3Sigma: Distribution-based cluster scheduling for runtime uncertainty

Jun Woo Park
Carnegie Mellon University
junwoop@cs.cmu.edu

Alexey Tumanov
UC Berkeley
atumanov@berkeley.edu

Angela Jiang
Carnegie Mellon University
ahjiang@cs.cmu.edu

Michael A. Kozuch
Intel Labs
michael.a.kozuch@intel.com

Gregory R. Ganger
Carnegie Mellon University
ganger@ece.cmu.edu

ABSTRACT

The 3Sigma cluster scheduling system uses job runtime histories in a new way. Knowing how long each job will execute enables a scheduler to more effectively pack jobs with diverse time concerns (e.g., deadline vs. the-sooner-the-better) and placement preferences on heterogeneous cluster resources. But, existing schedulers use single-point estimates (e.g., mean or median of a relevant subset of historical runtimes), and we show that they are fragile in the face of real-world estimate error profiles. In particular, analysis of job traces from three different large-scale cluster environments shows that, while the runtimes of many jobs can be predicted well, even state-of-the-art predictors have wide error profiles with 8–23% of predictions off by a factor of two or more. Instead of reducing relevant history to a single point, 3Sigma schedules jobs based on full distributions of relevant runtime histories and explicitly creates plans that mitigate the effects of anticipated runtime uncertainty. Experiments with workloads derived from the same traces show that 3Sigma greatly outperforms a state-of-the-art scheduler that uses point estimates from a state-of-the-art predictor; in fact, the performance of 3Sigma approaches the end-to-end performance of a scheduler based on a hypothetical, perfect runtime predictor. 3Sigma reduces SLO miss rate, increases cluster goodput, and improves or matches latency for best effort jobs.

ACM ISBN 978-1-4503-5584-1/18/04…$15.00
https://doi.org/10.1145/3190508.3190515

CCS CONCEPTS

• **Computing methodologies → Planning under uncertainty**; • **Software and its engineering → Scheduling**; **Cloud computing**;

ACM Reference format:
Jun Woo Park, Alexey Tumanov, Angela Jiang, Michael A. Kozuch, and Gregory R. Ganger. 2018. 3Sigma: Distribution-based cluster scheduling for runtime uncertainty. In *Proceedings of Thirteenth EuroSys Conference 2018, Porto, Portugal, April 23–26, 2018 (EuroSys '18),* 17 pages.
https://doi.org/10.1145/3190508.3190515

1 INTRODUCTION

Modern cluster schedulers face a daunting task. Modern clusters support a diverse mix of activities, including exploratory analytics, software development and test, scheduled content generation, and customer-facing services [20]. Pending work should be mapped to the heterogeneous resources so as to satisfy deadlines for business-critical jobs, minimize delays for interactive best-effort jobs, maximize efficiency, and so on. Cluster schedulers are expected to make that happen.

Knowledge of pending jobs' runtimes has been identified as a powerful building block for modern cluster schedulers [4, 13, 28]. With it, a scheduler can pack jobs more aggressively in a cluster's resource assignment plan [4, 13, 28, 32], such as by allowing a latency-sensitive best-effort job to run before a high-priority batch job provided that the priority job will still meet its deadline. Runtime knowledge allows a scheduler to determine whether it is better to start a job immediately on suboptimal machine types with worse expected performance, wait for the jobs currently occupying the preferred machines to finish, or to preempt them [2, 28]. Exploiting job runtime knowledge leads to better, more robust scheduler decisions than relying on hard-coded assumptions.

