasing eigenvalues – the ones with the highest eigenvalues are the principal components. The size of each eigenvalue is proportional to the percent of variability in the original data its corresponding eigenvector captures.

The original data can be transformed to uncover principal variables by multiplying the sorted eigenvectors matrix (called a feature vector) by the transposed matrix of original data. The first columns of the resulting matrix will contain the principal component values of the data for each observation.

Our input matrix for sessions has a row for each of the 145,582 sessions (in all logs combined), and 18 columns, one for each variable defined in section 9.2.2. Our focus is on finding the dimensionality of the data – hopefully discovering that a select subset of our variables is enough to explain most of the variance between sessions in parallel workloads.

A positive answer to this question would mean that the variables we chose indeed capture most of the variance between sessions. This is the most important point in using PCA – in contrast

to some statistical techniques, the wrong variables won't produce arbitrary results that can be misinterpreted. We have experienced this over a long period of time during this research – starting only with the 12 traditional workload variables from Table 19, many eigenvalues were near equal, and almost no dimensionality reduction was possible. The gradual addition of the temporal structure variables raised the percentage to their current level, in which the first 8 of 18 eigenvectors capture 85% of the variance.

9.2.4. PCA of Sessions

Table 20 describes the nine largest eigenvectors of the sessions' PCA analysis; the last line contains the weight of each vector – its eigenvalue divided by the sum of all eigenvalues – and the cumulative weight.

The next step is to interpret what each of the vectors means in terms of the original variables. To do so, the largest coefficients of each vector are highlighted. For example, the second vector gives most of its weight to the weekly cycle – the coefficients of the other variables in that vector are negligible in comparison. Analogically, the third vector is focused on locality, the fourth on the daily cycle, the fifth and seventh on parallelism, and the sixth on inter-arrival time.

The first – most important vector – contains several variables with similar coefficients. The reason is that these variables are correlated: The (linear) correlation coefficients between Tm and Ti is -0.75 (!), between Ti and Ri is 0.41, between Ti and Ii is 0.38, and between Ii and D is 0.42. Other pair-wise correlations of these variables are high as well, and match the findings in (Talby, Feitelson and Raveh, 1999).

	#1	#2	#3	#4	#5	#6	#7	#8	#9
J	-.09	.11	-.36	-.01	.40	.27	-.45	-.17	.17
D	-.34	-.02	.11	-.04	.03	.21	-.26	.00	-.71
Rm	-.26	-.09	.22	.06	.10	-.18	.01	-.69	-.25
Ri	-.34	.01	.00	-.05	.10	-.19	.35	-.42	.40
Tm	.39	.02	-.12	.07	-.18	.36	.16	-.37	-.06
Ti	-.44	.00	.07	-.09	.14	-.33	-.06	.33	.12
Im	-.28	-.02	.17	-.06	-.23	.56	.15	.03	.22
Ii	-.39	-.01	.16	-.08	-.16	.42	.12	.11	.12
?D	.08	.03	.05	-.70	.00	-.04	-.05	-.15	-.04
%D	.14	.05	.03	-.69	.00	.00	-.02	.01	.02
UP	-.17	.18	-.50	-.06	-.17	-.08	.30	.01	-.21
UR	-.17	.16	-.50	-.03	.27	.16	-.06	-.03	.08
?W	.04	.67	.21	.06	.05	-.01	.02	-.01	-.01
%W	.03	.67	.22	.06	.05	.00	.02	-.01	-.02
Pm	-.08	.06	.00	.05	-.56	-.13	-.66	-.19	.29
Pi	-.16	.16	-.36	-.04	-.52	-.19	.12	.01	-.17
%	19	12	12	11	8	8	6	4	4
C%	19	31	43	54	62	69	75	81	85

Table 20. Principal Eigenvectors of Sessions' PCA

The intuition behind these correlations is clear: Think time is computed by summing runtimes and inter-arrival times, and high runtimes or inter-arrival times imply higher session duration. As a result, these variables' weights are correlated as well. However, there is still a difference between the first vector, focused on the overall session (duration), and the eighth vector, focused on the structure of jobs in the session (note how the duration and inter-arrival variables are negligible there).

High linear correlations explain the pairing of variables in the other vectors as well: the correlation between ?W and %W is 0.93, between ?D and %D is 0.74, between UP and UR is 0.45, between Pm and Pi is 0.24, and between Im and Ii is 0.61. PCA captures these correlations by placing correlated variables in the same vectors; this allows us to relate to the "feature" each vector represents, rather than perfecting the way each feature is measured, or making sure that it's measured once (and that variables are uncorrelated). In contrast, this will be an issue in the clustering analysis in the next section, where highly correlated variables will be filtered out from the analysis.

To verify the stability of the results in Table 20, we repeated the analysis several times, slightly varying the variables set each time. The resulting vectors and the corresponding features they represent always stay the same; however, the order of the vector may change. For example, if only 1 out of 16 variables measures locality (for example, if UR is removed), then the locality vector would move from 3rd vector to 5th place, because the locality feature is now less evident in the dataset. However, the main features and proportion of explained variance is about the same in all analyses.

To conclude, the following features explain most of the variance between sessions:

- Interval of Inter-arrival / think times
- Weekly cycle
- Locality
- Daily cycle
- Parallelism
- Inter-arrival time

Arguably, these variables alone are not enough, since although they dominate certain eigenvectors, there are also non-zero coefficients in each eigenvector, which are required to build it, and ignoring them spoils the results. However, this is more than compensated in the seven smallest eigenvectors (not shown in Table 20), all of which are also dominated by the above variables.

Note the high dominance of the temporal structure variables, which synthetic models to date have largely ignored. In contrast, the runtime seems to play a surprisingly minor role.

9.3. Clusters of Sessions

9.3.1. Methodology

Identifying and characterizing a small number of consistent session clusters is of high practical importance to both algorithm design and workload modeling. We will use the classic K-means clustering algorithm (Giudici, 2003), which in a nutshell works by iterating until convergence a two step process: compute estimated centers of clusters, and tag each observation to belong to the cluster to which it is closest.

The algorithm requires the number of clusters as input, and finding the "right" number of clusters is highly problem-specific and sometimes subjective (Giudici, 2003). We have experimented with a large number of clustering results (a practice required anyway to verify the stability of our results), and concluded that using five clusters gives the most stable and useable results.

Another methodological issue is the variables set by which the clustering is performed. In contrast to the PCA analysis in which we used all candidate variables, here it is desirable to remove highly correlated variables, so that each feature is represented once and the algorithm is not distorted to cluster by any one particular feature. The variables used for the clustering shown here are Duration, Think-time interval, Day time part, Work week part, Unique processors count, Parallelism median and Runtime median.

Table 21 shows the mean, median and interval of each variable in each cluster. Figure 24 shows some of the full distributions of sessions, ordered and colored by cluster using the same colors of Table 21 and Figure 23. Analysis of the data confirms that the clusters correspond to intuitive session types: interactive versus batch work, day versus night, and weekday versus weekend. This enables giving each cluster a significant name.

	Int. Workday Daytime			Int. Workday Night			Interactive Weekend			Batch Highly Parallel			Batch High Duration		
	Mean	Med	Int	Mean	Med	Int	Mean	Med	Int	Mean	Med	Int	Mean	Med	Int
J	3.4	2.0	10.0	5.0	2.0	15.0	4.2	2.0	12.0	2.5	1.0	6.0	22.4	6.0	48.0
D	5634	2626	28371	45445	16452	179584	19560	4186	93611	55788	20713	237677	363495	186533	1246071
Rm	2418	383	12006	11322	2008	53114	5076	485	29682	8068	2774	34182	24304	10828	92299
Ri	2265	9	12821	5583	18	37010	3723	14	24118	3786	0	19284	32825	18104	119838
Tm	-1123	0	6948	-6101	0	40581	-3311	0	21876	-5863	0	34675	-103705	-40434	396074
Ti	4035	112	23819	13423	155	75626	8620	161	53907	13782	0	83244	259850	200832	750768
Im	428	32	1966	1160	17	5751	701	28	2921	2086	0	10984	19722	420	102109
Ii	1435	136	7529	6264	87	41378	3126	128	14006	7822	0	44333	90006	32286	344532
?D	0.95	1.00	0.00	0.37	0.00	1.00	0.72	1.00	1.00	0.67	1.00	1.00	0.66	1.00	1.00
%D	0.97	1.00	0.27	0.18	0.00	0.52	0.65	1.00	1.00	0.55	0.46	1.00	0.47	0.42	1.00
UP	1.28	1.00	2.00	1.27	1.00	2.00	1.30	1.00	2.00	1.25	1.00	1.00	3.33	2.00	7.00
UR	2.49	2.00	6.00	2.75	2.00	7.00	2.63	2.00	7.00	1.93	1.00	4.00	6.55	4.00	17.00
?W	1.00	1.00	0.00	0.95	1.00	0.42	0.02	0.00	0.16	0.78	1.00	1.00	0.73	0.76	1.00
%W	1.00	1.00	0.00	0.95	1.00	1.00	0.04	0.00	0.00	0.79	1.00	1.00	0.77	1.00	1.00
Pm	4.99	3.56	15.94	5.54	2.56	21.27	4.32	1.78	15.94	61.76	64.00	60.62	9.59	6.00	31.38
Pi	1.77	0.00	10.24	1.95	0.00	12.00	1.80	0.00	10.00	9.52	0.00	56.89	13.54	5.29	62.00

Table 21. Characteristics of the Five Session Classes

9.3.2. The Five Session Classes

Interactive, workday, daytime sessions. This is the most common session class (43%). It occurs always in weekdays, and starts during daytime in 95% of its sessions. Each session has very few jobs (median is 2.0 and mean is 3.4), resulting in high locality. Runtimes and parallelism are low, typical for interactive work.

Interactive, workday, nighttime sessions. These 29% of the sessions are active mostly during the night (82%) although only 63% start at night. The number of jobs, inter-arrival times and think times are short and typical of interactive work; locality is high as well. On the other hand, runtimes are much higher (median of 33 minutes in contrast to 6 minutes in the daytime cluster). The most plausible explanation is that these sessions often represent someone who works interactively during the day, and towards the evening starts one or more long job that run during the night.

Interactive, weekend sessions. The weekly cycle is the next important differentiator between sessions, according to these results. 98% of sessions in this cluster start during weekends. The statistics are typical of interactive work, and are mostly in between the values of the previous two weekday clusters.

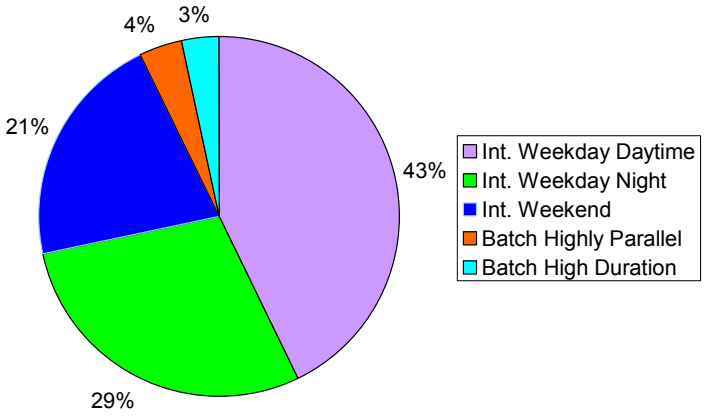


Figure 23. Distribution of Session Classes

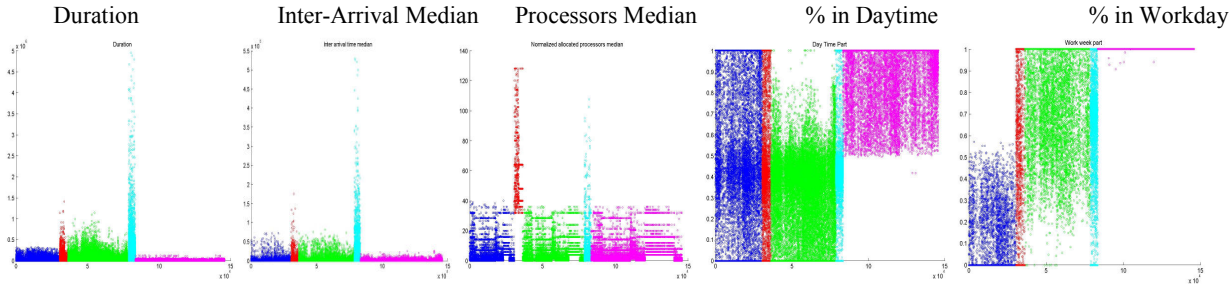


Figure 24. Distribution of Selected Variables by Session Cluster

Batch, highly parallel sessions. The last two clusters obviously represent batch work, and are divided between parallelism and runtime. Both clusters are active near-equally at day vs. night and weekday vs. weekend – note that as we defined it daytime is 42% of each 24-hour day, and workdays are 71% of a week. This cluster has sessions with usually one job (median 1.0, mean 2.5), very high parallelism (median 64.0, meaning half the machine since it's normalized across logs), and much higher runtimes than the interactive sessions.

Batch, high duration sessions. These batch sessions have higher parallelism than the interactive sessions (5.29 median), and most evidently – a runtime median of 51.8 hours and mean of 100.9 hours, hinting that the runtime distribution has a long tail. This is in sharp contrast to a median runtime of 5.75 hours in the other batch sessions cluster, and much less in the interactive ones. This session class has the most jobs (median 6.0, mean 22.4), but this may be caused in part by the way we defined sessions using think times. Most of the think times in this session class are highly negative – that is, the user does not wait for a job that runs several hours to complete and then submits another one within 20 minutes; the typical case is submitting several jobs concurrently.

9.4. Principal Components of Users

9.4.1. Variables Set

Our data has 2,048 users in all the logs combined. The traditional variables were kept, computed on all the jobs submitted by each user: median and 90% interval of runtime, parallelism, inter-arrival time and think time. The total number of jobs J and the total number of sessions S are used to quantify the user's activity. To measure how continuous and intensive that activity was, a jobs per week (JW) variable was added; we also computed a sessions per week variable, but its linear correlation to JW was full (1.0), so it was dropped. Duration D in days was not used in the PCA analysis, but was measured and will be given in the table of clusters.

Some of the temporal structure variables make no sense to measure at the user level – the UP, UR, ?D and ?W variables lose their meaning when jobs from different sessions are combined. The %D and %W variables over each user's jobs were measured, as they describe the user's habits outside a single session scope.

The suggested way to analyze users' temporal behavior, including aspects such as locality, is to take advantage of our analysis of sessions. For each user, we add five more variables, each counting the proportion of each session type in that user's sessions. This encapsulates many aspects of the user's work patterns, and as the analysis will show, provides good results. The variable names correspond to the initials of the session classes' names from section 5.2, and in the same order are

%IWD, %IWN, %IW, %BHP and %BHD. Note that the proportion of session of each class is used and not the number of sessions, to decouple the amount of work (measured using the J and S variables) from the measurement of work patterns.

The first nine eigenvectors of the PCA analysis of the above 18 variables are given in Table 22. The two think time variables (Tm, Ti) are absent, since they have a correlation of 1.0 and 0.99 with the Im and Ii variables and therefore have exact similar values.

9.4.2. PCA of Users

The first 9 out of 18 eigenvectors in the users' PCA capture 83% of the variable between users. However, the next five eigenvectors are a repetition of eigenvectors 5,6,2,3 and 4 respectively – meaning they have the same dominant variables – so with the same set of variables used for the first eight eigenvectors, 96.7% of the variability is captured.

As with sessions, each eigenvector represents a feature, and variables are grouped by correlation. These features explain most of the variance between users:

- Daily Cycle
- Parallelism
- Runtime
- Number of jobs / sessions
- Weekly cycle
- Inter-arrival time
- Jobs per week

	#1	#2	#3	#4	#5	#6	#7	#8	#9
J	-.01	.13	-.01	-.64	.06	-.26	.15	-.06	.04
S	.04	.19	.00	-.62	.05	-.23	.03	.20	.01
Rm	-.24	.07	-.44	.12	.30	-.07	.06	.36	-.16
Ri	-.20	.16	-.50	.07	.22	.00	.07	.22	-.01
Pm	-.08	.51	.18	.16	-.04	.00	.02	.13	-.07
Pi	.01	.50	.13	.03	-.07	.05	-.05	-.17	.05
Im	.01	.01	-.08	.19	-.25	-.63	.21	-.04	-.12
Ii	.09	-.02	-.13	.25	-.26	-.54	.06	-.09	.25
%D	.49	.11	-.28	.08	.10	.06	.12	.18	-.03
%W	.07	-.02	.20	.08	.47	-.32	-.32	-.22	-.66
%IWD	.57	-.05	.08	.05	.25	-.02	.13	.10	.13
%IWN	-.53	-.19	.18	.01	.10	-.13	-.10	.13	.19
%WD	.01	-.05	-.35	-.17	-.59	.17	-.16	-.08	-.51
%BHP	-.04	.54	.21	.15	-.14	.02	-.03	.20	-.07
%BHD	-.14	.23	-.30	.01	.23	.09	.26	-.75	.11
JW	-.12	-.13	.26	.03	-.02	.12	.83	.08	-.36
%	16	15	11	10	8	7	6	5	5
C%	16	30	41	51	60	67	73	78	83

Table 22. Principal Eigenvectors of Users' PCA

	Long-Term Light User			Long-Term Heavy User			Short-Term Weekend User			Short-Term Workday User		
	Mean	Med	Int	Mean	Med	Int	Mean	Med	Int	Mean	Med	Int
J	259	68	1252	526	171	1529	48	8	242	170	10	683
S	70	26	281	82	50	223	12	4	47	26	4	117
Rm	2,569	241	13,092	12,292	4,668	48,951	1,653	91	9,679	1,984	107	10,822
Ri	11,853	5,269	43,228	54,460	56,064	117,519	7,609	821	43,205	3,750	256	23,333
Pm	3.22	1.78	14.94	8.45	3.56	31.75	2.98	1.00	10.18	3.20	1.25	10.18
Pi	11.06	3.75	60.00	18.31	8.98	64.00	5.40	0.19	30.00	5.30	0.00	28.00
Im	33,349	390	71,072	-7,063	6	125,444	50,996	147	29,381	21,784	0	21,522
Ii	1,135,229	261,020	5,025,950	776,389	396,090	2,684,896	1,058,137	119,341	4,899,134	471,280	34,469	2,499,302
Tm	39,299	1,419	88,965	31,567	2,659	133,638	56,819	548	69,745	24,742	204	38,180
Ti	1,134,561	256,335	5,023,631	693,984	283,564	2,340,145	1,060,989	106,129	4,899,777	471,995	28,530	2,497,021
%D	0.82	0.86	0.47	0.66	0.67	0.61	0.60	0.67	1.00	0.18	0.18	0.48
%W	0.80	0.78	0.46	0.72	0.72	0.42	0.22	0.25	0.50	0.90	1.00	0.33
%IWD	55%	51%	83%	24%	23%	48%	36%	30%	100%	17%	10%	50%
%IWN	22%	20%	53%	38%	38%	69%	36%	25%	100%	65%	60%	100%
%WD	20%	20%	44%	19%	19%	38%	25%	10%	100%	17%	3%	67%
%BHP	2%	0%	11%	6%	0%	47%	2%	0%	3%	1%	0%	2%
%BHD	1%	0%	7%	13%	8%	43%	1%	0%	7%	1%	0%	4%
JW	126.30	6.03	502.65	25.13	4.86	47.57	1139.86	10.51	6951.26	1247.26	20.48	7114.54
D(days)	239	125	789	323	260	848	97	12	583	110	7	636

Table 23. Characteristics of the Four User Classes

9.5. Clusters of Users

Users were clustered with the same methodology used for session clustering. The variables selected to perform the clustering are Day time part, Parallelism median, Runtime median, Number of jobs, Work week part, Inter-arrival time median, Jobs per week, and the proportion of batch high-duration sessions. These variables include the dominant coefficients in all eigenvectors of the users' PCA analysis. The best results were obtained with four clusters, and as with sessions, it is possible to assign meaningful names to each user class, corresponding to intuitive user types.

