The Dynamics of Backfilling: Solving the Mystery of Why Increased Inaccuracy May Help

Dan Tsafrir Dror G. Feitelson
School of Computer Science and Engineering
The Hebrew University, 91904 Jerusalem, Israel
{dants,feit}@cs.huji.ac.il

Abstract

Parallel job scheduling with backfilling requires users to provide runtime estimates, used by the scheduler to better pack the jobs. Studies of the impact of such estimates on performance have modeled them using a "badness factor" $f \geq 0$ in an attempt to capture their inaccuracy (given a runtime r, the estimate is uniformly distributed in $[r, (f+1) \cdot r]$). Surprisingly, inaccurate estimates ($f > 0$) yield better performance than accurate ones ($f = 0$). We explain this by a "heel and toe" dynamics that, with $f > 0$, cause backfilling to approximate shortest-job first scheduling. We further find the effect of systematically increasing f is V-shaped: average wait time and slowdown initially drop, only to rise again later on. This happens because higher fs lead to increased randomness (more long jobs appear as short and vice versa) and to backfilling longer jobs, overshadowing the "heel and toe" dynamics and limiting the preference for short jobs. Finally, we show that the badness factor fails to capture the badness of real estimates, because these are modal and bounded by a maximal value.

1 Introduction

The workload on a parallel supercomputer consists of a sequence of jobs submitted for execution. These jobs are characterized by their arrival time, their size (number of processors they need), their runtime, and a runtime estimate provided by the user to aid the scheduler in planing ahead. Such estimates are known to be inaccurate, a fact that prompted many researchers to check their actual effect on the scheduling. Much of this research used the "f-model", in which estimates are assumed to be some multiple of the real runtime, and led to some surprising conclusions, such as the claim that inaccurate estimates lead to improved performance. We explain this behavior, and show that this model is actually inappropriate for such research, as it does not reflect salient features of user estimates in real workloads. But we also show that a relatively minor modification improves the model and enables realistic evaluations.

Backfilling. The EASY backfilling algorithm [21, 24] is the most commonly used method for parallel job scheduling at the present time [6], and is the default setting of prominent schedulers like Moab, Maui [15], and LoadLeveler [18]. Upon submittal, users specify the number of processors required by their jobs, and these are placed in a first-come first-serve (FCFS) queue. When enough free processors become available to fill the needs of the first queued job, it is allocated exclusive use of these processors and runs to completion. The problem with this approach is that many

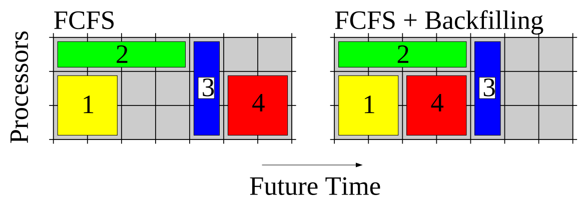


Figure 1: *EASY backfilling reduces fragmentation. It would have been impossible to backfill job 4 had its length been more than 2 time units, as the reservation for job 3 would have been violated.*

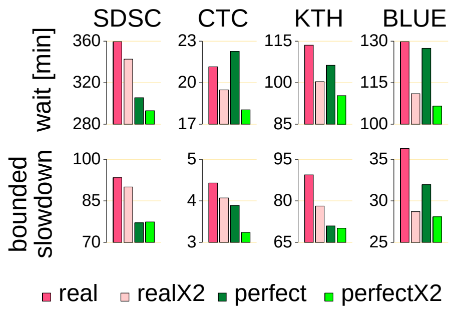


Figure 2: *Average performance usually improves if replacing user estimates ("real") by actual runtimes ("perfect"; f=0). But in both cases, making estimates less accurate (by doubling) tends to help.*

processors may be left idle as they accumulate. To solve this problem, the scheduler makes a *reservation* for the first queued job (the one that needs more processors than are available), for the earliest future time at which enough free processors are expected to be available. The scheduler then continues to scan the queue for smaller jobs (requiring less processors) that can be started immediately without interfering with the reservation. The action of selecting such jobs for execution before their time is called *backfilling*. This is illustrated in Fig. 1.

User Runtime Estimates. Note that backfilling requires the scheduler to know in advance how long each job will run: to compute the reservation for the longest-waiting job (need to know the runtimes of job 1 and job 2 to determine when their processors will be freed), and to know if smaller jobs are short enough to be backfilled (need to make sure job 4 will terminate before the reservation of job 3). Therefore, EASY and other schedulers require users to provide a runtime estimate for all submitted jobs [21]. Jobs that exceed their estimates are killed, so as not to violate subsequent reservations. The assumption was that users would be motivated to provide accurate estimates, because (1) jobs would have a better chance to backfill if their estimates are tight, but (2) would be killed if they are too short. Nevertheless, user estimates are highly inaccurate [10, 2, 24, 3, 7, 20, 33].

Modeling Inaccuracy. To study the sensitivity of backfilling to poor estimates, Feitelson and Mu'alem Weil proposed the "f-model" [10] (IPDPS'98). Given a job J with runtime r, the model postulates that its estimate is chosen at random from a uniform distribution in the range $[r, (f+1)\cdot r]$, where $f \geq 0$ is a predetermined constant.[1] They termed f the "badness factor" because estimates become increasingly inaccurate as f grows, with $f = 0$ indicating completely accurate estimates. The f-model has been used when simulating workloads that lacked estimates data [36, 12, 13], but much more importantly, the model and its variants have been extensively used to study the impact of inaccurate estimates on backfilling algorithms [30, 10, 38, 36, 24, 1, 28, 37, 26, 5, 14]. One simple variant of interest is the "deterministic f-model", in which there is no randomness and estimates are a direct multiple of the runtime and the badness factor plus one: $(f+1)\cdot r$ [38, 1, 5].[2]

[1] f is nonnegative because a job is killed by the system if it tries to run beyond its estimate, so the estimate is never smaller than the runtime.

[2] The first known use of the deterministic model was by Suzuoka et al. [30] in 1995 (the same year in which the first paper about EASY was published [21]), which utilized artificial estimates that are 50% bigger than real runtimes ($f = \frac{1}{2}$) to evaluate the impact of inaccuracy.

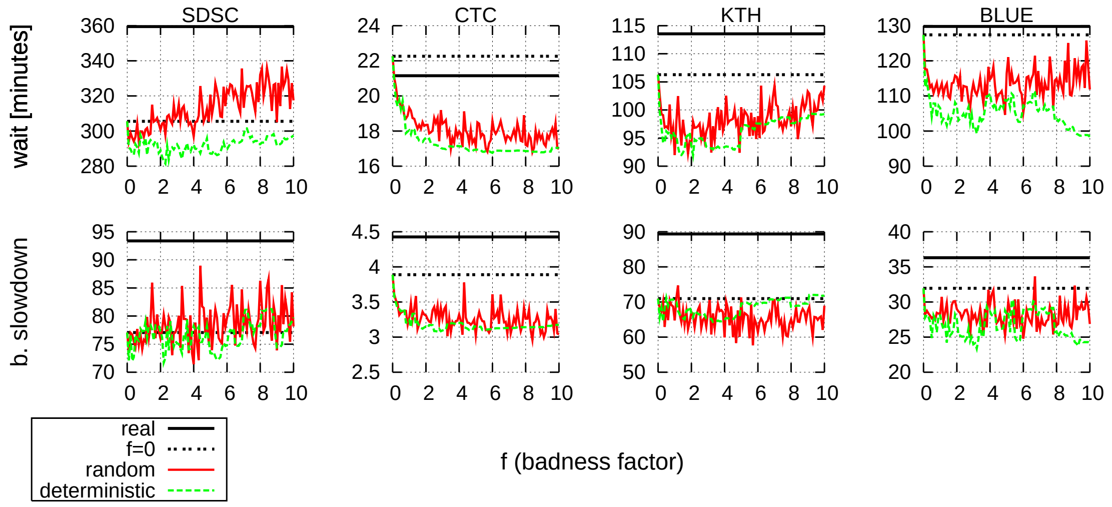


Figure 3: *Performance as a function of f for the random and the deterministic f-model ("real" corresponds to real user estimates). Excluding SDSC, most results associated with positive f values are better (smaller) than the performance associated with f=0.*

The Inaccuracy Mystery. A very surprising result repeatedly reported by most of the aforementioned papers was that, in terms of performance, inaccurate estimates are usually preferable over accurate ones. This is illustrated in Fig. 2 that shows average wait time and slowdown of jobs when simulating the run of four different workloads with various estimates. Evidently, performance improves when deliberately making estimates less accurate by doubling them. This is true both when doubling perfectly accurate estimates and when doubling the original inaccurate user estimates.[3]

While there's wide agreement that making estimates less accurate by multiplying them with some factor is usually beneficial, the effect of the chosen f is less obvious. This is illustrated in Fig. 3. Faced with (usually a small subset of) such results, researchers claimed that the improvement in performance is largely "insensitive" to f [38, 36, 5, 14]. Further, England et al. suggested a new "robustness" metric for computer systems [5] and claimed (in one case-study demonstrating the usefulness of their metric) that

ROBUSTNESS CLAIM

"Our results support those of a previous work and also indicate that backfilling is robust to inaccurate run time estimates in general. It seems that, with respect to backfilling, what the scheduler doesn't know won't hurt it." [5]

The Failure to Explain the Mystery. The fact nonzero badness ($f > 0$) usually improves performance was unanimously explained by what we call the *"holes argument"* [10, 38, 24, 1, 28], as articulated by Chiang et al. [1]:

HOLES ARGUMENT

"We note that for large f (or when multiplying [real] estimates by two), jobs with long runtimes can have very large runtime overestimation, which leaves larger 'holes' for backfilling shorter jobs. As a result, average slowdown and wait may be lower" [1]

[3]In HPDC'99, Zotkin and Keleher conjectured that the improvement obtained when multiplying *perfect* estimates by some factor (as was reported in IPDPS'98 [10]), might also be obtained if multiplying *real* user estimates [38]. This was later verified to be true by Mu'alem and Feitelson in TPDS'01 [24].

