Modeling User Runtime Estimates

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

Abstract

User estimates of job runtimes have emerged as an important component of the workload on parallel machines, and can have a significant impact on how a scheduler treats different jobs, and thus on overall performance. It is therefore highly desirable to have a good model of the relationship between parallel jobs and their associated estimates. We construct such a model based on a detailed analysis of several workload traces. The model incorporates those features that are consistent in all of the logs, most notably the inherently modal nature of estimates (e.g. only 20 different values are used as estimates for about 90% of the jobs). We find that the behavior of users, as manifested through the estimate distributions, is remarkably similar across the different workload traces. Indeed, providing our model with only the maximal allowed estimate value, along with the percentage of jobs that have used it, yields results that are very similar to the original. The remaining difference (if any) is largely eliminated by providing information on one or two additional popular estimates. Consequently, in comparison to previous models, simulations that utilize our model are better in reproducing scheduling behavior similar to that observed when using real estimates.

1 Introduction

EASY Backfilling [19, 21] is probably the most commonly used method for scheduling parallel jobs at the present time [7]. The idea is simple: Whenever the system status changes (a new job arrives or a running job terminates), the scheduler scans the queue of waiting jobs in order of arrival. Upon reaching the first queued job that can not be started immediately (not enough free processors), the scheduler makes a reservation on the job's behalf. This is the earliest time in which enough free processors would accumulate and allow the job to run. The scheduler then continues to scan the queue looking for smaller jobs (require less processors) that have been waiting less, but can be started immediately without interfering with the reservation. The action of selecting smaller jobs for execution before their time is called *backfilling*.

To use backfilling, the scheduler must know in advance the length of each job, that is, how long jobs will run[1]. This information is used when computing the reservation time (requires knowing when processors of currently running jobs will become available) and when determining if a waiting job is eligible for backfilling (must be short enough so as not to interfere with the reservation). As this information is not generally available, backfilling schedulers require their users to provide runtime estimates for submitted jobs. Obviously jobs that violate these estimates are killed. This is essential to insure that reservations are respected. Indeed, backfilling is largely based on the assumption that users would be motivated to provide accurate estimates, because jobs would have a better chance to backfill if the estimates are tight, but would be killed if the estimates are too short.

However, empirical investigations of this issue found that user runtime estimates are actually rather inaccurate [21]. Results from four different installations are shown in Fig. 1. These graphs are histograms of the estimation accuracy: what percentage of the requested time was actually used. The promising peak at 100% actually reflects jobs that reached their allocated time and were then killed by the system according to the backfilling rules. The hump near zero was conjectured to reflect jobs that failed on startup, based on the fact that all of them are very short (less than 90 seconds). The rest of the jobs, that actually ran successfully, have a rather flat uniform-like histogram.

The issue of user runtime estimates has since become the focus of intensive research. A number of studies have suggested that inaccurate runtime estimates are actually good, as they provide the scheduler with more flexibility and eventually lead to better performance; as a result, it was even proposed to simply double the user runtime estimates before using them [29, 21], or further, randomizing them [22]. In contrast, other studies contend that accurate runtime estimates are actually better, as they can lead to even better performance if used correctly, e.g. by scheduling in some SJF (shortest job first) based order [14, 23, 1, 25]. Still other studies have shown that the accuracy of user runtime estimates can have non-trivial effects on the results of performance evaluations [8].

[1] This is true for any backfilling scheduler, not just EASY.

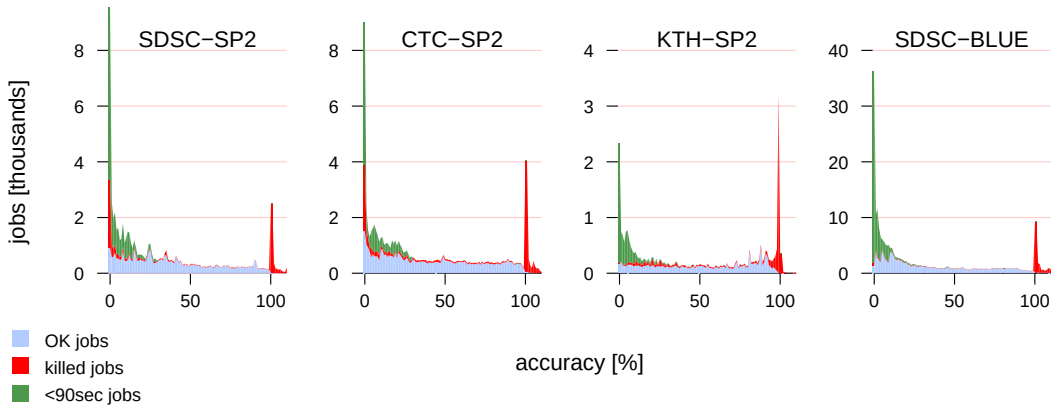

Figure 1: *Accuracy histogram of user runtime estimates:* $accuracy = 100 \times \frac{runtime}{estimate}$.

1.1 Motivation

All this activity spurred a search for ways to model user runtime estimates. Such a model is needed for three reasons. First, it is useful as part of a general workload model that can be used to study different job scheduling schemes, e.g. by means of simulation. Second, it is often the case that existing log files from production systems (used to drive simulations) are missing this information; a model can help in artificially manufacturing it. Third, a model may provide insights that will be useful in the study of whether and how the inaccuracy of estimates may be exploited by the scheduler.

We would like to make it clear that this paper targets the first two reasons mentioned above, that is, we aim to model and reflect reality, not to make it better. Indeed, in a different study, we show how backfilling schedulers can produce and utilize better runtime predictions that dramatically improve performance [25]. But even this novel technique often relies on user estimates under various conditions. Additionally, recall that user estimates have a role that is different than just serving as approximated runtimes, as they are also part of the user contract: the system guarantees a job will never be killed before its user estimate is reached. Consequently, system generated predictions (or other conceivable future mechanisms that are similar) can't "just" replace estimates.

At the same time, estimates ensure that jobs will indeed be killed at some point. Systems with no user estimates at all (that is, no runtime upper bound) are also undesirable, as these will allow jobs to run indefinitely, potentially overwhelming the system. At the very least we would expect users to choose some runtime upper-bound from a predefined set of values. However, this scenario is rather similar to reality, in which most users are already limiting themselves to very few canonical "round" estimates (as will be shown below), and jobs that exceed their estimates are immediately killed. It turns out there is actually no fundamental difference between allowing users to choose "any value", or from within a limited set.

Therefore, regardless of any possible scheduling improvements or changes, it seems a parallel workload model will not be complete if realistic user estimates are not included. Importantly, we will show that systems perform better if real user estimates are replaced with artificial ones, generated by existing models. This uncaptured "badness" quality of real user estimates constitutes a serious deficiency of existing models, as the purpose of these is to reflect reality, not to paint a brighter (false) picture. While counter intuitive, our goal in this paper is to produce estimates such that performance is worsened, not improved. Only when such a model is available, we can take the next step and consider ways to improve performance, based on a truly representative workload.

In the reminder of this section we survey the estimate models that have been proposed, and point out their shortcomings. This motivates the quest for a better model, which we propose in this paper.

1.2 Existing Models

The simplest possible model is to assume that user estimates are accurate. For example, such a model was used by Feitelson in [8]. This approach has two advantages: it is extremely simple, and it avoids the murky issue of how to model user estimates correctly. However, as witnessed by the data in Fig. 1, it is far from the truth.

A generalization of this model is to assume that a job's estimate is uniformly distributed within $[R, (f+1)R]$, where R is the job's runtime, and f is some non negative factor (f can't be negative because jobs are killed once their estimates are reached). If $f = 0$, this means that the estimates are identical to runtimes; if $f = 4$, they are distributed between R and $5R$, with an average of $3R$. Arguably, higher f values model increasingly inaccurate users. This model, which we call the "f-model",

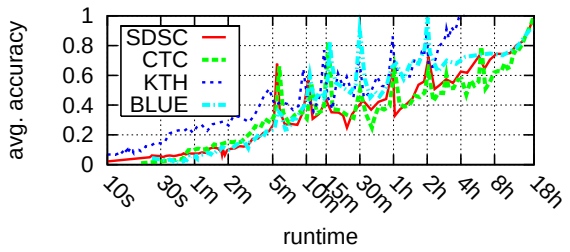

Figure 2: *Average accuracy as a function of binned-jobs' runtime, in four different production traces.*

was proposed by Mu'alem and Feitelson [11] and several variants of it were used to investigate the effects of inaccuracy [29, 21, 1]. It was also used by several researchers in simulations using workloads that did not contain estimates data [13, 8]. The main problem with this model is that the estimates it creates are overly correlated with the real runtimes, so it actually gives the scheduler considerable amount of valuable information that is unavailable when real user estimates are used. In particular, it enables the scheduler to effectively identify shorter jobs and select them for backfilling, leading to SJF-like behavior. For example, under this model, a one-hour job will always appear longer than a one-minute job (in reality, this is often not the case). This leads to better performance results than those observed when using real user estimates.

A third model, also proposed by Mu'alem and Feitelson, attempts to reproduce the histograms of Fig. 1. These flat histograms imply that $R/E = u$, i.e. that the ratio of the actual runtime R to the estimate E can be modeled as a uniformly distributed random variable ($u \in [0, 1]$). By changing sides we find that given a runtime R divided by u results in an artificial estimate E. While unrelated to the actual user estimate for this particular job, this is expected to lead to the same general statistics of all the estimates taken together. The model also created the peak at 100% and the hump at low values. Finally, if E came out outrageous (because u happened to be very small), it was truncated to 24 hours. This was called the "ϕ-model" by Zhang et al. [27] (ϕ denoted the fraction of jobs in the 100% peak), who used it in various simulations.

The problem with this model is that it is missing a "hidden" factor which is often overlooked: that all production installations have a limit on the maximal allowed runtime. For example, on the SDSC SP2 machine this limit is 18 hours. Naturally, the limit also applies to estimates, as it is meaningless to estimate that a job will run for say 37 hours if all jobs are limited to 18 hours.

Consider Fig. 2 which displays the average accuracy of jobs grouped to 100 equally sized bins according to their runtime. It has previously been conjectured that the apparent connection between longer runtimes and increased accuracy, is because the more a job progresses in its computation, the grater its chances become to reach successful completion [3]. However, this false hypothesis ignores the existence of a maximal allowed runtime, which suggests long jobs are guaranteed to have high accuracy. For example, if a job runs for 17 hours, its estimate must be in the range of 17 to 18 hours, so it's using at least 94.4% of its estimate. In other words, in contrast to the underlying assumption of the ϕ-model, the distribution of jobs in the accuracy histogram (Fig. 1) is not uniform. Rather, long jobs must be on the right, where accuracy is high, while short jobs tend to be on the left, at lower accuracies.

A fourth rather similar model was proposed by Cirne and Berman [3], which took the opposite direction in comparison to the previous model and chose to produce runtimes as multiples of estimates and accuracies, while generating direct models to the latter two. This decision was based on the argument that accuracies correlate with estimates less than they do with runtimes. In their model, accuracies were claimed to be well-modeled by a gamma distribution (this seems to be the result of trying to model the uniform part of the histogram along with the hump at low accuracies, by using one function for both). Estimates were successfully modeled by a log-uniform distribution. This methodology suffers from the same problem as the previous model, because accuracy is again independent of runtime. In addition, this model is not useful when attempting to add estimates to existing logs that lack them, or to workloads that are generated by other models which usually include runtimes and lack estimates [10, 6, 15, 20].

