A Case for Conservative Workload Modeling: Parallel Job Scheduling with Daily Cycles of Activity

Dror G. Feitelson and Edi Shmueli
School of Computer Science and Engineering
The Hebrew University of Jerusalem, Israel

Abstract

Computer workloads have many attributes. When modeling these workloads it is often difficult to decide which attributes are important, and which can be abstracted away. In many cases, the modeler only includes attributes that are believed to be important, and ignores the rest. We argue, however, that this can lead to impaired workloads and unreliable system evaluations. Using parallel job scheduling as a case study, and daily cycles of activity as the attribute in dispute, we present two schedulers whose simulated performance seems identical without cycles, but then becomes significantly different when daily cycles are included in the workload. We trace this to the ability of one scheduler to prioritize interactive jobs, which leads to implicitly delaying less critical work to nighttime, when it can utilize resources that otherwise would have been left idle. Notably, this was not a design feature of this scheduler, but rather an emergent property that was not anticipated in advance.

1 Introduction

Computer workloads have many attributes of which some are more important than others. When modeling these workloads it is often impractical, and sometimes even infeasible, to represent each and every attribute accurately. When this is case, the natural course of action is to focus on the seemingly important attributes which are expected to dominate system behavior, and vaguely represent or even ignore the rest.

Identifying which attributes are important and which can be abstracted away is one of the most difficult problems facing workload modelers. While statistical techniques such as principal component analysis have been developed to assist in this task, they are limited by the fact that they only consider the structure of the workload itself, and not its interaction with the system. In many cases, the modeler therefore relies on experience and intuition to choose the attributes that are believed to be important. However, as we argue and demonstrate in this paper, this might not be enough, and could lead to impaired workloads and unreliable system evaluations.

Daily cycles of activity are a good example of a controversial attribute that is often ignored. In many types of systems there is a significant difference between the workload experienced during the day and the one experienced during the night, mainly in the volume of activity, which is much higher during the day. It might therefore seem reasonable to ignore the daily cycle and model only the daytime workload when evaluating new systems, since a system that can handle the high-volume daytime workload can obviously handle the lower load requirements during the night. This also better matches common simulation methodology, which calls for stationary workloads and steady-state conditions.

But ignoring the daily cycle may have unforeseen consequences. In this paper we present a case study that deals with recent developments in scheduling of parallel jobs. A few years ago Shmueli and Feitelson suggested a workload model that is based on users and sessions, in which users arrive and depart, and sessions are started and aborted dynamically, in reaction to the performance observed from the system [16]. This model has inspired the design of a new class of *user-aware* schedulers, which try to reduce the chances for session aborts, with the goal of improving user satisfaction and the overall throughput and utilization of the system. The question is what is the effect of daily cycles on the evaluation of such schedulers, which in particular seem to target mainly the daytime interactive workload.

To answer this question, we consider a specific scheduler called *CREASY*, which prioritizes interactive jobs that are known to be important to the users, in order to extend their sessions of activity with the system [18]. We compare the performance of CREASY, in simulation, to the performance of the well-known EASY scheduler, that pri-

oritizes jobs solely according to their arrival order in the interest of fairness. The comparison is done in two steps: first without, and then with daily cycles included in the workload.

We show that without daily cycles the performance of CREASY and EASY is the same, which might lead one to mistakenly conclude that there is no advantage for user-aware scheduling over the traditional user-oblivious approach. However, things change when daily cycles are added to the workload. First, the performance levels predicted by the simulation for both schedulers become much more realistic. But more importantly, throughput and utilization improve by up to 50% under CREASY relative to EASY. The daily cycles of activity thus result in not only a quantitative difference for CREASY, but also in a qualitative one, preventing it from being dismissed as ineffective.

We further analyze the behavior of the two schedulers in order to uncover the underlying mechanism that produces this effect. We note that both schedulers use the exact same algorithm to backfill[1] jobs, and neither of them explicitly accounts for the existence of daily cycles in the workload. However, CREASY's prioritization scheme causes more jobs to be submitted during the day, leading to the increase in throughput — and more interestingly, to implicitly delaying less critical work to the nighttime. These delayed jobs could then utilize resources that otherwise would have been left idle, as happens with EASY. This case study thus joins other work [3, 9] in demonstrating that conservative workload modeling — in which workload features of a-priori unknown importance are included rather than being ignored — may lead to more reliable evaluations and have a significant effect on the outcome.

The paper is organized as follows. Section 2 provides background on workload modeling for parallel job scheduling, and reviews the user-aware CREASY scheduler. Section 3 describes our simulation environment and CREASY's performance results, with and without daily cycles in the workload. Section 4 analyzes the results, and uncovers the exact effect of the cycles on performance. Section 5 concludes this study, and provides operational recommendations for workload modeling applicable to both parallel and general computer systems.

[1] Allowing small jobs from the *back* of the queue to jump ahead and *fill* holes in the schedule.