Long-term, Light users. 55% of users belong to this class, whose members have a median of 26 sessions over a period of 125 days. The means are much higher, indicating a long tail of the respective distributions. According to the proportions of session classes for these users, their focus on interactive work is higher than the overall average – this class seems to represent people whose day job involves use of the parallel computer. The runtimes, parallelism, inter-arrival times and number of jobs per week are all in accord with a mainly interactive style of work.

Long-Term, Heavy users. These 20% of the users are the source of most of the load on the computer. 6% of their sessions are BHP and 13% of their sessions are BHR, in contrast to 1-2% in all other user classes. These users produce most of the sessions of these two batch classes. These users are also the heaviest users of the machine in terms of number of jobs, sessions and duration, by a significant margin. They work both day and night, workday and weekends in equal proportions. Their runtime, parallelism and inter-arrival statistics are high, a mix of their interactive and batch sessions.

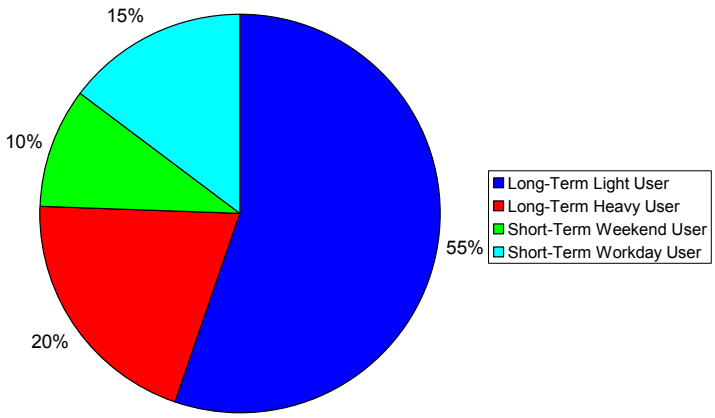


Figure 25. Distribution of User Classes

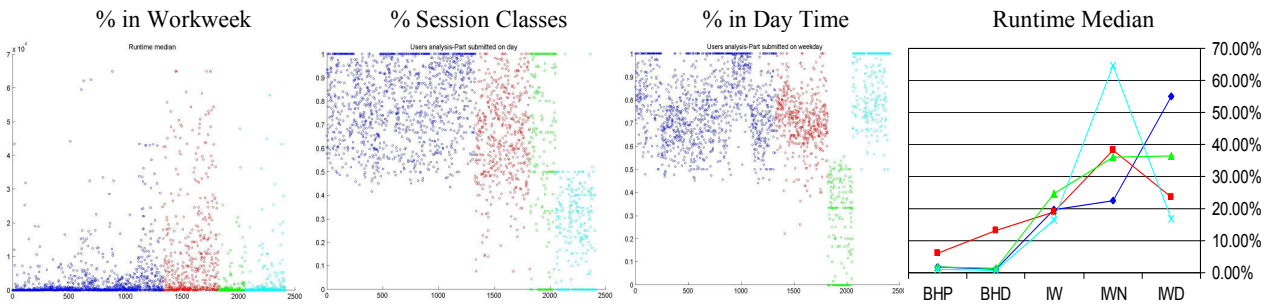


Figure 26. Distribution of Selected Variables by User Cluster

Short-term, Weekend users. The last two user classes represent users who worked on the computer for a short period – a median of 4 sessions, with median durations of 12 and 7 weeks, and a small total number of jobs (8 and 10, although the means are much higher). These seem to be users who received access to the computer for one computationally demanding project. The two user classes differ by their temporal work patterns, as measured by %D, %W and the session classes' proportions. The third "weekend" cluster, consisting 10% of the user population, has 78% of jobs starting on weekends, and the proportions between the three interactive session classes are much closer than in the overall distribution of session classes.

Short-term, Workday users. 15% of the users belong to this fourth cluster, composed of short-term users with two strong habits: they prefer workdays over weekends (90% of jobs ran in workdays), and they prefer the night over day time (82% of jobs ran during the night). This is reflected by the fact that 65% of their sessions are interactive workday night sessions.

Both short-term user classes consist mostly of interactive work, reflected by low runtime and parallelism statistics, and very intensive work, reflected by a very high number of jobs per week. Figure 26 contains several graphs that visualize these differences between the user clusters, and Figure 25 summarizes the distribution of user classes.

Since we have conducted all our analyses on the combined workload of seven logs, it is necessary to remap the clusters – of both users and sessions – to the original logs, to verify that we haven't clustered according to the logs. The full data is not given here due to lack of space, but the bottom-line results are that the session and user classes that we identified exist clearly in all logs, and are indeed unrelated to particular locations or architectures.

9.6. Summary

This study provides the basis for our user-based workload model. The PCA analysis answers a basic methodological question – what needs to be modeled? The temporal structure variables in particular have been shown to be of great importance, while some other variables have not. The PCA analysis was also a precondition to a meaningful clustering, since it ensures us that we are clustering according to a suitable set of variables – which wasn't known before.

In addition, we provided a very specific blueprint for constructing such a model: the set of variables, the user classes and their distribution, and the session classes and their distribution. To complete the model, full distribution fitting of each modeled variable in each user and session class is required, as well as a distribution for the inter-arrival times of new users and new sessions. Before doing so, the next section will address the issues of parameterization and load manipulation, which will be used to adjust these distributions.

A second use of this work is algorithm design. Consider for example an algorithm that relies on good runtime prediction, such as a backfilling scheduler (Section 8), a grid management system (Jarvis et al., 2004) or a soft real-time task mapping service (Dinda et al., 1999). When a job arrives, it needs to be attached to a session, made simple using the same-user, 20-minute think time rule. If the job is starting a new session, then its class must be determined, which is also easy: if it exceeds a certain requested runtime or parallelism – half the machine's maximum, for example – the session is tagged as BHD or BHP respectively. Otherwise, the session type is determined solely by the day of the week and the time of the day.

Once the session class is determined, we know its distribution of runtimes. This distribution is far narrower – and thus more informative – than what can be known a-priori by other means. Three layers of narrowing are at work here: the user level, the session level, and the session class level.

If a given job is not the first within its session, then locality enables us to predict its runtime with great accuracy, based on the past jobs of that session. For example, In the IWD session class – to which 43% of sessions belong – the mean unique number of runtimes is just 1.28. Sessions, as we defined them, are a natural way to capture and exploit locality, especially since determining a

session's class is so easy. Using only the user, or the last hour or day, is open to much more noise and thus to inferior results.

This is not to say that a learning algorithm is not necessary – but given the a-priori knowledge embodied in this study, there is much less left to learn. We believe this should lead to reassessing existing predictors, and in particular to the preference of simple "knowledge-packed" algorithms overly highly sophisticated AI techniques (Yu et al., 1997; Jarvis et al., 2004) that assume little in advance.

Similar to runtime prediction is the problem of load prediction, found in load balancing (Yu et al., 1997), grid or multi-cluster scheduling (Jarvis et al., 2004) and soft real-time or general QoS enabled systems (Dinda et al., 1999). Here we must predict both runtime and parallelism, and sometimes the number of jobs as well. Regardless of the specific problem and of whether the algorithm that tries to solve it is predictive, adaptive, dynamic, learning or plain heuristic – this work provides a lot of sound prior knowledge on parallel workloads it can easily use. Due to the large size of our data, the architecture-neutrality of the analysis, and the stability of the results, we believe that they can be highly useful for a large variety of applications.

10. Parameters for a Parallel Workload Model

10.1. Introduction

The goal of this section is to find and validate the parameters for our new workload model, thus addressing both the parameterization and load manipulation methodological issues. The output of this section is a meta-model, which directs how the two selected parameters define all the other workload-level variables. This section is largely based on (Talby, Feitelson and Raveh, 2007), where the Co-Plot technique is used to find the number of required parameters, the parameters themselves, and then to validate the resulting meta-model.

Load manipulation is tackled by enabling load to be one of the two parameters, and defining the required equations to compute the other workload-level variables accordingly. This will later enable load to affect the user and session models, rather than the jobs model directly. However, the results in this section are relevant to parameterizing any parallel workload model – including models that are not user- or session-based.

Note that this section only builds a meta-model of summary statistics of entire workloads, and therefore its validation is also by statistical means. Validation against the adaptive and prediction-based schedulers will require the full model.

10.2. The Two Axes of Parallel Workload Variation

Perhaps more important than analyzing individual variables, the Co-Plot analysis enables us to understand the correlations between all the variables at once, and to define clusters of variables and the relations between them. In order to do so, Figure 27 includes the arrows from Figure 2, plus two imaginary perpendicular dotted axes, to make our explanation easier.

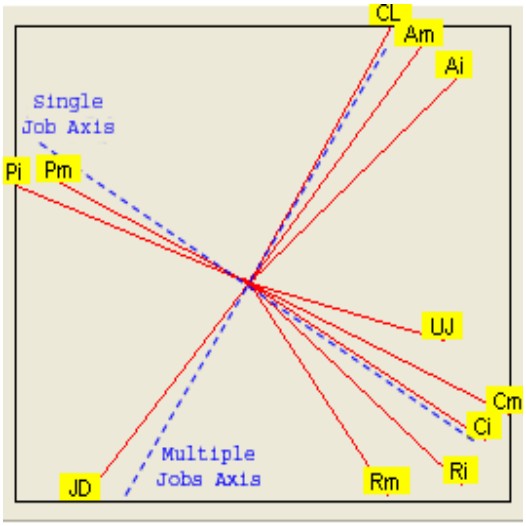


Figure 27. Variable Arrows from Figure 2

The two dotted axes are perpendicular – meaning not correlated in the plot. All the variables along each axis are highly correlated – for example, the CPU load (CL) and normalized number of jobs (JD) are very highly correlated, although that correlation is negative. This correlation does not have to be linear – note the definition of distance and alienation in Appendix B – but it means that the variables grow together, by some relationship. The two axes were drawn since most of the variable arrows fall very close to one of them, and they define the two main variable clusters of this data set:

- **The Multiple Jobs Axis** – includes the number of jobs per day (JD), the median and interval of the inter-arrival times (Am and Ai), and the CPU load (CL). The interval of the normalized degree of parallelism (Ni) and the total number of nodes in the machine (TN) are also members of this axis (this conclusion is drawn from other runs of the analysis as well), although their correlation with the axis is not full.
- **The Single Job Axis** – includes the median and interval of runtimes (Rm and Ri), the median and interval of total CPU work (Cm and Ci), and the median and interval of parallelism (Pm and Pi). The normalized number of users (UJ) also belongs to this cluster; its correlation to the axis is not full, but actually stronger in most runs than it appears in Figure 27.

The runtime load (RL) and the median of normalized parallelism (Nm) seem to be between the two axes. However, as mentioned above, these two variables have a low correlation in this Co-Plot, so it's hard to conclude anything about them. Since the runtime load is an important parameter in many studies in this area, it should be noted that the CPU load – which exhibits high correlation in this Co-Plot – can possibly be used to represent it, as the two variables are highly correlated, in the usual linear sense of correlation.

The names of the two axes reflect the fact that one axis gathers the variables that define a single job, while the other axis gathers variables that define the relations between jobs. Although we are making a generalization here, as some important variables are not fully correlated with either axis, we can still say that these two axes define the two-dimensional workload space. The low coefficient of alienation (0.10 and below) achieved in our analyses means that the workloads space can be hosted in a 2-D space without much loss of information. The two axes we identified clarify what each dimension stands for.

The Single Job Axis provides us with the following information. Logs with high runtimes (on average, relative to other logs) also have high total CPU work, and in contrast have low parallelism. This is true for both the median and the interval of the three distributions involved. Also, logs with high runtimes have more users per job. This is important, because the number of users is one of the only variables that can be estimated in advance, before a system is built. The strong correlation between the number of users and the medians and intervals of runtimes or degrees of parallelism means that these variables too can now be predicted in advance. This is why the number of users was normalized by the number of jobs, and not by the job's duration (like the JD variable): this was required in order to decouple it – i.e. make it uncorrelated – to the Multiple Jobs Axis. The number-of-users-per-day (UD) variable was also computed: it enters the Co-Plot with a high correlation, and is usually drawn between the two axes.

The Multiple Jobs Axis also provides valuable information about its variables. First, logs with many jobs per day (JD) are usually found on machines with a lot of processors (TN). The TN variable is not fully correlated to this axis, since – as expected – it has a small positive correlation with parallelism (it is expected that machines with more nodes will have higher parallelism, although this effect seems to be weak). As expected, logs with many jobs per day have very low inter-arrival times (more jobs over the same time frame must be temporally closer to one another). On the other hand, they also have very low CPU loads and runtime loads, which is counterintuitive.

The median and interval of the normalized number of processors are between axes. The median of the normalized parallelism has an obvious positive correlation to the (un-normalized)

parallelism, and is uncorrelated to the total number of nodes in the machine (as expected by its definition). The interval is somewhat closer to the Multiple Jobs Axis, but the correlation is far from full, so it's safest not to conclude anything about it.

One should be careful not to misinterpret these findings – note that they relate to whole workloads, not jobs. For example, the negative correlation between the degree of parallelism and runtime means that systems with high average parallelism exhibit lower runtimes, not that jobs that use many processors are shorter – the opposite of which was indicated in (Feitelson and Rudolph, 1998; Downey and Feitelson, 1999). The reason for this may be, for example, that systems with more processors tend to enforce tighter runtime limits on jobs. This would fit in with the conjecture that systems with fewer processors may try to compensate for this by offering more flexible policies.

10.3. A Parametric Meta-Model

As described in section 4.1, one of the first results of our analysis was the fact that real workloads are rather different from one another, so a single model cannot represent all systems. As any specific model will also occupy a given place in the "workloads space" represented by the Co-Plot, it too will be far from most of the logs.

The good news is that another result of the Co-Plot analysis was the identification of variables that can be used to parameterize a workload model. As we saw the analyzed workloads fit well into a two-dimensional space, and each can be placed in this space by defining its position on two axes. The variables on each axis are highly correlated with each other, so that once one is given, the others can be well approximated. Moreover, the two axes are largely independent, so there is no need for multiple regression, for example. It is therefore clear that a parametric workload model should take one parameter from each axis.

The main criterion for selecting a representative parameter from each axis is that this should be a variable that can be estimated for a future system, before that system is built. This is required because a model of a system's workload is most useful before it is built and deployed. Specifically, we found that the Multiple Jobs axis can be represented by the number of jobs per day (JD), or to a lesser degree (but trivial to know in advance) the total number of nodes in the machine. The Single Jobs Axis does not contain an easy-to-predict variable, except for the users per job parameter, to which it is only weakly correlated. Experimenting with different possible lead variables led to the conclusion that the highest correlations are achieved when the Single Jobs Axis is represented by the median of CPU work (Cm). Table 24 summarizes the equations and R^2 values for fitting the other main variables to the two selected representative variables. These values were generated based on all the data from the production workloads (Table 2).

Equation	R^2
$Ci = 14052 \times \ln(Cm) - 31068$	0.75
$Rm = 0.8346 \times Cm + 131.0012$	0.60
$Ri = 10423 \times \ln(Cm) - 14441$	0.76
$Pm = -1.531 \times \ln(Cm) + 12.261$	0.28
$Pi = 74.69 \times e^{-0.002 \times Cm}$	0.44
$UJ = 0.7633 \times Cm^{0.3073}$	0.45
$CL = 2.2024 \times JD^{-0.2534}$	0.75
$Am = 2764.4 \times JD^{-0.6582}$	0.74
$Ai = 580400 \times JD^{-1.1079}$	0.96

Table 24. Fitting equations and coefficients to the *JD* and *Cm* variables

Note that the correlation is not always linear, as this is not the type of correlation that Co-Plot uses (see Appendix B). If variables are plotted along the same axis, it means that the variables grow together, by some relationship. Limiting this relationship to be linear would make Co-Plot far less useful in practice, since far fewer datasets would fit into a two-dimensional space under such a limitation. The imperfection of the R^2 values in Table 6 are caused by a combination of the imperfect plot they are based upon (a coefficient of alienation of 0.10 and not 0.0), and our choice to use only simple fitting equations, to keep the model simple and avoid over-fitting.

In order to test whether this meta-model is good enough for practical use, we test our ability to generate parameter combinations that will locate a model in a specific place in the Co-Plot map. For this we generated four synthetic observations, using combinations of two pairs of "low" and "high" values of the two proposed parameters, as shown in Table 25.

The resulting Co-Plot is shown in Figure 28, using the same variable set as in Figure 1, and the same dataset augmented by Gen1 to Gen4. The coefficient of alienation is 0.11 and the average of correlations is 0.85. The results are very positive in two respects. First, the Figure itself is not affected at all by the addition of the four new observations: All the observations and all the variables are exactly where they were in Figure 2, and retain all their relative positions and directions.

Observation	JD	Cm
Gen1	100	25
Gen2	300	25
Gen3	100	600
Gen4	300	600

Table 25. Parameter values for the four synthetically generated workloads

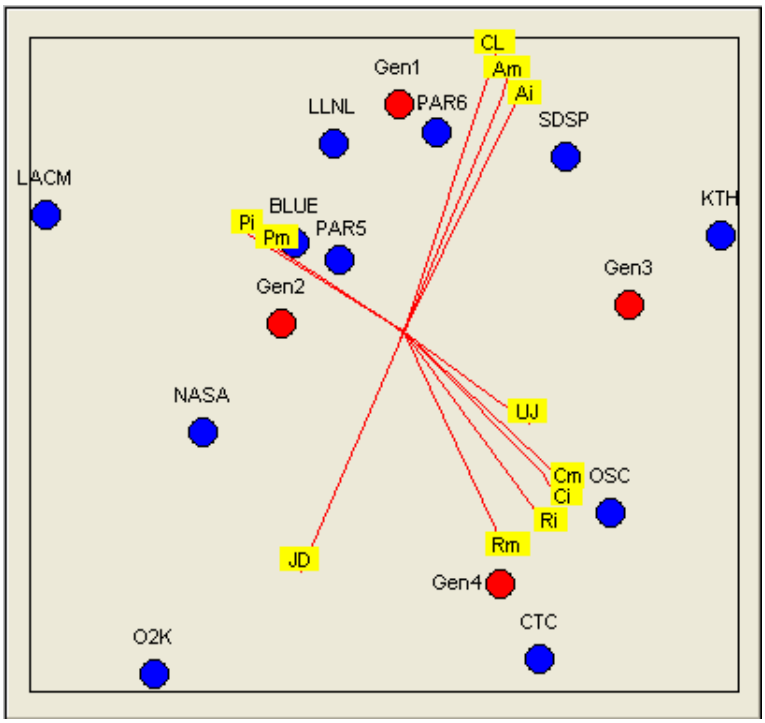


Figure 28. Verifying the parametric JD/Cm meta-model via synthetic observations

Second, the four synthetic observations are each placed in a different quadrant of the plot, as marked by the center and directions of the two axes. Moreover, each is placed in the quarter it is expected to belong to, as dictated by Table 25. We therefore conclude that the meta-model specified by Table 24 is a good representation of the current known world of production parallel workloads.

