Using Inaccurate Estimates Accurately

Dan Tsafrir

Department of Computer Science
Technion – Israel Institute of Technology
Haifa 32000, Israel
dants@cs.cs.technion.ac.il

Abstract. Job schedulers improve the system utilization by requiring users to estimate how long their jobs will run and by using this information to better pack (or "backfill") the jobs. But, surprisingly, many studies find that deliberately making estimates *less* accurate boosts (or does not affect) the performance, which helps explain why production systems still exclusively rely on notoriously inaccurate estimates.

We prove these studies wrong by showing that their methodology is erroneous. The studies model an estimate e as being correlated with $r \cdot F$ (where r is the runtime of the associated job, F is some "badness" factor, and larger F values imply increased inaccuracy). We show this model is invalid, because: (1) it conveys too much information to the scheduler; (2) it induces favoritism of short jobs; and (3) it is inherently different than real user inaccuracy, which associates 90% of the jobs with merely 20 estimate values, hindering the scheduler's ability to backfill.

We conclude that researchers must stop using multiples of runtimes as estimates, or else their results would likely be invalid. We develop (and propose to use) a realistic model that preserves the estimates' modality and allows to soundly simulate increased inaccuracy by, e.g., associating more jobs with the maximal runtime allowed (an always-popular estimate, which prevents backfilling).

Keywords: Supercomputing, scheduling, backfilling, user runtime estimates.

1 Context and Background

In a typical supercomputing environment, the supercomputer is a machine that's comprised of up to tens of thousands of nodes, servicing work that is generated by hundreds of users, who collectively submit tens- to hundreds of thousands of jobs. The runtime of jobs ranges from several seconds to tens of hours or more. Jobs can be serial, but more often than not they are parallel. In the context of this paper, "parallel" doesn't mean embarrassingly parallel; rather, each job is comprised of a collection of threads that cooperate and communicate to solve one problem. It is therefore crucial that a job's threads run simultaneously.

Jobs have several attributes, notably, the ID of their submitters (uid), their arrival time, runtime, and size (the number of nodes or processors they require).

E. Frachtenberg and U. Schwiegelshohn (Eds.): JSSPP 2010, LNCS 6253, pp. 208–221, 2010.

Table 1. A typical log file that records the activity of a supercomputer; each line is associated with one submitted job; each column is associated with one job attribute

jobID	*arrival time*	*size*	*runtime*	*estimate*	*uid*	...
1	2010, Apr 24, 12:00:01	2	00:15:37	00:30:00	1013	...
2	2010, Apr 24, 12:05:37	128	01:50:01	18:00:00	1013	...
3	2010, Apr 24, 13:25:20	49	18:00:00	18:00:00	1237	...
...	...	...	...	...	...	...

Table 2. Activity logs we use; see [16] for more details regarding the machines and their workload. We refer to logs in their abbreviated name (leftmost column)

abbrev.	*ver.*	*site*	*cpus*	*jobs*	*duration*	*util.*
CTC	1.1	Cornell Theory Ctr	512	77,222	6/96–5/97	56%
KTH	1.0	Swedish Royal Instit. of Tech.	100	28,490	9/96–8/97	69%
SDSC	2.1	San-Diego Supercomput. Ctr	128	59,725	4/98–4/00	84%
BLUE	2.1	San-Diego Supercomput. Ctr	1,152	243,314	4/00–6/03	76%

We normally think of jobs as rectangles, whereby the vertical dimension is the size, and the horizontal dimension is the runtime. The size if often referred to as the "width" of the jobs (hence jobs can be narrow or wide), and the runtime is often referred to as the "length" of the job (hence jobs can be short or long).

The work submitted by users throughout the lifetime of the machine is recorded in activity logs similar to the one depicted in Table 1.

Many activity logs were collected over the years in the parallel workload archive [16]. Table 2 lists the ones that are used in this study.

2 Backfilling

The baseline scheduling algorithm of most supercomputers is typically rather simple. When a user submits a job, (s)he specifies how many nodes the job needs. The job is then placed in a First-Come First-Served (FCFS) wait queue until enough nodes become free (due to previously submitted jobs that terminate), at which point the job is started, and it runs to completion in batch mode, on a dedicated partition, without ever being preempted. This is illustrated in Figure 1(a).

The problem with FCFS is fragmentation. So all mainstream schedulers employ the optimization that allows job 4 to jump over job 3, provided job 4 doesn't delay job 3, as depicted in Figure 1(b). The act of small jobs jumping ahead before their turn to fill holes in the schedule is called *backfilling* [13].