At the same time, the observed "insensitivity" of performance to the exact badness value for $f > 0$, was explained by what we call the *"balance argument"* [38, 36, 37, 14], as articulated by Zhang et al. [36]:

BALANCE ARGUMENT

"We can understand why backfilling is not that sensitive to the estimated execution time by the following reasoning. On average, overestimation impacts both the jobs that are running and the jobs that are waiting. The scheduler computes a later finish time for the running jobs, creating larger holes in the schedule. The larger holes can then be used to accommodate waiting jobs that have overestimated execution times. The probability of finding a backfilling candidate effectively does not change with the overestimation." [36]

For example, doubling the lengths of all the jobs in Fig. 1 only means the X-axis is scaled by a factor of two, but doesn't change anything regarding the backfilling decision: indeed, after doubling, job 4 looks twice as long in the eyes of the scheduler, but the same applies to the 2-time-units-hole opened by job 2, so job 4 can still backfill.

While both arguments seemingly make sense, one obvious problem with them is that they are contradictory: If the balance-argument is correct, then there is no benefit in opening those "larger holes" as suggested by the holes-argument, because backfilling candidates would become proportionally larger and cancel the effect. On the other hand, the "holes argument" implies a performance improvement that is proportional to f, in contrast to the balance-argument rationale. Regardless of the contradiction, both arguments fail to explain the results shown in Fig. 3, for example the noisiness of BLUE (performance is actually quite sensitive to f), or the opposite trends observed in SDSC/wait vs. CTC/wait (CTC/wait supports the holes-argument while SDSC/wait contradicts it; both contradict the balance-argument).

Failure to Capture Users' "Badness". The role of a model is, among other things, to truthfully reflect reality. In this respect, according to Fig. 3, the popular f-model fails: It yields unrealistically improved performance results that are consistently better than those obtained when real user estimates are utilized (the only exception is very small f values in CTC/wait). Surprisingly, this issue has been usually brushed aside: In contrast to the other key parameters in parallel workloads (runtimes, interarrival times, number of processors) that received a lot of attention in terms of realistic modeling [9, 4, 17, 2, 3, 37, 22, 23], the dominant estimate model has been the f-model [30, 10, 38, 36, 24, 1, 28, 37, 26, 5, 14], or simply using actual runtimes instead of estimates (f=0) [19, 29, 35, 8, 27]. We conjecture that this can be attributed, to some extent, to the perception that estimates are unimportant because "inaccuracy improves performance" and "what the scheduler doesn't know won't hurt it". But results associated with real estimates (Fig. 3) suggest reality is more complex.

Preview. In this paper we try to solve all the questions raised above: Why are result so sensitive to f, as seen in Fig. 3? Why does multiplying the estimate by a factor usually help? Can we make the model more realistic? To do so, we perform a detailed study of what really happens when f grows, both in terms of performance (Section 3) and in terms of backfilling activity (Section 4). This leads to the heel-and-toe dynamics, which explain the improved performance as resulting from a shift in system behavior towards SJF scheduling (Section 5). We then show why this breaks down with higher f values (Section 6). The conclusion is that the f-model is not very realistic, and a more precise model

Abbreviation	Site	CPUs	Jobs	Start	End	Utilization
CTC	Cornell Theory Center	512	77,222	Jun 96	May 97	56%
KTH	Swedish Royal Instit. of Tech.	100	28,490	Sep 96	Aug 97	69%
SDSC	San-Diego Supercomputer Center	128	59,725	Apr 98	Apr 00	84%
BLUE	San-Diego Supercomputer Center	1,152	243,314	Apr 00	Jun 03	76%

Table 1: *The trace files used to drive the simulations.*

like we propose in [33] should be used; however, the f-model can be improved by imposing realistic restrictions on the estimates (Section 7). Finally, our conclusions are presented in Section 8.

2 Methodology

The experiments are based on an event-based simulation of EASY scheduling, where events are arrivals and terminations. Upon arrival, the scheduler is informed of the number of processors the job needs and its estimated runtime. It can then start the job's simulated execution or place it in a queue. Upon a job termination, the scheduler is notified and can schedule other queued jobs on the freed processors. Job runtimes are part of the simulation input, but are not given to the scheduler.

Tab. 1 lists the four traces we used to drive the simulations. These are available through the Parallel Workload Archive [25]. As recommended, we used the sanitized version of the traces [11, 34]. Since these traces span the past decade, were generated at different sites, on machines with different sizes, and reflect different load conditions, we have reason to believe consistent results obtained in this paper are representative. The traces are simulated using the exact data provided, with the possible modification of replacing real user estimates with those generated by the f-model, as noted.

Scheduling performance is measured using average *wait time* and *bounded slowdown*. The wait time is between a job's submittal and its start time. Slowdown is response time (wait-time plus runtime) normalized by runtime. Bounded slowdown eliminates the emphasis on very short jobs (e.g. jobs with zero runtime), a consequence of having the runtime in the denominator. A commonly used runtime lower-bound of 10 seconds was set, yielding the formula: $\max\left(1\ ,\ \frac{w+r}{\max(10,r)}\right)$, where w and r are the job's wait and running times in seconds, respectively. To reduce warmup effects, the first 1% of terminated jobs were not included in the results; to reduce cooldown effects, jobs terminating after the last arrival were also not included [16].

Finally, we have chosen f's minimal value to be zero, because this seems to be best aligned with our perception of "badness", namely, that "zero badness" implies perfect accuracy. However, it is often more convenient to set the minimum to be one. In such cases we use an uppercase F, defined here to be: $f + 1$. Under these terms, the random model uniformly draws an estimate of a job with runtime r from $[r, r \cdot F]$. Likewise, the deterministic model sets the estimate to be $r \cdot F$. Note that in *all* figures where badness is shown along the X-axis, the random F-model is plotted against the deterministic $F/2$-model, such that both have the same mean.

3 Performance as a Function of Badness

Statistical Confidence. The first step we take in trying to uncover the impact of increased "badness" ($= f$) on performance is to expose the trends underlying the very noisy Fig. 3. To this end, we note that (somewhat surprisingly) no previous work has used the f-model in a statistically sound manner, that is, researchers have consistently inferred performance results associated with a given f from a *single* simulation, despite the model's random component. Fig. 4 shows that plotting performance in terms of mean ("random") and 90% interval ("90% confidence", from the 5th percentile to the 95th percentile) is beneficial, turning the initial noisy results (Fig. 3) into relatively smooth curves.[4]

V Trend vs. L Trend. Fig. 4 reveals two trends:[5] The first is V shaped (most pronounced for SDSC), and the second is L shaped (CTC). In both cases, random performance curves initially drop (improve) for small f values. Then, the curves either asymptotically converge to some value (L shape), or the trend is first reversed and only then converges (V shape). This general tendency continues to larger f values: BLUE is actually V shaped in both metrics (its curves are quite similar to that of SDSC if changing the X scale to $f \in [0, 100]$ and bigger); KTH/wait and KTH/slowdown are L and V shaped, respectively. The deterministic model obviously stays noisy (only one sample per f), but it is evident that its curves are usually found in the proximity of the lower (better) performance bound of the random model.

Load Correlates with Trends. If grouping SDSC and BLUE (V shapes only) and comparing them to CTC and KTH (some L shapes), then Tab. 1 reveals they can be characterized as having higher and lower load, respectively. (In this paper the term "load" means "offered load" or "utilization".) To check whether the load determines if curves are V or L shaped, we simulated all the logs under "high" and "low" load conditions. Load is typically artificially varied by multiplying all arrival times by a constant (e.g. if BLUE's original load is 76%, we can raise it to 80% by multiplying all arrival times by $\frac{76}{80}$). We set the high-load to be that of SDSC (84%), and the low-load to be that of CTC (56%). The results[6] are shown in Fig. 5-6 and suggest that average load is an influencing, yet not the exclusive, factor in determining the performance trend: Trends of SDSC and CTC are invariant to the load change. However, for high load, BLUE and KTH clearly become very similar to SDSC. In contrast, with lower load, the inner-angle of the V curves becomes less "sharp" and somewhat closer to CTC's L curves.

FINDING #1

Expressed in terms of confidence intervals, performance is either V or L shaped. Higher or lower average load implies a tendency towards a V or L shape, respectively. The deterministic model is usually closer to the best performance results of the random model.

[4]Each "random" data point in Fig. 4 is the average of a hundred simulations (with different seeds). There are 101 values of f, that is, $f = 0, \frac{1}{10}, \frac{2}{10}, \frac{3}{10}, ..., 10$), which means each sub-figure involves about 10,000 simulations of scheduling the jobs in the given trace.

[5]While not shown due to space limitations, we have conducted simulations with bigger f values and the result reported here are also based on these additional experiments.

[6] As there are four traces simulated under three (KTH/BLUE) or two (SDSC/CTC) different load conditions (a total of ten trace/load pairs), Figs. 4-6 are the summary of more than 100,000 simulations. Further, to check the consistency of the trends reported in this section, we have also simulated the X ranges: $f \in [0...100]$ ($\Delta X = 1$), $f \in [0...1,000]$ ($\Delta X = 10$), and $f \in [0...10,000]$ ($\Delta X = 100$), which amounts to more than 400,000 simulations.