10.4. A Meta-Model for Load Manipulation

In section 4.3, we concluded that the three most common methods to manipulate the load of a workload – multiplying its inter-arrival times, runtimes or parallelism by a factor – are flawed. A possible solution to this problem, based on the parametric meta-model, is to make load a first-class model parameter. This will later enable load to affect workload attributes other than the above three, particularly at the user and session levels, in ways that will preserve other correlations.

Although the runtime load is not fully correlated to either axis, the CPU load (CL) is highly correlated to the Multiple Jobs Axis, and it can replace the jobs-per-day variable as the lead variable representing that axis in the meta-model. Note that current models do not differentiate between runtime load and CPU load – they implicitly assume that jobs use 100% of the available CPU during their entire run. Table 26 defines the meta-model equations based on the CL variable rather than the JD variable; only equations for CL are given, since the ones for Cm remain as they were in Table 24.

Equation	R^2
$JD = 39.309 * CL^{-2.9549}$	0.75
$Am = 285.72 * CL^{2.208}$	0.72
$Ai = 10103 * CL^{3.3032}$	0.73

Table 26. Fitting equations and coefficients to the *CL* variables instead of the *JD* variable

Note that under this meta-model, changing the load affects the runtimes, parallelism, and inter-arrival times as specified in section 4.3. This is not a recipe for manipulating a given workload to achieve a desired load level; but a model that will be based on this meta-model will handle load manipulations correctly. Again we validate this model using four synthetically generated observations, as done in Table 25 for the JD/Cm meta-model. The low and high values used for the CPU load were 0.45 and 0.65 respectively, and the Co-Plot which merges Figure 2 with the four generated observations is given in Figure 29. Its coefficient of alienation is 0.11, and its average of correlations is 0.85. As the figure shows, the CL/Cm meta-model is just as successful as the JD/Cm meta-model.

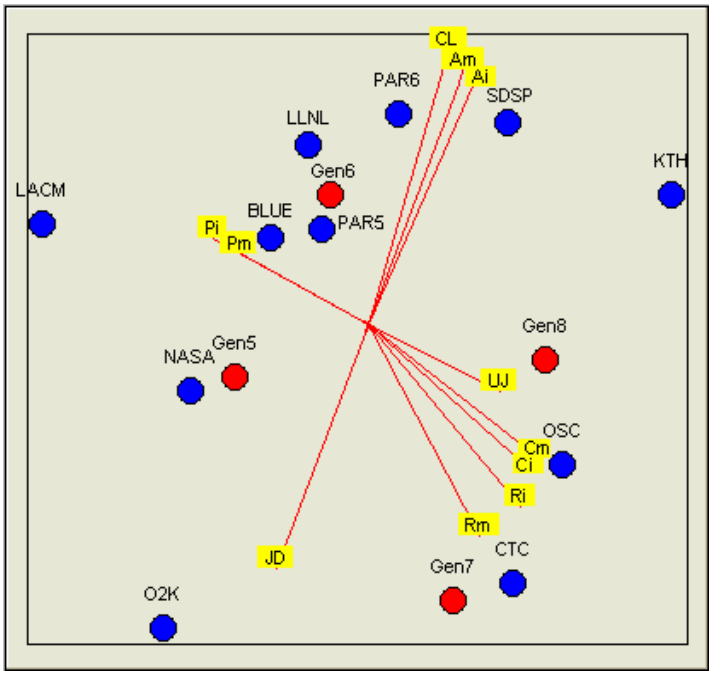


Figure 29. Verifying the parametric CL/Cm meta-model via synthetic observations

10.5. Machine Size

Another workload parameter which must be addressed by a workload model is the machine size: How many processors does the computer have? On one hand, this parameter was removed at the early stages of the Co-Plot analysis, because it did not correlate well with all the other variables. On the other hand, this parameter must be determined, and in the case of a synthetic workload built to simulate a future system, the machine size may be the only parameter known with certainty (in

contrast to CPU time, jobs per day or load). In addition, the machine size determines the range of job sizes that the model must produce.

The user-based model is based on the same seven production logs used to find user and session clusters and principal components. These seven logs are broadly assumed to represent "mainstream" parallel computer workloads well, and are the most heavily used in current research. These logs have a range of machine sizes – 100, 128, 400, 430, 1024, 1152 and 2048 processors – representative of current supercomputers. In comparison, the most recent Top 500 supercomputers list (Top500.Org, 2006) includes 48 computers with 257-512 processors, 264 computers with 513-1024 processors, and 124 computers with 1025-2048 processors. Note that the average size of a new parallel computer today would be less than the average observed on the world's Top 500 list.

To understand how to relate machine size (MS) to the model's parameters as given in the previous sections (Cm, JD, CL), and how to cope with the situation in which the machine size is given as input to the model, we computed the correlation between machine size and the model's parameters, using only the seven logs on which the model is built. The results were surprising.

Correlation of **MS** and **Cm**	–0.67
Correlation of **MS** and **JD**	0.93
Correlation of **MS** and **CL**	–0.74
Correlation of **Cm** and **JD**	–0.45

Table 27. Correlations between machine size and model parameters

The strongest observation is that a bigger machine results in more jobs – not bigger jobs or higher load. People use bigger machines to complete more jobs, but these jobs are actually "lighter" on average (in terms of CPU work). Why this happens can only be speculated: It may be because large supercomputers are badly managed and highly underutilized, or it may be because users program their jobs to split into many small jobs, to enhance their backfilling opportunities. In addition, there's a negative correlation between machine size and load: Smaller machines are more heavily utilized.

These correlations are higher than those observed for the Co-Plot analysis which involved all logs, so they should be considered with caution. However, although they are based on a small dataset, it is a representative one, and it offers the best advice which can be given today on how to handle machine size within a workload model.

If only the machine size is given as the basis for a workload model, then it can be used to compute the other parameters according to the formulas given in Table 28. The formulas are based on best curve fitting to the available data, with an adjustment to ensure their scalability where several fits had a similar R^2 value. For example, the relation between machine size and load cannot be linear, since this would predict that a machine with 10,000 nodes would have a negative load.

Equation	R^2
$Cm = 361860 \times MS^{-1.3417}$	0.68
$JD = 0.3694 \times MS - 1.3102$	0.86
$CL = -0.09 \times Ln(MS) + 1.1356$	0.57

Table 28. Computing model parameters based on machine size

11. Modeling User Arrivals and Class

11.1. Analysis of User Arrivals

To build the dataset for the model construction, three data files were built from the cleaned version of each of the seven logs as published in the Parallel Workloads Archive (Feitelson, 1999): A list of jobs, a list of sessions and a list of users. The file formats are detailed in Appendix D.

The input to model construction is either the machine size, from which the median of CPU work and the load are computed, or all of these three parameters (for purpose of load manipulation, for example). The first step is to create the user population, and for this purpose see Figure 30, which shows the number of new users per week in each of the seven investigated logs.

The granularity of the X axis is weekly, because plots based on a daily or hourly arrival of users are affected by the weekly and daily cycles respectively, which we would like to avoid at this stage. To prevent such interdependencies, we will model the number of new users which appear every week, and the cycles will be taken into account later when building the session model. Therefore, the "arrival" of a user in a given week doesn't mean that her first job would start as the week begins, and users already active when the log starts may not have a job in the first week at all.

As Figure 30 shows, a large number of users are already active at the start of each log, and other new users appear with time. This means that we need to model the arrival process of new users separately from the snapshot of users already active when a workload begins. Note that the majority of active users don't always appear on week 1, but sometimes on weeks 2 or 3: This is either because the first week is not a full week (most logs don't start on Monday), or because the logging process started when the machine was still in "warm-up" mode and not open to all potential users. Our model will always begin from Monday at midnight and assume no "warm-up" period, i.e. the computer is already in normal operations with an active user population. This makes the most sense if the model is intended to use for activities such as comparing schedulers.

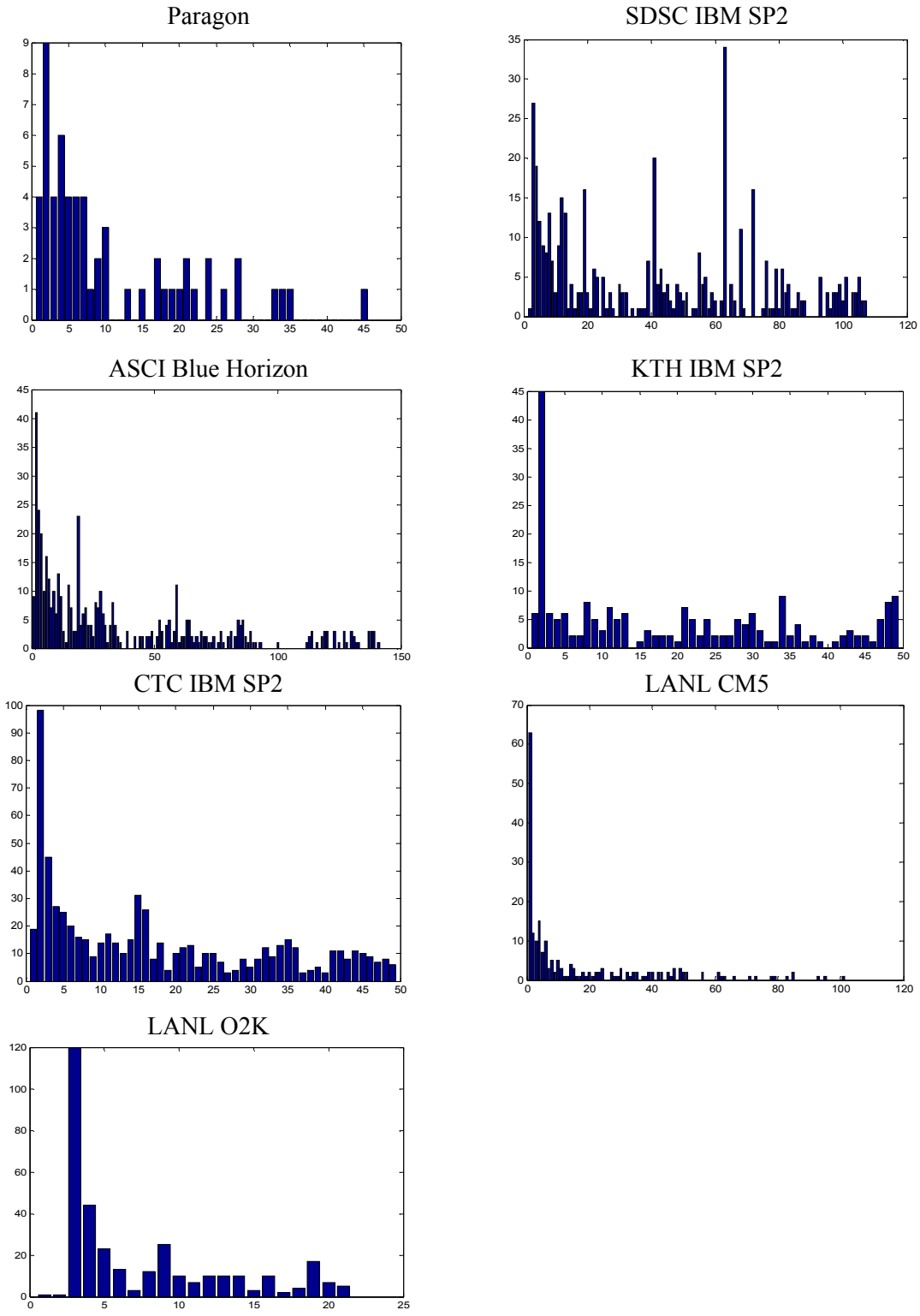


Figure 30. Number of new users per week in the modeled logs

In most logs, the number of new users appearing in weeks 2-10 is greater than the average later on in the log. This suggests that these are active users who already existed when the log began, but only had their first session (that was captured by the published log) several weeks later.

There is one occasion in which a large population of new users appears in the middle of the log: Week 63 of the SDSC IBM SP2 log. The reason for this is unknown – perhaps another computer at that center was shut down, or a new organization gained access to the computer. In any case, such a sole case is clearly an outlier, and is therefore excluded from the model.

11.2. Number of new users per week

To model the number of new users per week in the steady state, Figure 31 shows the number of new users per week, for all logs combined. The logs were aligned to the first week (Starting Monday at midnight) in which a majority of their users were active, to exclude "warm-up" weeks in which the computer worked but users apparently didn't. This meant discarding the first (partial) week of the Paragon, BLUE, KTH and CTC logs, and the first two weeks of the SDSP2 log. Under this alignment Figure 31 shows the first 44 weeks of all logs except LANL O2K. The O2K log was excluded because it only has 21 weeks, so including it would distort the picture for the other 23 weeks. Figure 31 can therefore be considered as an average of new user arrivals, multiplied by six.

The next step is to consider only the users which arrived in the steady state. Figure 32 does just that: It is a histogram of the number of new user per week, for one log (i.e. after dividing by six), based on the data from Figure 31 but starting from week 12. This is the empirical distribution of new users per week to be modeled, and it has two useful properties.

First, it is random, and therefore can be modeled by drawing from a distribution. To determine this we used the runs test for randomness (Bradley, 1968), which is appropriate for this task for two reasons. First, it is a non-parametric test, so its validity does not depend on assuming any particular distribution of the data. Second, it tests for both locality (clustering of nearby values) and cycles (disparity of nearby values) in one test. The null hypothesis – that the values in Figure 31 are random – cannot be rejected, and the probability to observe the given values by chance given that the null hypothesis is true is 0.77.

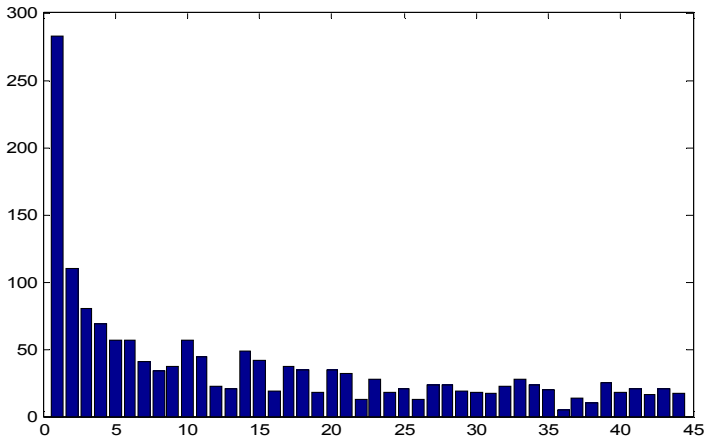


Figure 31. Number of new users per week, adjusted for warm-up, six logs combined

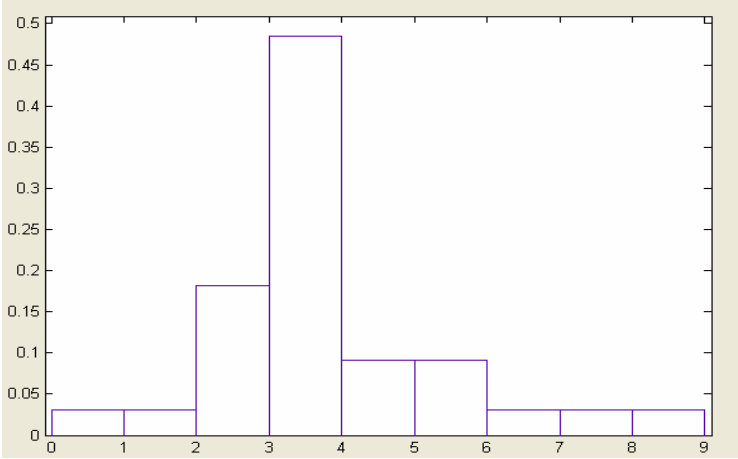


Figure 32. Number of new users per week, which were not active in the beginning of the log

Second, it can be modeled independently of the model's parameters. The correlation between the number of new users not active when the log began and the log's machine size, for the six logs used to build Figure 32, is -0.05. The correlations with Cm, JD and CL are slightly higher (around 0.35), but we can still afford to ignore them, since they will have a chance to affect the total number of users in the system by affecting the number of users who are already active when the log begins.

Figure 32 is therefore our model's distribution of the number of new users per week. Applying distribution fitting results in a good fit to the Log-Logistic distribution, with $\mu=1.25$ and $\sigma=0.2214$.

11.3. Number of active users at startup

We assume that those users in the first 11 weeks, which are above the average expected number of new users, are users which were already active when the log began (but not necessarily had their first session start on week 1). The number of these users varies by log, and correlation-wise is mostly affected by load: High load workloads tend to have *less* users. This correlation is more evident when looking at the number of active users at startup, then when looking at all users during the log. Allowing the load to affect only the users active at startup does not cause this effect to diminish quickly, since these users remain a significant part of the workload for a long time. For example, in the 32-months-long BLUE log the users which appeared in the first four weeks account for 20% for all users through the log, and this ratio is 47% for the 24-months-long LANL CM5 log.

The best fit between the number of active users at startup (AUS) and the load was found to be linear, as given in Equation 13. Since the correlation is not perfect, the residuals are quite high, and this also adds randomness to the modeled number of active users on startup. The residuals are uniformly distributed, based on the empirical values from the production logs. Note that AUD may be negative for high loads, an issue that we'll discuss in section 14.1.

$$AUS = -543.37 \times CL + 481.77 + R$$

$$R \sim U(-85, 85)$$

Equation 13. Modeling the number of active users at startup

11.4. User Classes

The next step is to draw the user class, or cluster, for each user. We rely heavily on this decision: In the session model, each distribution will be provided separately for each user class. We also assume that all correlations and some of the temporal relations between session variables will be reflected in the model this way. For example, we will not directly model the correlation between a user's number of sessions and average parallelism (heavy users tend to submit more highly parallel jobs). Instead this will emerge from the fact that the distributions of both number of sessions and parallelism for the two 'heavy' user types will have higher probabilities for larger values.

This approach has two advantages over direct modeling of all dependencies. First, it reduces the number of dependencies and temporal relations that need to be modeled – beyond elegance, the number of constraints deems direct modeling of all pair-wise relations impractical anyway. Second, it attempts to model directly the root cause of the dependencies, instead of just imitating them statistically. In other words, having many interactive jobs and starting them on weekdays are related but probably do not cause each other – more likely both of them are caused by the fact that the user submitting them is a light-weight, long-term user of the machine. Modeling the cause directly increases the likelihood that dependencies not modeled directly will self-emerge, and also adds to the "insight value" of the model: explaining *why* in addition to *how much*.

The overall distribution of user classes is given in Figure 25. However, we have to check if this distribution depends on the values we have already selected – The model's parameters and the number of users – and also if there is some kind of temporal correlation between its values. It turns out that this distribution depends heavily on the model's parameters – rounded correlations are presented in Table 29 for all seven logs – and also exhibits temporal locality.