2 User-Based Workload Modeling and Scheduling

The parallel jobs executed on large-scale parallel systems are *rigid*: they require a certain number of processors for a certain duration. To eliminate the detrimental effects of paging, such jobs are conventionally run to completion without preemption. When a job is submitted, the user provides the desired number of processors and an estimate of the runtime. The system's parallel job scheduler uses this information to pack queued jobs together and execute them on the available processors.

Parallel job schedulers have traditionally been evaluated using trace-driven simulations, in which traces of real, production-use parallel systems are played-back to generate the workload [20, 12, 19, 21, 7, 15]. This practice abstracts away the interactive system users who in reality generate the workload for the scheduler. It produces a *static* stream of jobs whose arrival rate is predetermined by the timestamps from the trace. As a result schedulers were designed to focus solely on the packing of jobs, and ignore the users. Likewise, evaluations were based on indirect performance metrics such as the mean job response time and slowdown, that are conjectured to be correlated with user satisfaction.

A few years ago Shmueli and Feitelson suggested a workload model that is based on users and sessions [16]. In their model, the workload is generated by users-models that arrive and depart, and whose sessions of activity with the system are started and aborted *dynamically* during the simulation. Within each session, the activity is regulated by a feedback loop, where system performance affects subsequent submittals as in a closed system model [14]. But in addition, sessions may be *aborted* as a reaction to the (bad) performance observed from the system. This model has immediately inspired the design of a new class of schedulers, which are *user-aware*, and try to reduce the chances for session aborts as a means to improve user satisfaction. This is naturally expected to also improve the overall throughput and utilization of the system.

A detailed study of the behavior of users in parallel systems, on which the above models were based, was presented in [17]. One of the important insights from this study was that the decision of users to continue or abort their sessions with the system depends on the *response times*[2] of their jobs: the shorter the response time, the higher the probability for the users to continue their interactive sessions with the system. Empirical data about this relationship, derived from several production-use parallel

[2] Response time is the total time the job spends in the system, from submission to completion.

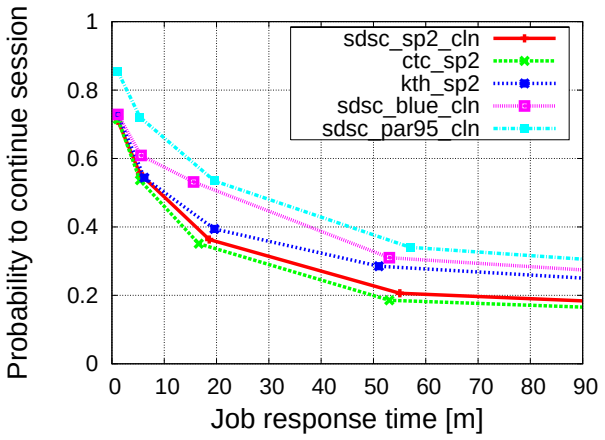


Figure 1: *The relationship between the response times of jobs and the probability for the users to continue their sessions is not linear. Data is derived from several production-use parallel systems.*

system, is shown in Figure 1 and may be approximated using Equation 1. This relationship is not linear, which indicates that jobs with short response times are much more *critical* to the users, in the sense that delaying them, even by the smallest amounts, dramatically increases the chances for session aborts.

$$prob(continue\ session) = \frac{0.8}{0.05 \times job\ resp.\ time + 1} \quad (1)$$

The CREASY scheduler, first introduced in [18], exploits this information in a most intuitive manner. It uses the derivative of Equation 1 to assign high priorities to critical jobs whose *expected response time*[3] is short, and lower priorities to those whose effect on user behavior is already marginal. The rationale is to promote the execution of recently submitted short jobs whose owners are still active and waiting for them to respond, which should motivate them to continue the interaction and submit additional jobs to the system, all within the same session.

CREASY stands for "CRiticality-based EASY", which suggests that it is based on the well-known EASY scheduler, originally developed for the IBM parallel SP system [10]. In fact, CREASY inherits its backfilling algorithm from EASY. The only actual difference between the two schedulers is in the way they prioritize the jobs: while the user-oblivious EASY only accounts for the jobs' arrival order in the interest of fairness, CREASY assigns it priorities to improve user satisfaction, as described above.

The complete prioritization scheme of CREASY comprises two factors:

$$priority(j) = \alpha \cdot criticality(j) + seniority(j) \quad (2)$$

The *criticality* factor is the derivative of Equation 1 which assigns high priorities to the critical jobs, while the *seniority* factor is the job's waiting time in the scheduler's queue, in minutes. The role of the seniority factor is to prevent the starvation of old jobs whose criticality, by definition, is already low, by making sure their priority steadily increases with time, until it exceeds the priority any newly submitted critical job.

Finally, the weight α is used to set the relative importance of the two factors in the calculation, and at the same time to adjust the different units used. With $\alpha = 0$ only seniority takes effect, and CREASY effectively reverts to the original EASY behavior. With $\alpha = 6000$ there is a strong preference for the critical short jobs, which should boost the performance of CREASY. Experiments show that beyond 6000, improvements in performance for CREASY are marginal.