2.1 Pros of Backfilling: Simple, Effective, and Popular

There are several properties that make backfilling an attractive algorithm: (1) it's simple for users to understand and for developers to implement; (2) it's a batch

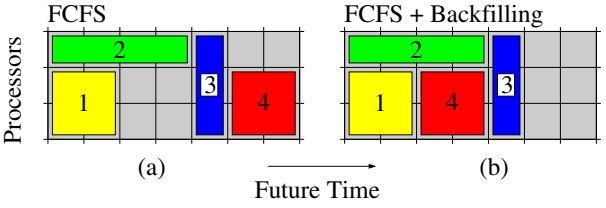

Fig. 1. (**a**) A space/time Gantt chart displaying a FCFS schedule. The X and Y axes denote time and the nodes comprising the machine, respectively. Each rectangle represents a job, such that the rectangle's width and height are the job's runtime and size, respectively. The job numbers indicate arrival order (not arrival time). (**b**) Backfilling reduces fragmentation and improves the utilization by allowing narrow/short jobs to start ahead of their time. Note that it would have been impossible to backfill job 4 had its length been more than 2 time units, as job 3 would have been delayed.

scheduler, which is a virtue in the context of high-performance computing, because applications are often tailored to make use of all available memory, in which case not sharing the memory with others is important; (3) empirical studies show that backfilling improves the utilization of the machines by 10–30 percentage points [12]; and, (4) as it turns out, despite its simplicity, backfilling produces performance results that are a close second to more sophisticated scheduling schemes that, e.g., employ preemption and migration [2,26].

The consequence of the above attractive properties of backfilling is that it became the de-facto standard for supercomputer scheduling. Backfilling is nowadays supported by all the relevant mainstream production products [7], including Load Leveler (by IBM), Maui and Moab (by Cluster Resources), LSF (by platform), OpenPBS and PBS-Pro (by Alair), and GridEngine (by Sun). A survey of the top 50 machines within the top-500-list [4] indicated that 60% of them employ backfilling as their scheduling algorithm [6]. Probably due to its popularity and success, there are many research efforts and papers that deal with backfilling, and many variants were suggested [9].

2.2 Cons of Backfilling: Mandating User Runtime Estimates

There is a price to pay for all the aforesaid attractive properties: in order to operate correctly, a backfilling scheduler must know what's going to happen in the future, namely, it must know in advance how long each job will run.

For example, in Figure 1(b), assume we've just reached T_2 (time unit 2), J_1 (job 1) has just ended, and J_3 (which is the next job in the queue) cannot be started, because there are currently not enough free processors. To enforce the backfilling rule (small jobs can backfill only if, by so doing, they don't delay the first queued job), the scheduler needs to know the runtime of J_2 so as to be able to compute the earliest start time of J_3 (which is T_4). Likewise, the scheduler needs to know that J_4 is short enough so as not delay J_3 if it is backfilled.

To make such determinations possible, users are mandated to provide runtime estimates for each job they submit. And jobs that attempt to exceed their estimates are killed by the system so as not to violate subsequent commitments.

3 Studying the Impact of User Inaccuracy: Wrong Way

The impact of user runtime estimates on the performance of backfilling systems has intrigued many researchers. The first published work we are aware of that investigated the issue was a 1995 technical report from Carnegie Mellon University by Suzuoka et al. [19]; this work came out in the same year as the paper that introduced backfilling [13]. As of this writing, the most recent published work on the subject is an IPDPS 2010 paper by Tang et al. [20], which was awarded best paper (attesting the continued interest in this topic).

These two studies frame 15 years of research (surveyed below) that attempted to understand how inaccurate user estimates affect performance. We argue that the conclusions of most of these research efforts regarding inaccuracy are wrong.

The canonical (and possibly the only) way to study the impact of (in)accuracy of estimates on performance is to: (1) take a workload as depicted in Table 1; (2) manipulate the values within its estimates' column; (3) feed the modified log into a simulator that simulates the run with those artificial estimates; and (4) observe the change in the resulting performance metrics that the simulator outputs. By repeatedly invoking this procedure (initially using completely accurate estimates and then systematically making them less accurate) it is possible to tabulate the performance as a function of the "magnitude" of inaccuracy.

The question is, of course, how to artificially generate those increasingly inaccurate estimates, and how to define and quantify the said magnitude of inaccuracy. We contend that previous studies got this point wrong and that this is why their results are invalid.

The reminder of this Section is dedicated to describing how inaccuracy is typically modeled (Section 3.1); to highlighting the strange, contradictory results such models have yielded and the conflicting attempts to explain them (Section3.2); and to resolving the aforementioned contradiction, while providing the explanation to the counterintuitive results (Section 3.3).

3.1 Modeling Increased Inaccuracy with the *F*-Model

In 1998, to study the sensitivity of backfilling to poor estimates, Feitelson and Mu'alem proposed the "F-model" [8] as follows:

- let r be the runtime of job J,
- let e denote the (artificially-generated) estimate of J,
- let $F \geq 1$ be a "badness factor",
- then e is chosen at random from a uniform distribution $e \in [r, F \cdot r]$.