In addition to the per-model shortcomings mentioned above, there are two drawbacks from which all of them collectively suffer: The first is lack of repetitiveness: The work of users of parallel machines usually takes the form of bursts of very similar jobs, characterized as "sessions" [8, 28]. In the SDSC-SP2 log for example, the median value of the number of different estimates used by a user is only 3, which means most of the associated jobs look identical to the scheduler. It has been recently shown that such repetitiveness can have decisive effect on performance [26]. The second shortcoming is a direct result of the first: estimates form a modal distribution composed of very few values, a fact that is not reflected in any existing model. This is further discussed in the next section.

The conclusion from the above discussion is that all currently available models for generating user estimates are lacking in some respect. Consequently, using them in simulations leads to performance results that are generally unrealistically better than those obtained when real user estimates are used. Our goal in this paper is to capture the "badness" of real user estimates by finding a model that matches all known information about them: their distribution, their connection with each job's runtime, and their effect on scheduler performance.

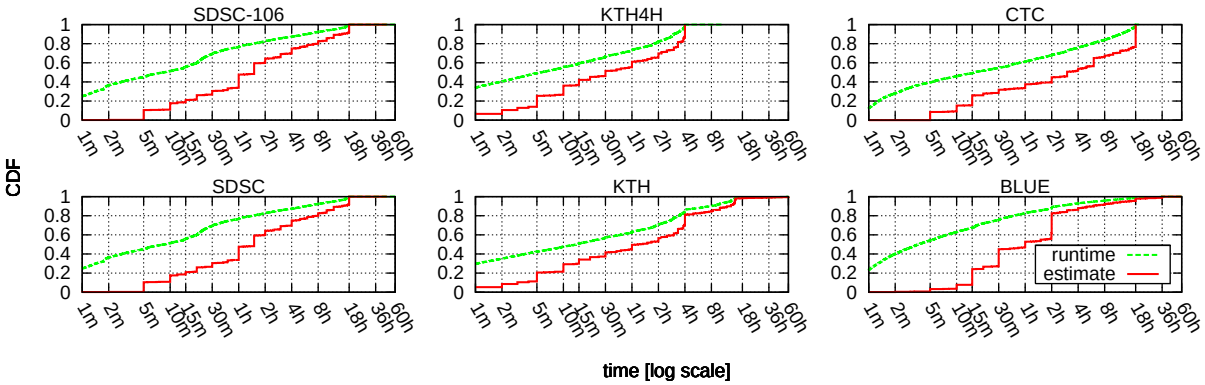

Figure 3: *Runtime and estimate CDFs (cumulative distribution functions) of various workload traces (Section 4 discusses the traces in detail). The fact that runtime-curves are much higher than estimate-curves means runtimes are indeed much shorter than estimates. For example, in CTC, 40% of the estimates are shorter than one hour (60% are longer), while for runtimes the situation is reversed (only 40% are longer than one hour).*

2 Modality

We require a model capable of generating realistic user estimates. The usual manner in which such problems are tackled is by fitting observed data to well known distributions, later to be used for producing artificial data. To some extent, this methodology is applicable when modeling estimates, which appear to be well captured using the log-uniform distribution [3] as shown in Fig. 3.

The difficulty lies in that user estimates embody another important characteristic: they are inherently modal [21, 2, 17], because users tend to repeatedly use the same "round" values (e.g. ten minutes, one hour, etc.). This is reflected in the staircase-like estimate curves of Fig. 3, in which each mode corresponds to a popular estimate value.

In particular, note the significant modes located at the maximal estimate of each trace, where the runtime and estimate curves finally meet[2]. Evidently, the maximal allowed estimate is always a popular value. For example, this value is used by a remarkable 24% of CTC jobs. This phenomenon probably reflects users' lack of knowledge or inability to predict how long their jobs will run, along with their tendency to "play it safe" in the face of strict system policy to kill underestimated jobs.

In the context of job scheduling, this observation is quite significant, as maximal-estimate jobs are the "worst kind" of jobs in the eyes of a scheduler (too long to be backfilled within all existing scheduling "holes"). In fact, if all jobs chose their estimates to be the maximal value, all backfilling activity would stop completely[3].

The observation about the maximal estimate mode may also be applied, to some extent, on other (shorter) modes: Consider the scenario in which an SJF scheduler must work with estimates that are highly inaccurate. If these estimates nevertheless result in a relatively correct ordering of waiting jobs, performance can be dramatically improved (up to an order of magnitude according to [1]). However, if estimates are modal, many jobs look the same in the eyes of the scheduler, which consequently fails to prioritize them correctly, and performance deteriorates. In general, if the estimate distribution is dominated by only a few large monolithic modes, performance is negatively effected, as less variance among jobs means less opportunities for the scheduler to perform backfilling.

Modality is absent from existing estimate models. An immediate heuristic that therefore comes to mind when trying to incorporate modality, is to "round" artificially generated estimates (e.g. by one of the models described above) to the nearest "canonical" value: values smaller than 1 hour are rounded to (say) the nearest multiple of 5 minutes, values smaller than 5 hours are rounded to the nearest hour, and so on. Experiments have shown that this heuristic fails in capturing the badness of user estimates, and performance results are similar to those obtained before this artificial modality was introduced. Additionally, arbitrary "rounding" fails to reproduce the various properties of the estimate distribution, as reported in the following sections.

The fact of the matter is that modes have a different (worse) nature than produced by the above. For example, when examining the number of jobs associated with the most popular estimates, we learn that these decay in an exponential manner e.g. half of the jobs use only 5 estimate values, 90% of the jobs use 20 estimates values etc. In contrast, the decay of less popular modes obeys a power law. In fact, almost every estimates-related aspect exhibit clear "model-able" (that can be modeled) characteristics.

[2]With the apparent exception of BLUE and KTH; this is further discussed in Section 4 which reveals that 2h and 4h effectively serve as BLUE and KTH's maximal estimate values, respectively.

[3]Except for when using the "extra" nodes, see [21] for details.

3 Methodology

The modal nature of estimates motivates the following methodology. When examining a trace, we view its estimate distribution as a series of K modes given by $\{(t_i, p_i)\}_{i=1}^{K}$. Each pair (t_i, p_i) represents one mode, such that t_i is the estimate-value in seconds (t for time), and p_i is the percentage of jobs that use t_i as their estimate (p for percent or popularity). For example, the CTC mode series includes the pair $(18h, 23.8\%)$ because 23.8% of the jobs have used 18 hours as their estimate. Occasionally, we refer to modes as *bins* within the estimate histogram. Note that $\sum_{i=1}^{K} p_i = 100\%$ (we are considering all the jobs in the trace). The remainder of this section serves as a roadmap of this paper, describing step-by-step how the $\{(t_i, p_i)\}_{i=1}^{K}$ mode-series is constructed.

3.1 Roadmap of This Paper

Each of the following paragraphs correspond to a section or two (sections are listed in order), and may contain some associated definitions to be used later on.

Trace Files We build our model carefully, one component at a time, in order to achieve the desired effect. Each step is based on analyzing user estimates in traces from various production machines, in an attempt to find invariants that are not unique to a single installation. The trace files we used and the manipulations we applied on them are discussed in Section 4.

Mass Disparity Our first step is showing that there exists a natural partition within the mode series that divides it into two: About 20 "head" estimate values are used throughout the entire trace by about 90% of the jobs that compose the trace. The rest of the estimate values are considered "tail" values. This is discussed in Section 5. Throughout the paper we will see that these two mode groups have distinctive characteristics. Naturally, the efforts we invest in modeling the two are proportional to the mass they entail.

Number of Estimates We start the modeling in Section 6 by finding out how many different estimates there are, that is, modeling the value of K. Note that this mostly effects the tail as we already know the head size ($\sim$20).

Time Ranks The next step is modeling the values themselves, that is, what exactly are the K time-values $\{t_i\}_{i=1}^{K}$. The indexing of this ascendingly sorted series is according to the values, with t_1 being the shortest and t_K being the maximal value allowed within the trace (also denoted T_{max}). The index i denotes the *time rank* of estimate t_i. This concept proved to be very helpful in our modeling efforts. We also define the *normalized time* of an estimate t_i to be t_i/T_{max} (a value between 0 and 1). Section 7 defines the function F_{tim} that gets i as input (time rank), and returns t_i (seconds).

Popularity Ranks Likewise, we need to model the mode sizes/popularities/percentages: $\{p_j\}_{j=1}^{K}$. This series is sorted in order of decreasing popularity, so p_1 is the percentage of jobs associated with the most popular estimate. The index j denotes the *popularity rank* of the mode to which p_j belongs. For example, the popularity rank of 18h within CTC is 1 (p_1 = 23.8%), as this is the most popular estimate. We also define the *normalized popularity rank* to be j/K (a value between 0 and 1). Section 8 defines the function F_{pop} that gets j as input (popularity rank), and returns p_j, the associated mode size.

Mapping Given the above two series, we need to generate a mapping between them, namely, to determine the popularity p_j of any given estimate t_i, which are paired to form a mode. Section 9 defines the function F_{map} that gets i as input (time rank) and returns j as output (popularity rank). Using the two functions defined above, we can now associate each t_i with the appropriate p_j. This yields a complete description of the estimates distribution. The model is then briefly surveyed in Section 10.

Validation Finally, the last part of this paper is validating that the resulting distribution resembles the reality. Additionally, we also verify through simulation that the "badness" of user estimates is successfully captured, by replacing the original estimates with those generated by our model. The replacement activity mandates developing a method according to which estimates are assigned to jobs (recall that an estimate of a job must be bigger than or equal to its runtime). This is done in Section 11. The paper is concluded in Section 12.

3.2 Input, Output, and Availability

As we go along, the number of *model parameters* accumulates to the neighborhood of two dozens. Most are optional and are supplied with reasonable default values. The only mandatory parameters are the number of jobs N (the number of estimates to produce), and the maximal allowed estimate value T_{max}. Another important parameter is the percentage of jobs associated with T_{max}, as this popular mode exhibits great variance and has decisive effect on performance. The *output of the model* is the series of the modes: how many jobs use which estimate.

The model we develop is somewhat sophisticated and involves a number of technical issues with subtle nature. As it is our purpose to allow simulations that are more realistic, the C++ source code of the model is made available for download from the parallel workload archive [9]. Its interface is composed of two function: The first gets a structure containing all the model parameters (all but two are assigned default values), and returns an array of K modes. The second function gets the mode array and another array composed of job structures (ID and runtime). It than associates each job with a suitable estimate.