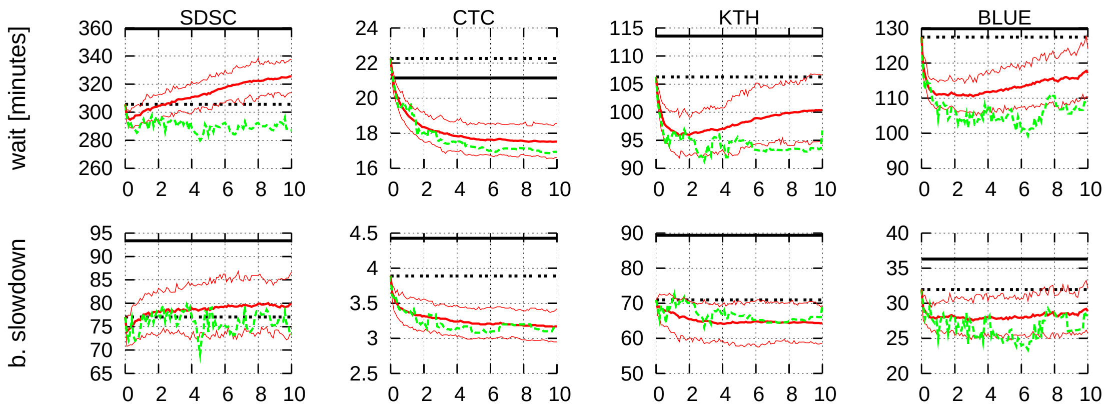


Figure 4: *Confidence intervals expose clearer performance trends (compare with Fig. 3).*

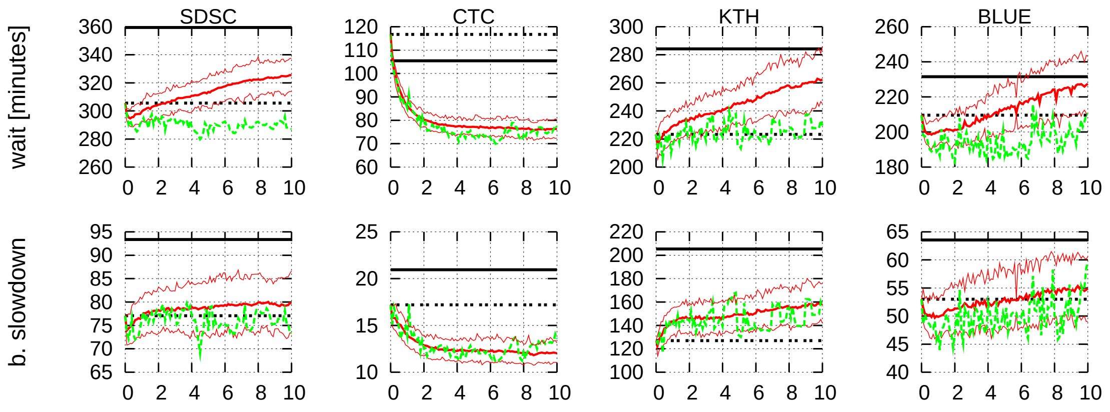


Figure 5: *The high load conditions of SDSC make KTH and BLUE very similar to SDSC.*

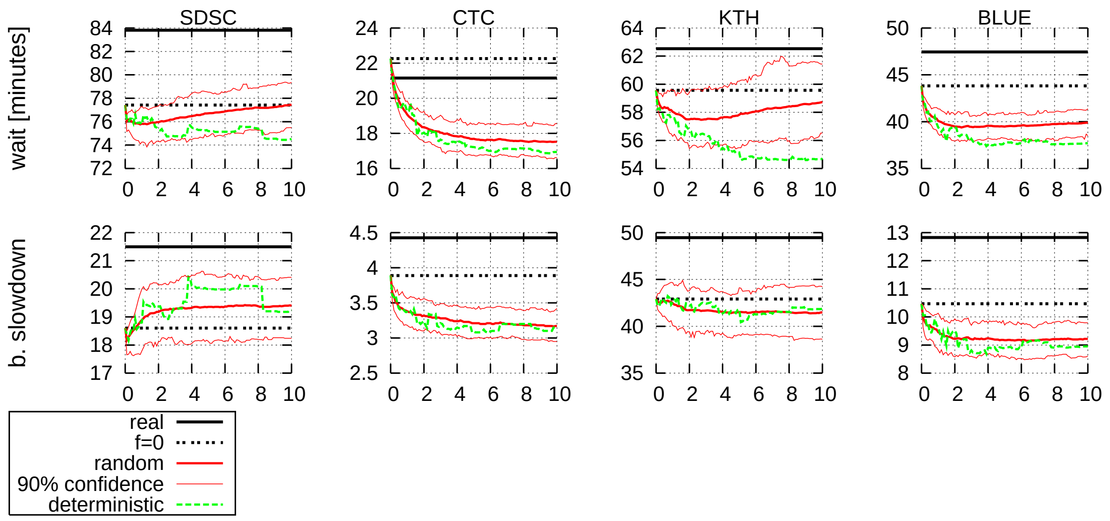


Figure 6: *The low load conditions of CTC make V-shapes less pronounced and closer to L.*

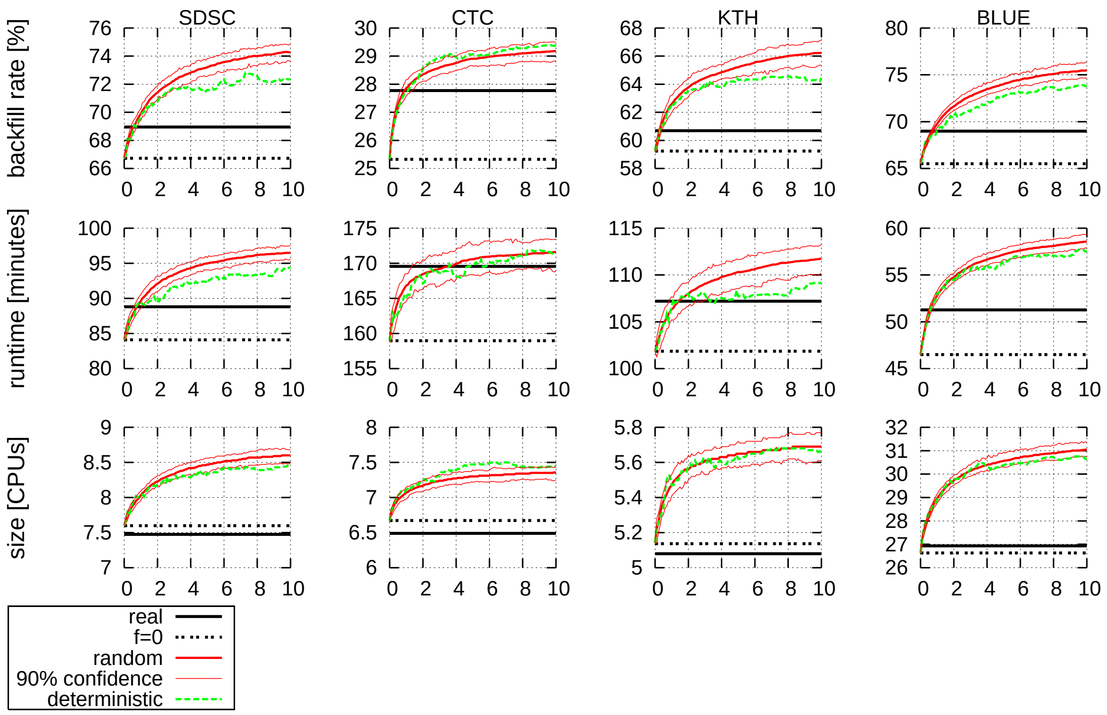


Figure 7: *The percent of backfilled jobs and their average runtime and size monotonically increase with f. In all cases, the relative increase is roughly similar, e.g. the rates/runtimes/sizes associated with f=10 are 10-20% bigger than that of f=0.*

4 Backfilling as a Function of Badness

Holes vs. Balance. Our goal is to understand the reason for the system behavior as reported in Finding 1. A reasonable first step is to validate or disprove the (contradicting) claims underlying the "holes" and "balance" arguments. Though we already know both fail to provide a full explanation to the observed performance trends (e.g. the V shape), determining which argument (if any) better describes the effect of increased f on backfilling is essential. Recall the holes argument implies backfilling activity intensifies with f, whereas the balance argument claims the effect of bigger holes evens out by backfill candidates appearing proportionally longer.

Results. Fig. 7 shows the percent of jobs that were backfilled, as a function of f, along with the main characteristics of these jobs. The trends are consistent and the confidence intervals are tight. Backfilling rates clearly increase with f: The exact numbers are workload dependent in that higher loads (Tab. 1) imply higher rates. But when simulating the logs under equal high/low load conditions (as in Sec. 3), the rates become remarkably similar. The runtime/size of backfilled jobs also follow the same pattern, though in this case the increase is invariant to the examined loads.

FINDING #2

In accordance to the holes argument and in contrast to the balance argument, bigger f implies more jobs that enjoy backfilling. On average, these jobs are longer and wider.