F was termed the "badness factor", because the artificial estimates start off completely accurate when $F = 0$, and then they become increasingly inaccurate as F grows. Note that, according to the backfilling rules (Section 2.2), F cannot be smaller than 1, since a job must be killed if it tries to exceed its estimate (so as not to violate subsequent commitments), which means $r \leq e$ must always hold.

We note that it is often more convenient to normalize the badness factor so that it would start from zero. We thus set f to be $f = F - 1$ and use the upper- or lowercase notation as is convenient; for the lowercase notation, the following holds:

- $f \geq 0$ (when $f = 0$, the estimates are completely accurate), and
- e is chosen at random from a uniform distribution $e \in [r, (f + 1) \cdot r]$.

The F-model has been used when simulating workloads that lacked estimates data [10,15,25], but, much more importantly, it and its variants have been extensively used to study the impact of inaccurate estimates on backfilling algorithms [1,3,5,8,11,14,17,18,19,20,26,27]; one simpler variant that has been likewise used is the "deterministic F-model" [1,5,15,19,27], in which there is no randomness and each estimate is set to be a direct multiple of the runtime and the badness factor: $e = r \cdot F$.

3.2 The Inaccuracy Mystery

Many of those that utilized the F-model to study inaccuracy reported a surprising, counterintuitive result. They found that inaccurate estimates are usually preferable over accurate ones. This is illustrated in Figure 2 that shows the overall average wait time and bounded slowdown of jobs obtained when simulating the run of the workloads from Table 2 with completely accurate estimates ($F = 1$) and with estimates that are set to be exactly double the runtime ($F = 2$ in the deterministic model). The studies that observed this surprising phenomenon explained it with what we call the *"holes argument"* [1,8,14,15,18,27], as articulated by Chiang et al.:

> ***The Holes Argument:*** *"We note that for large F (or when multiplying 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]

Other researchers that utilized the F-model observed a different, though equally counterintuitive, phenomenon. They found that the performance is insensitive to the (in)accuracy of estimates. This is illustrated in Figure 3.[1] Faced with (a tiny fraction of) such results, researchers concluded that the performance is uncorrelated to F [5,11,20,25,27]. For example, England et al. suggested a

[1] In all plots depicting the behavior of the deterministic and the random model along the same X axis, we divide F by 2 in the deterministic case, so as to make both models have the same mean.

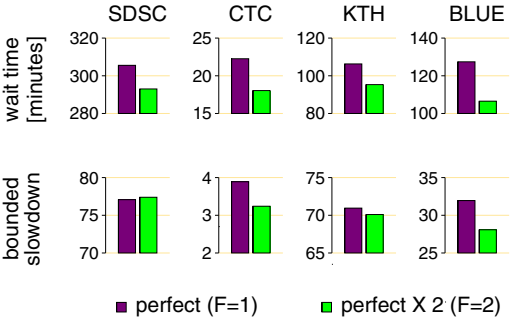

Fig. 2. Lower values mean better performance. Thus, counterintuitively, using completely accurate estimates ("perfect") typically produces inferior results to when estimates are set to be double the runtime ("perfect X 2"). A job's wait time is the duration that elapses between its submission time and the time is starts to run. A job's bounded slowdown is defined to be $\max\left(1\ ,\ \frac{w+r}{\max(10,r)}\right)$, where w and r are the job's wait- and run-times in seconds, respectively; this is a bounded from of the slowdown metric ($\frac{w+r}{r}$) that eliminates the emphasis on very short jobs (shorter than 10 seconds). Performance results throughout this paper are averages across all job.

"robustness" metric for the evaluation of computer systems, and claimed (in a case-study attempting to demonstrate the usefulness of their metric) that:

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

Likewise, Tang et al. argued (in their IPDPS'10 best paper) that:

> ***The Insensitivity Claim:*** *"Our analysis indicates that FCFS [with backfilling] is not sensitive to user runtime estimates."* [20]

Those that attempted to explain this surprising finding have done so with the help of what we call the "balance argument" [11,25,26,27], as articulated by Zhang et al.:

> ***The 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."* [25]

For example, doubling the lengths of all the jobs in Figure 1 only means the X-axis is scaled by a factor of two, but doesn't change anything regarding the backfilling

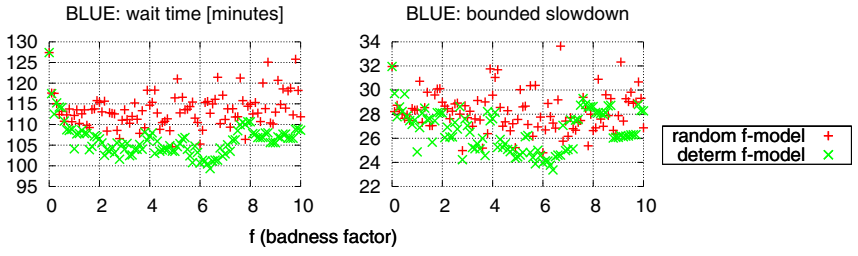

Fig. 3. Performance as a function of the badness factor f, using the random and the deterministic f-models, with a resolution of 0.1 (that is, $f = 0, \frac{1}{10}, \frac{2}{10}, \frac{3}{10}, ..., 10$). Counterintuitively, there is no clear connection between f and the associated performance.