	% of long-term heavy users (%LTHU)	% of long-term light users (%LTLU)	% of short-term weekend users (%STWE)	% of short-term workday users (%STWD)
Median CPU Time	0.4	-0.7	0.9	0.0
Jobs per Day	-0.6	0.5	-0.2	-0.2
CPU Load	0.7	-0.6	0.1	0.3
Machine Size	-0.6	0.7	-0.4	-0.4

Table 29. Correlations between model parameters and % of each user class

Equation	R^2
$\%LTHU = 5.979147\times10^{-5}\times Cm + 0.45250\times CL - 0.084420$	0.57
$\%LTLU = -26.2512\times10^{-5}\times Cm - 0.53006\times CL + 0.944925$	0.63
$\%STWE = 21.47628\times10^{-5}\times Cm - 0.08845\times CL + 0.087351$	0.84
$\%STWD = -1.20394\times10^{-5}\times Cm + 0.16601\times CL + 0.052141$	0.12

Table 30. Model Distribution of user classes given by the Cm and CL parameters

To account for these correlations – and define the main way in which the model's parameter affect the model – user classes will be drawn from the distribution defined in Table 30. Values are based on linear regression using the Cm and CL variables, which gave the highest R^2 values. Note that they imply that a higher load is achieved mainly by replacing long-term light users by long-term heavy users, and a bigger jobs workload (in terms of CPU work) is achieved by replacing long-term heavy users with occasional weekend users, and some long-term light users.

The temporal structure of user classes does not have a daily or weekly cycle, but some locality is evident. Table 31 presents the empirical probability of a user being of each class, based on the class of the previous user (i.e. the empirical Markov chain). The consistent feature is that whatever the previous user's class was, the current user has a higher chance of being of the same class than the general distribution would suggest. That higher chance is similar across all four user classes, and using a least squares approach to minimize the difference from the empirical distributions, we found it to be 10%. Thus, the class of each user is decided as follows:

- In 10% of the cases, it is the class of the previous user
- In the other 90%, it is drawn according to the distribution dictated by Table 30.

Class of previous user:	*Empirical distribution class of current user:*			
	%LTLU	%LTHU	%STWE	%STWD
Long term, light user	**60.16**	18.00	7.76	14.08
Long term, heavy user	45.10	**33.88**	10.20	10.82
Short term, weekend user	52.34	13.19	**19.15**	15.32
Short term, workday user	52.54	14.97	10.45	**22.04**
All classes:	55.19	20.35	9.76	14.70

Table 31. Empirical distribution of user class, based on the class of the previous user

12. Modeling Session Arrivals and Class

12.1. Analysis of Session Arrivals

Correctly identifying the factors that control the distribution of session arrivals and session classes lies at the core of our workload model. Since the job layer of our model reduces to mainly drawing job attributes from narrow distributions defined by the current session's class, the sessions model becomes "responsible" for generating most of the interesting features of the model: long-range dependence, heavy tailed inter-arrival times, cycles, and correlations between job attributes to themselves (runtime and parallelism) and to temporal features (such as the time of day). The following analysis provides a basis as well as new insights for all of them.

We begin by analyzing the number of sessions per week, since this number is independent of the daily and weekly cycles. This number is decided per user and per week, and potentially depends on every value selected before it. Since we have accounted for the model parameters and inter-user dependencies by embedding them in the distribution of the user class, the number of sessions per week can only depend on the user class and the user's number of sessions in the previous weeks.

The data reveals that the number of sessions per week depends heavily on the user class and on the user's week of activity. The week of activity is defined using "human weeks" – starting on Monday at midnight – so if a user ran her first job on Friday noon and the second one on the next Tuesday morning, the second job occurred in this user's second week of activity. Figure 33 plots the empirical distributions of the number of sessions per user class and per week of activity, showing this dependency. Not surprisingly, users are more active when they begin using the computer. This is the most significant temporal influence on the number of sessions: There is a correlation between the number of sessions in a week to the number of sessions in the previous weeks, but this correlation is caused by the effect of the week of activity: High numbers are expected to be clustered at the beginning of each user's activity and gradually decrease.

Figure 33 visually presents the dependency of the number of sessions on the user's class and week of activity. It is based on information from all users of all logs, but accounts for the variation in log durations and the fact that users arrive in the middle of the log. For example, if a user arrives at week 50 of the Paragon log, then we only have five weeks of information about that user, since that log only has 54 weeks. To account for this, the average for each week of activity in Figure 33 is computed by normalizing it for the number of users for which information is available for that week.

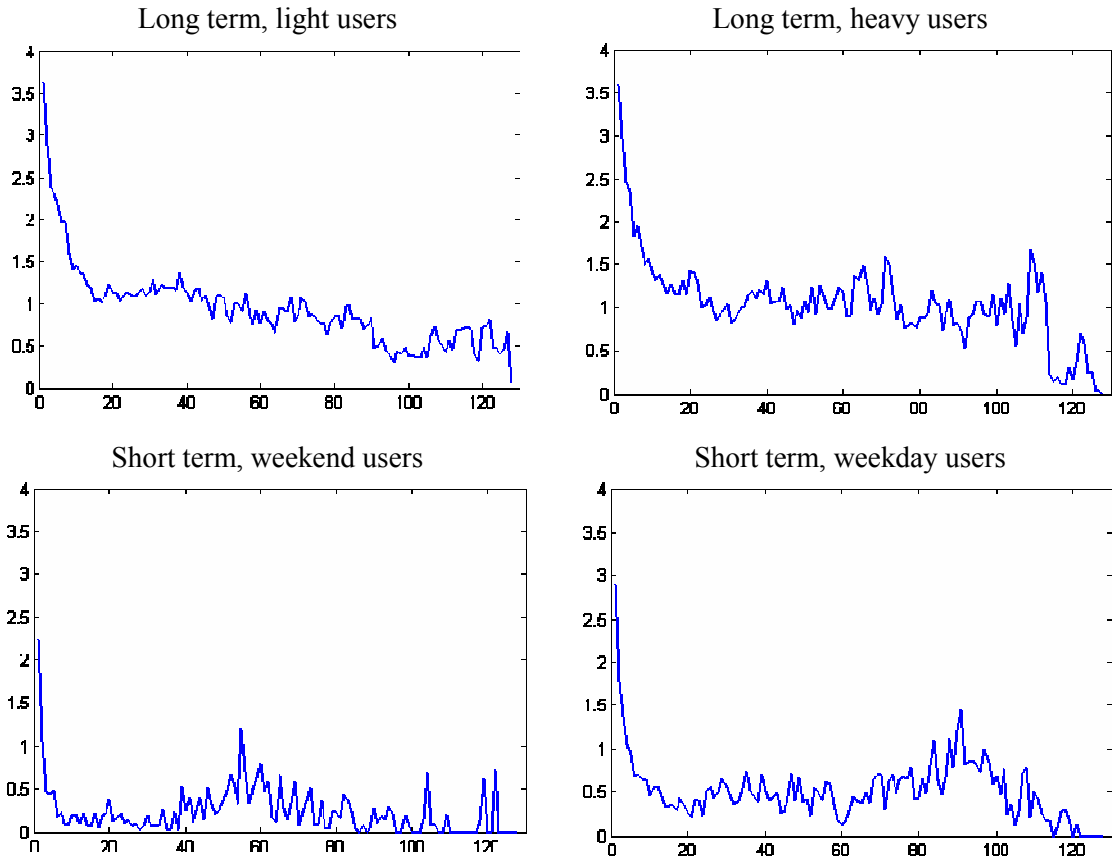


Figure 33. Average number of sessions per week, as a function of user class and week of activity

We don't know the distribution of a user's number of active weeks – only the censored distribution up to each log's duration – but several consistent features can be noted. The activity pattern of the two long-term user classes is similar, and all user classes start with a high level of activity which quickly drops to a constant or slowly decreasing rate. The non-monotonic "humps" for the two short-term user classes originate from users in the BLUE log, and do not seem to representative. The 'short-term' user classes earn their name from a very high empirical probability (> 0.5) of disappearing after just one or two weeks of activity, but there's still a small chance of seeing the same user active even after two years, often after many weeks of inactivity. This translates to a heavy-tailed distribution of inter-arrival times for each user, and also creates long-range dependence since the jobs of sessions of the same user, even months apart, will be correlated.

The data in Figure 33 led us to two modeling decisions. First, once a user enters the system it would never leave it – modeling the (high) probability of being inactive after 20 weeks of activity is simpler and more accurate. Second, since the probability of inactivity in any given week is high (except the first week of a user's activity, when it is by definition zero), we will model separately the probability of inactivity (having zero sessions in a given week) and the distribution of the number of sessions given activity (having at least one session). The next two sections model these two distributions.

Another key detail is that Figure 33 doesn't include data from all users, but only users for which we (approximately) know their first week of activity: users that arrived from the 12th week of activity in each log onwards. These are also the only users used to build the inactivity and session count models in the next two sections. Section 12.4 deals with the problem of modeling the week of activity of users who are already active at startup, which we must address since our user model includes them. The final section of this chapter models the distribution of session classes.

12.2. Inactivity

Figure 34 plots the distribution of inactivity – starting zero sessions in a given week – as a function of the user's class and week of activity. This analysis takes into account the fact that information about users' inactivity is censored: we don't know if a user is inactive after the end of its log. For each week, only users for which information is available are included in the analysis, and only weeks for which there are at least 50 such users were used to build the equations.

The probability of inactivity given the week of activity best fits a logarithmic curve for all user classes, and the best fit parameters are given in Table 32. The logarithmic fit needs to be adjusted for the first weeks of activity, where the empirical probability is consistently lower than it predicts, and for the very late weeks of activity, where the probability has to be kept below 1.0.

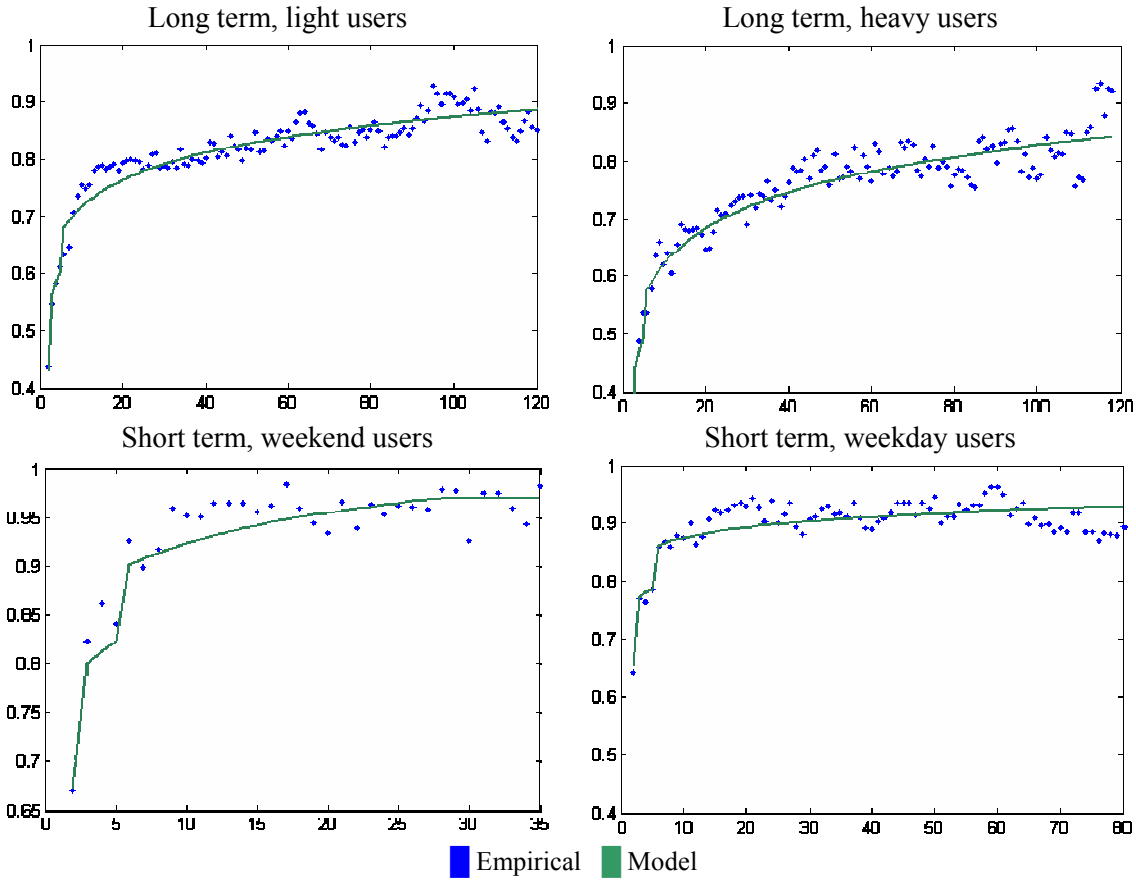


Figure 34. Probability for zero session per week, given user's week of activity and class

$$P(S=0\mid WoA,UC)=\begin{cases} a\cdot\ln(WoA)+b-0.18 & if \quad WoA=2 \\ a\cdot\ln(WoA)+b-0.07 & if \quad 3\le WoA\le 5 \\ \min(a\cdot\ln(WoA)+b\ ,0.97) & if \quad WoA\ge 6 \end{cases}$$

Where a,b are determined by UC as given in Table 32

Equation 24. Probability of a user's inactivity given the user's week of activity and class

Equation	R^2
P(S=0|Woa,UC=LTLU) = 0.0683×Ln(WoA) + 0.5594	0.82
P(S=0|Woa,UC=LTHU) = 0.0895×Ln(WoA) + 0.4149	0.84
P(S=0|Woa,UC=STWE) = 0.0451×Ln(WoA) + 0.8201	0.60
P(S=0|Woa,UC=STWD) = 0.0254×Ln(WoA) + 0.8164	0.31

Table 32. Unadjusted probability of inactivity given the user's week of activity and class

12.3. Number of sessions per week

The next challenge is to model the distribution of a user's number of sessions per week, given the user's class and week of activity, and given that the user has at least one session that week. This is a complete distribution for each combination of user class and week of activity (rather than a single number), as shown in Figure 35 for the case of Week of activity = 1. Fortunately, results from performing distribution fitting on all of these combinations using the relevant empirical data shows that the number of sessions per week can be modeled using the same distribution, and the user class and week of activity control that distribution's parameters in a predictable way.

Distribution fitting was done using EasyFit (MathWave, 2006), which supports 42 discrete and continuous distributions, mostly using maximum likelihood estimators, and computes the Kolmogorov-Smirnov, Anderson-Darling and Chi-Squared goodness of fit tests to compare them. The program's inputs were a datasets of for different user class and week of activity combinations, using only users who arrived after the 12th week of their log, so that their first week of activity is known. The program was set to assume a lower bound of one when estimating parameters, therefore results are adjusted to fit the distribution of the number of sessions given that it is at least one. There are 360 combinations of user class and week of activity where there is information for at least 50 users in that week of activity, and after manually analyzing a third of them in EasyFit, it became clear that one distribution should be used to model all cases.

$$f(x)=\begin{cases} \frac{1}{\sigma}\cdot\left(1+k\cdot\frac{x-\mu}{\sigma}\right)^{-1-\frac{1}{k}} & k\neq 0 \\ \frac{1}{\sigma}\cdot\exp\left(-\frac{x-\mu}{\sigma}\right) & k=0 \end{cases}$$

Equation 35. Probability density function of the Generalized Pareto distribution

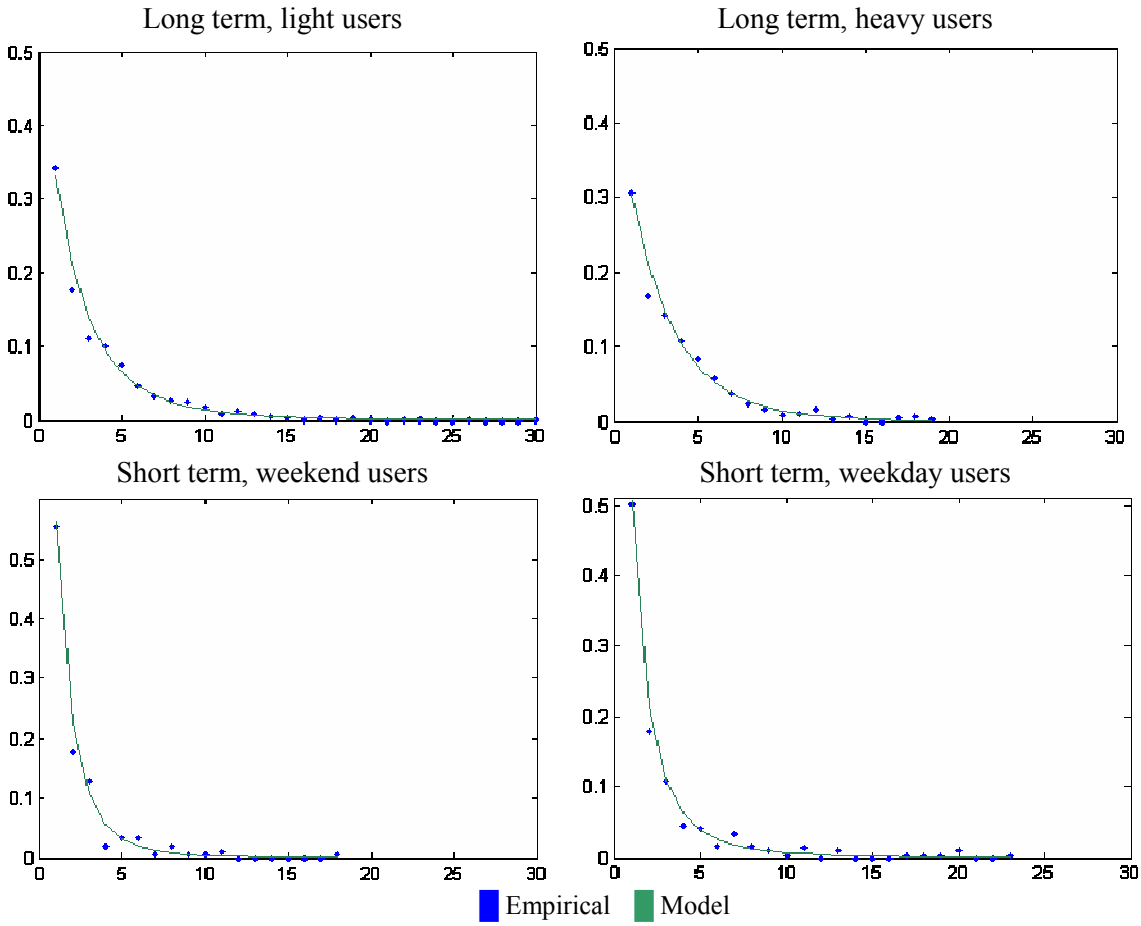


Figure 35. Empirical distribution of the number of sessions for the first week of activity

The *Generalized Pareto distribution* (Kotz and Nadarajah, 2001) is almost consistently the best distribution to describe the number of sessions per week users have. It is marginally outperformed by the Wakeby distribution, but Generalized Pareto was preferred because it has 3 parameters instead of 5 and they are easier to understand, so the marginal gap did not warrant the extra complexity.

The distribution's three parameters are shape k, scale $\sigma > 0$ and location μ. If $k = 0$ and $\mu = 0$, the generalized Pareto distribution is equivalent to the exponential distribution, and if $k > 0$ and $\mu = \sigma$, it is equivalent to the Pareto distribution. The shape parameter dictates how "heavy" the distribution's tail is: when $k > 0$, the tail decreases as a polynomial; when $k = 0$, it decreases exponentially; and when $k < 0$, the tail is finite.

To model the number of sessions, manual analysis suggested setting the location parameter μ to 0.6 for all cases – this was the approximate MLE for most cases. An automated script was used to produce MLE parameter estimates for k and σ for every combination of user class and week of activity. Then, a linear fit was found to be a good predictor of how these parameters depend on this combination. Table 33 summarizes the results.