A possible interpretation of this finding is as support for the L-shaped performance curves (CTC, Fig. 4-6). This is based on the notion that jobs can be partitioned into "light" or "heavy" based on whether their characteristics allow

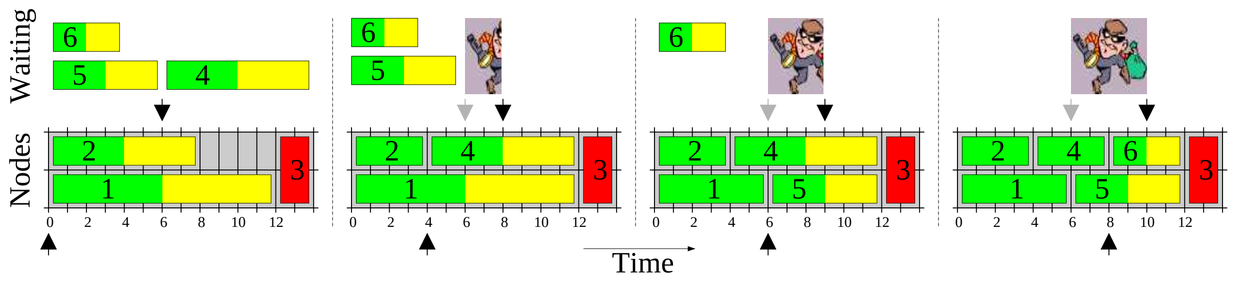


Figure 8: *Illustrating heel-and-toe. Job numbers indicate arrival order. Job estimates are exactly double their runtime (F=2). The left portion of jobs (green/dark) indicates their real runtimes. Due to the doubling, the scheduler views jobs as twice as long (right portion; yellow/bright). The bottom arrows show the progress of time, whereas the top black arrows show the earliest time at which job 3 would have been started, had real runtimes been known (at that particular point in time). The thief's width shows the amount of "stolen" time, at the expense of job 3.*

them to be backfilled or not, and it seemed that bigger f simply means that more jobs become "light" and can enjoy better service. However, as we will show below, our finding doesn't just mean *"more"* jobs. It can also mean *different* jobs, and specifically longer jobs, possibly at the expense of shorter ones.

5 The Heel-and-Toe Dynamics

Heel-and-Toe Hypothesis. The question that follows Finding 2 is why is it so? What's wrong with the balance argument? Why isn't the effect of bigger holes canceled by the backfill candidates that are proportionally longer? After reexamining the backfilling rules, we come up with a possible explanation, as illustrated in Fig. 8. To simplify, assume all estimates are exactly double the runtime (F=2 under the deterministic model as defined in Sec. 2). Based on the information available to the scheduler at T_0 (time 0), it appears the earliest time for J_3 (job 3) to start is T_{12}, even though the *real* earliest start time is actually T_6. Thus, the scheduler makes a reservation on J_3's behalf for T_{12} and can only backfill jobs that honor this reservation. At T_4, J_2 terminates. As J_1 is still running, nothing has changed with respect to J_3's reservation, and so the scheduler scans the wait queue in search of appropriate candidates for backfilling. J_4 (the first backfill candidate under FCFS) fits the gap between T_4 and the reservation (T_{12}) and so it is backfilled, effectively pushing back the real earliest time at which J_3 could have started from T_6 to T_8.

SJFness. This "heel-and-toe" scenario, of repeatedly pushing away the earliest starting point of the first queued job, step by step, may continue until T_{12} is reached. During this time, the window between the current time and the reservation time is continuously shortened, such that waiting jobs that fit this open gap get shorter and shorter, effectively nudging the system towards Shortest-Job First (SJF) scheduling. (Note that the initial open gap can be very short to begin with). And so, if the heel-and-toe dynamic does in fact occur, this limited form of "SJFness" contributes to the performance improvement reported in Finding 1, namely, the first (descending) part of the V-curves, and the L-curves in their entirety. This effect is directly quantified in the next section.

Tendency towards SJFness with positive f was also observed (but not explained) by Zotkin and Keleher [38], which conducted an "off-line" simulation of what happens when *all* the jobs in a trace arrive at the same exact time instance. They found that, in comparison to f=0, shorter jobs leave the system at a faster rate when estimates are set to be five times the actual runtime.[7] The heel-and-toe dynamics explain this phenomenon.

Verifying Heel-and-Toe Occurs. Let J_h be the first queued job (meaning J_h isn't backfilled, but rather, it waits for its turn, becomes first, and gets a reservation). Let S_h denote the *real shadow time* of J_h, defined to be J_h's (hypothetical) start-time, if all estimates suddenly become completely accurate. For example, the initial real shadow of J_3 in Fig. 8 is T_6. During the time J_h is first, we say that a backfill operation is *wild* if S_h is pushed away because of it, or that it's *mild*, otherwise. All the backfill operations in Fig. 8 are wild, because all resulted in a change of the real shadow. By definition, showing that wild backfilling happens means proving that heel-and-toe dynamics indeed occur. Fortunately, detecting wild backfilling is easy within a simulation: We compute S_h by traversing the run-list in (real) termination order and finding the earliest time in which enough free processor accumulate to satisfy J_h. By doing this before/after a backfill operation, we can tell if the operation is wild (S_h changed) or mild (stayed the same).

Fig. 9 clarifies that the heel-and-toe dynamic is not just hypothetical, e.g. with f=10, 2-5% of the jobs are wildly backfilled. The X-axis doesn't start at zero, because there can be no wild backfilling with perfect estimates (the first real shadow is always the last). The consequences of wild backfilling are *delayed* jobs that suffer from at least one wild backfill operation while they are at the head of the queue (as J_3 in Fig. 8). Fig. 10 (top) shows that around 1% of the jobs are delayed. Any performance improvement obtained by the f-model is at the expense of these jobs. The average number of times S_h is pushed away is shown in the middle of Fig. 10 (three times for J_3 in Fig. 8). Finally, the bottom of Fig. 10 shows the average delay duration. This is the elapsed time between J_h's initial real shadow and its eventual start time (the "stolen" time in Fig. 8).

FINDING #3

The heel-and-toe dynamic (1) is verified to occur in practice, (2) reconciles between the balance and holes arguments, and (3) leads to a limited form of SJFness. Thus, it explains the performance improvement due to positive f values.

Let us now explain why performance can also become worse.

6 Countering the SJFness of Heel-and-Toe

We now focus on the second, ascending, part of the V-shaped performance curves where performance continuously degrades (Finding 1; Fig. 4-6). The explanation has two components: the increased f, and the resulting amplification of randomness (for the non-deterministic model). These components increasingly contradict the SJFness reported earlier:

Increased f. As shown in Fig. 7 (and highlighted in Finding 2), backfilling activity monotonically increases with f, while at the same time, the runtime of backfilled jobs becomes longer. Longer average runtime wouldn't have been problematic by itself, had short jobs been nevertheless prioritized. But this is not the case. To illustrate why, let us

[7] Though paradoxically this didn't prevent Zotkin and Keleher from using the balance argument [38].

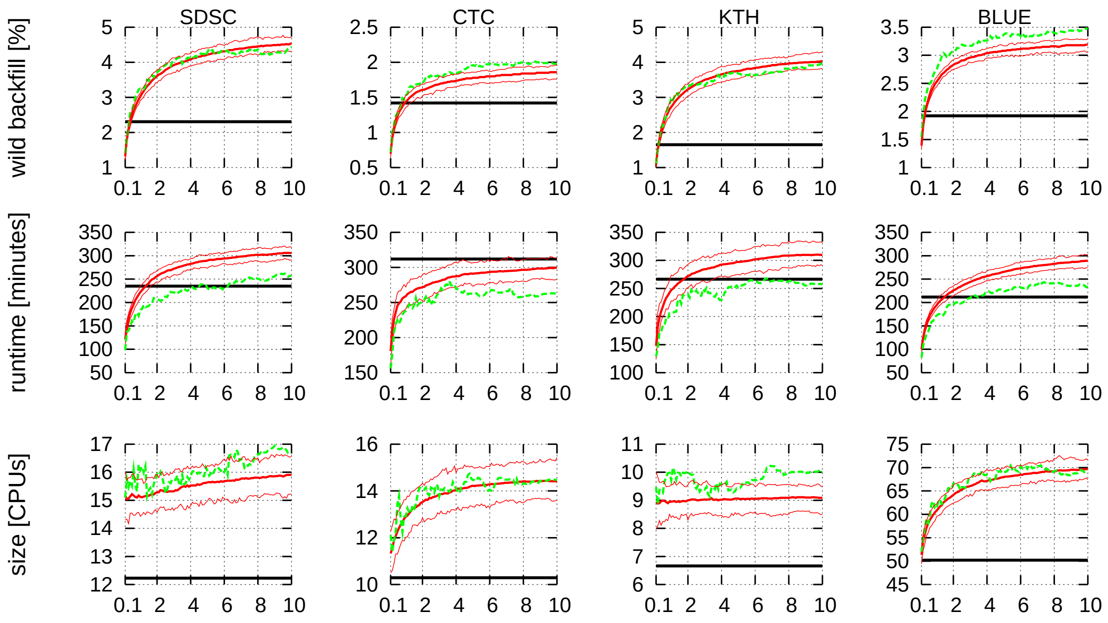


Figure 9: *Existence of wild backfilling demonstrates heel-and-toe dynamics occurs. The rate of wild jobs and their average runtime/size follow the same trends as in the general case (Fig. 7), but wild jobs are longer and wider. Increasing the load has similar effects to those witnessed in Sec. 4 for the general case.*