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 backfill before the doubling if and only if it can do so after the doubling.

While both the holes argument and the balance argument 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 longer and cancel the effect. Conversely, the holes argument implies a performance improvement that is proportional to F, in contrast to the balance argument rationale. Indeed, the holes argument seems contradictory to Figure 3, and the balance argument seems contradictory to Figure 2.

3.3 Solving the Mystery: The Heel-and-Toe Backfilling Dynamic

To make sense of the counterintuitive, contradictory findings, we do what inaccuracy studies should have done in the first place but for some reason didn't. Namely, we exploit the random component of the F-model in order to quantify performance in terms of statistical mean and confidence intervals. As it turns out, doing so transforms the noisy results (presented in Figure 3) into well-behaved curves that expose a clear trend as demonstrated in Figure 4.

The fact that a clear trend exists means that all the papers that argued that performance is insensitive to accuracy were mistaken. Their mistake was caused by conducting only a few experiments (a tiny fraction of Figure 3), instead of achieving statistical confidence.

Three of the four simulated logs (SDSC and KTH not shown) produce results similar to that of BLUE as depicted in the left of Figure 4. The performance trend for these logs can be characterized as "V shaped", namely, initially the curves drop (performance improves) and then the trend is reversed (performance worsens). The performance of CTC is "L shaped", asymptotically converging to some value after the initial drop.

We will now explain why the curves behave as they do, starting with the initial drop that indicates performance improvement across all four logs for small

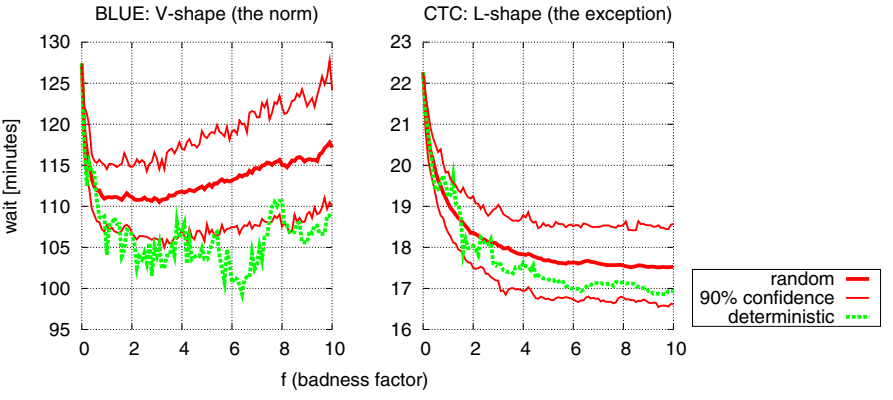

Fig. 4. Averaging multiple experiments exposes a clear trend (compare with Figure 3). For every f (where $f = \frac{1}{10}, \frac{2}{10}, \frac{3}{10}, ..., 10$), we conduct 100 simulations with different seeds; the "random" curve shows the mean of these runs, and the matching "90% confidence" curves show the 5th percentile to the 95th percentile. (We thus performed $100^2 = 10,000$ simulations of scheduling the jobs for each trace.) The deterministic model is unaffected by different seeds (lacking a random component), and so the associated results remain noisy, but we can see that the deterministic curve roughly approximates the best case scenario of the random experiments. Although not shown, the SDSC and KTH logs produce qualitatively similar results to that of BLUE (for both bounded slowdown and wait time) [24].

f values. We begin by noting that, in accordance to the holes argument (and in contrast to the balance argument), backfilling activity intensifies when inaccuracy is increased as shown in Figure 5. Namely, the larger f is, the more jobs enjoy backfilling.

The question is why? What's wrong with the balance argument? Why aren't the bigger holes canceled out by the proportionally bigger backfill candidates? The answer is the "heel-and-toe" backfilling dynamic,[2] which we illustrate in Figure 6 and characterize next. For simplicity, we assume all estimates are exactly double the runtime (F=2 under the deterministic model). 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 it is therefore backfilled, effectively pushing back the real earliest time at which J_3 could have started from T_6 to T_8. (Likewise, when J_1 terminates,

[2] In the Talmud, the expression "heel-and-toe" describes a slow and careful motion, whereby a person advances by repeatedly moving the heel of the back foot adjacent to the toe of the front foot.