3 Daily Cycles Effect on Performance

The CREASY scheduler described above prioritizes jobs whose expected response time is short, in order to promote their execution and have them respond while their owners are still active in the system. Intuitively, this should reduce the chances for session aborts, motivate the users to submit more jobs, and result in higher overall system throughput and utilization.

To evaluate this hypothesis we used the *SiteSim* simulator, which is a C++ framework designed to accurately emulate the interaction between the users and their system, in order to produce reliable simulation results [16]. The users in SiteSim generate the workload by submitting jobs to the scheduler, and the scheduler in turn schedules the jobs and notifies the users when they complete. This interaction between the users and the scheduler continues throughout the entire course of the simulation, which results in a workload that is generated dynamically and adjusts to the temporal load conditions in the simulated system. An interesting result of this approach is that comparing different schedulers "under the same conditions" does not mean that they will be required to schedule exactly the same sequence of jobs. Instead, it means that they will serve the same workload generation process. This is

[3] Expected response is the sum of the time the job has already spent in the scheduler's queue, and the time it is expected run, which is based on a user estimate.

essential in order to enable the use of different schedulers to lead to different throughput and utilization levels.

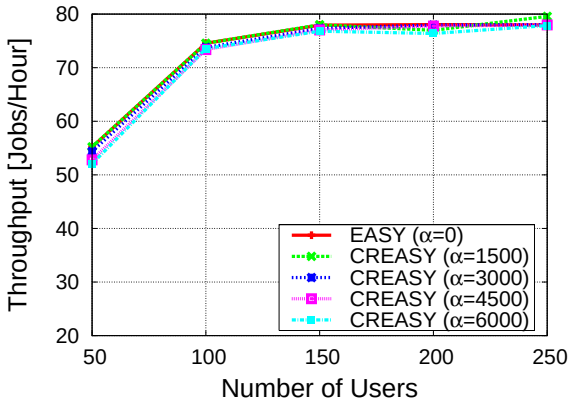
Modeling the scheduler actions upon job arrival in SiteSim is relatively straightforward; it requires the use of an internal wait queue to hold the jobs, and the implementation of a processor allocation algorithm, to process the queue and select jobs for execution, every time a new job arrives or a running job terminates. The scheduler model assumes that jobs are run to completion without preemption on a dedicated set of processors — a common usage pattern typically found in large-scale parallel supercomputers.

Modeling the users, on the other hand, is of course much more involved, and requires a preliminary study of user behavior, like the one by Zilber et al. presented in [22]. This specific study has established the notion of *user sessions* in parallel systems, and defined sessions to be periods of continuous activity by the users, during which they submit one or more jobs to the scheduler. Sessions end when the think time between the completion of a job and the submission of the next exceeds a certain threshold, which the study has identified to be twenty minutes, by analyzing various parallel systems traces.

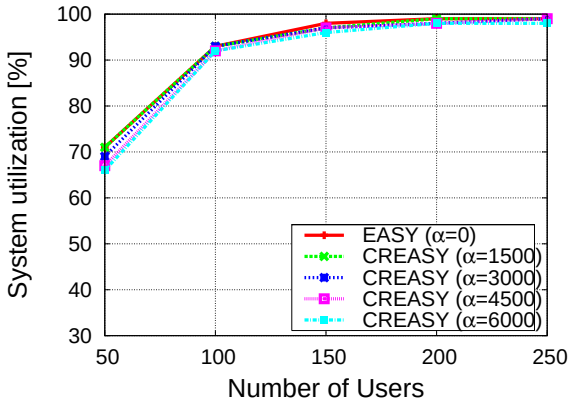
As described above in Section 2, Shmueli and Feitelson complemented Zilber's study by identifying the relationship between the response times of the jobs, and the decision of users to continue or abort their interactive sessions with the system. Together, the two studies provide enough data to allow the development of simple models of the users, which we found to be sufficient for our purpose. More detailed models require the conduction of live experiments with real users, which is beyond the scope of this paper. Actual model parameters, such as the characteristics of the jobs, or the think times in-between jobs, are based on empirical data drawn from five different parallel system traces[4] that are available from the Parallel Workloads Archive [13].

Load in SiteSim is governed by the size of the user population, and is configured at simulation start. We begin by simulating 50 users to experiment with low loads, and gradually added 50 users each time, up to a total of 250 users, to simulate a truly loaded system. For each of five simulated loads, we compared 5 different versions of CREASY, using α values that grow from 0 to 6000 in jumps of 1500. As explained above, when $\alpha = 0$ CREASY behaves identically to EASY, whereas for higher α values, its behavior becomes increasingly user-oriented, which should result in a higher overall throughput. The simulations continue for 4320 simulated hours, equivalent to 180 days — about half a year.