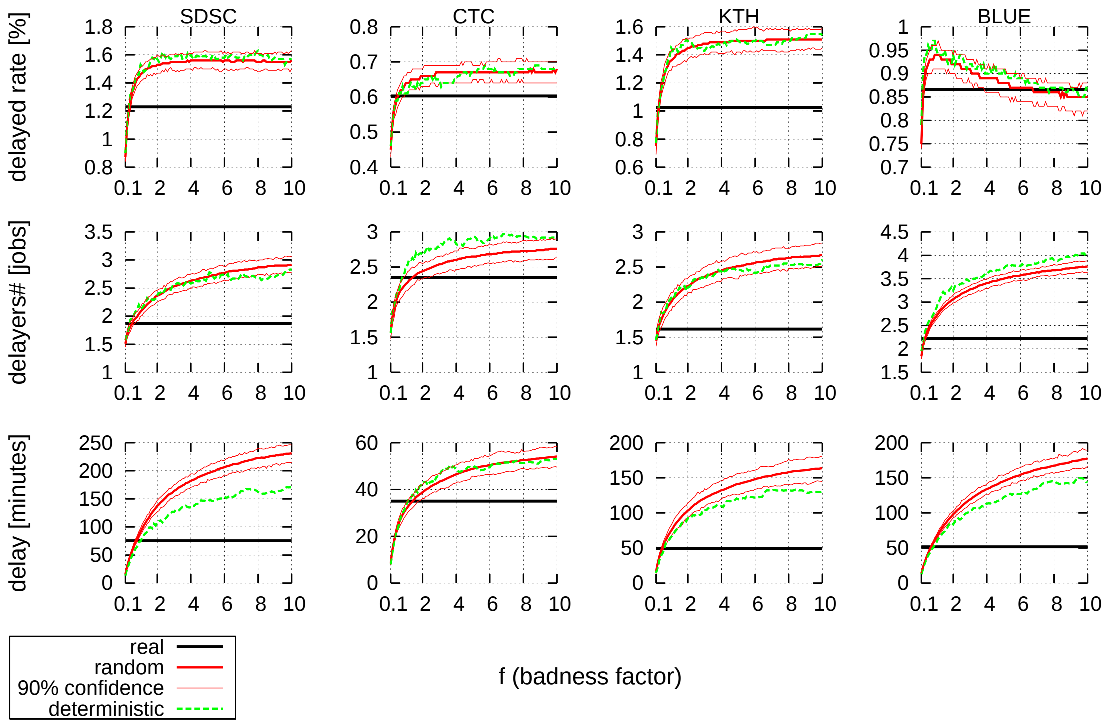


Figure 10: *The rate of jobs that suffer from wild backfilling (top), the average number of wild events per such job (middle), and the average delay (bottom). Note that multiplying the top and middle curves results in the top of Fig. 9. The unique trend observed in BLUE (top) is also displayed by the other logs if simulating high load conditions as in Sec. 3 and examining a slightly wider f range; on the other hand, BLUE becomes like all the others if simulating low load conditions.*

F	estimates				hole length
	J_1	J_4	J_5	J_6	at T_4
1	6	4	3	**2**	2
$1\frac{1}{3}$	8	$5\frac{1}{3}$	**4**	$2\frac{1}{3}$	4
2	12	**8**	6	4	8

Table 2: *The length of the hole in the schedule and the estimates of jobs in Fig. 8, for various F values. The first row (complete accuracy) lists job runtimes and therefore estimates in later rows can be obtained by multiplying this row with the appropriate F. The hole size is J_1's estimate minus 4 (it's now T_4, thus 4 time-units have already elapsed). For each F, the estimate of the first job that fits the hole appears in bold.*

reconsider the scenario depicted in Fig. 8. Tab. 2 lists the estimates of jobs at time T_4 (after J_2 terminates) for various F values, as well as the length of the resulting hole. The last row simply specifies what is shown in Fig. 8 (F=2). Recall that job indexes indicate arrival order, used by the scheduler when searching for backfill candidates. Thus, J_4 is the first candidate and since it fits the existing hole it is chosen for backfilling. However, if the value of F had been $1\frac{1}{3}$ instead of 2 (second row in Tab. 2), then the hole would have been proportionally smaller and the scheduler would have deemed J_4 as too long for backfilling, favoring instead the shorter J_5 for execution. If F was further reduced to 1 (complete accuracy; first row), than J_5 would also appear as too long, effectively making J_6 (the shortest waiting job) the only eligible candidate. We can therefore see there's a subtle tradeoff here:

FINDING #4

While bigger f means more backfilling (which short jobs enjoy more than longer ones), the bigger holes do in fact allow longer jobs to backfill.

This finding is verified in Fig. 11. First, the top row shows the average runtime of non-backfilled jobs: this usually becomes shorter with increased f, suggesting the scheduler indeed makes "wrong" decisions by forcing shorter jobs to wait and preferring longer jobs for backfilling (Fig. 7). More important is the bottom row that directly quantifies the effect: "SJFness" is the percent of jobs that are the shortest in the waiting queue at the time they are chosen to run. Evidently, SJFness intensifies with very small f values, only to monotonically drop later on (perfectly coinciding with our explanation above).

Increased Randomness. The situation gets worse when randomness is introduced, as now, in addition, long jobs can masquerade as short jobs and vice versa. To illustrate this, let J_1/J_2 be two jobs within the wait queue with runtimes r_1/r_2 and estimates e_1/e_2 that were generated by the random model, respectively. This is depicted in Fig. 12 (left), assuming $r_1 < r_2$ without loss of generality. We are interested in $Pr\,(e_1 > e_2)$, that is, the probability the scheduler is erroneously told that J_1 is longer than J_2. By conditioning (Bayes' theorem) this is

$$Pr\,(e_1 > e_2) \;=\; Pr\,(e_1 > e_2 \mid e_1, e_2 \in \alpha\,) \cdot Pr\,(\; e_1, e_2 \in \alpha\;) \;+\; Pr\,(e_1 > e_2 \mid \overline{e_1, e_2 \in \alpha}\,) \cdot Pr\,(\;\overline{e_1, e_2 \in \alpha}\;)$$

where $\alpha \equiv [r_2, Fr_1]$ is the intersection between the two domains from which e_1 and e_2 are drawn. The second term in the above summation is obviously zero (when either e_1 or e_2 are outside α then $e_1 < e_2$) and so we are left with

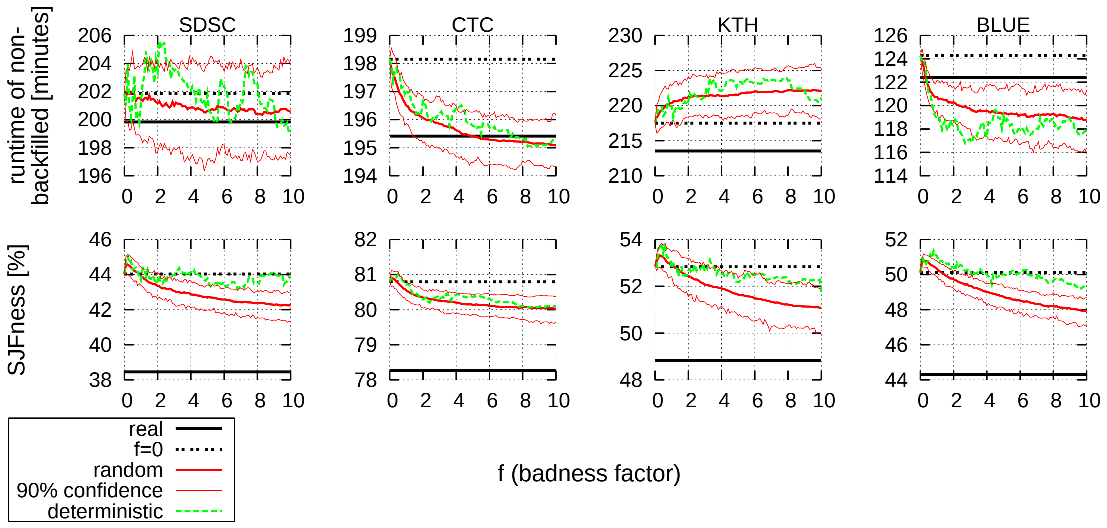
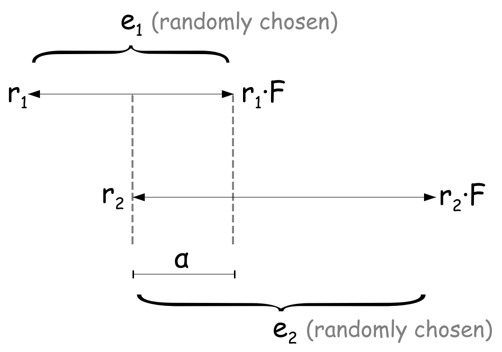


Figure 11: *Average runtime of non-backfilled jobs is usually made shorter when increasing f (top). Average SJFness initially rises, but there's a quick trend change as backfilled jobs become longer.*



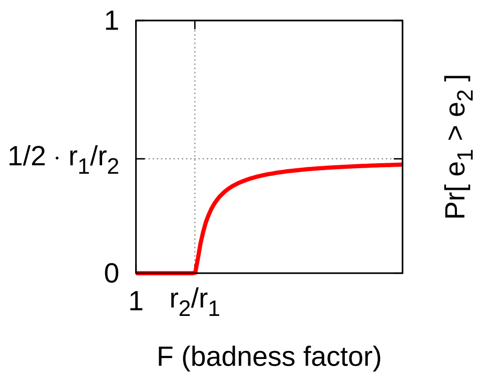


Figure 12: *Left:* r_i *and* e_i *are the runtime and estimate of job* J_i*, such that* e_i *is uniformly chosen from* $[r_i, Fr_i]$ *(*$i = 1, 2$*). Right: the probability* J_2 *would erroneously appear to the scheduler as shorter than* J_1*.*

$$Pr\,(e_1 > e_2) \;=\; \underbrace{Pr\,(e_1 > e_2 \mid e_1, e_2 \in \alpha)}_{\lambda_1} \cdot \underbrace{Pr\,(e_1 \in \alpha)}_{\lambda_2} \cdot \underbrace{Pr\,(e_2 \in \alpha)}_{\lambda_3}$$

If α exists ($Fr_1 > r_2$), then $\lambda_1 = \frac{1}{2}$, because it's simply the probability one number is bigger than another if both are uniformly chosen from the same domain. (If α is degenerated then $\lambda_1 = 0$.) As λ_2 and λ_3 represent standard events in a uniform setting, we get

$$Pr\,(e_1 > e_2) \;=\; \frac{1}{2} \cdot \frac{Fr_1 - r_2}{Fr_1 - r_1} \cdot \frac{Fr_1 - r_2}{Fr_2 - r_2} \;=\; \frac{1}{2} \cdot \frac{F^2}{(F-1)^2} \cdot \frac{r_1}{r_2} + O\left(\frac{1}{F}\right)$$

thus the error probability is monotonically increasing and converges to $\frac{1}{2} \cdot \frac{r_1}{r_2}$ when F goes to infinity (Fig. 12; right).