Abbrev.	Site	Start	End	CPUs	Number of jobs (N)			M	U	X	K
					original	cleaned	sane	*months*	*users*	*max*	*estimates*
SDSC-106	San-Diego Supercomp. Ctr.	Apr 98	Apr 00	128	73,103	59,332	53,673	24	428	18h	339
CTC	Cornell Theory Center	Jun 96	May 97	512	79,302	77,222	77,222	11	679	18h	265
KTH4H	Swedish Royal Instit. Tech.	Sep 96	Aug 97	100	23,070	23,070	23,070	11	209	4h	106
BLUE	San-Diego Supercomp. Ctr.	Apr 00	Jun 03	1,152	250,440	243,314	223,407	32	468	36h	525
SDSC	San-Diego Supercomp. Ctr.	Apr 98	Apr 00	128	73,496	59,725	54,053	24	428	18h	543
KTH	Swedish Royal Instit. Tech.	Sep 96	Aug 97	100	28,490	28,490	28,490	11	214	60h	271

Table 1: *The trace files. The variables M, U, X, and K are months duration, number of users, maximal estimate value, and number of estimate bins, respectively. BLUE relates to San-Diego's Blue-Horizon machine. The others are SP2 machines.*

4 The Trace Files

The analysis and simulations reported in this paper are based on four accounting logs from large-scale parallel machines that are listed in Table 1. These are all the logs from the parallel workload archive [9] that contain information about user estimates and were available at the time we began this research (the DAS2 log, which also contains this data, was added since). Since traces span the past decade, were generated at different sites, by machines with different sizes, and reflect different load conditions, we have reason to believe consistent results obtained in this paper are truly representative.

Table 1 contains data about the original traces, their recommended "cleaned" version which is also available from the archive (excludes various non-representative anomalies [26]), and a "sane" version. The latter applies a filter on "cleaned" logs to remove jobs that cannot be used in simulations (unknown size, runtime, or submission time). As our goal is providing a model for the sake of performance analysis through simulation, our modeling activity targets only sane jobs. In particular, the K column in Table 1 is related to the sane versions, as is all the data presented in this paper.

During the study we found that two of the sane logs need to be further manipulated to be useful in this context. The first is the SDSC log: We say an estimate mode is "owned" by a user if this estimate was exclusively used by only that user within the log. It turns out that user 106 is uniquely creative in comparison to others, owning 204 estimates of the 543 found in SDSC (38%). This is highly irregular[4] as shown in Fig. 4 which displays the number of modes owned by each user (only owners are shown). We therefore remove this unique activity from the log for the remainder of the discussion (regular activity of user 106, using estimates that are also used by others, is allowed to remain). The resulting log is called *SDSC-106*. This version proved beneficial when modeling K and F_{tim}, the number of different estimate values found within a log

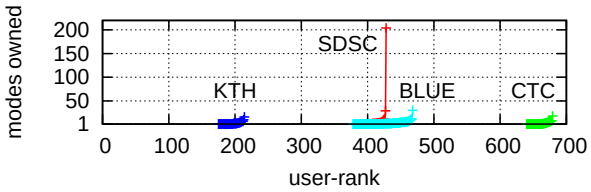

Figure 4: *Assume there are n users in a log. Each user is associated with the number of modes he owns m_i ($i = 1, 2, ..., n$), such that m_1 is the smallest and m_n is the largest. The i index is defined to be the user-rank and serves as the X-axis. The associated m_i is the Y-axis and only positive m_i-s are shown (that is, users with a zero m_i, that do not own any mode, are not shown). The SDSC outlier is associated with user 106 which is order of magnitude more "original" than other users, exclusively owning 38% of the SDSC modes.*

(Section 6), and the time values used (Section 7), respectively. Other aspects of the model where not affected.

The other problematic workload was KTH: This log is actually a combination of three different modes of activity: running jobs of up to 4 hours on weekdays, running jobs of up to 15 hours on weeknights, and running jobs of up to 60 hours on weekends. We have found that in the context of user estimates modeling, considering these three domains in an aggregated manner is similar to, say, aggregating CTC and BLUE to be a single log. We therefore focused on only one of them — the daytime workload with the 4-hour limit, which is the largest component of the log. This will be denoted by *KTH4H*.

Recall our claim that maximal estimate values are always popular (Fig. 3). We have argued that 4h and 2h are the effective maxima of KTH and BLUE, respectively. Obviously, this is the case for KTH (most of the time 4h is the maximum). As for BLUE, this machine had an "express" and "interactive" priority queues defined, with a limit of 2 hours on submitted jobs [9]. Indeed, the vast majority of 2-hours estimate jobs are from within these queues, which means here too users provided the maximal value available to them (while still allowing their jobs to be accepted to the higher priority queues).

[4] In fact, as this activity is concentrated within about 2 months of the log, it actually constitutes a workload flurry [26].

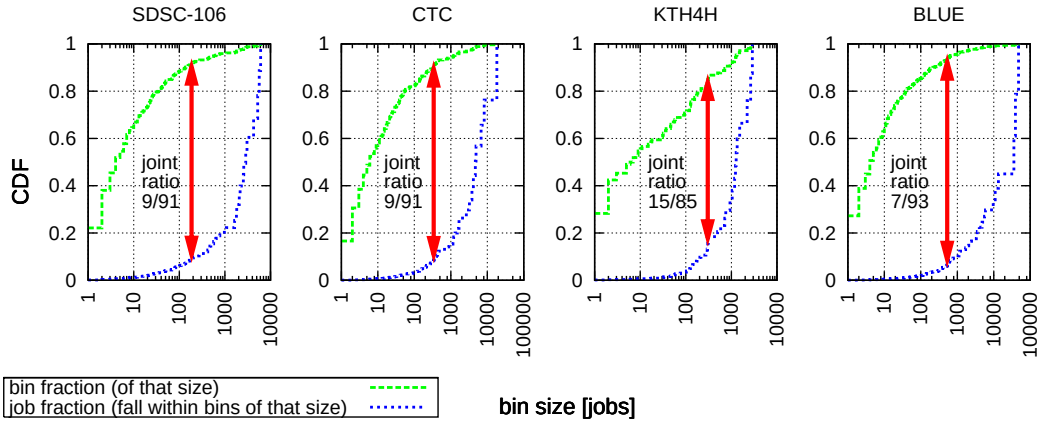

Figure 5: *Distributions of bins and of jobs, showing that a small fraction of the bins account for a large fraction of the jobs and vice versa. The actual fractions are indicated by the joint ratio, which is a generalization of the proverbial 10/90 rule.*

jobs	10%	50%	75%	90%	95%	98%	99%	100%
SDSC-106	1	6	12	22	39	77	116	339
CTC	1	4	10	22	36	62	89	265
KTH4H	1	6	12	21	28	36	43	106
BLUE	1	3	8	23	42	76	116	563
SDSC	1	6	12	23	43	91	156	543
KTH	1	8	21	41	60	89	122	270

Table 2: *Mass disparity: per-log minimal number of estimate bins needed to cover the specified percent of the jobs.*

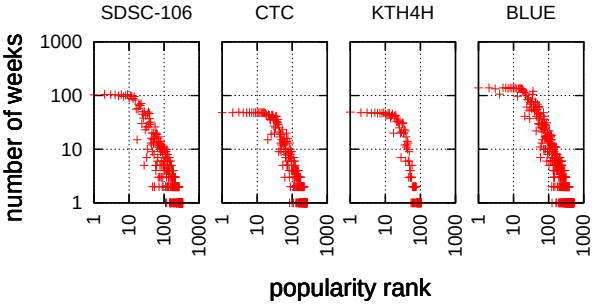

Figure 6: *Weeks in which an estimate appears, as a function of its popularity-rank (note that estimates are sorted from the most popular to the least). The top-20 appear throughout the logs.*

5 Mass Disparity of Estimates

Examining the histogram of estimates immediately reveals that the distribution is highly modal: A small number of values are used very many times, while many other values are only used a small number of times. In this section, we establish the mass disparity among estimate bins.

Human beings tend to estimate runtime with "round" or "canonical" numbers: 10 minutes, one hour etc. [21, 1, 17]. This has two consequences. One is that the number of bins in the histogram (K) is very small relative to the number of jobs in the trace (N). According to Table 1, N may be in the order of tens to hundreds of thousands, while K is invariably in the order of only a few hundreds.

The other consequence is that a small set of canonical bins dominates the set of values. Similar phenomena have been observed in many other types of workloads. They are called a "mass disparity", because the mass of the distribution is not spread out equally; rather, a small set of values gets a disproportionally large part of the mass [5].

The mass disparity of user runtime estimates is illustrated in Fig. 5. These are CDFs related to the bin size (the number of jobs composing a bin). In each graph, the top line is simply the distribution of bin sizes. This line grows sharply at the beginning, indicating that there are very many small bins (i.e. values that are used by only a small number of jobs). The other line is the distribution of jobs, showing the fraction of jobs with estimates that fall into bins of the different sizes. This line starts out flat and only grows sharply at the end, indicating that most jobs belong to large bins (i.e. most estimate values are the popular values that are repeatedly used very many times).

The figure also shows the joint ratio for each case. This is a generalization of the well-know 10/90 rule. For example, the joint ratio of 9/91 for the CTC log means that 9% of the bins account for 91% of the jobs, and vice versa: the other 91% of the bins contain only 9% of the jobs. Further details about the shape of the distributions are given in Table 2. This shows the absolute number of bins involved, rather than their fraction; for example, the CTC row shows that a mere 4 bins cover 50% of the jobs, 10 bins cover 75% of the jobs, and 22 bins contain 90%. Indeed, a bit more than 20 head bins are enough to account for 90% of the jobs in all four logs.

"Head" bins dramatically vary in size: While the most popular is used by $10-25\%$ of the jobs, only $\approx 1\%$ use the 20-th most popular. Regardless, all head bins, whether large or small, have a common temporal quality: their use is not confined to a limited period of time. Rather, they are uniformly used throughout the entire log. This is shown in

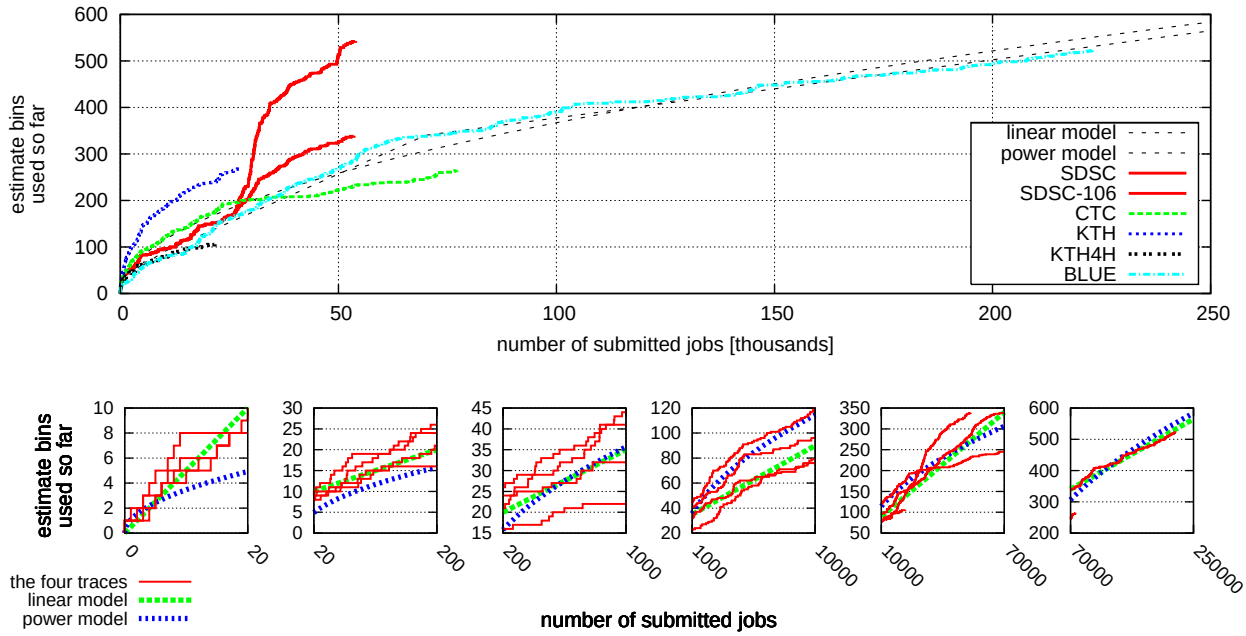

Figure 7: *Modeling K using a power model $K = \alpha N^{\beta}$ ($\alpha = 1.1$, $\beta = 0.5$) and a liner model which is defined by the points as specified in Table 3. Curves associated with SDSC share the same color, the higher being SDSC-106.*

Fig. 6 that plots the number of weeks in which estimates are used, as a function of their popularity ranks. The horizontal dot sequence associated with head bins indicates they are spread out evenly throughout the log. Further, the point of intersection between this sequence and the Y-axis is always the duration of the trace, e.g. for SDSC this is 2 years (a bit more than 100 weeks).