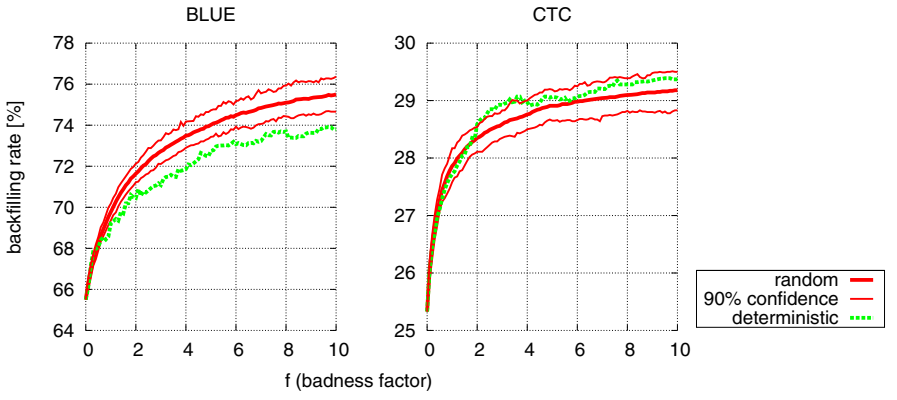

Fig. 5. The percent of backfilled jobs monotonically increases with f

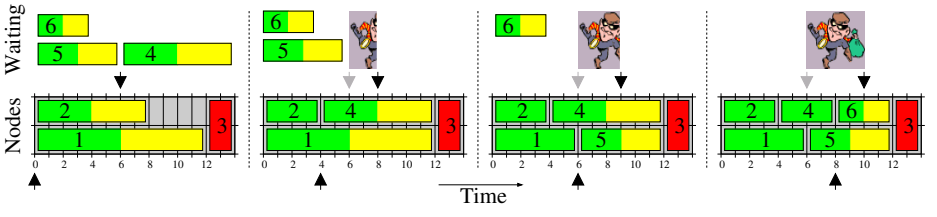

Fig. 6. Illustrating the heel-and-toe backfilling dynamic. Job numbers indicate arrival order. Job estimates are exactly double their runtime (F=2). The left portion of jobs (green/dark) indicates their real runtimes. Due to the doubling, the scheduler views jobs as twice as long (right portion; yellow/bright). 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 to the scheduler (at that point in time). The thief's width shows the amount of "stolen" time, at the expense of job 3.

then J_5 is backfilled, and when J_4 terminates, then J_6 is backfilled, respectively pushing J_3's real earliest start time to T_9 and then T_{10}.)

To exemplify that the heel-and-toe dynamic does indeed occur, we define "wild backfilling" to be a backfill decision that result in a delay of the earliest start time possible of the first queued job (all backfill decision in Figure 6 are wild). We further define the "stolen time" to be the duration of the time interval by which the earliest start time got delayed (in Figure 6 this is 4 time units, from T_6 to T_{10}). Figure 7 confirms that the heel-and-toe dynamic does in fact occur, by showing the wild backfilling rate and average stolen time within the SDSC simulations (again, the other logs are similar).

The heel-and-tow dynamic induces a state whereby shorter jobs (those that fit the steadily shrinking holes) are favored. This explains the performance improvement. In particular, Figure 7 shows that the dynamic intensifies as F grows, explaining the observed performance improvement trend caused by steadily growing inaccuracy (initial part of the V and L curves in Figure 4).

Importantly, notice that the heel-and-tow dynamic reconciles between the contradictory holes argument and balance argument. The performance improvement attributed to positive Fs is not because of wider holes in the schedule that allow for more backfilling (in accordance to the holes argument), because backfill candidates are indeed widened proportionally (in accordance to the balance argument). Rather, it is the result of a heel-and-toe effect that manages to *keep the holes open* by backfilling shorter jobs in a way that repeatedly delays the execution of the first queued job.

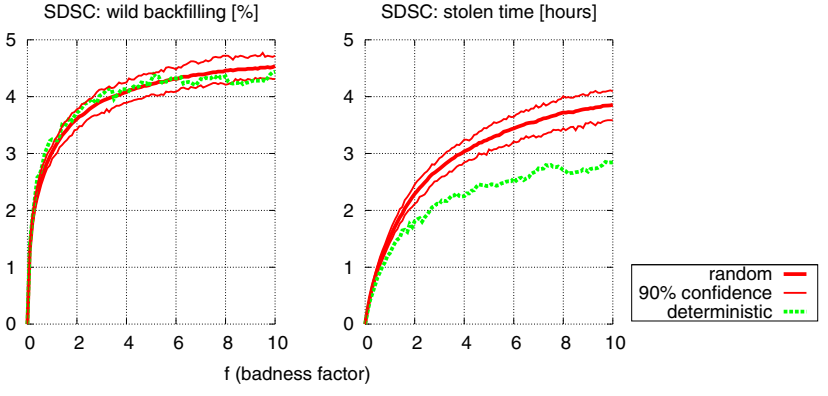

Fig. 7. The impact of the heel-and-toe backfilling dynamic. In SDSC, up to 5% of the jobs are started as a result of a wild backfilling decision (left), causing the first queued jobs to be delayed by up to 4 hours (right), on average.