Finding #5

Under the random model, the bigger the f, the more it is probable the scheduler would erroneously view short jobs as long and vice versa. This explains why SJFness is higher for the deterministic model (Fig 11) and hence why the deterministic model consistently outperforms the random model (Fig. 4-6).

7 Making the Model More Realistic

The Problem. The f-model is *the* dominant model for generating artificial user runtime estimates. It is used to complement workloads that lack estimates data [36, 12, 13], but more importantly, to evaluate the impact of inaccurate user estimates on backfilling algorithms [30, 10, 38, 36, 24, 1, 28, 37, 26, 5, 14]. Based on the f-model, researchers have drawn neat conclusions that range from "performance is independent of accuracy", through "what the scheduler don't know won't hurt it", to "inaccuracy actually improves performance" (see Introduction). Indeed, when employing *artificial* estimates as generated by the f-model, these claims may reflect certain aspects of the truth, as shown above. However, there is a fundamental, yet evidently very elusive and overlooked, problem with all the insights that are based on the f-model:

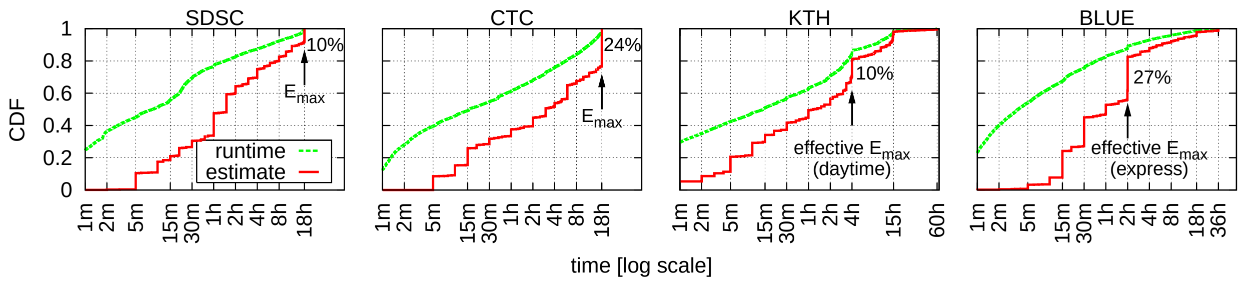
THE PROBLEM WITH THE f-MODEL

Increased inaccuracy that is modeled by greater f values effectively *spreads* the estimates across a larger domain. But with real estimates it's exactly the opposite! Namely, inaccuracy manifests itself by more jobs using the *same* estimate value. Thus, conclusions based on the theoretical f-model might not apply when real user estimates are involved.

Understanding results that are based on the f-model can be interesting and important. For example, the heel-and-toe dynamics turned out to be the reason why, as shown in Fig. 2, doubling of *real* user estimates improves performance. (Doubling is a legitimate scheduling optimization, and there are other related practical issues [32].) Nevertheless, such understandings can have only limited applicability for real systems that employ real user estimates. Importantly, a statement like "inaccuracy improves performance" is a misleading oversimplification: real inaccuracy is actually tightly correlated with degraded performance, as will be exemplified next.

Modality of Real Estimates. Human users don't choose estimates that are uniformly distributed between the real runtime and its multiple with some value. Instead, they use arbitrarily "round" estimates, e.g. 5 minutes, 1 hour, etc. In fact, we found about 90% of the jobs repeatedly use the same 20 "round" values [33]. This modality is reflected in the staircase-like CDF curves shown in Fig. 13, where each mode corresponds to a popular estimate. One particular value that is especially popular is E_{max}, the maximal estimate allowed. This is a derivative of the per-site administrative upper bound on runtimes, which may differ from site to site, but is always enforced. As noted, 4h and 2h serve as the "effective" E_{max} of KTH and BLUE in that most jobs were submitted during daytime or to the interactive/express queues, respectively. E_{max} is used by 10-27% of the jobs and is the most popular in three of the four logs (in SDSC it's ranked third). Its immense popularity can probably be attributed to (1) the fact underestimated jobs are killed by the system upon reaching their estimate, and (2) the inability of users to predict how long their jobs will run and their desire to "play it safe" and prevent their jobs from being prematurely killed.

Implications of Modality. Regardless of the reason for E_{max}'s popularity, the implications are dire in terms of performance. To understand why, consider an extreme case in which *all* jobs use E_{max} as their estimate. The outcome



trace	E_{max}	used by [jobs]	
SDSC	18h	9.7%	
CTC	18h	23.8%	
KTH	4h (weekdays) / 15h (weeknights) / 60h (weekends)	10.1%	(4h)
BLUE	2h (express/interactive) / 36h (others)	27.3%	(2h)

Figure 13: *Top: cumulative distribution function (CDF) of runtimes and estimates. Unlike runtimes, estimates are modal. Runtime- are higher than estimate-curves, as runtimes are always shorter than estimates (underestimates jobs are killed). Bottom: per-site maximal allowed estimate (E_{max}) and its popularity.*

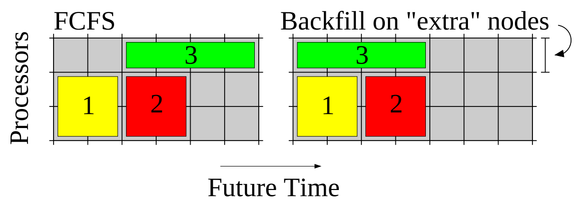


Figure 14: *FCFS can't start J_3 before J_2. But with backfilling, if more nodes than needed will be available for J_2 at its reservation time, these "extra" nodes can be allocated immediately.*

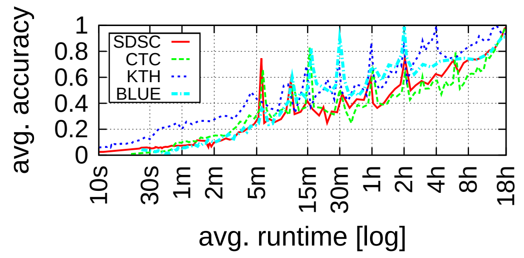


Figure 15: *Average accuracy* $\left(\frac{runtime}{estimate}\right)$ *as a function of runtime. (X-axis groups jobs to 100 equally sized bins according to their runtime.)*

is that all backfilling activity, as shown in Fig. 1, completely stops: The reservation of the first queued job is computed based on estimated termination times of *currently running* jobs, and these will all occur before E_{max} time, by definition. Hence, the reservation itself will occur before E_{max} time and therefore backfilling holes (from the present time until the reservation) are *always* smaller than E_{max}. Since we assume all estimates of waiting jobs are exactly E_{max}, none of them will fit the holes in the schedule. The consequence is that scheduling largely reverts to plain FCFS, which result in a serious performance degradation. (The only remaining backfill activity is on the expense of the "extra" nodes [24], as shown in Fig. 14.)

Surprisingly, the mere existence (and hence popularity) of E_{max} is completely overlooked by researchers. For example, Cirne and Berman conjectured that the apparent connection between longer runtimes and increased accuracy, shown in Fig. 15, is because the more a job progresses in its computation, the grater its chances become to reach successful completion [3]. This false hypothesis ignores the existence of E_{max}, which suggests long jobs are guaranteed to have high accuracy. E.g. assuming E_{max} is 18h, if a job's runtime is 17h, then its estimate must be bigger (between 17h-18h) and thus at least 94% accurate. In other words, long jobs are on the right of Fig. 15, where accuracy is high, while short jobs tend to be on the left, at lower accuracies.