N (jobs)	0	20	200	1,000	10,0000	70,000	250,000
K (ests)	0	10	20	35	90	340	565
K/N (slope)		1/2	1/18	1/53	1/164	1/240	1/800

Table 3: *Points defining the linear model of K using N. The slope indicates the arrival rate of new estimates.*

6 Number of Estimates

We have established that about 20 popular "head" bins represent about 90% of the jobs' estimate distribution mass. We are left with the question of modeling the number of the other "tail" bins used by the remaining 10%.

Examining the four traces of choice in Table 1, we see that K tends to grow with the size of the trace, where this "size" can be measured in various ways: as the number of jobs executed (N), as the duration of time spanned (M), as the maximal estimate (X), or as the number of different active users (U). Note that the U metric also measures size, as new users continue to appear throughout each log. This is relevant because after all, users are the ones generating the estimates. In fact, in each of the four traces of choice, about 40% of the estimate modes are exclusively owned (as defined above) by various users[5].

We have experimented in modeling K as a function of the aspects mentioned above (individually or combined), and most attempts revealed some insightful observations. In fact, we are convinced K is the product of a combination of all factors, and that they all effect it to some degree. However, in the interest of being short while avoiding unwarranted complications (considering this only affects the tail of the distribution), we have chosen to model K as a function of N alone, which obtains tolerable results.

Fig. 7 plots K as a function of the number of jobs submitted so far (if n is an X value, its associated Y is the number of estimate bins in use, before the n-th job was submitted). Note how the vanilla version of KTH and SDSC stands out: the former due to the three estimate domains it contains, and the latter due to user 106. All curves can be rather successfully fitted with a power model on individual bases (we present one such power model that was simultaneously fitted against all four traces of choice). Accordingly, we allow the user of our model to supply the appropriate coefficients (as optional parameters). However, as this only effects tail bins, we set an ad-hoc linear model (defined by Table 3) as the default configuration. This provides a tolerable approximation of K for any given job number N.

[5] A surprising anecdote is that the actual number of bin-owners is also (exactly) 40, in three of the four traces.

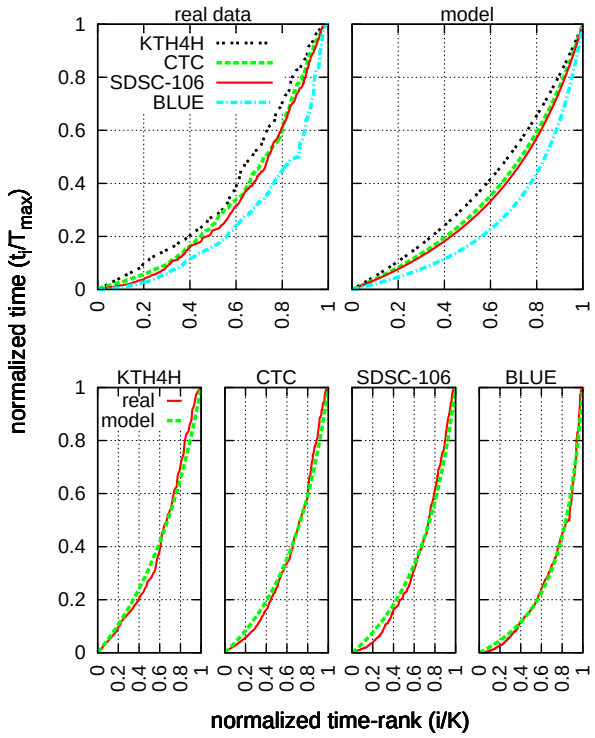

Figure 8: *Modeling estimate times using* $f(x) = \frac{(a-1)x}{a-x}$.

trace	KTH4H		CTC		SDSC-106		BLUE
a	1.91	$>$	1.57	$>$	1.50	$>$	1.24
K	106	$<$	265	$<$	339	$<$	525

Table 4: *The a parameter of the fractional fit presented in Fig. 8 is correlated with the number of different estimates (K).*

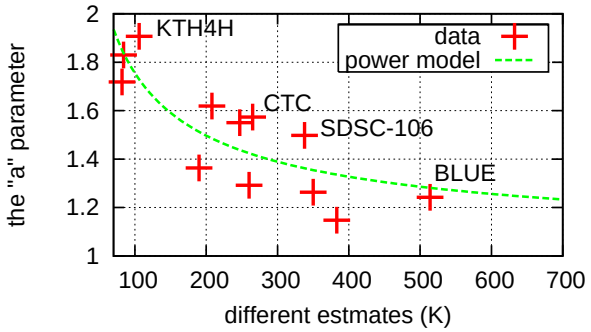

Figure 9: *Modeling a as a function of K using $1 + \alpha K^{\beta}$ (with $\alpha = 12.1$, $\beta = -0.6$). Note that a bigger K results in an a parameter that is closer (but never equal) to 1, as required.*

7 Time Values of Estimates

Having computed a K approximation (order of a few hundreds), we know how many estimate bins should be produced by our model. Let us continue to generate these K values, namely manufacture the $\{t_i\}_{i=1}^{K}$ series. It has already been noted that users tend to give "round" estimates [21, 2, 17], but this loose specification is not enough. In this section we develop a simple method to generate K such appropriate values. We are currently not considering the most popular (20) estimates in a separate manner. These will be addressed in detail later on (Section 9), complementing the model we develop in this section.

Recall that the time-ranks of estimates are their associated *indexes*, when ascendingly numbered from shortest to longest. Evidently, this concept can be very helpful for our purposes. We define a function F_{tim} that upon a time-rank input i, return the associated time value t_i (seconds), such that $F_{tim}(i) = t_i$.

The top-left of Fig. 8 plots normalized estimate time (t_i/T_{max}, where T_{max} is the maximal estimate) as a function of its associated normalized time-rank (i/K), for all four traces. According to the top-right and bottom of Fig. 8, it turns out the resulting curves can be modeled with great success when using the fractional function $f(x) = \frac{(a-1)x}{a-x}$ for some $a > 1$ (x is normalized time-rank). Further, the actual values of a (Table 4) are correlated with K, in that bigger K implies smaller a.

An obvious property of $f(x)$ in the relevant domain ($x \in [0, 1]$) is that when a gets closer to 1, its numerator goes to zero and therefore the curve gets closer to the bottom and right axes. On the other hand, as a gets further from 1 (goes to infinity), its numerator and denominator get more and more similar, which means the function converges to $f(x) = x$ (the main diagonal). The practical meaning of this is that less estimate values (smaller K, bigger a) means estimates' temporal spread is more uniform. In contrast, more estimate values (bigger K, smaller a) means a tendency of estimates to concentrate at the beginning of the Y-axis, namely, be shorter.

In order to reduce the number of user-supplied parameters of our model, we can try to approximate a as a function of K (which we already know how to reasonably deduce from the number of jobs). The problem is that we only have four samples (Table 4), too few to produce a fit. One heuristic to overcome this problem is splitting the traces in two and computing K and a for each half. This enlarges our sample space by eight (two additional samples per trace) to a total of twelve. The results of fitting this data to the best model we could find are shown in Fig. 9 and indicate a moderate success.

We can now define the required F_{tim} to be

$$F_{tim}(i) = T_{max} \cdot f\left(i/K\right) = T_{max} \cdot \frac{(a-1)\frac{i}{K}}{a - \frac{i}{K}}$$

Generating the $\{t_i\}_{i=1}^{K}$ series of time values is done by simply assigning $1, 2, ..., K$ to the time-rank i in an iterative manner. Finally, as almost 100% of the estimates are given in a minute resolution, the generated values are rounded to the nearest multiple of 60 (if not colliding with previously generated estimates).

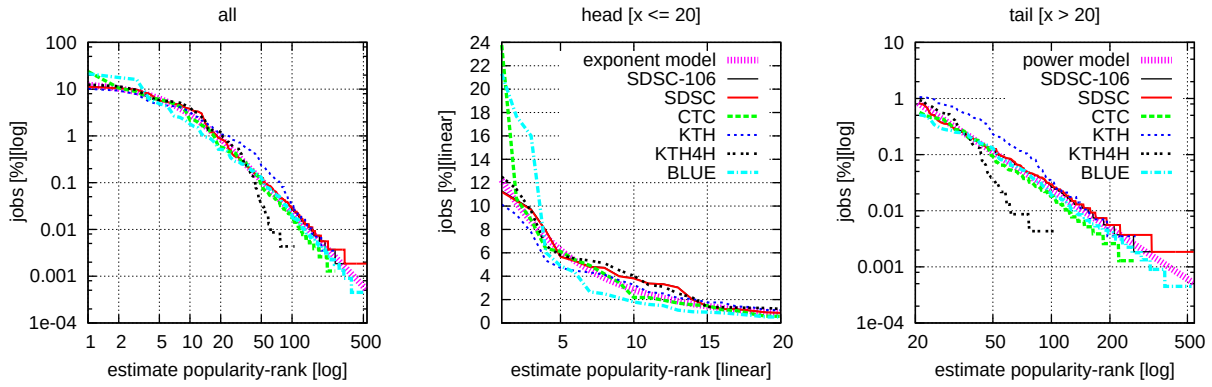

Figure 10: *Modeling percentage of jobs associated with estimate bins, as a function of popularity rank. The head (middle) is modeled by the exponential function* $\alpha e^{\beta x} + \gamma$ *(with* $\alpha = 14.05$*,* $\beta = -0.18$*, and* $\gamma = 0.46$*). The tail (right) is modeled by the* ωx^{ρ} *power law (with* $\omega = 795.6$ *and* $\rho = -2.27$*). Note that the middle figure has linear axes, while the other two are log scaled. The left figure concatenates the head and tail models.*

8 Popularity of Estimates

In the previous section we have modeled the time values of estimates. Here we raise the question of how popular is each estimate, that is, how many jobs are actually using each estimate value? Answering this question implies modeling the $\{p_i\}_{i=1}^{K}$ percentage series. Once again, like in the previous section, ranking the estimates (this time based on popularity) proves to be highly beneficial. Recall that $\{p_i\}_{i=1}^{K}$ is descendingly sorted such that p_1 is the percentage of jobs using the most popular estimate value, p_i is the percentage of jobs using the i-most popular estimate value, and i serves as the associated popularity rank. We seek a function F_{pop} such that $F_{pop}(i) = p_i$. Note that the constraint of $\sum_{i=1}^{K} F_{pop}(i) = 100$ must hold.