In most cases, the job runtime estimates are based on previous runtimes observed for similar jobs (e.g., from the same user or by the same periodic job script)—a point estimate (e.g., mean or median) is determined from the relevant history. When such estimates are accurate, schedulers relying

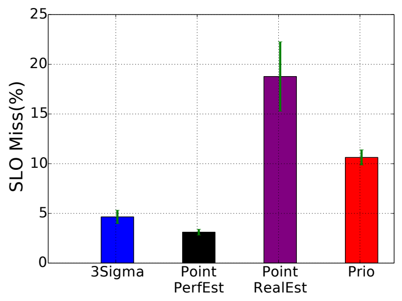


Figure 1: Comparison of 3Sigma with three other scheduling approaches w.r.t. SLO (deadline) miss rate, for a mix of SLO and best effort jobs derived from the Google cluster trace [20] on a 256-node cluster. (Details in §5) 3Sigma, despite estimating runtime distributions online with imperfect knowledge of job classification, approaches the performance of a hypothetical scheduler using perfect runtime estimates (`PointPerfEst`). Full historical runtime distributions and mis-estimation handling helps 3Sigma outperform `PointRealEst`, a state-of-the-art point-estimate-based scheduler (detailed in §2.2). The value of exploiting runtime information, when done well, is confirmed by comparison to a conventional priority-based approach (`Prio`).

on them outperform those using other approaches. Further, previous research [28] has shown that these schedulers can be robust to a reasonable degree of runtime variation (e.g., up to 50%).

However, we find that the estimate errors, while expected in large, multi-use clusters, cover an unexpectedly larger range. Applying a state-of-the-art ML-based predictor [27] to three real-world traces, including the well-studied Google cluster trace [20] and new traces from data analysis clusters used at a hedge fund and a scientific site, shows good estimates in general (e.g., 77–92% within a factor of two of the actual runtime and most much closer). Unfortunately, 8–23% are not within that range, and some are off by an order of magnitude or more. Thus, a significant percentage of runtime estimates will be well outside the error ranges previously reported.

Worse, we find that schedulers relying on runtime estimates cope poorly with such error profiles. Comparing the middle two bars of Fig. 1 shows one example of how much worse a state-of-the-art scheduler does with real estimate error profiles as compared to having perfect estimates.

This paper describes the 3Sigma cluster scheduling system, which uses all of the relevant runtime history for each job rather than just a point estimate derived from it. Instead, it uses expected runtime *distributions* (e.g., the histogram of observed runtimes), taking advantage of the much richer information (e.g., variance, possible multi-modal behaviors, etc.) to make more robust decisions. The first bar of Fig. 1 illustrates 3Sigma's efficacy, showing that it approaches the hypothetical case of a scheduler with perfect point estimates.

By considering the range of possible runtimes for a job, and their likelihoods, 3Sigma can explicitly consider the various potential outcomes from each possible plan and select a plan based on optimizing the expected outcome. For example, the predicted distribution for one job might have low variance, indicating that the scheduler can be aggressive in packing it in, whereas another job's high variance might suggest that it should be scheduled early (relative to its deadline). 3Sigma similarly exploits the runtime distribution to adaptively address a significant problem with point over-estimates, which may suggest that the scheduler avoid scheduling a job based on the likelihood of missing its deadline.

Full system and simulation experiments with production-derived workloads demonstrate 3Sigma's effectiveness. Using its imperfect but automatically-generated history-based runtime distributions, 3Sigma outperforms both a state-of-the-art point-estimate-based scheduler and a priority-based (runtime-unaware) scheduler, especially for mixes of deadline-oriented jobs and latency-sensitive jobs on heterogeneous resources. 3Sigma *simultaneously* provides higher (1) SLO attainment for deadline-oriented jobs and (2) cluster goodput (utilization). In most cases, 3Sigma performs nearly as well as the hypothetical system with perfect estimates.

This paper makes four primary contributions. First, it exposes a major problem with applying recent runtime-estimate-guided schedulers to large, multi-use clusters: significant numbers of bad estimates including some large outliers. Second, it describes an approach, which leverages full runtime distributions, that solves this problem as well as an implemented scheduling system (3Sigma) based on this solution. Third, it describes new core scheduler mechanisms, also implemented in 3Sigma, needed to make distribution-based scheduling efficient and scalable—as well as to mitigate the effects of outliers falling outside the observed history. Fourth, it reports on end-to-end experiments on a real 256-node cluster, showing that 3Sigma robustly exploits runtime distributions to improve SLO attainment and best-effort performance, dealing gracefully with the complex runtime variations seen in real cluster environments.