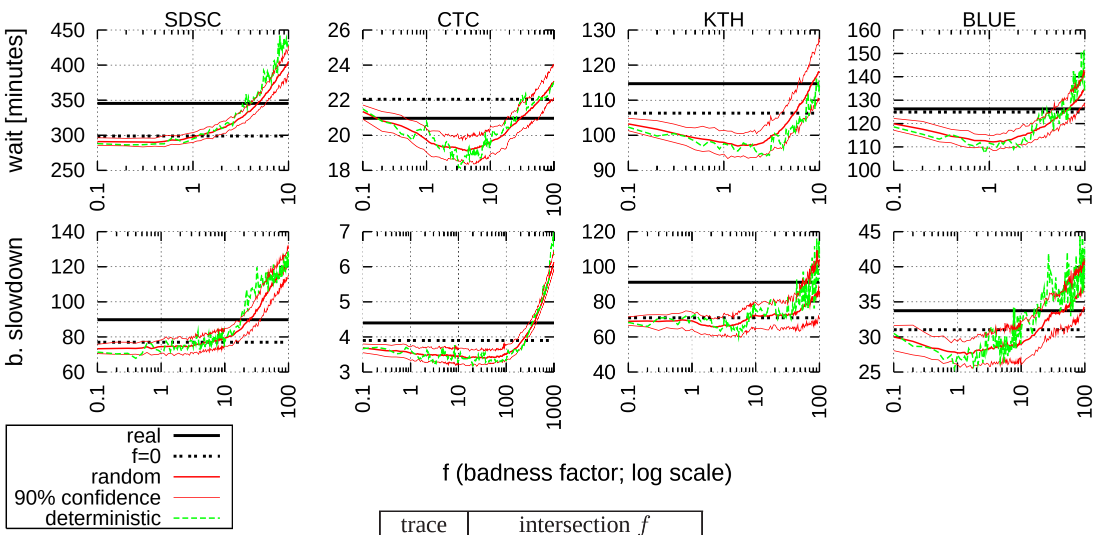
The peaks in Fig. 15 are due to popular estimates (that are smaller than E_{max}) and the many *underestimated* jobs that used them: as these jobs are killed upon reaching their estimates, they have 100% accuracy. But many other jobs that use these popular values are in fact significantly *overestimated*. The problem is that the scheduler has no way to distinguish between such jobs, in contrast to when the f-model is used. To clarify, consider a scheduler that explicitly favors shorter jobs for backfilling [38, 1, 32] and must work with inaccurate estimates. If these estimates nevertheless result in a relatively correct ordering of waiting jobs (as would happen with the f-model), performance can dramatically improve (up to an order of magnitude according to [1]). However, if estimates are modal (as generated by real users), many jobs look the same in the eyes of the scheduler, which consequently fails to prioritize them correctly, which means performance deteriorates. As shown earlier, heel-and-toe dynamics nudge a FCFS-based scheduler towards SJFness, and therefore the same argument applies. Further, an estimate distribution that is dominated by only a few monolithic modes (E_{max} and others) negatively effects performance, because less variance among waiting jobs means less opportunities for the scheduler to exploit existing holes (with various sizes) for backfilling.

Enforcing an Upper Bound on Estimates. The bottom line is that if one wants to model increasing user inaccuracy, one should focus on the modality of user estimates. For example, 10% of the jobs using E_{max} is an optimistic scenario relative to 20%, which in turn is more optimistic than 30%, etc. Modeling increased inaccuracy by gradually associating more jobs with E_{max} is certainly more realistic than using the vanilla f-model. Fortunately, E_{max} can be naturally incorporated within the f-model if instead of using artificial estimates as is, we truncate them to be E_{max} in case they are bigger. Namely, if the artificial estimate is e, we instead use $\min(e, E_{max})$. Let this be denoted as the *truncated f-model*. This model has the property that bigger f values imply more jobs associated with E_{max}.

Fig. 16 shows the results.[8] The truncation has negligible impact for very small f values, because at this points very few artificial estimates exceed E_{max}. The common trend is therefore of improved performance, similarly to the vanilla f-model. Truncation gradually becomes the dominant factor as f increases and so the trend is reversed. The difference between the truncated (Fig. 16) and vanilla (Fig. 4) models when f goes to infinity is that the ascending part of the latter never[9] intersects the curves associated with real user estimates (verified till f=10,000), whereas the former always does. At the intersection point, the truncated model is successful in "capturing the badness" of the real estimates. Thus, with big enough f, the behavior of the truncated model coincides with our claim above that performance degrades if inaccuracy is increased by making the estimate distribution more modal.

An Accurate Model. While the truncated f-model is more realistic than the vanilla one, its output is still fundamentally different than the real thing. A key difference is that only one mode is created (at E_{max}), whereas real estimates exhibit several modes (Fig. 13). Indeed, the E_{max} mode is the most influential, but other modes are also essential in that they significantly contribute to the overall observed effect of bad performance in the face of real user estimates. Further, the E_{max} mode as created by the model is poorly constructed: it consists of long jobs only (with big enough runtimes such that multiplying them with F results in estimates bigger than E_{max}). In reality, many short jobs are

[8] Of an additional 400,000 simulations, in continuation to those mentioned in Footnote 6.
[9] With the exception of BLUE/wait.



trace	intersection f	
	wait	b. slowdown
SDSC	4.77	24.50
CTC	29.82	376.09
KTH	8.36	124.35
BLUE	6.68	47.63

Figure 16: *Performance results obtained with the truncated f-model (compare with Fig. 4). The table specifies the intersection point between curves associated with the "random" model and those associated with real estimates. (Slight differences exist between results associated with real user estimate of the vanilla and the truncated models. This is due to runtimes bigger than E_{max} that unexplainably exist in the original logs and were truncated to make sure they are not bigger than the associated estimates.)*

estimated by users to run E_{max}. Of these, most notable are jobs that fail on startup. Thus, even with the truncated model, the scheduler can still identify shorter jobs better than when real estimates are employed (until a certain F).

For these reasons, we find ourselves in an undesirable situation where each trace/metric combination requires a different f to obtain performance results comparable to that of the real thing (bottom of Fig. 16). This serious drawback is contrasted with the model's simplicity and ease of implementation and use. We therefore view it as the "quick and dirty" substitute for the vanilla version, namely, if faced with the choice of using either one of them, we strongly support the truncated version. It is our opinion that while it is not perfect, it is also not "garbage".

In general, however, we advocate using the more sophisticated estimate model we have developed in [33], instead of the f-model variants. This paper serves in part as motivation. The input of our new model is E_{max} and optionally the percent of associated jobs.[10] The optional argument allows to gradually increase inaccuracy in a truly realistic manner. The output of the new model is a series of modes, where each mode is a pair consisting of an estimate value and the percent of jobs that use it (twenty of which cover 90% of the jobs). This means that in contrast to common practices, estimates are not generated on a per-job basis, but rather, collectively, before hand. Thus, our model also provides a way to map the generated distribution onto a set of jobs with predetermined runtimes, such that each job's assigned estimate is equal to or bigger than its runtime, as required by the backfilling rules. The model is available for download at [31], and was verified to produce results that are almost identical to the real thing [33].

[10]We show that the dissimilarity between estimate distributions of different traces is largely embodied in the percent of jobs that use E_{max} as their estimate; the distributions are otherwise remarkably similar.

8 Conclusions

User runtime estimates are required for backfilling, the most popular scheduling scheme of parallel systems to date. Many studies have claimed that system performance is either unaffected by, robust to, or improves with increasingly inaccurate estimates. The de-facto standard model for obtaining such results has been the f-model that given a runtime r, uniformly chooses the associated estimate from $[r, (f+1) \cdot r]$ at random, or deterministically sets it to be $(f+1) \cdot r$. With this model, bigger fs imply increased inaccuracy. Studies that reported a performance improvement explained it with the "holes" argument, claiming that increased overestimation of long jobs opens larger holes in the schedule for backfilling shorter jobs. In contrast, studies reporting performance is unaffected have used the "balance" argument, claiming the larger holes average out by the fact backfill candidates appear proportionally longer.

We found performance is extremely sensitive to minor changes in f, and that within the noisy results space the contradictory observations about performance-trends are both possible, when using only few samples in a non-systematic manner. However, conducting a statistically sound analysis revealed that the mean effect of increasing f is usually V-shaped: average wait time and slowdown drop at low inaccuracies and the trend is gradually reversed for larger fs (though positive fs yield better results than f=0). To explain this, we show that the seemingly contradictory "balance" and "holes" arguments are both incorrect, or rather, correct to some extent, but miss the key issue that reconciles between them: Performance improvement due to increased f is not simply the result of more backfilling due to more holes in the schedule ("holes" argument), because inflated runtime estimates not only create holes in the schedule, but also enlarge potential backfill jobs, making it harder for them to fit into these holes ("balance" argument). Rather, it is the result of a "heel-and-toe" dynamic: a distinctive sequence of events where small backfill jobs gradually *prevent the holes from closing up* and leading to a preference for short jobs and the automatic production of an SJF-like schedule. When f is very small, the proportionally narrow holes make sure only jobs that are truly short enjoy the effect (explains the initial descending part of the V-shape). However, as f gets bigger, increasingly longer jobs can enjoy it too (explains the ascending part). The situation is worse for the random model, which allows long jobs to masquerade as short and vice versa (explains why the deterministic model yields better performance). We have directly quantified this by measuring the "SJFness" as a function of f, defined to be the percent of jobs that are the shortest in the wait-queue at the time they are started. The result was consistently Λ-shaped, a kind of mirror image to the V performance curves.

Fully understanding of the f-model highlights its fundamental flaw: it leads to a limited SJF-like scheduling, and indeed, SJF is insensitive to multiplying runtimes by some factor as long as the relative ordering of jobs is preserved. But *real* user estimates provide no such ordering! Rather, they are inherently modal, with 90% of the jobs using only 20 "round" estimate values (e.g. 1 hour) and, in particular, 10-27% using E_{max} – the maximal allowed.[11] Any popular estimate is bad for backfilling, as the scheduler can't differentiate between the associated jobs, e.g. they can have 0% accuracy (zero runtime) if they fail on startup, 100% accuracy if they are underestimated and killed by the system,

[11]Probably due to a combination of the inability of users to accurately predict how long their jobs will run and the backfilling policy too kill underestimated jobs.

or anything in between if they reach successful completion. However, E_{max} is especially bad, as the associated jobs appear too long for backfilling and the more there are jobs that use it, the more the schedule resembles plain FCFS.

The bottom line is that the popular claim that "increasingly inaccurate estimates improve performance" is only correct if "inaccurate" means "multiplied by a factor" (as in the f-model), which is far from the truth when real estimates are involved. Inaccuracy of real estimates manifests itself in the form of modality, and "increasing it" means making estimates more modal (e.g. by adjusting the number of jobs associated with E_{max} from 10% to 20%). In this case, *increased inaccuracy actually worsen performance*, as one would intuitively expect. Put in another way, this paper refutes the overwhelmingly accepted myth that inaccuracy improves (or doesn't effect) performance, on the grounds that it is based on false (= unrealistic) assumptions.