To finish, we need to explain why performance worsens for all but the CTC log (the ascending, right part of the V shape) and why CTC is different (L-shaped). The explanation is detailed elsewhere [21,24], and, to keep the discussion focused, we only provide the intuition here. The performance is worsened, because the probability that the scheduler will "mistake" short jobs for long (and vice versa) monotonically increases with F. Formally, if the runtime of two jobs J_1 and J_2 is r_1 and r_2, respectively, and we assume (without loss of generality) that $r_1 < r_2$, then we can prove that the probability $Pr(e_1 > e_2)$ monotonically increases with F, where $e_1 \in [r_1, r_1 \cdot F]$ and $e_2 \in [r_2, r_2 \cdot F]$ are the randomly chosen estimates of J_1 and J_2, respectively. (And this is, by the way, the reason why the deterministic curve roughly follows the best case scenario of the random model in Figure 4, as this probabilistic argument doesn't apply.) The reason CTC is largely unaffected by the probabilistic argument, is that it lacks the bursty nature that the other workloads have (meaning, the wait queue is typically short and so the chances of making the $Pr(e_1 > e_2)$ mistake are smaller); when artificially introducing burstiness to CTC, the associated performance curves become V-shaped like that of the other workloads.

4 Studying the Impact of User Inaccuracy: Right Way

The previous section might seem to suggest that increased inaccuracy improves performance (thereby corroborating the conclusions of many past studies). Nothing can be further from the truth. The conclusions of the previous section are based on *artificial* inaccuracy as generated by the F-model, whereas *real* inaccuracy (as manifested by real users) is inherently different, and it affects performance in an entirely different way. In reality, less accurate estimates worsen the performance, in contrast to what we've learned in the previous section [23].

As it turns out, inaccuracy of human users takes the form of utilizing very few "round" values, such as 15 minutes, 1 hour, and oftentimes the maximal runtime allowed. In fact, in all of our logs, merely 20 such canonical values are used by 90% of the jobs as estimates. This user behavior is clearly evident from Figure 8, which plots the cumulative distribution function of the runtimes and estimates of jobs. In contrast to the smooth runtime curve, the estimates form a rigid staircase-like structure, whereby each stair is associated with a single popular estimate value.

Importantly, the modality of estimates hampers backfilling systems, because jobs with different runtimes all look the same to the scheduler, preventing it from distinguishing between short and long jobs and limiting its ability to utilize existing holes in the schedule. (Conversely, by definition, the F-model provides a fairly good relative ordering of the jobs and a lot of variability for the scheduler to work with.)

Especially harmful to performance is the fact that the maximal allowed runtime is always a very popular estimate value among users (e.g., in the case of

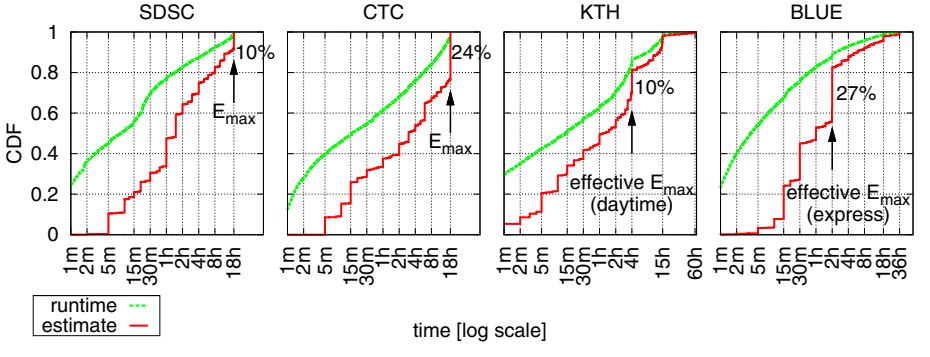

Fig. 8. Cumulative distribution function (CDF) of jobs' runtimes and estimates. The runtime curves appear higher because runtimes are always shorter than estimates (underestimated jobs are killed). We denote the maximal allowed runtime as E_{max}. (This is also the maximal allowed estimate, as jobs are allowed to run until their estimate is reached.) In SDSC and CTC the E_{max} is 18h; in KTH and BLUE, 4h and 2h serve as the "effective" E_{max}, because most jobs were submitted during daytime or to the express queue, respectively, and 4h and 2h are the associated limits enforced on those systems. Clearly, E_{max} is a popular value.