Number of Sessions per Week given user class:	Is distributed by Generalized Pareto with parameters μ=0.6 and:
Long term, light user	K = 0.11 Sigma = -0.0097×WoA + 4.74
Long term, heavy user	K = 0.00076×WoA + 0.013 Sigma = -0.0166×WoA + 3.31
Short term, weekend user	K = -0.0251×WoA - 0.147 Sigma = 0.0168×WoA + 3.806
Short term, workday user	K = -0.0178×WoA + 0.337 Sigma = -0.1030×WoA + 2.897

Table 33. Model of the number of sessions as a function of user class and week of activity

The results provide some intuition about how the activity pattern of different users evolves over times. The number of sessions of the two long-term user classes has a polynomial tail, which is constant over time for light users, and start smalls but gradually grows for heavy users. The scale for both long-term user classes decreases over time, and is generally higher for light users. Short term weekday users start with the heaviest tail of all user classes, which very slowly decreases. On the other hand, their scale parameter is the smallest of all user classes, and it also decreases rapidly – about ten times faster than the rate of decreases in other user classes. The short-term weekend users' distribution is the only one with a finite tail – which shrinks further over time.

Table 34 presents parameter estimates specifically for the first week of activity. These numbers somewhat differ from the numbers computed from Table 33 – goodness of fit tests suggest that the first week of a user's activity is unique, and may warrant separate modeling. Table 34 is such a separate model, and the model in Figure 35 is based on it rather than on Table 33. While this clearly provides a more accurate fit, empirical studies are required to decide if this extra complexity is required in practice, for example to correctly compare schedulers.

Number of Sessions per Week given WoA=1 and user class:	Is distributed by Generalized Pareto with μ=0.6 and:	
	k	σ
Long term, light user	0.1795	2.5012
Long term, heavy user	0.0557	2.8330
Short term, weekend user	0.3008	1.1597
Short term, workday user	0.4661	1.2969

Table 34. Model of the number of sessions given user class in the 1st week of activity

12.4. Week of activity for active users at startup

The two most important properties of a user in our model are class and week of activity. However, for the large group of users who were already active at startup, we do not know their actual week of activity. We removed these users from the dataset to model the number of sessions per week, but when generating a synthetic workload, a current week of activity must still be assigned to each user generated according to equation 13 from section 11.3.

Assigning each such user with a week of activity of 1 is a mistake, since it would cause them to create a much higher load than empirically evident. In the same way, assigning the week of activity by drawing from a uniform distribution between 1 and 140 weeks (or some other arbitrary upper bound) is erroneous as well, since most users produce very little activity after 50 weeks for the long-term user class or 20 weeks for the short-term user classes. To account for the fact that users are typically more active during their first weeks of activity – and accordingly skew the distribution of the week of activity for active users at startup – we need an accurate measure of the distribution of a user's week of arrival, given its user class and the fact that s/he has at least one session in the next 12 weeks (that's how users active at startup are defined). Fortunately, this distribution can be easily described using distributions that we have already modeled.

Let *WoA* be the random variable that states a given user's week of activity, T_w the Boolean random variable stating if the user has at least one session in the twelve weeks starting at week w, S_w the random variable that counts the number of sessions the user has in week w, *UC* the random variable stating the user's class, and W_{max} the dataset's maximal number of weeks. Then:

$$P(WoA = w \mid T_w, UC = c) = \frac{P(T_w \mid WoA = w, UC = c) \cdot P(WoA = w \mid UC = c)}{P(T_w \mid UC = c)}$$

$$= \frac{P(T_w \mid WoA = w, UC = c) \cdot P(WoA = w \mid UC = c)}{\sum_{k=0}^{W\max} P(T_k \mid WoA = k, UC = c) \cdot P(WoA = w \mid UC = c)}$$

$$= \frac{P(T_w \mid WoA = w, UC = c)}{\sum_{k=1}^{W\max} P(T_k \mid WoA = k, UC = c)}$$

$$= \frac{1 - P(not\ T_w \mid WoA = w, UC = c)}{\sum_{k=1}^{W\max} \left[1 - P(not\ T_k \mid WoA = k, UC = c)\right]}$$

$$= \frac{1 - \prod_{x=w}^{w+11} P(S_x = 0 \mid WoA = x, UC = c)}{\sum_{k=1}^{W\max} \left[1 - \prod_{y=k}^{k+11} P(S_y = 0 \mid WoA = y, UC = c) \right]}$$

Equation 46. Probability that a user active at startup is at a given week of activity

The last expression only involves the probability of inactivity given a user's class and week of activity, which is exactly what we modeled by equation 14 in section 12.2. It is therefore possible to compute these probabilities directly. However, the computed results pose a problem: Based on our inactivity model from section 12.2, a user's week of activity has a small but positive chance to be well over 200 (circa four years). This is problematic because such results are only based on interpolation – we don't have any concrete data on users that "old"; they are "out of bounds" for the models of inactivity and number of sessions, and may produce unpredictable results; and they are non-realistic, since all our logs were started not long after the computer was in operation. This issue is addressed by adding the a-priori realistic assumption that all users existing at startup have been active for at most one year (52 weeks). Under this assumption, the distribution of the week of activity for users existing at startup becomes:

$$P(WoA = w \mid WoA \leq 52) = \frac{P(WoA \leq 52 \mid WoA = w) \cdot P(WoA = w)}{P(WoA \leq 52)}$$

$$= \begin{cases} 0 & w > 52 \\ \dfrac{P(WoA = w)}{P(WoA \leq 52)} & w \leq 52 \end{cases}$$

$$= \begin{cases} 0 & w > 52 \\ \dfrac{P(WoA = w)}{\sum_{x=1}^{52} P(WoA = x)} & w \leq 52 \end{cases}$$

Equation 57. Probability that a user is at a given week of activity, given that it is active for no more than a year

As equation 17 shows, a simple normalization by a known probability is required to compute an existing user's week of activity, given that the user is at active for at most one year. Figure 36 shows the resulting distributions, per user class. The values are computed based on the inactivity defined by equation 14, meaning that it correctly accounts for the censored distribution of users' inactivity, and that it only relies on data from weeks where data from over 50 users was available.

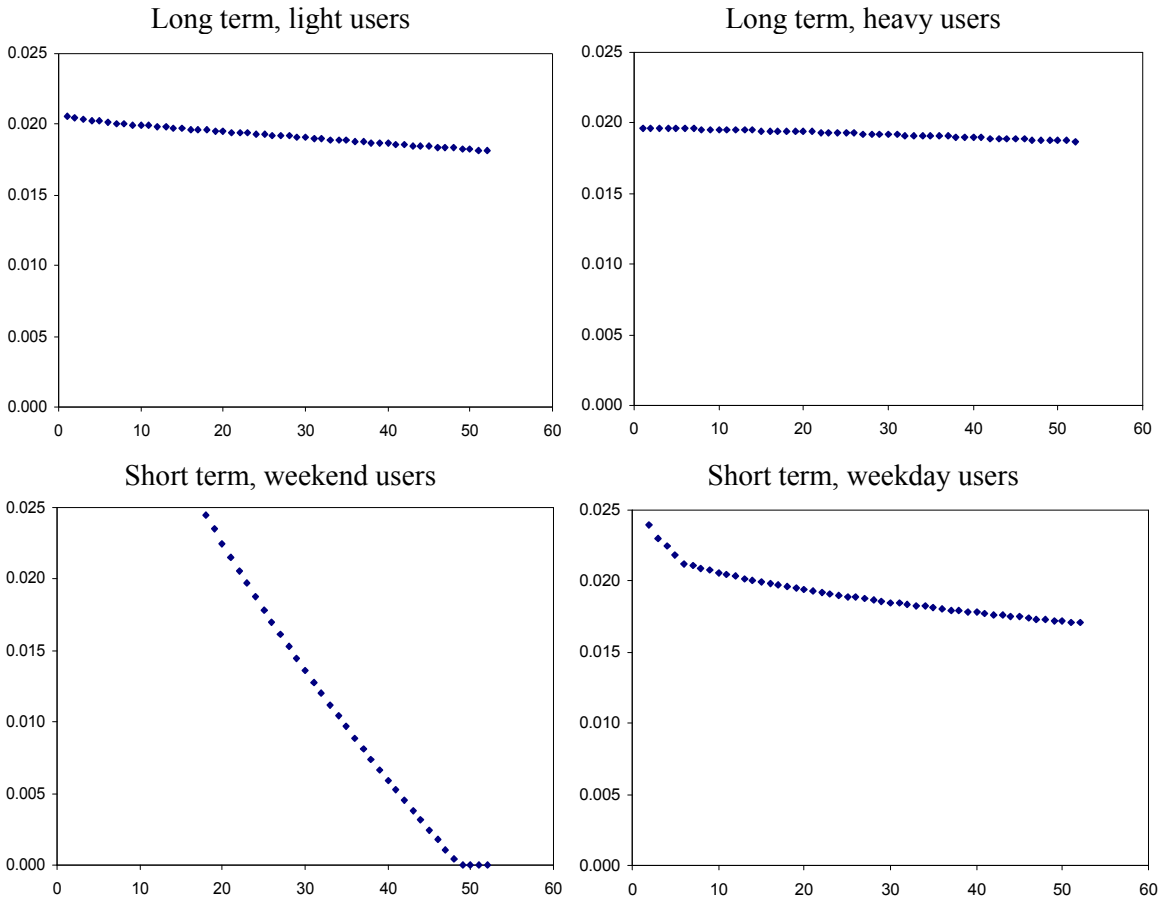


Figure 36. Model Distribution of the week of activity of users active at startup

12.5. Session classes

Given a user of a certain class and week of activity, and given that this user has *n* sessions this week, the next step is to determine the class of each session. The distribution of session classes may depend on the user's class, the user's week of activity, and the class of previous sessions of the same user. The results indicate that all three dependencies exist, in a non-negligible way.

Table 35 summarizes the distribution of session classes given the user class and previous session class. The dependency on the previous session class is the only strong temporal dependency which was identified, making a Markov chain model a natural choice. All sessions from all users (not just those who arrived from the 13th week) were used to produce these tables. The number of sessions for each user class and session class is also given, as an indicator of the results' robustness.

The main consistent observation that can be made from Table 35 is that a session's class is more likely to be of the same class as the previous session of the same user than the general distribution would suggest. This is true for all session classes, but is especially strong for the batch session classes: a Batch Highly Parallel or Batch High Duration session has a 44% chance of coming after a session of the same class. Only 4% and 3% of all sessions are of these two classes respectively, meaning that sessions of these classes tend to appear in groups.

		This session's class					# of Sessions
		IW	BHP	IWD	BHD	IWD	
Previous s. class	IW	27%	2%	29%	2%	39%	30,220
	BHP	10%	44%	18%	3%	25%	5,682
	IWN	21%	3%	45%	4%	27%	41,269
	BHD	11%	4%	30%	44%	11%	4,192
	IWD	20%	2%	19%	1%	58%	61,811

Table 35a. Empirical Markov chain of session classes, ***independent of user class***

		This session's class					# of Sessions
		IW	BHP	IWD	BHD	IWD	
Previous s.	IW	27%	2%	24%	1%	46%	19,339
	BHP	12%	31%	19%	2%	35%	2,831
	IWN	21%	3%	39%	2%	35%	21,427
	BHD	17%	5%	33%	23%	21%	1,060
	IWD	19%	2%	17%	1%	62%	47,684

Table 35b. Empirical Markov chain of session classes, ***given user class = LTLU***

		This session's class					# of Sessions
		IW	BHP	IWD	BHD	IWD	
Previous s.	IW	26%	3%	39%	4%	29%	8,088
	BHP	7%	58%	16%	4%	15%	2,707
	IWN	20%	3%	53%	7%	17%	15,010
	BHD	8%	3%	28%	53%	7%	2,967
	IWN	23%	4%	25%	2%	46%	10,700

Table 35c. Empirical Markov chain of session classes, ***given user class = LTHU***

		This session's class					# of Sessions
		IW	BHP	IWD	BHD	IWD	
Previous s.	IW	46%	1%	21%	1%	30%	827
	BHP	23%	20%	37%	7%	13%	30
	IWN	25%	2%	42%	3%	28%	645
	BHD	28%	3%	33%	18%	20%	40
	IWD	27%	1%	20%	1%	51%	939

Table 35d. Empirical Markov chain of session classes, ***given user class = STWE***

		This session's class					# of Sessions
		IW	BHP	IWD	BHD	IWD	
Previous s.	IW	29%	1%	46%	1%	24%	1,966
	BHP	16%	22%	27%	7%	28%	114
	IWN	21%	1%	53%	1%	23%	4,187
	BHD	17%	5%	35%	22%	21%	125
	IWD	18%	1%	40%	1%	39%	2,488

Table 35e. Empirical Markov chain of session classes, ***given user class = STWD***

Apart from this locality, the previous session class seems to have a different effect on different user classes: $P(SC \mid UC,PSC) \neq P(SC \mid UC) \neq P(SC \mid PSC)$. Numerous approaches to simplify Table 35 into a model with fewer parameters result in significant deviations from the empirical distributions, and since the proper selection of session classes is so central to our model, this is a risky distribution to over-simplify. It is interesting by itself that some of the intuitive approaches to simplify the Markov chain don't always work:

- Drawing from the distribution given the previous session class alone, with a constant modifier per user class; or drawing from the distribution given the user class alone, with a constant modifier per previous session class
- Using a single constant or a per-user-class constant that determines the likelihood of using the previous session class again, otherwise drawing it from a global distribution
- Using a hidden Markov model assuming that the user is either in 'Interactive' or 'Batch' mode, with constant in-mode session class distributions and per-user-class probabilities of switching between the two modes

To conclude, it seems that the Markov chain is a unique property of each user class, and should be directly modeled as such. For example, a long-term heavy user is almost three times more likely to have two consecutive highly parallel batch sessions than a short-term weekend user, who in turn is 2.5 times more likely to have a batch session after an interactive night session than a short-term weekday user is.

The distribution of session classes depends on one more variable – the user's week of activity. Figure 37 presents the distribution of session classes, independent of the user class and previous session class, at three different stages of a user's lifecycle. Consistently, as the user's week of activity rises, the proportion of both batch sessions increases, at the expense of the interactive daytime sessions. A user who has been active on the machine for over 60 weeks is four times more likely to start a batch (BHP/BHD) session than a user who has been active for less than nine weeks. This is not surprising: The more experienced a user is with the system, the more s/he tends to use it heavily. New uses, on the other hand, apparently take the time to learn the system and do test runs, which results in a higher number of sessions in the first weeks of activity, which are mostly interactive. The interactive weekend sessions are the only ones who are unaffected by the week of activity. Table 36 presents the regression results which best model this dependency.

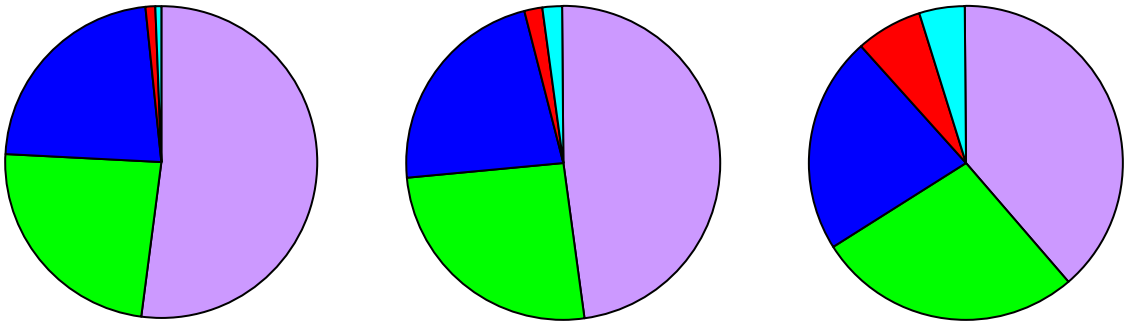


Figure 37. Distribution of session classes given the session's user's week of activity

Equation	R^2
%IWD = -0.0022×WoA + 0.5107	0.90
%IWN = 0.0011×WoA + 0.2437	0.64
%IW = 0.2161	N/A
%BHP = 0.0006×WoA + 0.0182	0.74
%BHD = 0.0005×WoA + 0.0113	0.76

Table 36. Model distribution of session classes, given the session's user's week of activity

Repeating this process for specific user classes and previous session classes suggests that the effect of the week of activity is roughly independent of these two variables. Therefore, to define our complete model for selecting session classes, the distributions in tables 35 and 36 need to be combined. This is done by starting from the distribution as dictated by the week of activity, and using Table 36 to define not the direct distribution, but rather the adjustment from the global distribution that the user class and previous session class dictate:

$$P(SC \mid WoA,UC,PSC) = P(SC \mid WoA) + [P(SC \mid UC,PSC) - P(SC)]$$

Equation 18. Distribution of session classes, given all dependencies

Where $P(SC \mid WoA)$ is given by Table 35, $P(SC)$ is given by Figure 23 in section 9.3.2, and $P(SC \mid UC,PSC)$ is given by Table 35. For example, for a long-term light user in the 10th week of activity who just finished an interactive weekend session, the probability that the next session will be an interactive daytime session is -0.0022×10+0.5107+[0.46-0.43] = 51.87%. Note that since each of the three distributions always sums to 1.0, equation 18 always defines a valid distribution as well.

13. Modeling Parallel Jobs

13.1. Model Pseudo-code

Given the user and session models, we now turn to modeling the generation of a stream of parallel jobs. This integrates together the model's three layers, in a way that is required to be "as simple as possible, but no simpler" – that is, a simple method that still captures all the statistical attributes which are considered important. Chapters 4 to 8 of this thesis define what is "important", in particular in the context of correctly evaluating parallel schedulers. Chapters 9 to 12 investigate the key empirical dependencies, ensuring that each correlation and auto-correlation is accounted for.

Listing 5 provides the complete model's pseudo-code. Whenever drawing from a distribution is required, a reference to the section that defines it is given in brackets. The first layer is the user model, generating the number of users and the class and week of activity for each one, based on the model's parameters. This layer is responsible for addressing these limitations of current workload models, as summarized in chapter 6: direct user modeling, load manipulation, parameterization, and an aspect of long-range dependence (since a user's class remains constant over time).

The second layer of the model is the session model, generating the number of sessions per user and per week, and the class of each session. This layer is responsible for generating three statistical phenomena. The first is self-similarity – users have an explicit "on-off" behavior, with a heavy tailed "off" duration distribution caused by our week-long inactivity model. The second is the correlation between individual job attributes: The correlation between jobs' runtime and parallelism, for example, is not modeled directly but rather by having both attributes depend on the session class. The dependency between these attributes and the time of day, day of week and inter-arrival time between jobs is created in the same manner. And third, the session model is responsible for the daily and weekly cycles.