[4]The trace files used are SDSC-Par-1995-2.1-cln, CTC-SP2-1996-2.1-cln, KTH-SP2-1996-2, SDSC-SP2-1998-3.1-cln, and SDSC-BLUE-2000-3.1-cln.



(a) Job Throughput



(b) System Utilization

Figure 2: *Average throughput and utilization* ***without*** *daily cycles: Increasing α does not seem to produce any improvement for CREASY.*

3.1 CREASY *without* Daily Cycles

Figure 2(a) shows the throughput of CREASY for the different α values and load conditions describe above, *without* having daily cycles in the workload. As can be seen, there is virtually no difference in throughput as α increases; for the entire load scale, CREASY with $\alpha = 0$ performs similarly to CREASY with $\alpha = 6000$. Figure 2(b) further complements the above results, showing that there is no difference in system utilization as well.

Given these results one might mistakenly conclude that there is no advantage for user-aware scheduling over the traditional user-oblivious approach, and dismiss CREASY for having no advantage whatsoever over

EASY. All this however will be reversed, when daily cycles are included in the workload, as described next.

3.2 Including Cycles in the Workload

Daily cycles are an obvious and well-known phenomenon, occurring in contexts that range from stock trading [1] to on-line gaming [5]. They are also observed in practically all computer workloads, and are in fact one of the main reasons that such workloads are not stationary. In particular, daily cycles in parallel system workloads have been observed and reported many times [4, 2, 11, 8]. Examples are shown in Figure 3 using data from the Parallel Workloads Archive [13].

We include daily cycles in the workload by partitioning the day into two parts. Users are active and submit jobs only during the daytime, which we define, based on this figure, to be from 8:00 AM to 18:00 PM. During the rest of the time no new jobs are submitted. Thus if the time at which the next job should be submitted is after 18:00 PM in simulated time, this job is delayed to the next day. To make the transition from night to day somewhat less abrupt, users arrive randomly between 8:00 AM and 10:00 AM rather than all arriving at 8:00 AM sharp.

This plain model is similar to the model used by Downey [2], who partitioned the day into two equal parts of 12 hours each. The main difference is that Downey assumed Poisson arrivals during the day, whereas in our model the arrivals are governed by user sessions and dynamics as described above. Furthermore, the model used here is much simpler than the one used in [18], which first introduced CREASY. The latter had four user classes, to support combinations of users who are active during the days and nights, weekdays and weekends. As discussed below, for the purpose of demonstrating the importance of the cycles, the simpler model suffices.

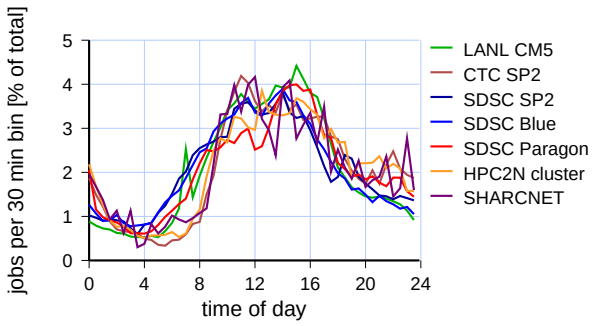
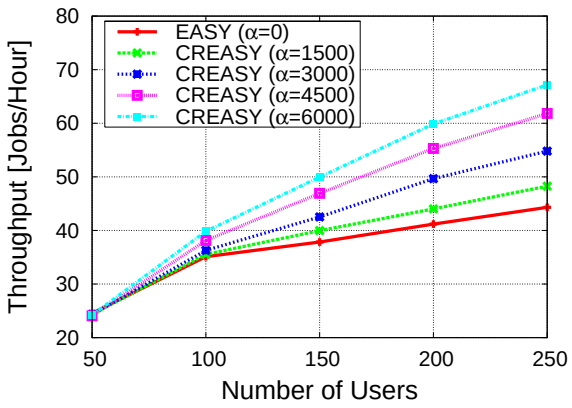
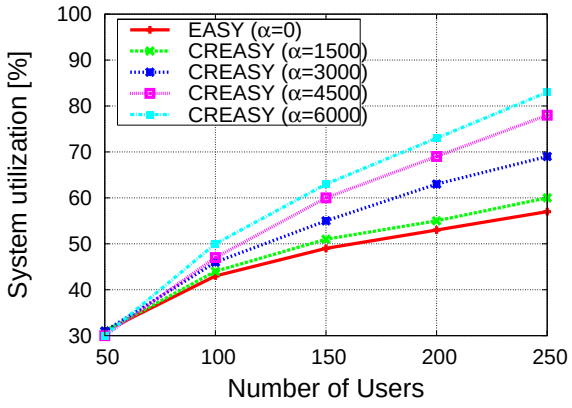


Figure 3: *Examples of daily cycles observed on large parallel systems.*



(a) Job Throughout



(b) System Utilization

Figure 4: *Average throughput and utilization* ***with*** *daily cycles: for $\alpha = 6000$, CREASY outperforms EASY by more than 50%.*

Figure 4 shows the throughput and utilization of CREASY under the workload with the cycles. As clearly seen, both throughput and utilization improve for CREASY as α increases. Under the highest simulated load of 250 users, and for $\alpha = 6000$, the improvement over EASY surpasses 50%. Thus the introduction of daily cycles not only produces quantitative improvement for CREASY, but also a qualitative one, clearly demonstrating the benefits of user-aware scheduling for the system.