Fig. 10 plots the percentile size of each estimate bin, as a function of its popularity-rank. Again, there is a clear distinction between the top twenty most popular estimates (head of the distribution) and the others (tail), as sizes of head-bins decay exponentially, whereas the decay of the tail obeys some power law.

The suggested fits are indeed very successful ($R^2 > 0.95$ in both cases). However, when concentrating on the head (left or middle of Fig. 10), it is evident the exponential model is less successful for the first few estimates. For example, in CTC the most popular estimate is used by about 24% of the jobs, while in SDSC this is true for only 11%. In BLUE the situation is worse as the three top ranking estimates "break" the exponential curve. (Indeed, the exponential fit was produced after excluding these "abnormal" points.) Obviously, no model is perfect. But this seemingly minor deficiency (at the "head of the head") is actually quite significant, as a large part of the distribution mass lies within this part (differences in less popular estimates are far less important).

We note that the observed differences among the traces at the "head of the head" expose an inherent weakness in any estimate model one might suggest, because the effect of the variance among these 1-3 estimates is decisive. Consequently, our model will allow (though not mandate) the user to provide information regarding top-ranking estimates as model parameters (this will be further addressed in the next section). As for the default, recall that a job estimating to run for the maximal allowed value (T_{max}) is the worst kind of job in the eyes of a backfilling scheduler (Section 2). For this reason, we prefer the default model to follow the CTC example by making the (single) top ranking estimate "break" the exponential contiguity. This significant job percentage will later be associated with T_{max} to serve as a realistic worst case scenario. We therefore define F_{pop} as follows

$$F_{pop}(i) = \begin{cases} 89 - \sum_{j=2}^{20} \left(\alpha e^{\beta \cdot j} + \gamma\right) & i = 1 \\ \alpha e^{\beta \cdot i} + \gamma & i = 2, 3, ..., 20 \\ \omega \cdot i^{\rho} \cdot \frac{100-89}{\lambda} & i = 21, 22, ..., K \end{cases}$$

Starting with the (simplest) middle branch, F_{pop} is determined by the exponential model for all head popularity ranks but the first (the default values for the coefficients are specified in the caption of Fig. 10). The first branch is defined so as to preserve the invariant shown in Table 2 that the twenty top ranking estimates are enough to cover almost 90% of the jobs. Finally, the third branch determine sizes of tail estimates according to a power law (again, coefficient values are specified in Fig. 10). But to preserve the constraint that $\sum_{i=1}^{K} F_{pop}(i) = 100$, tail sizes are scaled by a factor of $\frac{100-89}{\lambda}$, where λ is the sum of the tail: $\sum_{i=21}^{K} \omega \cdot i^{\rho}$. The resulting default curve is almost identical to the one associated with the model as presented in Fig. 10, with a top rank of a bit more than 20% (to be associated with T_{max}).

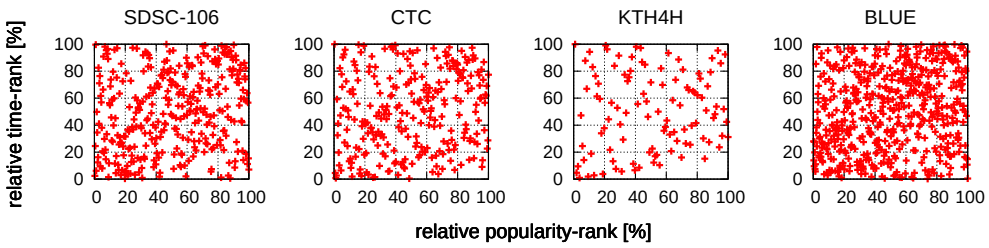

Figure 11: *Scatter plots of relative popularity-ranks vs. relative time-ranks appear to reveal a uniform distribution across all traces.*

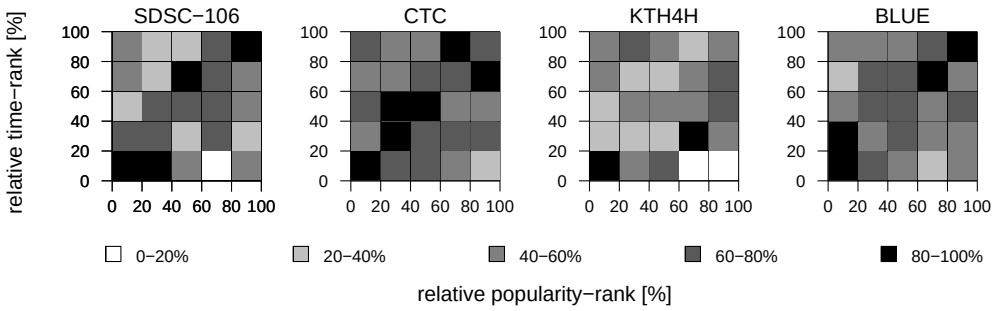

Figure 12: *Aggregating the data shown in Fig. 11 into a grid-based heat-map reveals no further insight, other than a consistent tendency of popular estimates to be short (bottom-left black cells).*

9 Mapping Time to Popularity

The next step after separately generating the estimates' time $\{t_i\}_{i=1}^{K}$ and popularity $\{p_j\}_{j=1}^{K}$ is figuring out how two construct a bipartite matching between the two. We seek a function F_{map} such that $F_{map}(i) = j$, that is, we want to map each time-rank to a popularity-rank in a manner that yields an estimate distributions similar to those found in the original traces (Fig. 3).

9.1 Mapping of Tail Estimates

As a first step towards constructing F_{map}, let us examine this mapping as it appears in the four traces. Fig. 11 scatter plots normalized popularity-ranks vs. normalized time-ranks: one point per estimate[6]. The points appear to be more or less uniformly distributed, which means there is no apparent mapping rule.

In an effort to expose some trend possibly hidden within the "disorder" of the scatter plots, we counted the number of points in each grid-cell within Fig. 11. We then generated an associated heat-map for each sub-figure by assigning a color based on the point-count of each cell: cells that are populated by 80-100% of the maximal (cell) point-count found within the sub-figure (denoted C), are assigned with black; cells populated by 0-20% of C are assigned with white; the remaining cells are assigned with a gray intensity that is linearly proportional to their point-count, batched in multiples of 20% of C.

[6] A scatter plot of actual values turns out to be meaningless.

The result, displayed in Fig. 12, appears to strengthen our initial hypothesis that the mapping between popularity-ranks and time-ranks is more or less uniformly random, as other than the bottom-left cell being consistently black (top twenty popular estimates show tendency of being shorter), there is no consistent pattern that emerges when comparing the different traces.

Our next step was therefore to randomly map between time and popularity ranks. Regrettably, this resulted in failure, as the generated CDFs were inherently different than those displayed in Fig. 3. The major guilty party of this failure were (unsurprisingly) the "big modes" that fell in the wrong places. The fact of the matter is that when (uniformly) randomly mapping between time and popularity ranks, there is a nonnegligible probability that the 4-5 most popular estimates are assigned to (say) times in the proximity of the maximal value, which means that the majority of the distribution mass is much too long. Alternatively, there is also a nonnegligible probability that the opposite will occur, namely, that none of the more popular estimates will be assigned to a time in the proximity of T_{max}, contrary to our previous findings.

We conclude that it is tail estimates (in terms of popularity) that are roughly randomly mapped to times in a uniform manner, forming the relatively balanced scatter plot observed in Fig. 11. This appearance is created due to the fact there are much more tail estimates (few hundreds) than head's (20). The head estimates minority, which nevertheless constitute 90% of the mass, distributes differently and requires a greater modeling effort.

9.2 Determining Head Times

We have reached the point where the effort to model user estimates is reduced to simply determining twenty actual time values and mapping them correctly to the appropriate (head) sizes. In other words, our task is as simple as producing twenty (t_i, p_i) pairs. These are good news, as the number of samples is so small, that a thorough examination of the entire sample-space becomes a feasible task. The bad news is that unlike previous parts of the model that are actually relatively trivial, and in spite of considerable efforts we have made, we have failed to produce a *simple* method of accomplishing the task. In the interest of practicality and space, we will not describe our various unsuccessful attempts to produce a simple straightforward solution. Instead, we will concentrate on describing the sophisticated algorithm we developed that has finally managed to deliver satisfactory results.

Let us examine the relevant sample space. Table 5 lists the twenty most popular estimates in each trace, and their associated (job) percentile sizes. It is immediately apparent that of the 36 values displayed, a remarkable 15 are *joint times* across all traces (note that we do not consider values higher than 4 hours within the KTH4H log, when determining which values are joint). The joint times are highlighted in bold font, and have values one would expect from humans to ordinarily use. Note that this is regardless of the different per-trace maximal estimate limits. We conclude that joint values should be hard-coded in our model, as it is fairly reasonable to conjecture humans will always extensively use values like 10 minutes, 1 hour, etc. We therefore define the first head-mapping step — determining the twenty time values that are the most popular — as follows:

1. Choose T_{max}, the maximal estimate (which is a mandatory parameter of our model). As previously mentioned, this is always a top ranking value.

2. Choose all hard-coded joint times that are smaller than T_{max}.

3. Choose in order (from largest to smallest) multiples of T_{round} that are smaller than T_{max}, where T_{round} is 200 hours, then 100 hours, then 50h, 10h, 5h, 2h, 1h, 20min, 10min, and 5 minutes. This process is stopped when the number of (different) chosen values reaches twenty.

The role of the third item above is to add a *relative* aspect to the process of choosing popular estimates, which is largely hard-coded (second item). As will later be shown, this manages to successfully capture KTH4H's condensed nature. At the other end, traces (machines) of larger estimate domains containing jobs that span hundreds of hours do in fact exist [2]. Regrettably, their owners refuse to share them with the community. Nevertheless, our algorithm generates longer times based on the modes they report (400h, 200h, 100h, and 50h in the NCSA O2K traces).