Cycles are created as follows. We need to model the distribution of sessions over days of the week and hours of the day, the number of jobs and repetitions inside each session, and the inter-arrival times of jobs within a session. All these distributions – in the empirical data as well as the model – implicitly assume that it is given that all separate sessions are indeed sessions, i.e. they are at least 20 minutes apart. To model these distributions and constraints correctly, this scheme is used: Draw the day of week, hour of day, number of jobs and their intra-session arrival times independently for each session – and only check for collisions after all sessions are drawn. If any collisions exist, erase everything and redraw all sessions again.

```
GenerateWorkload (Model parameters, duration in weeks):
    Draw number of active users at startup [11.3]
    For each such user
        Draw its user class [11.4]
        Draw its week of activity [12.4]
    For each week
        Draw number of new users this week [11.2]
        For each new user
            Draw its user class [11.4]
            Week of activity = 1
        For each active user
            If user is active this week [12.2]
                Draw its number of sessions this week, given activity [12.3]
                GenerateWeeklySessions(user, week, number of sessions)

GenerateWeeklySessions (user, week, number of sessions)
    While true
        For i = 1 to number of sessions
            Draw the i'th session's class [12.5]
        For attempts = 1 to 100 do
            NewSessions = new empty list of sessions
            For i = 1 to number of sessions
                NewSessions.Add ( GenerateSession(i'th session's class) )
            If not CollisionsInSessionsList(NewSessions) then
                Return NewSessions

GenerateSession(session class) [13.2]
    Create a new session
    Draw the session's start day of the week
    Draw the session's start time of the day (hour, minute, second)
    Draw the number of unique job sequences in the session
    For each unique job sequence
        Draw its runtime, parallelism and number of repetitions
        For each repetition
            Draw the inter-arrival time from the previous job, and add job to session
    Return the session

CollisionsInSessionsList (sessions)
    For every pair s1, s2 in sessions
        If s1.StartTime <= s2.StartTime
        and s1.EndTime > s2.StartTime – 20minutes
        then Return true;
    Return false;
```

Listing 5. Pseudo-code of the User-Based Parallel Workload Model

To intuitively understand why this is a valid approach, consider the following analogy. Assume that you are required to model a Bernoulli experiment with p=0.5, given a three-sided cube that can be used to draw 0, 1 or 2 with p=0.33 for each result. Then, using the cube and re-throwing until the result is not 2 yields the desired model. Similarly, re-drawing session start and end times until we get a valid set of 5 IWD sessions and two IWN sessions in one week (for example) is valid, since the modeled number of sessions as drawn from the empirical dataset also implicitly assumes that this is the number "given that these are legally defined sessions".

The only problem with this approach is that there is a small probability to draw a set of session classes which can't normally fit within a one week in reality (for example, five high-duration batch sessions). To handle this edge case, the *GenerateSession* function from Listing 5 will only make 100 attempts at drawing a valid session set after drawing the session classes, and redraw the session classes if all of them fail. Again, we do not consider this a distortion of the overall distribution, since such cases do not (and can not) exist in the empirical distribution.

The *CollisionsInSessionsList* function tests whether a list of a user's weekly sessions includes a collision. Note that as defined it covers all cases: Two sessions that partially overlap, a session that consumes another session, or two sessions that start at the very same time. This function should also check that this week's sessions do not collide with sessions that began last week and ran into this week – this is not included in the pseudo-code to improve readability.

13.2. Modeling Jobs within a Session

The third layer of the model is the jobs model, defined by the *CreateSession* function from Listing 5. It is responsible for creating locality of sampling, and potentially flurries. Due to our construction, this is the simplest layer, for two reasons:

- It directly depends only on the session class – not on the user, previous sessions or jobs.
- Each of its attributes directly depends only on the session class – not on one another.

This simplicity at the jobs layer justifies the exhaustive modeling of dependencies at the user and session layers. It means that modeling the distributions of each of the intra-session attributes is a simple matter of fitting the empirical distribution of each attribute, per session class. There is no need to consider the temporal structure of each attribute, or dependencies to the others.

As demonstrated in sections 5.1 and 8.5.1, sessions typically exhibit high locality of sampling, which can be effectively modeled by generating jobs' attributes from a global distribution,

and then repeating them by a number of times distributed by a Zipf-like distribution (Feitelson, 2006). This approach has been previously applied to model locality of sampling: The model of (Feitelson and Jette, 1997) repeats jobs but assumes a zero think time between repetitions, and (Song, Ernemann, and Yahyapour, 2004) use a Markov chain model which accounts for both repetitions and correlations between subsequent unique job sequences, but does not result in a Zipf-like distribution of repetitions. By modeling the correlation between runtime and parallelism using the session class, which narrows the distributions of these two variables in the same direction for each session class, it becomes simpler to directly model the number of unique job sequences and repetitions by fitting the empirical distributions.

The *CreateSession* function can easily be used to produce flurries. If it is desirable to include flurries in the model, then this function should begin by deciding whether this session is a flurry – and if so, all values should be taken from separate distributions and parameters. The exact modeling of flurries is outside the scope of this work, and is a challenging task with the current datasets because there are very few known flurries, which are very different.

DISCUSSION AND CONCLUSIONS

14. Discussion

14.1. Using the Model

The obvious next research task regarding the user-based parallel workload model is validation: Both statistical validations to compare features that were not modeled directly, such as the overall distribution of inter-arrival times and the correlation between runtime and parallelism, as well as practical validations, such as comparing how the adaptive and prediction-based schedulers perform on model-generated workloads versus the production logs. This work is outside the scope of this document and will be published separately. This section discusses guidelines that govern such work, as well as any other application of this model.

The first guideline is to *generate multiple instances* of the model and rely on their average performance, rather than relying on a single generated workload. Using the early workload models which rely on generating values from distributions, scheduling simulation results would be very similar on a workload of 10,000 jobs, 50,000 jobs or 150,000 jobs. Due to the lack of long-range dependence, the variance of aggregated values – for example weekly or monthly – of all the distributions used diminished very quickly. On the other hand, the user-based workload model is similar to production logs in the sense that certain weeks and even full months may be high-load, low-load or possess some other similarity. Due to the many long-range dependencies, subsequent generations of two workloads from the model may be very different from one another. This reflects what should be expected from real workloads, but in contrast to the limited number of production logs, it is easy to generate fifty two-year workloads from our model, using the same underlying parameters. Simulation studies should report the average results, which should converge if the simulated algorithm performs consistently on a variety of realistic scenarios.

The second guideline is to initialize the model with *parameters within its range*. The model is based on the largest dataset used to build a workload model to date, and includes machines from 100 to 2,048 processors, load levels from 40% to 80%, and 80 to 300 jobs per day. Values in this range represent the majority of parallel computers deployed today, and slowly become more popular as machines of this capacity gradually move from the World's Top-500 list to be the mainstream shared departmental computational workhorse, in both academia and industry.

However, there is no evidence that the model is representative for values outside these ranges. This warning relates in particular to load, since many researchers tend to simulate algorithms on a variety of loads reaching 95% and above – using questionable methods to create that load, leading to questionable results. Equation 13 from section 11.3, used to generate the number of active users at startup as a function of the load, may result in a negative number of users for very high loads – this can be manipulated by decreasing the range of the uniform distribution in that equation, but is not an inherent problem with the model, since it's not designed for such loads.

This is not a limitation of this model, but rather of the available datasets: We have no good data on how parallel workloads really behave on very high loads, and it seems such loads are never reached in reality. Other models cannot generate a high-load parallel workload as well: The user-based model can be manipulated by multiplying the inter-arrivals, runtimes or parallelism by the same factors, but this still would not result in a representative workload. As an analogy, this is similar to simulating high load on a highway by multiplying the load each minute of the day by a constant, while in reality high loads are caused by heavy morning traffic during workdays. Decisions based on faulty models are likely to be faulty as well.

The third guideline in using the user-based model is to use it to create workloads that are at most two years long. This wasn't an issue in previous models, which do not feature long-range dependence. However, since our model has identified the week of activity as a key indicator of a user's activity pattern, it is important to remember that out of the seven logs used to build the model, only three are at least two years long. Evidence that production parallel workloads can significantly change over time (Talby, Feitelson and Raveh, 1999) suggests that the core parameters of a system's workload may change over several years of activity. Therefore, the best practice for generating a very large number of jobs from the user-based model is to generate several separate one-year-long workloads, instead of one multi-year workload. Again note that this shouldn't be viewed as a deficiency of the model: Neglecting this issue altogether, other models to date cannot provide a more accurate workload looking three or five years ahead.

14.2. Extending the Model

Due to its layered structure and the fact that it deals with many factors not considered before, the user-based workload model provides a solid basis for several future research directions. The first is the modeling of additional parallel job attributes, such as the user runtime estimates, which as

shown in (Tsafrir et al., 2005) also exhibits high locality and fits well the intra-session model of repetitions. Other possibilities are the memory and I/O requirements of parallel jobs, or application activity patterns. Since the intra-session model is based on the assumption that repetitions are caused by a human user running the same application over and over again, it is likely that all features of this application will be repetitive.

A second extension is modeling user feedback. This approach models the fact that while this work and others assume that the workload is an external input given to a computer system, in reality human users adjust their behavior according to the system's responsiveness. When the system is clogged, users will submit fewer new jobs, and vice versa. Modeling the user feedback is a complex problem, but it must certainly be based on the decisions made by users, therefore a direct model of the system's active users and their major indicators of activity (i.e. user class and week of activity) provide a sound basis.

A third research direction based on this work is replacing the low layer to model something other than jobs – network or storage capacity, for example. The user and session layers of the model provide a generic model of how human users use a shared, powerful computational device. They can be used to describe just as accurately the network requirements on that shared device during a session, or the storage requirements for storage related to work on that device. The jobs layer of our model is simple since most of the complexity is handled in the user and session layers, and this simplicity can be readily "reused" to model other things.

14.3. Improving Cluster and Grid Resource Management

This work also opens future research opportunities in the area of on-line cluster and grid resource management. This happens because the new user- and session-based model provides more accurate answers than possible to date for two common questions.

The first is "*What workload is expected an hour from now?*" – which by analyzing the currently active sessions, and perhaps sessions that are likely to start soon, our model can answer with a narrower distribution than was previously available. This is done by assigning a class for each session and user, effectively summarizing the a-priori information about them in a usable form. It is potentially useful for many on-line algorithms: Deciding how many processors to allocate to a moldable job; if, when and how much extra load to accept from another member in a computational grid; whether and when to migrate jobs; what capacity to reserve for interactive work; and others.

The second question is "*What workload is expected a week from now?*" – answered by analyzing the active users, based on assigning a class to each of them and recording their week of activity. This is potentially useful for a questions such a capacity planning of CPU, storage and I/O, and longer-term reservations in a grid environment.

15. Conclusions

Re-quoting the introduction, the goal of this research is to provide new information, discovered by means of sound statistical techniques, which can benefit both worlds – synthetic workload modeling, and on-line resource management algorithms.

With respect to workload modeling, this work focuses on analyzing the core reasons behind important statistical features and modeling them directly, rather than fitting distributions and then tweaking them to model specific observations. This enables our model to address a much larger set of constraints and dependencies than previous works, and also provide insights on *why* they exist. The identification of the main factors that govern the activity of users of parallel machines is new, and has many potential implications. The description of session classes enables more accurate short-term prediction than available before, of use to on-line cluster and grid resource management. The analysis is based on the largest and most scrutinized dataset used to build a workload model to date.

With respect to parallel scheduling, the bottom-line impact of improving a scheduler by 15-30% – as the adaptive and prediction-based schedulers can do – cannot be underestimated. The scheduler is a software-only, relative independent module of the operating system, and so is relatively easy to change. Realizing the improvements does not require any user training or involvement, and is available to all jobs regardless of programming language, compiler and libraries used. Alternatives to achieving the same performance gains require very substantial investments of time and money, so it remains to hope that these ideas will be implemented and widely available.

In the past decade, research in these areas matured from understanding the fundamental factors that drive parallel schedulers – backfilling dynamics, temporal structure and the impact of workloads and metrics – to optimizations based on acumen of statistical attributes of parallel computer workloads. Discovering that such attributes can be exploited in practice leads to a need for new workload models, which in turn identifies new statistical features, and raise ideas for a new generation of schedulers. This work is a stepping stone in this process.

REFERENCES

Agrawala, A.K., Mohr, J.M. and Byrant, R.M. 1976. An approach to the workload characterization problem. *Computer 9(6)*, 18-32.

Arlitt M. 2000. Characterizing Web User Sessions. In *Perf. Eval. Rev. 28(2)*, 50-56.

Beran, Jan 1994. Statistics for Long-Memory Processes (Monographs on Statistics and Applied Probability), Chapman and Hall, New York, NY.

Bode B., Halstead D., Kendall R., Lei Z. and Jackson D. 2000. The Portable Batch Scheduler and the Maui Scheduler on Linux Clusters. In *Proceedings of the 4th Annual Linux Showcase and Conference*, October 2000, Atlanta, USA.

Borg, Ingwer and Groenen, Patrick 1997. *Modern Multidimensional Scaling – Theory and Applications*. Springer Series in Statistics, Springer, pp.29-48 and pp. 199-205.

Bradley, James 1968. *Distribution-free statistical tests*. Chapter 12. Englewood Cliffs, NJ, Prentice-Hall.

Calzarossa, Maria and Serazzi, Giuseppe 1993. Workload Characterization: A Survey. In *Proceedings of IEEE 81(8)*, Aug 1993, 1136-1150.

Calzarossa, Maria and Serazzi, Giuseppe 1994. Construction and Use of Multiclass Workload Models. *Performance Evaluation 19(4)*, 341-352.

Cirne, Walfredo and Berman, Francine 2001. A Model for Moldable Supercomputer Workloads. In *Proceedings of 15th Intl. Parallel & Distributed Processing Symposium*, Apr 2001.

Cirne, Walfredo and Berman, Francine 2001. A Comprehensive Model of the Supercomputer Workload. In *4th Workshop on Workload Characterization*, Dec. 2001.

Steve J. Chapin, Walfredo Cirne, Dror G. Feitelson, James Patton Jones, Scott T. Leutenegger, Uwe Schwiegelshohn, Warren Smith, and David Talby, 1999. Benchmarks and Standards for the Evaluation of Parallel Job Schedulers. In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (Eds.), Springer-Verlag, 1999, Lect. Notes Comput. Sci. vol. 1659, pp. 66-89.

Chiang, S-H., Arpaci-Dusseau, A., and Vernon, M. K. 2002. The impact of more accurate requested runtimes on production job scheduling performance". In *Job Scheduling Strategies for Parallel Processing 2002*, D. G. Feitelson and L. Rudolph, Eds., Springer Verlag, Lecture Notes in Computer Science vol. 2537, 103-127.

Crovella, M.E. and Bestavros, A. 1996. Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes. In *Sigmetrics Conf. Measurement & Modeling of Computer Systems*, May 1996, 160-169.

Dinda P. 2002. Online Prediction of the Running Time of Tasks. *Cluster Computing, Volume 5, Number 3*.

Dinda P., Lowekamp B., Kallivokas L. and O'Hallaron D. 1999. The Case for Prediction-based Best-effort Real-time Systems. In *Proc. of the 7th Intl. Workshop on Parallel and Distributed Real-Time Systems*, April 1999, 309-318.

Downey, Allen B. 1997. A Parallel Workload Model and Its Implications for Processor Allocation. In *Proceedings of 6th Intl. Symposium on High Performance Distributed Computing*, Aug 1997.

Downey, Allen B. 1997b. Using Queue Time Predictions for Processor Allocation. In *Job Scheduling Strategies for Parallel Processing 1997*, D. G. Feitelson and L. Rudolph, Eds., Springer-Verlag, Lecture Notes in Computer Science vol. 1291, 35-57.

Downey, Allen B. 1997c. Predicting queue times on spacesharing parallel computers. In *11th Intl. Parallel Processing Symp.*, Apr 1997, 209-218.

Downey, Allen B. and Feitelson, Dror G. 1999. The Elusive Goal of Workload Characterization. *Performance Evaluation Review 26(4)*, 14-29.

Eiben A. E., Smith J. E. and Smith J. D. 2003. *Introduction to Evolutionary Computing*. Natural Computing Series, Springer-Verlag.

Etsion, Yoav and Tsafrir, Dan 2005. A Short Survey of Commercial Cluster Batch Schedulers. Technical Report 2005-13, School of Computer Science and Engineering, The Hebrew University of Jerusalem, May 2005.

Feitelson, Dror G. and Nitzberg, Bill 1995. Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860. In *Job Scheduling Strategies for Parallel Processing 1995*, D. G. Feitelson and L. Rudolph, Eds., Springer-Verlag, 1995, Lecture Notes in Computer Science vol. 949, 337-360.

Feitelson, Dror G. 1996. Packing schemes for gang scheduling. In *Job Scheduling Strategies for Parallel Processing 1996*, D. G. Feitelson and L. Rudolph, Eds., Springer-Verlag, Lecture Notes in Computer Science vol. 1162, 89-110.

Feitelson, Dror G. and Jette, Morris A. 1997. Improved Utilization and Responsiveness with Gang Scheduling. In *Job Scheduling Strategies for Parallel Processing 1997*, D. G. Feitelson and L. Rudolph, Eds., Springer-Verlag, 1997, Lecture Notes in Computer Science vol. 1291, 238-261.

Feitelson, D. G. 1999. *The Parallel Workloads Archive*. Available from World Wide Web: (http://www.cs.huji.ac.il/labs/parallel/workload)

Feitelson, Dror G. 2002. The forgotten factor: Facts; On performance evaluation and its dependence on workloads. In *Proceedings of EuroPar 2002*, Aug 2002, Springer-Verlag, Lecture Notes in Computer Science vol. 2400, 49-60.

Feitelson, Dror G. 2002a. Workload Modeling for Performance Evaluation. In *Perf. Eval. of Complex Systems: Techniques and Tools*, M.C. Calsarossa and S. Tucci (Eds.), Springer-Verlag, Sep 2002. LNCS vol. 2459, 114-141.

Feitelson, Dror G. 2003. Metric and workload effects on computer systems evaluation. *Computer 36(9)*, 18-25.

Feitelson, Dror G. 2006. Locality of Sampling. Technical Report 2006-16, School of Computer Science and Engineering, The Hebrew University of Jerusalem, April 2006.

Feitelson, Dror G. and Tsafrir, Dan 2006. Workload sanitation for performance evaluation. In *Proceedings of IEEE Intl. Symposium on Performance Analysis of Systems and Software*, Mar 2006.

Ferrari, D. 1972. Workload characterization and selection in computer performance measurement. *Computer 5(4)*, 18-24.

Frachtenberg E. and Feitelson D. G. 2005. Pitfalls in parallel job scheduling evaluation. In *11th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, D. G. Feitelson, E. Frachtenberg, L. Rudolph, and U. Schwiegelshohn (Eds.), Springer-Verlag, 2005. Lecture Notes in Computer Science Vol. 3834.

Gibbons, R. 1997. A historical application profiler for use by parallel schedulers". In *Job Scheduling Strategies for Parallel Processing 1997*, D. G. Feitelson and L. Rudolph, Eds., Springer-Verlag, 1997, Lecture Notes in Computer Science vol. 1291, 58-77.

Giudici, Paolo 2003. Applied Data Mining: Statistical Methods for Business and Industry. Wiley & Sons.

Guttman, L. 1968. A general non-metric technique for finding the smallest space for a configuration of points. *Psychometrica 33*, 479-506.

Gribble, S.D., Manku, G.S., Roselli, D., Brewer, E.A., Gibson, T.J. and Miller, E.L. 1998. Self-Similarity in File Systems. *Performance Evaluation Review 26(1)*, 141-150.

Hotovy, Steven 1996. Workload Evolution on the Cornell Theory Center IBM SP2. In *Job Scheduling Strategies for Parallel Processing 1996*, D. G. Feitelson and L. Rudolph, Eds., Springer-Verlag, 1996, Lecture Notes in Computer Science vol. 1162, 27-40.