Note that the simulations with the daily cycles are also much more realistic in terms of the predicted throughput and utilization. The results of Figure 2, using the simplistic cycle-less workload model, are unrealistically high and would never be achieved in reality. The values predicted with the cycles are much more realistic, and furthermore, match the observed utilization on production machines [6].

4 Detailed Performance Analysis

Given the above results, the natural question is what exactly is the underlying mechanism that produces the observed differences, or in other words, *why is the daily cycle so important for the prioritization of critical jobs to cause such an effect?*

We answer this question by presenting a detailed monitoring of the simulated system as the simulations unfold. We first analyze the system queue behavior, and then we present a detailed characterization of the system's state.

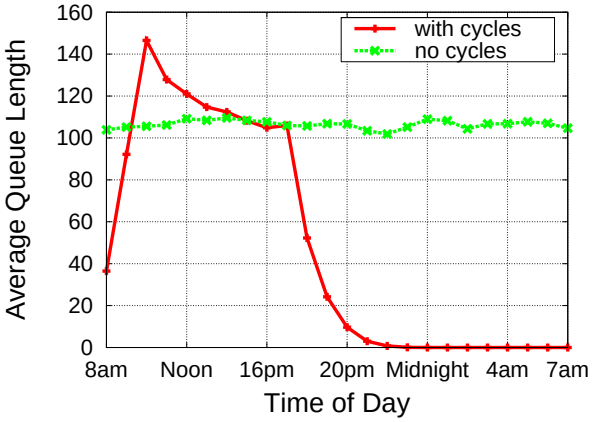
4.1 System Queue Behavior

Figure 5 shows one specific system element: the queue of waiting jobs. Sub-figure (a) shows the average queue length for the EASY scheduler as a function of simulated time of day, where the average is taken over the 180 simulated days. If daily cycles are *not* included in the workload model, this is essentially constant — as may be expected, because the workload arrives in a continuous manner with no specific meaning to events happening at 24-hour intervals. But when a cycle is included, the queue length quickly converges to the above steady-state value during the period when jobs are submitted, and quickly converges to zero when submittals are stopped.

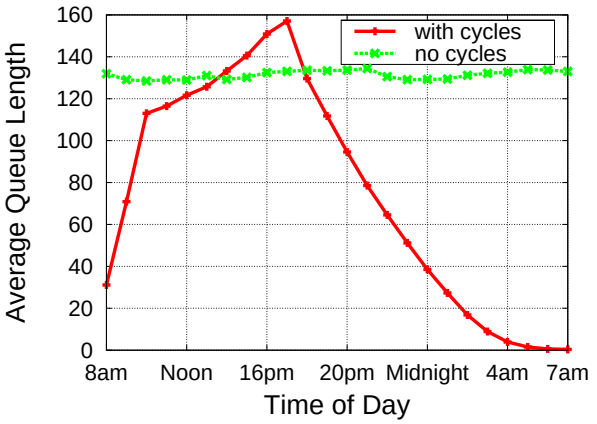
The CREASY scheduler, shown in sub-figure (b), behaves somewhat differently. Again, when there is no daily cycle, the queue length is constant, as explained above. But when a daily cycle is present, the queue length builds up continuously during the day as more jobs are submitted. Then, at night, when submittals stop, it takes a long time to drain and execute all the queued jobs. In effect, CREASY is constantly in a transient state[5], building up load during the day and draining it at night, but never reaching equilibrium. The difference from EASY indicates that CREASY is not queueing jobs randomly. Instead, by virtue of preferring critical jobs during the day, it tends to queue longer jobs that cannot be expected to respond quickly. These jobs accumulate during the day, and are then executed during the night, when the processors would otherwise be left idle, thus increasing the system utilization as we saw in Figure 4(b).

The fact that CREASY is so effective in delaying work for the night creates an interaction between the workload characteristics and the load level. In the simulations depicted here, job runtimes are limited to 1 hour, and the population of users is 250 strong; under these conditions, the queue finally drains around 7 AM, just in time for the new day's work which starts at 8:00. If jobs running for up to 2 hours are allowed, the queue does not drain completely, but the daily cycle still has a significant effect. If jobs up to 4 hours long are allowed, the spillover to the next day is so great that the simulation essentially becomes similar to a simulation without a daily cycle. This implies that with longer jobs sometimes seen in production parallel supercomputers, the maximal supported user base would actually be much smaller than 250, in order to take advantage of overcommitting the resources during the day.