#	estimate *hh:mm*	SDSC-106	CTC	KTH4H	BLUE
1	00:01			6.6 (4)	
2	00:02			4.0 (10)	
3	00:03			2.2 (14)	
4	00:04			1.2 (20)	
5	**00:05**	11.3 (1)	8.8 (3)	11.5 (2)	2.7 (7)
6	**00:10**	7.9 (4)	6.4 (4)	9.6 (3)	4.3 (6)
7	00:12	1.2 (17)			
8	**00:15**	3.0 (13)	10.6 (2)	5.3 (7)	16.0 (3)
9	**00:20**	4.8 (7)	2.0 (12)	3.1 (12)	2.5 (8)
10	**00:30**	4.7 (8)	3.5 (9)	5.5 (6)	17.7 (2)
11	00:40			1.3 (19)	0.5 (19)
12	00:45	1.1 (18)			
13	00:50				0.5 (20)
14	**01:00**	10.5 (2)	4.2 (8)	5.8 (5)	4.9 (5)
15	01:30		0.8 (18)	1.3 (18)	1.5 (12)
16	01:40			1.4 (16)	
17	01:59				6.0 (4)
18	**02:00**	5.3 (6)	5.4 (6)	4.5 (9)	21.3 (1)
19	02:10			1.3 (17)	
20	02:30	1.2 (16)		1.4 (15)	
21	**03:00**	3.8 (10)	4.9 (7)	2.5 (13)	1.8 (10)
22	03:20			5.1 (8)	
23	03:50			3.3 (11)	
24	**04:00**	5.7 (5)	2.2 (11)	12.5 (1)	1.6 (11)
25	04:50		0.6 (20)		
26	**05:00**	1.4 (15)	1.1 (16)		0.9 (15)
27	**06:00**	2.0 (14)	6.1 (5)		1.0 (14)
28	07:00	0.9 (19)			
29	**08:00**	3.4 (11)	1.5 (14)		0.8 (17)
30	**10:00**	3.3 (12)	1.7 (13)		0.9 (16)
31	**12:00**	4.0 (9)	2.2 (10)		0.6 (18)
32	15:00	0.9 (20)	1.5 (15)		
33	16:00		1.0 (17)		
34	17:00		0.6 (19)		
35	**18:00**	9.8 (3)	23.8 (1)		2.1 (9)
36	36:00				1.1 (13)
sum (all)		86.4	88.9	89.3	88.7
sum (joint)		81.2	84.4	60.4	79.1

Table 5: *Top twenty popular bins in the four traces. Each column contains exactly twenty job percentage values. Note that fifteen of the top twenty estimates are joint across all traces (excluding KTH4H for estimates bigger than 4 hours). Joint estimates appear in bold font. The subscript parentheses denote the popularity-ranks within each trace. Notice the percentile sum of the top twenty which is invariantly in the neighborhood of 89%, the value used in Section 8 to define F_{pop}.*

Finally, recall we have already generated K time values using F_{tim} defined in Section 7. Head times generated here, replace the twenty values generated by F_{tim} that are the closest to them (and so the structure reported in Fig. 8 is preserved).

ttr	$F_{sdsc-106}$	F_{ctc}	F_{kth4h}	F_{blue}
0	**3**	**1**	**1**	**1**
1	**1**	**3**	**4**	6
2	**4**	**4**	10	5
3	17	2	14	**3**
4	13	12	20	7
5	7	9	2	2
6	8	8	**3**	18
7	18	18	7	19
8	**2**	6	12	**4**
9	6	7	6	11
10	16	11	19	20
11	10	20	5	9
12	5	16	18	10
13	15	5	16	14
14	14	14	9	13
15	19	13	17	16
16	11	10	15	15
17	12	15	13	17
18	9	17	8	8
19	20	19	11	12

Table 6: *The F_{log} functions of the four traces. The four most popular ranks in each trace are highlighted in bold font.*

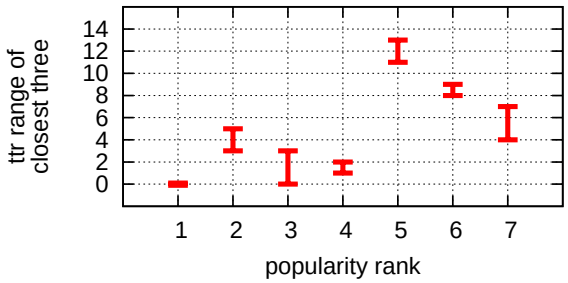

Figure 13: *There is only 0-3 difference between the closest three ttr-s that are associated with the more popular ranks (Table 6). For example, 3 of the ttr-s associated with popularity rank 2, are located in rows 3-5 in Table 6 (highlighted in a different color).*

9.3 Mapping of Head Estimates

Having both head times (seconds) and sizes (job percentages) we now go on to map between them. As usual, the mapping is made possible by using the associated ranks, rather than the actual values. For this purpose we need two new definitions:

First, we define a new type of time-rank, the *top-twenty time rank* (or *ttr* for short), which is rather similar to the ordinary time-rank: All top-twenty times, excluding T_{max}, are ascendingly sorted. The first is assigned a ttr=1, the second a ttr=2, and the last a ttr=19. For example, according to Table 5, in CTC, 00:05 has ttr=1, 00:10 has ttr=2, 01:30 has a ttr=7, and 17:00 has a ttr=19. T_{max} is always associated with ttr=0.

Second, for each trace-file log, we define a function F_{log} that maps ttr-s to the associated popularity ranks, within that log. For example, $F_{ctc}(0)$=1 as T_{max}=18h (associated with ttr=0) is its most popular estimate. Likewise, $F_{ctc}(1)$=3, as 5min is the smallest top-twenty estimate (ttr=1) and is the third most popular estimate within CTC. Table 6 lists F_{log} of the four traces. Recall that 2h is the effective T_{max} of BLUE and therefore this is the estimate we choose to associate with ttr=0. Additionally, note the BLUE 01:59 mode near its T_{max}=2h (Table 5). This is probably due to users trying to enjoy both worlds: use the maximal value, while "tricking" the system to assign their jobs a higher priority as a result of being shorter. We are not interested (nor able) to model such phenomena. Therefore, in the generation of Table 6 and throughout the reminder of this paper, we aggregate the 01:59 mode with that of 2h and consider them a single 27.3% mode.

The F_{log} functions in Table 6 reflect reality, and are in fact the reason for the log-uniform CDFs observed in Fig. 3. We therefore seek an algorithm that can "learn" these functions and be able to imitate them. Given such an artificial F_{log}, we would finally be able to match head-sizes (produced in Section 8, their size defines their popularity rank) to head-times (produced in Section 9.2, their value defines their ttr-s) and complete our model.

At first glance, the four F_{log} functions appear to have little similarities (the correlation coefficient between the columns of Table 6 is only 0.1-0.3), seemingly deeming failure on the generalization attempt. However a closer inspection reveals some regularities. Consider for example the more popular (and therefore more important to model) ranks: at least three of four values of each such rank are clustering across neighboring lines (ttr-s). This is made clearer in Fig. 13. Another observation is that when dividing popularity-ranks into two (1-10 vs. 11-20), around 75% of the more popular ranks are found in the top half of Table 6, which indicates a clear tendency of more popular ranks to be associated with smaller ttr-s. (This coincides with the log-uniformity of the original estimate distributions). It is our job to capture these regularities.

In the initialization part of our algorithm, which we call the *pool algorithm*, we associate ttr=0 (of T_{max}) with popularity rank=1, that is, the maximal estimate is also the most popular. The rationale of this decision is that

1. according to Table 6 this is usually the case in real traces,
2. as explained in Section 2, making T_{max} the most popular estimate constitutes a realistic worst case scenario, which is most appropriate to serve as the default setting, and
3. it is the "safest" decision due to the constraint that estimates must be longer than runtimes.

The last two items are the reason why we chose to follow the CTC example and enforce a sizable first rank on the

construction of F_{pop} (end of Section 8) that "breaks" the exponential contiguity observed in Fig. 10. To complete the initialization part, we allocate an empty vector V_{pool} designated to hold popularity ranks. Any popularity rank may have up to four occurrences within V_{pool}.

The body of the pool-algorithm iterates through the rest of the ttr-s in ascending order ($J_{ttr} = 1, 2, ..., 19$) and performs the following steps on each iteration:

1. For each trace file log, insert the popularity rank $F_{log}(J_{ttr})$ to V_{pool}, but only if this rank wasn't already mapped to some smaller ttr in previous iterations. (In other words, insert all the values from within the J_{ttr} line in Table 6, that weren't already chosen.)
2. If there exists popularity ranks that have four occurrences within V_{pool}, choose the smallest of these ranks R, map J_{ttr} to R, remove all occurrences of R from V_{pool}, and move on to the next iteration.
3. Otherwise, randomly choose two (not necessarily different) popularity ranks from within V_{pool}, map the smaller of these to J_{ttr}, and remove all its occurrences from V_{pool}.

A main principle of the algorithm is the gradual iteration over Table 6, such that no popularity-rank R is eligible for mapping to J_{ttr}, before we have actually witnessed at least one occasion in which R was mapped to a ttr that is smaller than or equal to J_{ttr}. This aims to imitate the original F_{log} functions, along with serving as the first safety-mechanism obstructing more popular ranks to be mapped to longer estimates (recall that estimate CDFs are log-uniform, which means most estimates are short).

Another important principle of the algorithm is that increased number of occurrences of the same R within V_{pool}, implies a greater chance of R to be randomly chosen. And so, an R that is mapped to a ttr $\leq J_{ttr}$ within two traces (two occurrences within V_{pool}), has double the chance of being chosen in comparison to a popularity rank for which this condition holds with respect to only one trace (one occurrence within V_{pool}). This aspect of the algorithm also aims to capture the commonality between the various traces.

Item number two in the algorithm tries to make sure an R will not be mapped to a ttr that is bigger than *all* the ttr-s to which it was mapped in the four traces. Like the first principle mentioned above, this item has the role of making sure the resulting mapping isn't too different than that of the original logs. It also serves as the second safety-mechanism limiting the probability of more popular ranks to be mapped to longer estimates.

The combination of the above "safety mechanisms" was usually enough to produce satisfactory results. However, on rare occasions, too many high popularity ranks have managed to nevertheless "escape" these mechanisms and be mapped to longer estimates. Adding a third safety-mechanism, in the form of using the minimum between two choices of popularity ranks (third item of the algorithm), has turned this probability negligible.

9.4 Embedding User-Supplied Estimates

While the estimate distributions of the traces bare remarkable resemblance, they are also very distinct within the "head of the head" (as discussed in Section 8), that is, the 1-3 most popular estimates. For example, considering Table 5, the difference between the percentage of SDSC and CTC jobs associated with 18h (10% vs. 24%) is enough to yield completely different distributions. Another example is BLUE's shift of the maximum from 36h to 2h, or its two huge modes in 15min and 30min; the fact that more than 60% of its jobs use one of these estimates (along with 01:59), cannot be captured by any general model. Yet another example is KTH4H's unique modes below 5min. This variance among the most important estimate bins, along with the fact users may be aware of special queues and other influential technicalities concerning their site, mandates a general model to allow its user to manually supply head estimates as parameters.

To this end, we allow the user to supply the model with a vector of up to twenty (t_i, p_i) pairs. The manner in which these pairs are embedded within our model is the following: The t_i values replace default-generated head times (Section 9.2) that are the closest to them, with the exception of T_{max} which is never replaced unless explicitly given by the user as one of the (t_i, p_i) pairs. (This is due to the reasons discussed in Section 9.3.) As an example, in order to effectively replace the maximal value of BLUE, the user must supply two pairs: $(36h, 1\%)$ to prevent the model from making the old maximum (36h) the most popular estimate, and $(2h, 27\%)$ to generate the new maximum.

Similarly to times, user supplied p_i percentile sizes replace default-generated sizes (Section 8) that are the closest to them. Once again, the biggest value (reserved for T_{max}) is not replaced if the user did not supply a pair containing T_{max}. Additionally, the remaining non-user head-sizes are scaled such that the total mass of the head is still 89% (scaling however do applies to the largest non-user size). If scaling is not possible (sum of user sizes exceed 89%), non-user head-sizes are simply eliminated, and the tail sizes are scaled such that the sum of the entire distribution is 100%.

Finally, the pool algorithm is refined to skip ttr-s that are associated with user-supplied estimates and to avoid choosing their associated popularity ranks for mapping.