Jackson D., Snell Q. and Clement M. J. 2001. Core algorithms of the Maui scheduler. In *Job Scheduling Strategies for Parallel Processing 2001*, D. G. Feitelson and L. Rudolph, Eds., Springer-Verlag, 2001, Lecture Notes in Computer Science vol. 2221, 87–102.

Jann, Joefon; Pattnaik, Pratap; Franke, Hubertus; Wang, Fang; Skovira, Joseph and Riodan, Joseph 1997. Modeling of Workload in MPPs. In *Job Scheduling Strategies for Parallel Processing 1997*, D. G. Feitelson and L. Rudolph, Eds., Springer-Verlag, 1997, Lecture Notes in Computer Science vol. 1291, 95-116.

Jarvis S. A., Spooner D. P., H. N. Lim Choi Keung, Cao J., Saini S. and Nudd G. R. 2004. Performance Prediction and its Use in Parallel and Distributed Computing Systems. In *Future Generation Computer Systems special issue on System Performance Analysis and Evaluation*, 2004.

Jones, J. P. and Nitzberg, B.1999. Scheduling for parallel supercomputing: a historical perspective of achievable utilization. In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (Eds.), Springer-Verlag, 1999. Lecture Notes in Computer Science vol. 1659, 1-16.

Karatza H. D. and Hilzer R. C. 2003. Performance Analysis of Parallel Job Scheduling in Distributed Systems. In *36th Annual Simulation Symposium (ANSS'03)*, April 2003, 109-116.

Koldinger, E.J.; Eggers, S.J., and Levy, H.M. 1991. On the validity of trace-driven simulation for multiprocessors. In *Proceeding of 18th Ann. Intl. Symp. Computer Architecture*, May 1991, 244-253,.

Kotz, S. and Nadarajah, S. 2001. Extreme Value Distributions: Theory and Applications. World Scientific Publishing Company, 2001.

Lazowska, E.D. 1977. The use of percentiles in modeling CPU service time distributions. In *Computer Performance*, K.M. Chandy and M. Reiser, Eds., North Holland, 53-66.

Lee, C. B., Schwartzman, Y., Hardy, J. and Snavely, A. 2004. Are user runtime estimates inherently inaccurate?. In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn (Eds.), Springer-Verlag, 2004. Lecture Notes in Computer Science Vol. 3277.

Leland W.E., Taqqu M.S., Willinger W. and Wilson D.V. 1994. On the self-similar nature of Ethernet traffic. *IEEE/ACM Trans. Networking* **2**(1), Feb 1994, 1-15.

Lipshitz, G., and Raveh, A. 1994. Applications of the Co-plot method in the study of socioeconomic differences among cities: A basis for a differential development policy. *Urban Studies 31*, 123-135.

Lo, V.; Mache, J. and Windisch, K. 1998. A comparative study of real workload traces and synthetic workload models for parallel job scheduling. In *Job Scheduling Strategies for Parallel Processing 1998*, D. G. Feitelson and L. Rudolph, Eds., Springer Verlag, 1998. Lecture Notes in Computer Science vol. 1459, 25-46.

Lublin, Uri and Feitelson, Dror G. 2003. The workload on parallel supercomputers: modeling the characteristics of rigid jobs. *Journal of Parallel & Distributed Computing 63(11)*, 1105-1122.

Maital, S. 1978. Multidimensional Scaling: Some Econometric Applications. *Journal of Econometrics 8*, 33-46.

Majumdar S.; Eager, D.L. and Bunt, R.B. 1988. Scheduling in multiprogrammed parallel systems. In *SIGMETRICS Conf. Measurement & Modeling of Computer Systems*, May 1988, pp. 104-113.

MathWave, 2006. EasyFit 3.0 Distribution Fitting Software. Available from World Wide Web: (www.mathwave.com/products/easyfit.html)

Menascé D. A., Almeida V. A. F., Riedi R., Ribeiro F., Fonseca R. and Meira W. Jr. 2003. A hierarchical and multiscale approach to analyze E-business workloads. In *Performance Evaluation 54(1)*, Sep 2003, 33-57.

Mu'alem A. W. and Feitelson D. G. 2001. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Trans. Parallel & Dist. Syst. 12(6)*, 529-543.

Raveh, Adi 2000. The Greek banking system: Reanalysis of performance. *European Journal of Operational Research 120*, 525-534.

Schroeder B. and Harchol-Balter M. 2000. Evaluation of Task Assignment Policies for Supercomputing Servers: The Case for Load Unbalancing and Fairness. In *9th IEEE Symposium on High Perf. Dist. Comp.*, August 2000.

Smith, W., Taylor, V. and Foster, I. 1999. 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. LNCS vol. 1659, 202-219.

Smith, W., Foster, I. and Taylor, V. 2000. Scheduling with advanced reservations. In *14th Intl. Parallel & Distributed Processing Symp.*, May 2000, 127-132.

Skovira J., Chan W., Zhoi H. and Lifka D. 1996. The EASY – LoadLeveler API project. In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), Springer Verlag, 1996, LNCS vol. 1162, 41-47.

Song B., Ernemann C. and Yahyapour R. 2004. Parallel Computer Workload Modeling with Markov Chains. In *11th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, D. G. Feitelson, E. Frachtenberg, L. Rudolph, and U. Schwiegelshohn (Eds.), New York, USA, June 2004.

Song B., Ernemann C. and Yahyapour R. 2005. User Group-based Workload Analysis and Modelling. In *Cluster Computing and the Grid 2005 (CCGrid 2005)*, May 2005, Vol. 2, 952-961.

Talby D. and Feitelson D. G. 1999. Supporting priorities and improving utilization of the IBM SP2 scheduler using slack-based backfilling. In *13th Intl. Parallel Processing Symp.*, 513-517, April 1999.

Talby D., Feitelson D. G., and Raveh A. 1999. 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. LNCS vol. 1659, 43-66.

Talby D. 1999. *Visual Co-Plot*. Available from World Wide Web: (http://www.cs.huji.ac.il/~davidt/vcoplot)

Talby D. and Feitelson D. G. 2005. Improving and Stabilizing Parallel Computer Performance Using Adaptive Backfilling. In *19th Intl. Parallel & Distributed Processing Symp.*, April 2005.

Talby D., Feitelson D. G. and Raveh A. 2007. A Co-Plot Analysis of Logs and Models of Parallel Workloads. *ACM Trans. on Modeling and Computer Simulation*, to appear.

Tsafrir, D., Etsion, Y. and Feitelson, D. G. 2005. Modeling User Runtime Estimates. In *11th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, D. G. Feitelson, E. Frachtenberg, L. Rudolph, and U. Schwiegelshohn (Eds.), Springer-Verlag, 2005. Lecture Notes in Computer Science Vol. 3834, 1-35.

Tsafrir, Dan, Etsion, Yoav and Feitelson, D. G. 2006. Backfilling Using Runtime Predictions Rather than User Estimates. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, to appear.

Tsafrir, Dan and Feitelson, Dror G. 2006. Instability in parallel job scheduling simulation: the role of workload flurries. In *20th Intl. Parallel & Distributed Processing Symposium,* April 2006.

Tsafrir, Dan and Feitelson, D. G. 2006b. The Dynamics of Backfilling: Solving the Mystery of Why Increased Inaccuracy May Help. In *IEEE Intl. Symp. Workload Characterization*, Oct 2006.

Vazhkudai S., Schopf J., Foster I. 2002. Predicting the Performance of Wide-Area Data Transfers. In *16th Int'l Parallel and Dist. Processing Symp.*, April 2002.

Windisch, K.; Lo, V.; Moore, R.; Feitelson, D. and Nitzberg, B. 1996. A comparison of workload traces from two production parallel machines. In *Proceedings of 6th Symposium on Frontiers Massively Parallel Computing*, Oct 1996, 319-326.

Yoo A. B. and Jette M. A. 2001. The Characteristics of Workload on ASCI Blue-Pacific at LLNL. In *1st Intl. Symp. on Cluster Computing and the Grid*, May 2001.

Yu, Kun-Ming, Wu, S.J.-W. and Hong, Tzung-Pei 1997. A Load Balancing Algorithm using Prediction. In *Proc. 2nd Aizu Intl. Symp.*, Mar 1997, pp. 159-165.

Zilber, J., Amit, O. and Talby, D. 2005. What is Worth Learning from Parallel Workloads? A User and Session Based Analysis". In *International Supercomputing Conference 2005 (ICS '05)*, June 2005.

Zotkin, D. and Keleher, P. J. 1999. Job-length estimation and performance in backfilling schedulers. In *8th Intl. Symp. High Performance Distributed Computing*, Aug 1999.

APPENDICES

A. Pseudo-Code of the EASY and SJBF Schedulers

A scheduling algorithm on a variable partitioning parallel computer is an event-driven algorithm, activated when a job arrives or terminates. At each of these events, the scheduler may decide to start one or more jobs from the waiting jobs queue:

```
struct Job { Time arrivalTime, int processors, int estimate, UserID user }
list<Job> OnJobArrival ( Job job )
list<Job> OnJobTermination ( Job job )
```

Listing 6. A Scheduler's Interface

The classic EASY scheduler is given in Listing 7. The SJBF algorithm is received by replacing the three underlined words by the word 'prediction'.

```
On Job Arrival (Job job):
    Add job to waiting jobs list
    Start Jobs According to FCFS with Backfilling

On Job Termination (Job job):
    Remove job from running jobs list
    AvailableProcessors += job.processors
    Start Jobs According to FCFS with Backfilling

Start jobs According to FCFS with Backfilling:
    // Start jobs in FCFS order first:
    For each job in waiting jobs list, by ascending order of arrival (FCFS):
        if job.Processors <= AvailableProcessors then StartJob (job)
    // Compute shadow time and extra nodes:
    Reserved = First job in waiting jobs list, by ascending order of arrival (FCFS)
    Shadow time = 0
    FreedProcessors = AvailableProcessors
    For each job in the running jobs list, by ascending time to termination:
        Shadow time = Max(0, job.estimate – (CurrentTime – job.startTime))
        FreedProcessors += job.processors
        if (FreedProcessors >= Reserved.processors) break;
    Extra nodes = FreedProcessors – Reserved.processors
    // Backfill:
    For each job in waiting jobs queue, by ascending order of arrival
        if job.processors <= AvailableProcessors and
           (job.estimate <= shadow time or job.processors <= extra nodes)
           then StartJob (job)

Start Job (Job job):
    Add job to the running jobs list
    Remove job from the waiting jobs list
    AvailableProcessors –= job.processors
```

Listing 7. Pseudo-code of the EASY and SJBF Scheduling Algorithms

B. Co-Plot

Classical multivariate analysis methods, such as cluster analysis and principal component analysis, analyze variables and observations separately. Co-Plot is a new technique which analyzes them simultaneously. This means that we'll be able to see, in the same analysis, clusters of observations, clusters of variables, the relations between clusters (correlation between variables, for example), and a characterization of observations (as being above average in certain variables and below in others). Co-plot is especially suitable for tasks in which there are few observations and relatively many variables – as opposed to regression based techniques, in which the number of observations must be an order of magnitude larger than the number of variables. The technique has been used before mostly in the area of economics (Lipshitz and Raveh, 1994; Raveh, 2000).

Co-plot's output is a visual display of its findings: It is based on two graphs that are superimposed on each other. We denote the number of observations by *n*, and the number of variables (the dimensionality) by *p*. The first graph maps the *n* observations into a two-dimensional space. This mapping, if it succeeds, conserves relative distance: observations that are close to each other in *p* dimensions are also close in two dimensions, and vice versa. The second graph consists of *p* arrows, representing the variables, and shows the direction of the gradient for each one.

Given an input matrix $Y_{n\times p}$ of *p* variable values for each of *n* observations (see for example Table 2), the analysis consists of four stages. The first is to normalize the variables, which is needed in order to be able to relate them to each other, although each has different units and scale. This is done in the usual way. If $\overline{Y_j}$ is the *j*'th variable's mean, and D_j is its standard deviation, then Y_{ij} is normalized into Z_{ij} by:

$$Z_{ij} := (Y_{ij} - \overline{Y_j}) / D_j \qquad (1)$$

In the second stage, we compute a measure of dissimilarity $S_{ik} \geq 0$ between each pair of observations (rows of $Z_{n\times p}$). A symmetric n×n matrix is produced from all the pairs of observations. In our analysis we use city-block distance – the sum of absolute deviations – as the measure of dissimilarity:

$$S_{ik} = \sum_{j=1}^{p} \left| Z_{ij} - Z_{kj} \right| \qquad (2)$$

In stage three, the *n* observations are mapped by means of a multidimensional scaling (MDS) method from the original *p*-dimensional space into a two dimensional Euclidean space, such that

'close' observations (with a small dissimilarity between them) are close to each other on the map, while 'distant' ones are also distant on the map. Note, however, that we are mainly interested in relative distances. This can be expressed as a relationship between the map distances d_{ik} (which are just the Euclidean distance in 2D) and the corresponding dissimilarity metrics S_{ik}, where we require:

$$S_{ik} < S_{lm} \text{ if and only if } d_{ik} < d_{lm}$$

The MDS we use is Guttman's Smallest Space Analysis, or SSA (Guttman, 1968). SSA is an iterative mapping technique, in which the need for additional iterations is guided by the coefficient of alienation Θ. The smaller it is, the better that the mapping reflects the original dissimilarities, and values below 0.15 are considered good. To evaluate Θ we first evaluate another metric μ which directly measures the correlation between the dissimilarity measures and the map distances:

$$\mu = \frac{\sum_{i,k,l,m} (S_{ik} - S_{lm})(d_{ik} - d_{lm})}{\sum_{i,k,l,m} |S_{ik} - S_{lm}||d_{ik} - d_{lm}|} \quad (3)$$

Thus μ can attain the maximal value of 1. This is then used to define Θ as follows:

$$\Theta = \sqrt{1 - \mu^2} \quad (4)$$

This functional form does not decrease much for mediocre values of μ, but drops sharply when μ approaches 1, so it is only low for very good correlations. Full details of the SSA algorithm are given in (Guttman, 1968). It is a widely used method in social sciences, and several examples and intuitive descriptions can be found in (Maital, 1978).

If the third stage results in a high coefficient of alienation, then the data does not fit well into two dimensions, and a different technique is required. On the other hand, if the coefficient of alienation is low, we know that the 2D map indeed captures the salient features of the data.

In the fourth stage of the Co-plot method, *p* arrows are drawn on the two dimensional Euclidean space obtained in the previous stage. Each variable *j* is represented by an arrow emerging from the center of gravity of the *n* points. The direction of each arrow is chosen so that the correlation between the actual values of the variable *j* and their projections on the arrow is maximized. Therefore, observations with a high value in this variable should be in the part of the space the arrow points to, while observations with a low value in this variable will be on the other side of the map. The length of the arrow is proportional to this correlation.

As a result of this construction, arrows associated with highly correlated variables will point in about the same direction, while arrows associated with uncorrelated variables will tend to be orthogonal. Thus the cosines of angles between these arrows are approximately proportional to the correlations between their associated variables.

The quality of the graph generated by the Co-plot technique is assessed by two types of measures, one for stage 3 and another for stage 4. In stage 3, a single measure – the coefficient of alienation – is used to determine the quality of the two-dimensional map. In stage 4, p separate measures – one for each variable – are given. These are the magnitudes of the p maximal correlations, which measure the goodness of fit of the p regressions. These correlations help in deciding whether to eliminate or add variables: Variables that do not fit into the graphical display, namely, have low correlations, should be removed, because they do not relate well to the principal features of the data as identified by the 2D mapping. The removal of a variable requires a re-computation of the Co-Plot, since it affects the earlier stages (SSA) as well. Note that since each variable's arrow is computed separately, there is no need to fit all the 2^p subsets of variables as in other methods that use a general coefficient of goodness-of-fit. The higher the variable's correlation, the better the variable's arrow represents the common direction and order of the projections of the n points along the axis it is on.

A free software package for Co-Plot analysis is available online (Talby, 1999).

C. Temporal Structure: Full Data Tables

Log	Runtime		# of Processors		Total CPU Work		Inter-Arrival Time		# of Jobs	
	15m	150m	15m	150m	15m	150m	15m	150m	15m	150m
NASA	0.05	0.17	0.45	0.34	0.21	0.11	0.01	-0.02	0.54	0.53
PAR95	0.14	0.07	0.37	0.23	0.15	0.05	0.19	0.18	0.59	0.50
PAR96	0.11	0.08	0.38	0.12	0.07	-0.03	0.14	0.15	0.38	0.33
BLUE	0.28	0.14	0.22	0.38	0.04	0.09	0.17	0.01	0.42	0.54
SDSP2	0.10	0.17	0.11	0.24	0.09	0.04	0.17	-0.01	0.48	0.60
CTC	0.17	0.20	0.16	0.18	0.03	0.07	0.21	-0.05	0.39	0.39
KTH	0.24	0.15	0.16	0.29	0.04	0.02	0.10	0.08	0.24	0.38
LACM5	0.25	0.32	0.78	0.56	-0.01	0.09	0.16	-0.05	0.92	0.74
LAO2K	0.34	0.49	0.68	0.35	0.04	0.16	0.05	0.03	0.52	0.52
LLNL	0.04	0.04	0.18	0.31	0.05	0.01	0.11	-0.05	0.50	0.52
OSC	0.15	0.10	0.44	0.51	0.08	0.07	0.19	0.02	0.62	0.62
Avg Logs:	**0.17**	**0.18**	**0.36**	**0.32**	**0.07**	**0.06**	**0.14**	**0.03**	**0.51**	**0.52**
DOW	-0.01	-0.01	-0.01	0.06	0.01	-0.07	-0.35	-0.33	0.01	-0.01
JANNctc	0.10	0.09	0.11	0.18	0.00	-0.01	-0.01	-0.33	0.21	0.16
FEIT96	0.29	0.21	0.25	0.04	0.28	0.22	-0.36	-0.30	0.01	-0.01
FEIT97	0.32	0.2	0.44	0.23	0.31	0.1	-0.24	-0.33	0.04	0.04
LUB	0.04	0.11	0.00	0.01	0.04	0.11	-0.01	-0.09	-0.01	0.01
CIRctc	0.02	0.12	0.07	0.29	0.00	0.03	0.74	-0.04	0.91	0.65
CIRkth	0.02	0.14	0.04	0.32	0.00	0.05	0.47	-0.13	0.81	0.60
CIRsd	0.02	0.06	0.06	0.20	0.01	0.02	0.62	0.20	0.88	0.62