CTC, 25% of the jobs utilized this value as estimate; see Figure 8). It is harmful because such jobs are *never* backfilled. Indeed, if all the jobs choose the maximum as their estimate, then there would be no backfilling activity, and the schedule would largely revert to plain FCFS.

The bottom line is that, when researchers wish to assess the impact of inaccuracy on performance, they should *not* use the F-model, as using it for this purpose constitutes a serious methodological error that would likely invalidate their results [23]. When using the F-model, researchers simply convey to the scheduler too much information that it would probably never enjoy in reality (fairly accurate relative ordering of jobs), and they additionally induce the heel-and-toe dynamic, which further unrealistically improves the results of their evaluation by artificially favoring shorter jobs.

The correct way to evaluate the impact of increased inaccuracy is by making the estimates distribution more modal. (For example, by associating an increasing number of jobs with the maximal runtime allowed.) In a different work, we have developed a detailed model that accurately captures the modal nature of the estimates distribution and that allows its users to control the "amount" of modality [23]. The model is freely available for download [22] from the parallel workload archive [16].

5 Conclusions

Backfilling drastically improves the system utilization [12] by allowing jobs to run ahead of their time, provided they do not delay higher-priority jobs. But in order to do so, backfill systems require users to estimate how long their jobs would run. Ever since the inception of backfilling, researchers wondered about the impact of inaccurate estimates on performance, and many studies addressed this issue (surveyed above).

To evaluate the impact of inaccuracy, researchers associated each job with an artificial estimate e that is a multiple of the actual runtime r with some "badness" factor F, such that $e = r \cdot F$ (or such that e is correlated with $r \cdot F$); with this "F-model", larger Fs supposedly imply increased inaccuracy. Relying on the F-model, researchers repeatedly reached a counterintuitive conclusion: that performance improves by (or is insensitive to) increased inaccuracy.

In this paper we have refuted this counterintuitive conclusion, by exposing the F-model to be erroneous. It artificially conveys too much information to the scheduler (the relative ordering of jobs), and, in addition, it implicitly nudges the system towards shortest-job scheduling through a "heel-and-toe" dynamic that manages to keep backfilling windows open at the expense of the first-queued job. In contrast, the inaccuracy of real user estimates worsens the performance, because users utilize very few "round" estimates (especially the maximal runtime), making it hard for the scheduler to distinguish between long and short jobs and hindering its ability to backfill effectively (e.g., jobs with the maximal runtime as estimate would never be backfilled).

We thus proclaim that the popular F-model is inappropriate for being used in studies that wish to learn the effect of inaccurate user estimates. Researchers

should stop using multiples of actual runtimes as estimates, or else they would likely get invalid results. To get trustworthy results, researchers should preserve the modal nature of user estimates [23]. We have made available a model that does so [22], and we recommend to prefer it over the F-model; with the suggested model, researchers can explore the impact of increased user inaccuracy by, e.g., increasingly associating more estimates with the maximal runtime allowed.

References

1. Chiang, S.-H., Arpaci-Dusseau, A., Vernon, M.K.: The impact of more accurate requested runtimes on production job scheduling performance. In: Feitelson, D.G., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2002. LNCS, vol. 2537, pp. 103–127. Springer, Heidelberg (2002)
2. Chiang, S.-H., Vernon, M.K.: Production job scheduling for parallel shared memory systems. In: 15th IEEE Int'l Parallel & Distributed Processing Symp (IPDPS) (April 2001)
3. Dimitriadou, S., Karatza, H.: Job scheduling in a distributed system using backfilling with inaccurate runtime computations. In: IEEE Int'l Conf. Complex, Intelligent & Software Intensive Systems (CISIS), pp. 329–336 (February 2010)
4. Dongarra, J.J., Meuer, H.W., Simon, H.D., Strohmaier, E.: Top500 supercomputer sites, `http://www.top500.org/` (updated every 6 months)
5. England, D., Weissman, J., Sadago-pan, J.: A new metric for robustness with application to job scheduling. In: 14th IEEE Int'l Symp. on High Performance Distributed Comput. (HPDC), pp. 135–143 (July 2005)
6. Ernemann, C., Krogmann, M., Lepping, J., Yahyapour, R.: Scheduling on the top 50 machines. In: Feitelson, D.G., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2004. LNCS, vol. 3277, pp. 17–46. Springer, Heidelberg (2005)
7. Etsion, Y., Tsafrir, D.: A Short Survey of Commercial Cluster Batch Schedulers. Technical Report 2005-13, The Hebrew University of Jerusalem (May 2005)
8. Feitelson, D.G., Mu'alem Weil, A.: Utilization and predictability in scheduling the IBM SP2 with backfilling. In: 12th IEEE Int'l Parallel Processing Symp (IPPS), pp. 542–546 (April 1998)
9. Feitelson, D.G., Rudolph, L., Schwiegelshohn, U.: Parallel job scheduling — a status report. In: Feitelson, D.G., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2004. LNCS, vol. 3277, pp. 1–16. Springer, Heidelberg (2005)
10. Frachtenberg, E., Feitelson, D.G., Petrini, F., Fernandez, J.: Adaptive parallel job scheduling with flexible coscheduling. IEEE Trans. on Parallel & Distributed Syst. (TPDS) 16(11), 1066–1077 (2005)
11. Guim, F., Corbalán, J., Labarta, J.: Prediction f based models for evaluating backfilling scheduling policies. In: 8th IEEE Int'l Conf. on Parallel & Distributed Computing, Applications & Technologies (PDCAT), pp. 9–17 (December 2007)
12. Jones, J.P., Nitzberg, B.: Scheduling for parallel supercomputing: a historical perspective of achievable utilization. In: Feitelson, D.G., Rudolph, L. (eds.) JSSPP 1999, IPPS-WS 1999, and SPDP-WS 1999. LNCS, vol. 1659, pp. 1–16. Springer, Heidelberg (1999)
13. Lifka, D.: The ANL/IBM SP scheduling system. In: Feitelson, D.G., Rudolph, L. (eds.) IPPS-WS 1995 and JSSPP 1995. LNCS, vol. 949, pp. 295–303. Springer, Heidelberg (1995)