10 Overview of the Model

Now that all the different pieces are in place, let us briefly review the default operation of the estimates model we have developed:

1. Get input. The mandatory parameters are maximal estimate value T_{max}, and number of jobs N (which is the number of estimates the model must produce as output). A third, "semi mandatory", parameter is the percentage of jobs associated with T_{max}. While the model can arbitrarily decide this value by itself, its variation in reality is too big to be captured by a model, whereas its influence on performance results is too detrimental to be ignored (T_{max} jobs are the "worst kind" of jobs in the eyes of the scheduler; Section 2).
2. Compute the value of K (different estimate times) as defined in Section 6.
3. Generate K time-values using F_{tim} as defined in Section 7.
4. Generate 20 "head" time-values using the algorithm defined in Section 9.2 and combine them with the K time-values produced in the previous item. Non-head times are denoted "tail" times.
5. Generate K percentile sizes using F_{pop} as defined in Section 8. The largest 20 sizes are the head sizes. The rest are tail.
6. Map between time- and size-values using F_{map} as defined in Section 9, by
   - Randomly mapping between tail-times and tail-sizes in a uniform manner (Section 9.1).
   - Mapping head-times and head-sizes using the pool algorithm (Section 9.3).
7. If received user supplied estimate bins, embed them within the model as described in Section 9.4.

10.1 About the Complexity

The only part which is non-trivial in our model is the pool algorithm: Generating the estimate time values by themselves is a trivial operation. Generating sizes (percentages of jobs) is equally trivial. Mapping between these two value sets is also a relatively easy operation, as all but the 20 most popular sizes can be randomly mapped. All the complexity of the model concentrates in solving the problem of deciding how many jobs are associated with each "head" estimate, or in other words, where exactly to place the larger modes. The question of whether a simpler alternative than the one suggested here exists, is an open one, and it is conceivable there's a positive answer. However, all the "immediate" heuristics we could think of in order to perform this task in a simpler manner have been checked and verified to be inadequate. In fact, it is these inadequacies that has lead us step by step in the development of the pool algorithm.

11 Validating the Model

Having implemented the estimate model, we now go on to validate its effectiveness. This is essentially composed of two parts. The first is obviously making sure that the resulting distribution is similar to that of the traces (Section 11.1). However, this is not enough by itself, as our ultimate goal is to allow realistic performance evaluation. The second part is therefore checking whether performance results obtained by using the original data are comparable to those produced when replacing original estimates with artificial values produced by the model (Section 11.3). The latter part mandates developing a method according to which artificial estimates are assigned to jobs (Section 11.2).

11.1 Validating the Distribution

Fig. 14 plots the original CDFs (solid line) against those generated by the "vanilla" model using various seeds. The only input parameters that are given to the model are those listed in Section 10, that is, the maximal estimate T_{max}, then number of jobs N, and the percentage of jobs associated with T_{max}. Recall that BLUE's maximum is considered to be 2 hours and that in order to reflect this we must explicitly supply the model with an additional pair (Section 9.4).

The results indicate the model has notable success in generating distributions that are remarkably similar to that of SDSC-106 and CTC; it is far less successful with respect to the other two traces. However, this should come as no surprise because, as mentioned earlier, the model has no pretense of reflecting abnormalities or features that are unique to individual traces. In the case of KTH4H, these are the large modes that are found below 5 minutes (Table 5). In fact, if aggregating these modes with that of 5 minutes, we get that a remarkable 25.5% of KTH4H's jobs have estimates that are 5 minutes or less, which is inherently different in comparison to the other traces. In the case of BLUE, its uniqueness takes the form of two exceptional modes located at 15 and 30 minutes. This distinctive quality is especially apparent in Fig. 10, where the three biggest modes "break" the log-uniform contiguity.

The practical question is therefore if the model can produce good results when provided with *minimal* additional information highlighting the trace-specific abnormalities. The amount of such information is inherently limited if we are to keep the model applicable and maintain its practi-

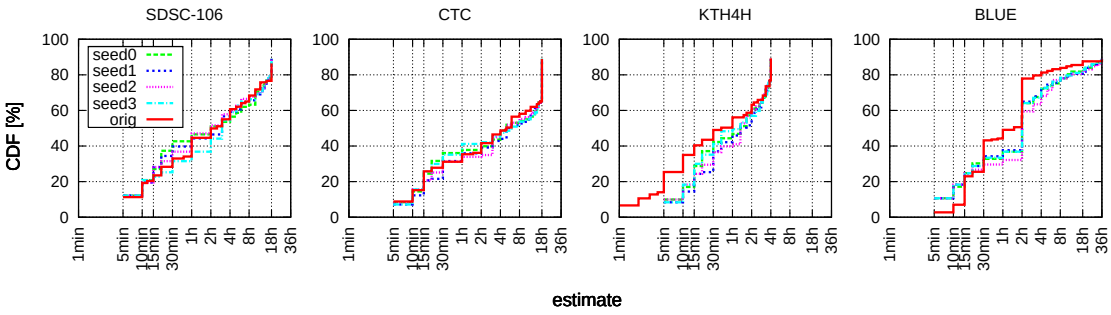

Figure 14: *The original estimate distribution of the traces (solid lines) vs. the output of the vanilla model, when used with four different seeds. Output is less successful for traces with unique features.*

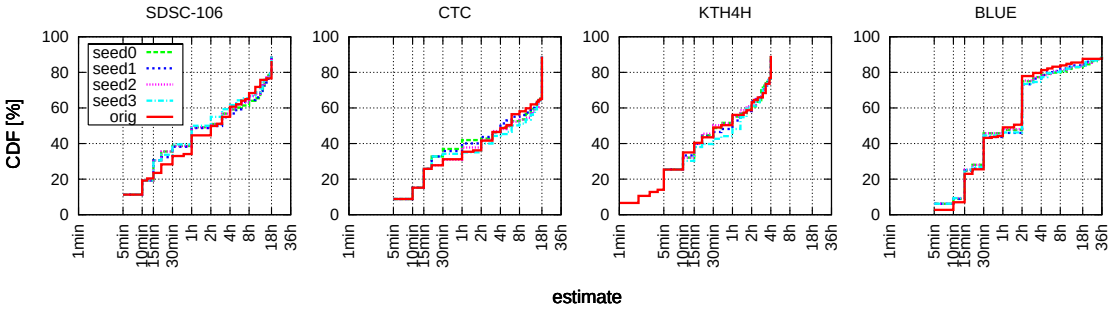

Figure 15: *Output of the model under the "improved" setting which provides minimal information identifying the unique features.*

cal value. We therefore define the "improved" setting in which the KTH4H model is provided with the additional $(5min, 25\%)$ pair. The BLUE model is provided with two additional pairs associated with its two exceptional modes: $(15min, 16\%)$ and $(30min, 18\%)$.

The results of the improved setting are shown in Fig. 15 and indicate that this additional information was all that the model needed in order to produce satisfactory results (also) with respect to the two "unique" traces. To test the impact of additional information on situations where the vanilla model manages to produce reasonable results by itself, the improved setting supplied three additional pairs (of the most popular estimates) when modeling CTC and SDSC-106. It is not apparent whether the additional information made a qualitative difference.

The important conclusion that follows from the successful experiment we have conducted in this section, is that estimate distributions are indeed extremely similar: Most of their variance concentrates within the 1-3 most popular estimates, and once these are provided, the model produces very good results.

11.2 Assigning Estimates to Jobs

The next step in validating the model is putting it to use within a simulation. For this purpose we have decided to simulate the EASY scheduler and evaluate its performance under the four workloads. This can be done with original estimates or after replacing them with artificial values that were generated by our model. Similar performance results would indicate success.

The common practice when modeling a parallel workload is to define canonical random variables to represent the different attributes of the jobs, e.g. runtime, size, interarrival time etc. [6, 15, 20]. Generating a workload of N jobs is then performed by creating N samples of these random variables. Importantly, each sample is generated *independently* of other samples.

In this respect, assignment of artificial estimates to jobs is subtle, as this must be done under the constraint that estimates mustn't be smaller than the runtimes of the jobs to which they are assigned. Here, we can't just simply randomly choose a value. However, if independence between jobs is still assumed, we can easily overcome the problem by using the *random shuffle algorithm*. This algorithm gets two vectors $V_{estimate}$ and $V_{runtime}$ that hold N values as suggested by their names. The content of both vectors is generated as usual, according to the procedure described above (under the assumption of independence). Now all that is needed is a random permutation that maps between the two, such that every estimate is equal to or bigger than its associated runtime. The random shuffle algorithm finds such a permutation by iterating through $V_{runtime}$ and randomly pairing each runtime R with some

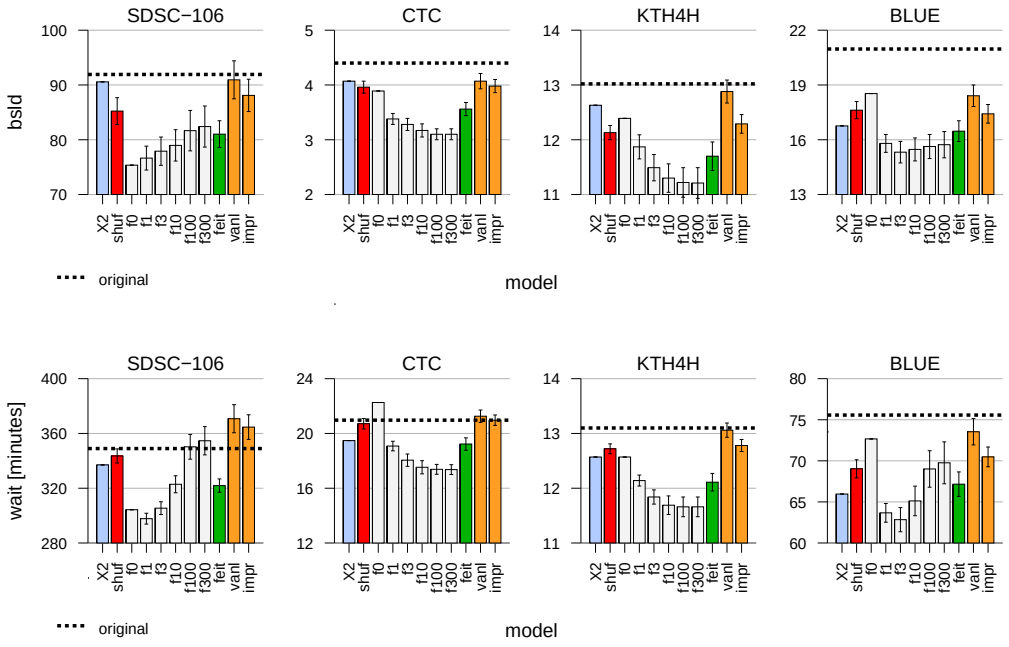

Figure 16: *Validating badness. The reason for the peculiar result associated with the average SDSC wait time remains unknown.*

estimate $E \in V_{estimate}$ for which $E \geq R$. After values are paired, they are removed from their respective vectors.

Note that we do not claim that the independence assumption underlying the random shuffle algorithm is correct. On the contrary. We only argue that this is the common practice. However, there is a way to transform the original data such that this assumption holds: The algorithm can be applied to the original data, that is, we can populate the $V_{estimate}$ vector with original trace estimates and reassign them to jobs using the shuffle algorithm. The outcome of doing this would be that the original estimates are "randomly shuffled" between jobs (which is the source of the algorithm's name). The result of such shuffling is to create independent "real" estimates. This is suitable as a basis for comparison with our model, as explained below.