Furthermore, the fact that the queue length increases during the day under CREASY does *not* imply that the



(a) EASY Scheduler ($\alpha = 0$)



(b) CREASY Scheduler ($\alpha = 6000$)

Figure 5: *Daily cycle and average queue length in heavy-load simulations of 250 users. The average at each point is based on 180 samples, from the 180 days of simulated activity. (a) EASY scheduler, with no prioritization of critical jobs. (b) CREASY scheduler, which prioritizes critical jobs.*

[5]This inherent non-stationarity also implies that conventional approaches to computing confidence intervals are invalid, but this is beyond the scope of the present paper.

system is unstable. Given the dynamic nature of the workload generation process, the system as a whole is inherently stable, as users will eventually cease to submit more work if their jobs get delayed in the queue. However, due to the selective nature of CREASY's job prioritization schema, CREASY does not converge to the long-term steady state within one day, and we see it only in its transient state, as opposed to EASY, which converges relatively quickly.

4.2 Characteristics of Running and Waiting Jobs

In order to understand the choices made by the two schedulers, we focus on the characteristics of the running and waiting jobs at different times during the day. Figure 6 uses color maps to show the distribution of jobs in *size*×*runtime* coordinates. The axes are discretized, with each combination of a certain range of sizes and a certain range of runtimes represented by a small square. The shade of the square indicates how many jobs have this combination of attributes. Note that the data is a sample at a particular point in time, not an integration over a certain time window.

The figure has four rows, and four columns. The first two rows correspond to the CREASY scheduler, and other two rows correspond to EASY. The first two columns show samples of the running and waiting jobs at the beginning (10:00AM) and at the end (16:00PM) of the work day. The third column shows a sample from the night time, at 23:00PM for CREASY and 20:00PM for EASY, since EASY's queue drains much faster. For comparison, the fourth column shows the distribution of running and waiting jobs for simulations without daily cycles.

We start by focusing on the sub-figures that are similar for both schedulers. As can easily be observed in the rightmost column, both the distribution of the running and the distribution of waiting jobs are very similar for CREASY and EASY for the simulations *without* the daily cycles. This is expected, as the performance results from Figure 2 also indicate a very similar behavior.

In addition, the distributions of *running jobs* at different times during the day are also largely similar for both schedulers (rows 1 and 3), except perhaps for CREASY running more small jobs at 10:00 AM (leftmost column). This similarity is attributed to the fact that we use statistical sampling, and thus only trace jobs at specific points in time; thus even if CREASY runs many more short jobs than EASY, it is difficult to capture this quantitatively using plain sampling, which results in figures that look alike.

The differences however are clearly seen for the *waiting jobs* in rows 2 and 4, which as opposed to the running jobs that typically run for short periods of time, accumulate in the scheduler queue for relatively long periods. These distributions exhibit two important differences, which we further circled in the sub-figures to help guide the explanation.

First, CREASY is much more effective in servicing small short jobs, whereas under EASY they tend to accumulate in the scheduler's wait queue (small oval at bottom left corner of the daytime waiting distributions, leftmost columns). This is a direct result of CREASY's policy of prioritizing critical jobs in the interest of improving user satisfaction and motivation to submit additional jobs. Second, as an indirect consequence of that policy, large long jobs tend to wait much longer under CREASY compared to EASY (large oval at 4:00PM and nighttime, second and third columns), which in contradistinction allows these jobs to freely compete for execution with the shorter jobs.

The de-prioritization of the large, long jobs is the core enabler of improved throughput of CREASY. By delaying these jobs in the queue, CREASY frees compute resources for the short interactive jobs, and encourages users to continue the interaction with the system. The delayed jobs are then executed during the night, utilizing resources that would otherwise remain idle. Thanks to the daily cycle, these jobs do not starve under CREASY, and the scheduler completely drains its queue prior to the beginning of the next day of activity. This also implies that explicit mechanisms for preventing starvation, such as reservations, may actually be redundant in the presence of daily cycles in the workload.

Another interesting observation from the distributions is that CREASY is quite different from SJF (shortest job first) scheduling, despite the fact that it also prioritizes the short jobs. With SJF, we would expect to see many more short jobs running, and fewer of them waiting in the queue. The figures show, however, that CREASY actually prefers *smaller* jobs over *short* ones.

The reason is that CREASY considers the response times of the jobs, which is the sum of time the job has already waited in the queue, and the time it is expected to run. Thus if short jobs wait for a long time in the queue, their priority decreases under CREASY, so the computation resources can be allocated to interactive jobs whose effect on the users is still significant.