Table 37. Locality of Sampling in Logs and Models

Log	Runtime		# of Processors		Total CPU Work		Inter-Arrival Time		# of Jobs	
	12h	24h	12h	24h	12h	24h	12h	24h	12h	24h
NASA	0.05	0.24	-0.06	0.39	0.01	0.30	-0.01	0.02	-0.29	0.40
PAR95	-0.08	0.18	0.14	0.23	-0.22	0.15	-0.03	0.04	0.45	0.35
PAR96	0.19	0.21	0.05	0.09	-0.09	0.05	0.10	0.07	0.21	0.18
BLUE	-0.09	0.17	0.01	0.23	-0.10	0.17	0.03	0.05	-0.01	0.43
SDSP2	0.01	0.29	0.02	0.11	-0.09	0.05	-0.07	-0.02	0.37	0.46
CTC	-0.18	0.34	0.02	0.20	-0.14	0.16	0.14	0.08	-0.04	0.18
KTH	-0.08	0.02	-0.02	0.07	-0.19	-0.06	-0.09	-0.15	-0.02	0.12
LACM5	-0.11	0.31	0.12	0.44	-0.01	0.28	-0.08	0.15	0.26	0.45
LAO2K	-0.38	0.62	0.01	0.00	-0.10	0.27	0.06	0.08	0.09	0.34
LLNL	0.25	0.32	-0.09	0.26	0.19	0.21	-0.07	-0.03	-0.25	0.37
OSC	0.09	0.12	0.35	0.25	0.04	0.10	0.09	-0.04	0.49	0.38
Avg Logs:	**-0.03**	**0.26**	**0.05**	**0.21**	**-0.06**	**0.15**	**0.01**	**0.02**	**0.11**	**0.33**
DOW	-0.01	-0.18	0.04	0.16	0.12	-0.07	0.01	-0.01	0.01	0.28
JANNctc	0.00	0.03	0.03	0.02	0.00	0.00	-0.04	-0.14	0.11	0.12
FEIT96	0.07	0.11	-0.03	0.09	0.01	-0.06	-0.11	0.11	-0.08	-0.08
FEIT97	0.05	0.35	0.02	-0.06	0.05	-0.08	0.02	0.05	0.05	-0.01
LUB	0.14	0.27	-0.02	0	0.14	0.15	-0.01	-0.01	-0.02	0.00
CIRctc	-0.24	-0.01	-0.53	0.61	-0.11	0.11	-0.05	0.77	-0.83	0.95
CIRkth	-0.31	0.00	-0.58	0.43	-0.13	0.09	-0.22	0.23	-0.84	0.63
CIRsd	-0.14	-0.01	-0.44	0.45	-0.05	0.04	-0.48	0.63	-0.96	0.99

Table 38. Daily Cycle in Logs and Models

Log	Runtime	# of Processors	Total CPU Work	Inter-Arrival Time	# of Jobs
	7d	7d	7d	7d	7d
NASA	0.29	0.27	0.28	-0.08	0.49
PAR95	0.30	0.07	0.12	-0.02	0.08
PAR96	0.22	0.05	0.04	-0.09	0.02
BLUE	0.20	0.16	0.19	0.04	0.53
SDSP2	0.38	0.21	0.11	0.02	0.21
CTC	0.21	0.14	0.15	0.05	0.11
KTH	0.46	0.46	0.12	0.02	0.43
LACM5	0.28	0.14	0.15	0.00	0.22
LAO2K	0.42	0.22	0.15	0.00	0.30
LLNL	0.18	0.31	0.18	0.04	0.46
OSC	0.40	0.21	0.34	0.07	0.20
Avg Logs:	**0.30**	**0.20**	**0.17**	**0.00**	**0.28**
DOW	0.13	-0.76	0.00	N/A	0.45
JANNctc	-0.05	-0.01	-0.03	-0.03	-0.07
FEIT96	0.00	-0.25	-0.02	N/A	0.23
FEIT97	0.05	0.05	-0.02	N/A	N/A
LUB	0.28	0.02	0.28	-0.01	0.02
CIRctc	-0.01	-0.02	0.01	0.54	1
CIRkth	-0.02	-0.02	0.00	0.11	1
CIRsd	0.00	0.01	0.02	N/A	1

Table 39. Weekly Cycle in Logs and Models

Log	Runtime	# of Processors	Total CPU Work	Inter-Arrival Time	# of Jobs
	avg H	avg H	avg H	avg H	avg H
NASA	0.65	0.72	0.60	0.45	0.74
PAR95	0.73	0.68	0.71	0.73	0.83
PAR96	0.71	0.71	0.62	0.64	0.78
BLUE	0.71	0.69	0.67	0.77	0.73
SDSP2	0.73	0.83	0.77	0.80	0.86
CTC	0.72	0.63	0.60	0.58	0.71
KTH	0.68	0.79	0.61	0.66	0.69
LACM5	0.82	0.81	0.80	0.83	0.99
LAO2K	0.70	0.90	0.73	0.59	0.72
LLNL	0.69	0.73	0.67	0.53	0.64
OSC	0.78	0.79	0.75	0.84	0.82
Avg Logs:	**0.72**	**0.75**	**0.69**	**0.67**	**0.77**
Avg >year:	**0.75**	**0.75**	**0.72**	**0.77**	**0.84**
DOW	0.46	0.48	0.47	0.46	0.46
JANNctc	0.51	0.59	0.50	0.56	0.45
FEIT96	0.57	0.62	0.54	0.44	0.49
FEIT97	0.68	0.70	0.64	0.45	0.53
LUB	0.63	0.47	0.63	0.48	0.59
CIRctc	0.50	0.47	0.49	0.08	0.50
CIRkth	0.52	0.50	0.48	0.14	0.50
CIRsd	0.45	0.50	0.50	0.33	0.50

Table 40. Self-Similarity in Logs and Models

D. Dataset File Formats

The model is based on seven production logs, freely available from the Parallel Workloads Archive (Feitelson, 1999). For each of the seven logs, the cleaned version of the log was used to create three other files: A jobs list, a sessions list and a users list. All three file formats are based on the Standard Workload Format (Chapin et al., 1999): They are text files, with one list per job/session/user, with a space-separeated list of numbers containing data for each. The file starts with a set of header comments (we use the same ones from the Standard Workload Format), a semicolon marks a comment line, and -1 stands for a missing value.

Table 41 lists the description of the 24 columns of numbers which appear in every line of each file. The Jobs Workload Format is an extension of the Standard Workload Format (which only contains a list of jobs). The added columns are the session ID of the session the job belongs to, the day in week and hour in day in which the job started (this is in local time, in contrast to the job's submission time which is in UTC), and class (cluster) of the session and user by which the job was submitted, and a log identifier (to enable mixing jobs from different logs in the same file, for joint analysis).

The sessions and users list are computed from the jobs list, and have been created as separate files only to simplify the analysis. Computed session fields include the percent of the session run during daytime (7:30-17:30 in local time, inclusive), the percent of the session run during workdays (Monday to Friday in local time, inclusive), number of unique runtimes (up to a 5% difference) and used processors for all the session's jobs (as a measure of locality), and the session and user class. Computed user fields include the number of sessions, percent of each session class among the user's sessions, the inter-session time median and interval, and the user class. Other fields are self-explanatory, and consistent with their definition in the Standard Workload Format.

The dataset is freely available for further research, on request.

Col #	Job Workload Format	Session Format	User Format
1	Job ID	Session ID	User ID
	Submit time (absolute)	Arrival time (absolute)	Arrival time (absolute)
	Wait Time	Duration	Total time in system
	Run Time	Original User ID	Numer of Sessions
5	Used Processors	Number of Jobs	Number of Jobs
	Used CPU time	Runtime median	Runtime Median
	Used memory	Runtime interval	Runtime Interval
	Reqequest Processors	Processors Median	Procs Median
	Reqested CPU time	Processors Interval	Procs Interval
10	Requested Memory	Jobs Inter-Arrival Time Median	Jobs IA Time Median
	Completion status	Jobs Inter-Arrival Time Interval	Jobs IA Time Interval
	User ID	Think time median	Think time median
	Group ID	Think time interval	Think time interval
	Executable ID	Did session start in Daytime?	Sessions IA Median
15	Queue ID	Percent of daytime during session	Sessions IA Interval
	Partition ID	Did session start in a workday?	Percent of daytime during user's activity
	Preceding Job	Percent of workdays during session	Percent of workdays during user's activity
	Think Time from Preceding Job	Number of Unique Runtimes	Percent of IWD sessions
	Session ID	Number of Unique Used Processors	Percent of IWN sessions
20	Day of Week at arrival	Day of Week at arrival	Percent of IW sessions
	Hour of Day at arrival	Hour of Day at arrival	Percent of BHP sessions
	Session Class	Session Class	Percent of BHD sessions
	User Class	User Class	User Class
24	Log ID	Log ID	Log ID

Table 41. Column descriptions of Job, Session and User Files in the user-based model dataset

תוכן העניינים

ת ק צ י ר

הבנת העומס הצפוי על מערכת הוא תנאי הכרחי לקבלת ההחלטות הנכונות בזמן העיצוב והקונפיגורציה שלה. לכן, ניתוח עומסים על מחשבים מקבליים הוא תחום מחקרי גדול, הפועל בשני כיוונים עיקריים. הראשון הוא הבנייה של מודלים של עומסים, שהם מודלים סטטיסטיים המבוססים על הבחנות מלוגים של מחשבים אמיתיים. מודלים אלו יכולים לשמש ליצירת עומסים מלאכותיים, על מנת להשוות אלגוריתמים לניהול משאבים תחת תנאים שונים (גודל המחשב, רמת עומס וכדומה) או כדי להפיק תובנות כלליות על השימוש במחשבים אלו.

כיוון המחקר השני מנצל מאפיינים סטטיסטיים של עומסים באופן ישיר, או ע"י בניית אלגוריתמים יוריסטיים המנצלים את המאפיינים שהתגלו או ע"י בניית אלגוריתמים אדפטיביים או מבוססי חיזוי שלומדים את העומס תוך כדי ריצה. אלגוריתמים אדפטיביים, לומדים או מבוססי חיזוי תמיד מבוססים על מידע קודם משמעותי לגבי העומס: אלו פרמטרים צריכים להיות אדפטיביים? לפי אלו רמזים כדאי לשנות אותם? אלו משתנים הם אלו שעל פיהם כדאי ללמוד את ההיסטוריה? פעמים רבות, החידוש העיקרי של אלגוריתמים כאלה היא גילוי מאפיין סטטיסטי חדש של העומס, וביצוע מידול מוצלח שלו.

בתחומים רבים במערכות מחשבים, החלופות הבסיסיות לניהול משאבים הן ידועות ומובנות היטב – ורוב שיפורי הביצועים בשנים האחרונות הם תוצאה של כוונון האלגוריתמים כדי לנצל מאפיינים סטטיסטיים שהתגלו שוב ושוב בלוגים של מחשבים אמיתיים. ניתן למצוא דוגמאות בתחומי התזמון, חלוקת משימות, ניהול grid מבוזר, חלוקת עומסים, מערכות זמן-אמת רכות, רפליקציה של מידע ועוד. כל התחומים האלה יכולים פוטנציאלית להרוויח מתוצאות מחקר זה.

המטרה של מחקר זה היא לספק מידע חדש, שהתגלה תוך שימוש בטכניקות סטטיסטיות מתאימות, ויכול להועיל לשני העולמות: מידול עומסים ופיתוח אלגוריתמים מקוונים. למרות שמחקר זה התחיל עם זיהוי מספר מאפיינים סטטיסטיים בלוגים של מחשבים אמיתיים שחסרים במודים הקיימים, השלב הראשון לא היה יצירה של מודל חדש הכולל מאפיינים אלו, אלא פיתוח אלגוריתמים מקוונים שינצלו

מאפיינים אלו. אלגוריתמים אלו מוכיחים את חשיבותם המעשית של המאפיינים החסרים, וכך מהווים מוטיבציה לשלב המידול.

הבעיה האלגוריתמית שנבחרה היא תזמון מקבילי, משתי סיבות. ראשית, מתזמנים מקביליים רגישים מאד לשינויים בעומס הניתן להם, ובמיוחד לשינויים במבנה העיתי (temporal) שלו. שנית, המתזמן הוא רכיב מפתח בכל מחשב מקבילי, בעל השפעה דרמטית על הביצועים הכוללים שלו. זהו רכיב תוכנה בלבד, ומודול עצמאי יחסית במערכת ההפעלה, כך ששיפור שלו מהווה הזדמנות מעשית וזולה לשדרוג ביצועי המחשב כולו.

אוסף האלגוריתמים הראשון שפותח במחקר זה הוא אלגוריתמים אדפטיביים. מוצגות תוצאות על פיהן המתזמנים הנמצאים בשימוש נרחב כיום סובלים מפערים גדולים ולא מוסברים בביצועים שלהם בן חודשי השנה. בנוסף, אלגוריתמי תזמון שונים הם שונים גם במיקום הפערים בביצועיהם, מה שרומז שבחירת האלגוריתם הנכון בכל פרק זמן בנפרד יכולה להביא לשיפור ביצועים כולל. מוצגים שני אלגוריתמים אדפטיביים שמשיגים מטרה זו: הראשון בוחר את המתזמן הפעיל לפי השוואת ביצועים בעבר הקרוב, והשני בוחר לפי רמת המקביליות הממוצעת של עבודות פעילות, שכפי שהמחקר מראה היא בעלת מתאם חיובי עם ביצועיהם היחסיים של המתזמנים השונים. מוצגות תוצאות סימולציה של האלגוריתמים האדפטיביים על לוגים אמיתיים, שמדגימות מאפיינים ייחודיים של המבנה העיתי של העומד על מחשבים מקביליים. בנוסף ניתנות קונפיגורציות פרמטרים מיטביות לכל אלגוריתם, שמשיגות שיפור ממוצע של 10% בביצועים ו-35% ביציבות עבור הלוגים שנבדקו.

האוסף השני של אלגוריתמי תזמון מבוסס על חיזוי זמן ריצה. תגליות מהתקופה האחרונה לגבי backfilling – אופטימיזציה נפוצה בתחום התזמון המקבילי – מראות שפרסומים קודמים, שהראו שדווקא זמני ריצה לא מדויקים מביאים לביצועים טובים יותר, בעצם שילמו בהוגנות (fairness) כדי לקבל ביצועים אלו. בנוסף, תזמון SJBF (Shortest-Job-Backfill-First) יכול להשיג ביצועים עדיפים כשהוא מופעל יחד עם אלגוריתם טוב לחיזוי זמן ריצה. האינדיקטור הטוב ביותר לחיזוי זמן ריצה של עבודה חדשה הוא המשתמש המריץ אותה: הודגם אמפירית שטכניקת חיזוי פשוטה, לפיה החציון של שלושת העבודה האחרונות של משתמש משמשות לחיזוי זמן הריצה של העבודה הבאה שלו, מביאה לביצועים טובים משמעותית לעומת מתזמנים קיימים.

מחקר זה בנה על תוצאות אלו וחקר חיזוי זמן ריצה מבוסס מושבים. זהו אלגוריתמים החיזוי הראשון המשתמש בכל ההיסטוריה של משתמש כדי לספק זמן ריצה חזוי. גילינו כי בנוסף לשימוש בעבודות מהעבר הקרוב ביותר כדי לבצע חיזוי, יש חשיבות גם לשימוש בעבודות הדומות ביותר לעבודה החדשה לפי מאפיינים נוספים. אנחנו מציגים גם אלגוריתמי חיזוי למקרה בו נתונה היסטוריה חלקית או אין שום מידע קודם לגבי עבודות קודמות, וגם למקרה בו לא נתונה הערכת המשתמש לזמן הריצה של העבודה החדשה.

מתזמנים אדפטיביים ומבוססי חיזוי מצטרפים לתוצאות מחקריות נוספות מהשנים האחרונות, שלנו ושל אחרים, כדי לבנות את הרשימה הבאה של מאפיינים סטטיסטיים החסרים ממודלים קיימים של עומסים על מחשבים מקביליים, שיש צורך מעשי למדל אותם:

1. לוקליות (Locality of Sampling)
2. דמיון עצמי ותלות ארוכת טווח (Self Seimilarity and Long-Range Dependence)
3. מחזוריות יומית ושבועית
4. צורך במודל פרמטרי
5. מתודולוגיה לשינוי רמת עומס
6. מידול של אירועים חריגים (Flurries)
7. מידול ישיר של משתמשים.

אנחנו טוענים שהפתרון לבעיות אלו יכול להיות מושג על ידי בניית מודל שכבתי: מודל משתמשים, אחריו מודל מושבים לכל סוג משתמש, ואחריו מודל עבודות לכל סוג מושב. לגישה כזו יש יתרון נוסף והיא קבלת תובנות חדשות לגבי התנהגות משתמשים אנושיים במחשבים מקביליים.

אנחנו מספקים בסיס סטטיסטי למודל כזה, על ידי מנת תשובה לשתי שאלות בסיסיות: אלו מאפיינים של משתמשים ומושבים הם מרכזיים מספיק כך שיש חשיבות למדל אותם? ובהינתן התשובה לכך, אלו סוגים של משתמשים ומושבים קיימים? התשובה לשאלה הראשונה ניתנת תוך שימוש ב-Principal Component Analysis, והשנייה ניתנת תוך שימוש ב-K-Means Clustering. אנחנו

מזהים משתנים המסבירים יותר מ-80% מהשונות בין משתמשים ובין מושבים וגם מזהים חמישה סוגים יציבים של מושבים וארבעה סוגים יציבים של משתמשים. הניתוח מבוסס על לוגים משבעה מחשבים מקביליים שונים, כולל יותר מ-87 חודשים, ומנתח את המידע מכל המחשבים יחד כדי להבטיח שהתוצאות נקיות מהשפעות של אתר או ארכיטקטורה מסוימים.

בניית המודל מתחילה בזיהוי אוסף פרמטרים שניתן להשתמש בהם כדי להבדיל בין מערכות. מודגם כיצד ניתן לספק למודל כקלט או את מספר המעבדים במחשב היעד, או את חציון כמות העיבוד שדורשת עבודה יחד עם רמת העומס הרצויה או מספר העבודות הממוצע ביום.

השלב הבא הוא בניית מודל המשתמשים, ולאחר ניתוח תבניות ההופעה של משתמשים אנחנו מסיקים שנכון למדל בנפרד את התפלגות מספר המשתמשים החדשים בשבוע, שאינה תלויה בפרמטרים או בהיסטוריה, ואת מספר המשתמשים הפעילים בתחילת המודל, התלויה ברמת העומס. מוצגת גם התפלגות החלוקה של משתמשים לארבעת הסוגים, והתלות שלה בפרמטרים ובהיסטוריה.

בניית מודל המושבים מתחילה בניתוח תבנית ההגעה של מושבים, וגילוי שהמשתנים העיקריים המשפיעים עליה היא הסוג ושבוע הפעילות של כל משתמש. כיוון שמשתמשים רבים לא פעילים במהלך שבועות שלמים, מוצג מודל נפרד של אי-פעילות ושל מספר המושבים בשבוע בהינתן פעילות. על בסיס מודלים אלו מוסק המודל להערכת שבוע הפעילות של משתמשים שהיו פעילים בתחילת המודל. לאחר מכן נבנה מודל סוג המושבים, המבוסס על מודל מרקובי שונה לכל סוג משתמש, ותיקון נוסף כדי לשקף את השינוי בהתפלגויות סוג המושבים לאורך זמן הפעילות של משתמש. על בסיס אוסף מודלים אלו נבנית ומוסברת תבנית המודל המלא, בעל שלוש השכבות, יחד עם קווים מנחים לשימוש בו ויישומים אפשריים הן בתחום המידול והן בתחום האלגוריתמי.

עבודה זו נעשתה בהדרכתם של:

ד״ר דרור פייטלסון

פרופ׳ עדי רווה

מידול משתמשים במחשבים מקביליים

חיבור לשם קבלת תואר דוקטור לפילוסופיה

מאת

דוד טלבי

הוגש לסינט האוניברסיטה העברית בירושלים

דצמבר 2006

מידול משתמשים במחשבים מקביליים

חיבור לשם קבלת תואר דוקטור לפילוסופיה

מאת

דוד טלבי

הוגש לסינט האוניברסיטה העברית בירושלים

דצמבר 2006