11.3 Validating Performance Results

Several estimate-generation models have been evaluated and compared against the original data:

- The *X2*-model: simply doubles user estimates on the fly [16, 21].
- The *shfl*-model: the result of applying the random shuffle algorithm (defined above) to the original data. As noted, assuming independence in this context is correct.
- The f-model: upon receiving a job's runtime R, uniformly chooses an estimate from the closed range $[R, R \cdot (f + 1)]$. In accordance with [21], six values of f were chosen: 0 (complete accuracy), 1, 3, 10, 100, and 300.
- The *feit*-model: targets accuracy (suggested by Mu'alem and Feitelson [21] and explained in the introduction).
- The *vanl*-model: the vanilla setting of the model developed in this paper (defined above).
- The *impr*-model: the improved setting of our model, supplying it with some additional information (defined above).

Note that *X2* and *shfl* are not models per-se, as both are based on real estimates. The competitors of our model are f and *feit* (which produce estimates based on runtime).

Performance results are shown in Fig. 16 in the form of average wait time and bounded slowdown. The black dotted lines present the results of running the simulations using the original data. Therefore, models that are closer to this line are more realistic. Recall that our aim here is not to improve performance. Rather, it is to produce trustworthy results that are closest to reality. All the results associated with models that contain a random component (all but *X2* and *f0*) are the average of one hundred different simulation runs employing different seeds. The errorbars associated with these models display the absolute-deviation (average of absolute value of deviation from the average).

When examining Fig. 16, it is clear the two variants of our algorithm are more realistic, in that they usually do a better job in capturing the "badness" of user estimates (compare with f-s and *feit*). Another observation is that using increased f-s (or *feit*) to model increased user inaccuracy (for the sake of realism) is erroneous, as *f0* usually produces results that are much closer to the truth. In fact, *f0* is usually comparable to the results obtained by our model with the exception of the SDSC trace. However, this is limited to the FCFS-based EASY scenario: if introducing a certain amount of limited SJF-ness to the scheduler (e.g. as in [25, 1]), *f0* yields considerably better performance results in comparison to the original, whereas our model stays relatively the same (figure not shown to conserve space). Another scenario in which *f0* can't be used is when evaluating system-generated runtime predictors that make use of estimates (along with other job characteristics) [14, 23, 18, 25]. Finally (returning to the context of EASY), unlike *f0*, our model has room for improvement as will shortly be discussed, and we believe it has potential to "go the extra mile".

A key point in understanding the performance results is noticing that the vanilla setting of our algorithm is surprisingly more successful in being closer to the original than its improved counterpart. This is troublesome as our entire case is built on the argument that models that are more accurate would yield results that are closer to the truth. The answer to the riddle is revealed when examining the *shfl* model. The fact of the matter is that one cannot get more accurate than *shfl*, as it "generates" a distribution that is *identical* to that of the original. Yet it too seems to be inferior to our vanilla model. This exposes our independence assumption (the random shuffle algorithm) as the true guilty party which is responsible for the difference between *impr* and the original. The correct comparison between *impr* and *vanl* should actually be based on which is closer to *shfl*, not to the original, as only with *shfl* can independence be assumed. Based on this criterion, *impr* is consistently better than *vanl*.

Once this is understood, we can also explain why the performance of *impr* (in terms of wait and slowdown) is always better than that of *vanl*. Consider the difference between the two models: *impr* simply has much more accurate data regarding *shorter* jobs (e.g. KTH4H's 25% of 5 minutes jobs). As short jobs benefit the most from the backfilling optimization, *impr* consistently outperforms *vanl* (in absolute terms).

11.4 Repetitiveness is Missing

We are not interested in artificially producing worse results by means of falsely boosting up estimates (as is done by *vanl* with respect to *impr*). This would be equivalent to, say, increasing the fraction of jobs that estimate to run

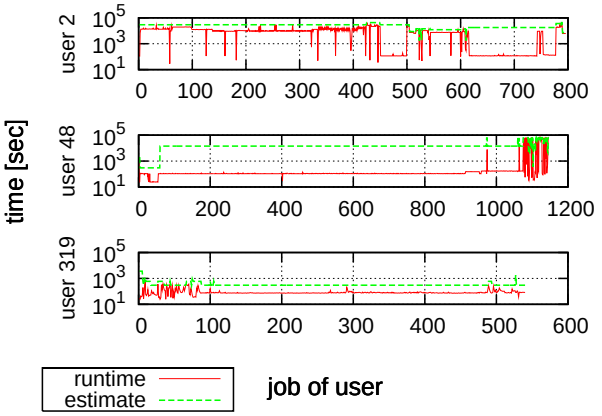

Figure 17: *Runtime and estimate of all the jobs submitted by three arbitrary users from the SDSC trace shows remarkable repetitiveness.*

T_{max}, which can arbitrarily worsen results. Our true goal is creating a reliable model. The above indicates that the problem lies in the assumption of independence, namely, the manner we assign estimates to jobs. While it is possible that this is partially because we neglected to enforce the accuracy to be as displayed in Fig. 1 (the accuracy histograms of even *shfl* are dissimilar to that of the original), we conjecture that the independence assumption is more acute.

It has been known for over a decade that the work generated by users is highly repetitive [12, 10]. Recent work [28, 24] suggests that the correct way to model a workload is by viewing it as a sequence of *user sessions*, that is, bursts of very similar jobs by the same user. This doctrine suggests that a correct model cannot just draw values from a given distribution while disregarding previous values as is done by most existing parallel workload models (e.g. [6, 15, 20, 4]). The rationale of this claim is that the repetitive nature of the sequence within the session may have a decisive effect on performance results[7].

Since users tend to submit bursts of jobs having the same estimate value (Fig. 17), the end result is somewhat similar to that of the existence of estimates modes, but in a more "temporal sense": At any time instance, jobs within the wait-queue tend to look the same to the scheduler, as jobs belonging to the same session usually share the same estimate value. Consequently, the scheduler has less flexibility in making backfilling decision and the performance is negatively effected. Our *shfl* algorithm, along with all the rest of the models, do not entail the concept

[7] A remarkable example stressing the importance of this phenomenon was recently published [26]: changing a runtime of only *one* job (within a log that spans two years) by a mere 30 seconds, resulted in a change of 8% in the average bounded slowdown of *all* the jobs; the reason was traced to be a certain user-session and its interaction with the scheduler.

of sessions and therefore result in superior performance in comparison to the original.

Accordingly, our future work includes developing an assignment mechanism that is session aware. This can be obtained if the procedure that pairs runtimes and estimates gets additional information associating jobs with users. User-based modeling [24] can supply this data.

12 Conclusions and Future Work

User runtime estimates significantly effect the performance of parallel systems [21, 1, 8]. As part of the effort to allow realistic and trustworthy performance analysis of such systems, there is a need for an estimates model that successfully captures their main characteristics.

A number of models have been suggested, but these are all lacking in some respect. Their shortcoming include implicitly revealing too much information about real runtimes, erroneously emulating the accuracy ratio of runtime to estimate, neglecting to take into consideration the fact that all production installations have a limit on the maximal allowed estimate, and that this value is always one of the more popular estimates. Importantly, two key ingredients are missing from existing models: the inherently modal nature of the estimates caused by users' tendency to supply "round" values [21, 2, 17], and the temporal repetitive nature of user estimates, assigning the same value to bursts of jobs (sessions) [26, 28]. These have decisive effect on performance results, as low estimate-variance of wait-queue jobs reduces the effectiveness of backfilling. The outcome is simulation results that are unrealistically better than those obtained with real estimates.

In this paper we produce a model that targets estimates modality. We view the estimates distribution as a sequence of modes, and investigate their main characteristics. Our findings include the invariant that 20 "head" estimates are used by about 90% of the jobs throughout the entire log. The popularity of head estimates (percentage of jobs using them) decreases exponentially, whereas the tail obeys a power-law. The few hundred values that are used as estimates, are well-fitted by a fractional model, while at the same time, 15 out of the 20 head estimates are identical across all the production logs we have examined. The major difficulty faced by this paper was determining how popular is each head estimate (how many jobs are associated with each). This was solved by the "pool algorithm", aimed to capture similarities between profiles of head-estimates within the analyzed production logs.

We found that all modeled aspects of the estimates distribution are almost identical across the logs, and therefore our model defines only two mandatory parameters: the number of jobs and the maximal allowed estimate (T_{max}). While considerable variance does in fact exist, it is mostly encapsulated within the percentage of jobs estimated to run for T_{max}. The remaining variance is attributed to another 1-2 very popular modes that sometimes exist, but are unique to individual logs. When provided this additional information, our model produces distributions that are remarkably similar to that of the original.

When put to use in simulation (by replacing real estimates with artificial ones), our model consistently yields performance results that are closer to the original than those obtained by other models. In fact, these results are almost identical to when real estimates are used and are randomly shuffled between jobs. This suggests that the temporal repetitiveness of per-user estimates may be the final obstacle separating us from achieving realistic results. Consequently, our future work includes developing an improved assignment scheme of estimates to jobs that will preserve this feature.

Our estimates model is available to download from the parallel workload archive [9]. Its interface contains two functions: generating the distribution modes, and assigning estimates to jobs. The latter is essentially random shuffling of estimates between jobs, under the constraint that runtimes are smaller than estimates. Our future work includes refining this function such that the user-session quality takes effect.

Acknowledgment

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*, Dec 2001.

[4] W. Cirne and F. Berman, "*A model for moldable supercomputer jobs*". In 15th *Intl. Parallel & Distributed Processing Symp.*, Apr 2001.

[5] M. E. Crovella, "*Performance evaluation with heavy tailed distributions*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 1–10, Springer Verlag, 2001. Lect. Notes Comput. Sci. vol. 2221.

[6] 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.

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

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

[10] 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.

[11] 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.

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

[13] 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.

[14] R. Gibbons, "*A historical application profiler for use by parallel schedulers*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 58–77, Springer Verlag, 1997. Lect. Notes Comput. Sci. vol. 1291.

[15] 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.

[16] P. J. Keleher, D. Zotkin, and D. Perkovic, "*Attacking the bottlenecks of backfilling schedulers*". *Cluster Comput.* **3(4)**, pp. 255–263, 2000.

[17] 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.), Springer Verlag, Jun 2004. Lect. Notes Comput. Sci. vol. 3277.

[18] H. Li, D. Groep, and J. T. L. Wolters, "*Predicting job start times on clusters*". In *International Symposium on Cluster Computing and the Grid (CCGrid)*, 2004.

[19] 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.

[20] 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.

[21] 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.

[22] D. Perkovic and P. J. Keleher, "*Randomization, speculation, and adaptation in batch schedulers*". In *Supercomputing*, p. 7, Sep 2000.

[23] W. Smith, I. Foster, and V. Taylor, "*Predicting application run times using historical information*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 122–142, Springer Verlag, 1998. Lect. Notes Comput. Sci. vol. 1459.

[24] D. Talby, *User Modeling of Parallel Workloads*. PhD thesis, The Hebrew University of Jerusalem, Israel, 200? In preparation.

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

[26] D. Tsafrir and D. G. Feitelson, *Workload Flurries*. Technical Report 2003-85, Hebrew University, Nov 2003.

[27] Y. Zhang, H. Franke, J. E. Moreira, and A. Sivasubramaniam, "*An integrated approach to parallel scheduling using gang-scheduling, backfilling, and migration*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 133–158, Springer Verlag, 2001. Lect. Notes Comput. Sci. vol. 2221.

[28] J. Zilber, O. Amit, and D. Talby, "*What is worth learning from parallel workloads? A user and session based analysis*". In *Intl. Conf. Supercomputing*, Jun 2005.

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