5 Conclusions

When coming to evaluate new systems, it is often hard to anticipate in advance exactly which workload attributes

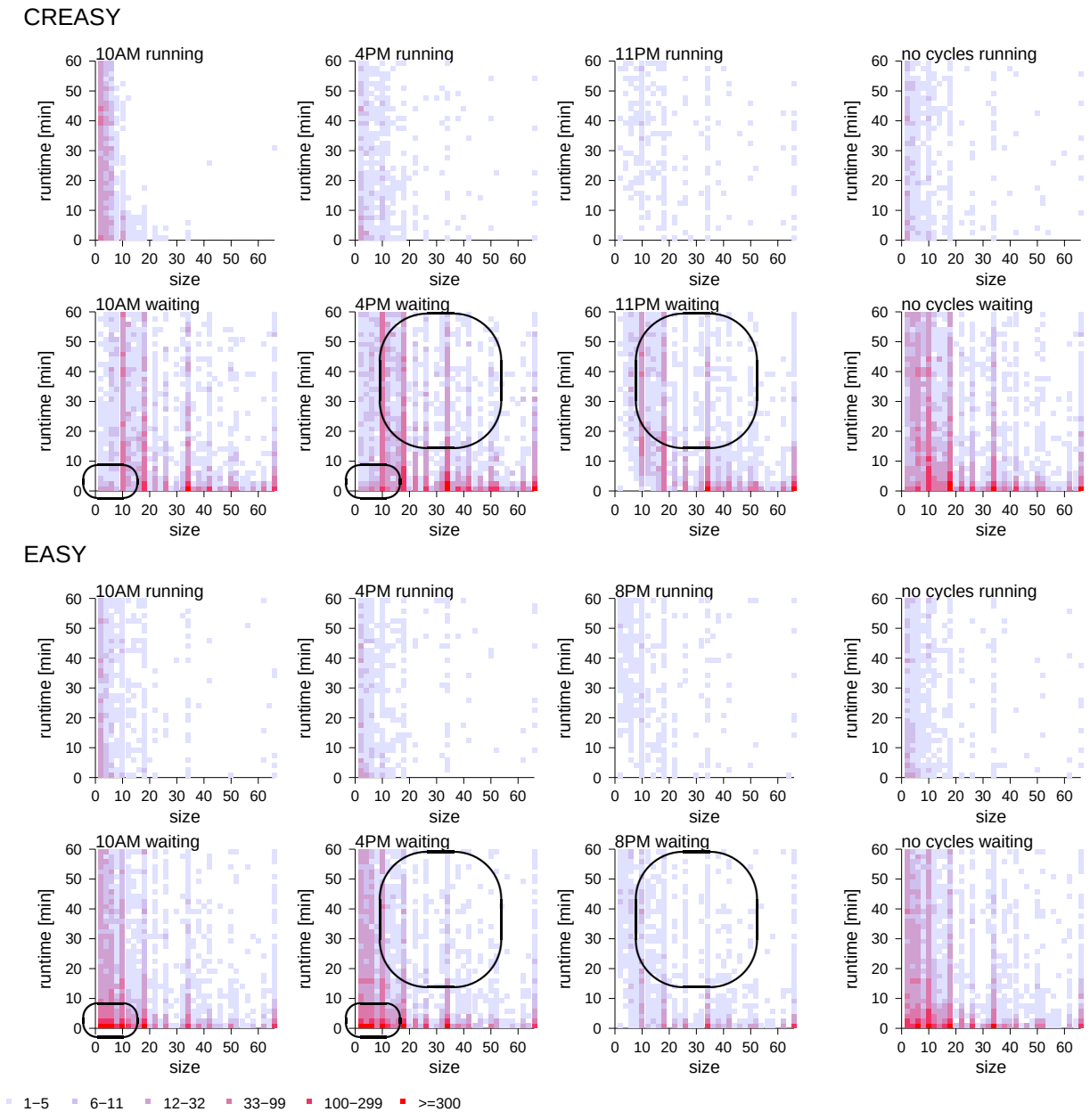


Figure 6: *Characteristics of running and waiting jobs as sampled at different times of the day, with the CREASY and EASY schedulers (250 users). The situation with no daily cycles is shown on the right for comparison.*

will turn out to be important, and which can be abstracted away [3, 9]. When this is the case, two courses of action become available. The more frugal approach is to strive for the most parsimonious model possible, and include in the model only those attributes that are positively known to be important. The alternative and more conservative approach is to exclude from the model only those attributes that have been positively identified as unimportant. Our study suggests the safety of the conservative approach as the default course of action.

Using daily cycles of activity as an example, we have shown that while it might sometimes seem reasonable to ignore them and only model the high loads of the daytime, this could in fact lead to unreliable evaluations. In our case study of user-aware scheduling, excluding daily cycles effectively eliminated significant differences in performance, and could lead one to choose EASY over CREASY as the preferable design option. Surprisingly, the 50% improvement under CREASY did *not* stem from any feature that explicitly takes advantage of the cycles, but rather emerged from the system dynamics — as a consequence of prioritizing interactive jobs, which led to sustaining higher throughput during the day and delaying the extra load to the nighttime. This underscores the importance of conservative modeling, as the effect was not anticipated in advance. Thus conservative modeling is more robust to lapses in our knowledge, which are ever present in the evaluation of new designs.