We demonstrate the correctness of our findings by suggesting the *truncated f-model*, which adjusts an estimate e that is generated by the vanilla f-model to be $\min(E_{max}, e)$. This creates a mode at E_{max}, such that bigger fs imply more jobs associated with E_{max}. Indeed, one can "manufacture" arbitrarily bad performance results by choosing a big enough f. Importantly, one can always find an f for which results obtained when using artificial estimates, are equal to those obtained when real estimates are employed, in contrast to the vanilla model. We view the truncated model as a simple "quick and dirty" substitute for the vanilla, and contend it should always be preferred over the latter.

Regrettably, the truncated model is not good (= realistic) enough. For example, it generates only one mode (at E_{max}) and only associates longer jobs with it, whereas with real estimates there are several modes and short jobs are associated with all of them. One consequence was that each trace/metric combination required a significantly different f in order to obtain results comparable to that of the real thing. We therefore advocate the use our accurate estimates model as suggested in [33], which was verified to produce results that are remarkably similar to the real thing (both in terms of the estimate distribution and the resulting performance). This model directly targets the modal nature of estimates and allows to gradually increase inaccuracy in a truly realistic manner. It is available for download at [31].

Finally, we note our results have a practical value for scheduling: heel-and-toe dynamics happen also with real user estimates, which actually exhibit a reasonable correlation with runtimes (many jobs get killed when reaching their estimates, leading to complete accuracy). In this context, future work includes the evaluation of estimates' effect on fairness — who pays for the average improvement in performance, and how much, if e.g. all (real) estimates are doubled. We also intend to check whether our findings apply to backfill schedulers with an explicit SJF component (such as those proposed in [1]) and see whether the heel-and-toe dynamics work there as well.

Acknowledgments. This research was supported in part by the Israel Science Foundation (grant no. 167/03).

References

[1] S-H. Chiang, A. Arpaci-Dusseau, and M. K. Vernon, "*The impact of more accurate requested runtimes on production job scheduling performance*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn (eds.), pp. 103–127, Springer Verlag, 2002. Lect. Notes Comput. Sci. vol. 2537.

[2] S-H. Chiang and M. K. Vernon, "*Characteristics of a large shared memory production workload*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 159–187, Springer Verlag, 2001. Lect. Notes Comput. Sci. vol. 2221.

[3] W. Cirne and F. Berman, "*A comprehensive model of the supercomputer workload*". In 4th *Workshop on Workload Characterization*, 2001.

[4] A. B. Downey, "*A parallel workload model and its implications for processor allocation*". In 6th *Intl. Symp. High Performance Distributed Comput.*, pp. 112–124, Aug 1997.

[5] D. England, J. Weissman, and J. Sadago-pan, "*A new metric for robustness with application to job scheduling*". In 14th *IEEE Intl. Symp. on High Performance Distributed Computing (HPDC)*, pp. 135–143, Jul. 2005.

[6] Y. Etsion and D. Tsafrir, *A Short Survey of Commercial Cluster Batch Schedulers*. Technical Report 2005-13, Hebrew University, May 2005.

[7] J. J. Evans, C. S. Hood, and C. S. Hood, "*Exploring the relationship between parallel application run-time and network performance in clusters*". In *IEEE Intl. Conf. on Local Comput. Networks (LCN)*, pp. 538–547, Oct. 2003.

[8] 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.

[9] D. G. Feitelson and M. A. Jette, "*Improved utilization and responsiveness with gang scheduling*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 238–261, Springer Verlag, 1997. Lect. Notes Comput. Sci. vol. 1291.

[10] D. G. Feitelson and A. Mu'alem Weil, "*Utilization and predictability in scheduling the IBM SP2 with backfilling*". In 12th *Intl. Parallel Processing Symp.*, pp. 542–546, Apr 1998.

[11] D. G. Feitelson and D. Tsafrir, "*Workload sanitation for performance evaluation*". In *IEEE Intl. Symp. Performance Analysis of Syst. and Software (ISPASS)*, pp. 221–230, Mar 2006.

[12] E. Frachtenberg, D. G. Feitelson, J. Fernandez, and F. Petrini, "*Parallel job scheduling under dynamic workloads*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn (eds.), pp. 208–227, Springer Verlag, 2003. Lect. Notes Comput. Sci. vol. 2862.

[13] E. Frachtenberg, D. G. Feitelson, F. Petrini, and J. Fernandez, "*Adaptive parallel job scheduling with flexible coscheduling*". *IEEE Trans. Parallel & Distributed Syst.* **16(11)**, pp. 1066–1077, Nov 2005.

[14] F. Guim, J. Corbalán, and J. Labarta, *Impact of Qualitative and Quantitative Errors of the Job Runtime Estimation in Backfilling Based Scheduling Policies*. Technical Report, Computer Architecture Department, Technical University of Catalonia (UPC)., 2006. Submitted.

[15] D. Jackson, Q. Snell, and M. Clement, "*Core algorithms of the Maui scheduler*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 87–102, Springer Verlag, 2001. Lect. Notes Comput. Sci. vol. 2221.

[16] R. Jain, *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, 1991.

[17] J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira, and J. Riodan, "*Modeling of workload in MPPs*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 95–116, Springer Verlag, 1997. Lect. Notes Comput. Sci. vol. 1291.

[18] S. Kannan, M. Roberts, P. Mayes, D. Brelsford, and J. F. Skovira, *Workload Management with LoadLeveler*. IBM, first ed., Nov 2001. ibm.com/redbooks.

[19] J. Krallmann, U. Schwiegelshohn, and R. Yahyapour, "*On the design and evaluation of job scheduling algorithms*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 17–42, Springer Verlag, 1999. Lect. Notes Comput. Sci. vol. 1659.

[20] C. B. Lee, Y. Schwartzman, J. Hardy, and A. Snavely, "*Are user runtime estimates inherently inaccurate?*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn (eds.), pp. 253–263, Springer Verlag, Jun 2004. Lect. Notes Comput. Sci. vol. 3277.

[21] D. Lifka, "*The ANL/IBM SP scheduling system*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 295–303, Springer-Verlag, 1995. Lect. Notes Comput. Sci. vol. 949.

[22] 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.

[23] E. Medernach, "*Workload analysis of a cluster in a grid environment*". In *Job Scheduling Strategies for Parallel Processing*, Jun 2005.

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

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

[26] G. Sabin and P. Sadayappan, "*On enhancing the reliability of job schedulers*". In *High Availability and Performace Computing Workshop (HAPCW)*, Oct. 2005.

[27] E. Shmueli and D. G. Feitelson, "*Using site-level modeling to evaluate the performance of parallel system schedulers*". In *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, Sep 2006.

[28] S. Srinivasan, R. Kettimuthu, V. Subrarnani, and P. Sadayappan, "*Characterization of backfilling strategies for parallel job scheduling*". In *Intl. Conf. Parallel Processing*, pp. 514–522, Aug 2002.

[29] V. Subramani, R. Kettimuthu, S. Srinivasan, and P. Sadayappan, "*Distributed job scheduling on computational grids using multiple simultaneous requests*". In 11th *IEEE Intl. Symp. on High Performance Distributed Computing (HPDC)*, p. 359, Jul. 2002.

[30] T. Suzuoka, J. Subhlok, and T. Gross, *Evaluating Job Scheduling Techniques for Highly Parallel Computers*. Technical Report CMU-CS-95-149, School of Computer Science, Carnegie Mellon University, Aug. 1995.

[31] D. Tsafrir, Y. Etsion, , and D. G. Feitelson, "*A model/utility for generating user runtime estimates and appending them to a standard workload format (SWF) file*". URL http://www.cs.huji.ac.il/labs/parallel/workload/m_tsafrir05, Feb. 2006.

[32] D. Tsafrir, Y. Etsion, and D. G. Feitelson, *Backfilling Using Runtime Predictions Rather Than User Estimates*. Technical Report 2005-5, Hebrew University, Feb 2005.

[33] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "*Modeling user runtime estimates*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson, E. Frachtenberg, L. Rudolph, and U. Schwiegelshohn (eds.), pp. 1–35, Springer Verlag, Jun 2005. Lect. Notes Comput. Sci. vol. 3834.

[34] D. Tsafrir and D. G. Feitelson, "*Instability in parallel job scheduling simulation: the role of workload flurries*". In 20th *Intl. Parallel & Distributed Processing Symp.*, Apr 2006.

[35] S. Vasupongayya, S-H. Chiang, and B. Massey, "*Search-based job scheduling for parallel computer workloads*". In *IEEE International Conference on Cluster Computing (Cluster)*, Sep. 2005.

[36] Y. Zhang, H. Franke, J. Moreira, and A. Sivasubramaniam, "*Improving parallel job scheduling by combining gang scheduling and backfilling techniques*". In 14th *IEEE Intl. Parallel & Distributed Processing Symp. (IPDPS)*, pp. 133–142, May 2000.

[37] Y. Zhang, H. Franke, J. Moreira, and A. Sivasubramaniam, "*An integrated approach to parallel scheduling using gang-scheduling, backfilling, and migration*". *IEEE Trans. on Parallel & Distributed Syst. (TPDS)* **14(3)**, pp. 236–247, Mar 2003.

[38] D. Zotkin and P. J. Keleher, "*Job-length estimation and performance in backfilling schedulers*". In 8th *IEEE Intl. Symp. on High Performance Distributed Computing (HPDC)*, p. 39, Aug 1999.