14. Mu'alem, A., Feitelson, D.G.: Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. IEEE Trans. on Parallel & Distributed Syst (TPDS) 12(6), 529–543 (2001)
15. Netto, M.A.S., Buyya, R.: Coordinated Rescheduling of Bag-of-Tasks for Executions on Multiple Resource Providers. Technical Report CLOUDS-TR-2010-1, U. of Melbourne, Australia, Submitted (TPDS) (February 2010)
16. Parallel Workloads Archive, `http://www.cs.huji.ac.il/labs/parallel/workload`
17. Sabin, G., Sadayappan, P.: On enhancing the reliability of job schedulers. In: High Availability & Performace Computing Workshop (HAPCW) (October 2005)
18. Srinivasan, S., Kettimuthu, R., Subrarnani, V., Sadayappan, P.: Characterization of backfilling strategies for parallel job scheduling. In: Int'l Conf. on Parallel Processing (ICPP), pp. 514–522 (August 2002)
19. Suzuoka, T., Subhlok, J., Gross, T.: Evaluating Job Scheduling Techniques for Highly Parallel Computers. Technical Report CMU-CS-95-149, School of Computer Science, Carnegie Mellon University (August 1995)
20. Tang, W., Desai, N., Buettner, D., Lan, Z.: Analyzing and adjusting user runtime estimates to improve job scheduling on the Blue Gene/P. In: IEEE Int'l Parallel & Distributed Processing Symp (IPDPS) (April 2010)
21. Tsafrir, D.: Modeling, Evaluating, and Improving the Performance of Supercomputer Scheduling. PhD thesis, The Hebrew University of Jerusalem (September 2006)
22. Tsafrir, D., Etsion, Y., Feitelson, D.G.: A model/utility for generating user runtime estimates and appending them to a standard workload format (SWF) file (February 2006), `http://www.cs.huji.ac.il/labs/parallel/workload/m_tsafrir05`
23. Tsafrir, D., Etsion, Y., Feitelson, D.G.: Modeling user runtime estimates. In: Feitelson, D.G., Frachtenberg, E., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2005. LNCS, vol. 3834, pp. 1–35. Springer, Heidelberg (2005)
24. Tsafrir, D., Feitelson, D.G.: The dynamics of backfilling: solving the mystery of why increased inaccuracy may help. In: 2nd IEEE Int'l Symp. on Workload Characterization (IISWC) (October 2006)
25. Zhang, Y., Franke, H., Moreira, J., Sivasubramaniam, A.: Improving parallel job scheduling by combining gang scheduling and backfilling techniques. In: 14th IEEE Int'l Parallel & Distributed Processing Symp. (IPDPS), pp. 133–142 (May 2000)
26. Zhang, Y., Franke, H., Moreira, J., Sivasubramaniam, A.: An integrated approach to parallel scheduling using gang-scheduling, backfilling, and migration. IEEE Trans. on Parallel & Distributed Syst. (TPDS) 14(3), 236–247 (2003)
27. Zotkin, D., Keleher, P.J.: Job-length estimation and performance in backfilling schedulers. In: 8th IEEE Int'l Symp. on High Performance Distributed Comput. (HPDC), p. 39 (August 1999)