Acknowledgments

This research was supported in part by the Israel Science Foundation (grant no. 167/03). Our workload data is based on traces from the Parallel Workloads Archive, and we thank all those who have made this data available.

References

[1] R. Almgren and J. Loren, "*Bayesian adaptive trading with a daily cycle*". *J. Trading*, Fall 2006.

[2] A. B. Downey, "*A parallel workload model and its implications for processor allocation*". *Cluster Computing* **1(1)**, pp. 133–145, 1998.

[3] D. G. Feitelson, "*Experimental analysis of the root causes of performance evaluation results: a backfilling case study*". *IEEE Trans. Parallel & Distributed Syst.* **16(2)**, pp. 175–182, Feb 2005.

[4] D. G. Feitelson and B. Nitzberg, "*Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860*". In *Job Scheduling Strategies for Parallel Processing*, pp. 337–360, Springer-Verlag, 1995. Lect. Notes Comput. Sci. vol. 949.

[5] T. Henderson and S. Bhatti, "*Modelling user behavior in networked games*". In 9th *ACM Intl. Conf. Multimedia*, pp. 212–220, Sep 2001.

[6] J. P. Jones and B. Nitzberg, "*Scheduling for parallel supercomputing: a historical perspective of achievable utilization*". In *Job Scheduling Strategies for Parallel Processing*, pp. 1–16, Springer-Verlag, 1999. Lect. Notes Comput. Sci. vol. 1659.

[7] B. G. Lawson and E. Smirni, "*Multiple-queue backfilling scheduling with priorities and reservations for parallel systems*". *SIGMETRICS Perform. Eval. Rev.* **29(4)**, pp. 40–47, 2002.

[8] H. Li, D. Groep, and L. Walters, "*Workload characteristics of a multi-cluster supercomputer*". In *Job Scheduling Strategies for Parallel Processing*, pp. 176–193, Springer-Verlag, 2004. Lect. Notes Comput. Sci. vol. 3277.

[9] H. Li and R. Buyya, "*Model-driven simulation of grid scheduling strategies*". In 3rd *IEEE Intl. Conf. e-Science & Grid Comput.*, pp. 287–294, Dec 2007.

[10] D. Lifka, "*The ANL/IBM SP scheduling system*". In *Job Scheduling Strategies for Parallel Processing*, pp. 295–303, Springer-Verlag, 1995. Lect. Notes Comput. Sci. vol. 949.

[11] U. Lublin and D. G. Feitelson, "*The workload on parallel supercomputers: modeling the characteristics of rigid jobs*". *J. Parallel & Distributed Comput.* **63(11)**, pp. 1105–1122, Nov 2003.

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

[13] "*Parallel workloads archive*". URL http://www.cs.huji.ac.il/labs/parallel/workload/.

[14] B. Schroeder, A. Wierman, and M. Harchol-Balter, "*Open versus closed: a cautionary tale*". In 3rd *Networked Systems Design & Implementation*, pp. 239–252, May 2006.

[15] E. Shmueli and D. G. Feitelson, "*Backfilling with lookahead to optimize the packing of parallel jobs*". *J. Parallel Distrib. Comput.* **65(9)**, pp. 1090–1107, 2005.

[16] E. Shmueli and D. G. Feitelson, "*Using site-level modeling to evaluate the performance of parallel*

system schedulers". In 14th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 167–176, Sep 2006.

[17] E. Shmueli and D. G. Feitelson, "*Uncovering the effect of system performance on user behavior from traces of parallel systems*". In 15th *15th Modeling, Anal. & Simulation of Comput. & Telecomm. Syst. (MASCOTS)*, pp. 274–280, Oct 2007.

[18] E. Shmueli and D. G. Feitelson, "*On simulation and design of parallel-systems schedulers: are we doing the right thing?*". *IEEE Trans. Parallel & Distributed Syst.*, to appear.

[19] S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan, "*Selective reservation strategies for backfill job scheduling*". In *JSSPP '02: Revised Papers from the 8th International Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 55–71, Springer-Verlag, London, UK, 2002.

[20] D. Talby and D. G. Feitelson, "*Supporting priorities and improving utilization of the IBM SP scheduler using slack-based backfilling*". In *IPPS '99/SPDP '99: Proceedings of the 13th International Symposium on Parallel Processing and the 10th Symposium on Parallel and Distributed Processing*, p. 513, IEEE Computer Society, Washington, DC, USA, 1999.

[21] J. William A. Ward, C. L. Mahood, and J. E. West, "*Scheduling jobs on parallel systems using a relaxed backfill strategy*". In *JSSPP '02: Revised Papers from the 8th International Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 88–102, Springer-Verlag, London, UK, 2002.

[22] J. Zilber, O. Amit, and D. Talby, "*What is worth learning from parallel workloads? a user and session based analysis*". In 19th *Intl. Conf. Supercomputing*, pp. 377–386, Jun 2005.