2 BACKGROUND AND RELATED WORK

Cluster consolidation in modern datacenters forces cluster schedulers to handle a diverse mix of workload types, resource capabilities, and user concerns [20, 23, 32]. One result of this has been a resurgence in cluster scheduling research. This section focuses on work related to using information about job runtimes to make better scheduling decisions.

Accurate job runtime information can be exploited to significant benefit in at least three ways at schedule-time.

1) Cluster workloads are increasingly a mixture of business-critical production jobs and best-effort engineering/analysis

jobs. The production jobs, often submitted by automated systems [11, 25], tend to be resource-heavy and to have strict completion deadlines [4, 13]. The best-effort jobs, such as exploratory data analytics and software development/debugging, while lower priority, are often latency-sensitive. Given runtime estimates, schedulers can more effectively pack jobs, simultaneously increasing SLO attainment for production jobs and reducing average latency for best-effort jobs [4, 13, 28].

2) Datacenter resources are increasingly heterogeneous, and some jobs behave differently (e.g., complete faster) depending upon which machine(s) they are assigned to. Maximizing cluster effectiveness in the presence of jobs with such considerations can be more effective when job runtimes are known [2, 28, 36].

3) Many parallel computations can only run when all tasks comprising them are initiated and executed simultaneously (gang-scheduling) [17, 18]. Maximizing resource utilization while arranging for such bulk resource assignments is easier when job runtimes are known.

Thus, many recent systems [4, 8, 9, 13, 28] make use of job runtime estimates provided by users or predicted from previous runs of similar jobs. Such systems assume that the predictions are accurate, and we find that they may face severe performance penalties if a significant percentage of runtime estimates is outside a relatively small error range. Worse, we find that this is to be expected in many environments.

2.1 Runtime variation and uncertainty

Analysis of job runtime predictability in production environments reveals that consistently accurate predictions should not be expected. Specifically, this section discusses observations from our analysis of job traces from three environments (details in §5): (1) analysts at a quantitative HedgeFund running a collection of exploratory and production financial analytics jobs on two computing clusters in 2016; (2) scientists at Los Alamos National Laboratory running data analysis, smaller-scale simulation, and development/test jobs on the Mustang capacity cluster between 2011 and 2016; (3) Google: the Google cluster trace[21] released in 2011 that has been used extensively in the literature. We observe the following:

First, job runtimes are heavy-tailed (longest jobs are much longer than others), suggesting that at least a degree of unpredictability should be expected. Heavy tails can be seen in the distribution of runtimes for each workload (Fig. 2(a)).

Second, job runtimes within related subsets of jobs exhibit high variability. We illustrate this with distributions of the Coefficient of Variation (CoV; ratio of standard deviation to mean), within each subset clustered by a meaningful feature, such as user id (Fig. 2(b)) or quantity of resources requested (Fig. 2(c)). CoV values larger than one (the CoV of an exponential distribution) is typically considered high variability. Large percentages of subsets in each of the workloads have high variability, with more occurring in the HedgeFund and Mustang workloads than in the Google workload.

Third, we evaluate the quality of the estimates from a state-of-the-art predictor and confirm that a significant percentage of estimates are off by factor of two or more. For this evaluation, we generated a runtime estimate for each job and compared with the actual observed runtime in the trace. We use JVuPredict, the runtime predictor module from the recent JamaisVu [27] project to generate runtime estimates. JVuPredict produces an estimate for each job by categorizing jobs (historical and new) using common attributes, such as submitting user or resources requested, and choosing the estimate from the category that has produced the best estimates in the past. Smith et al. [24] describe a similar scheme and its effectiveness for parallel computations.

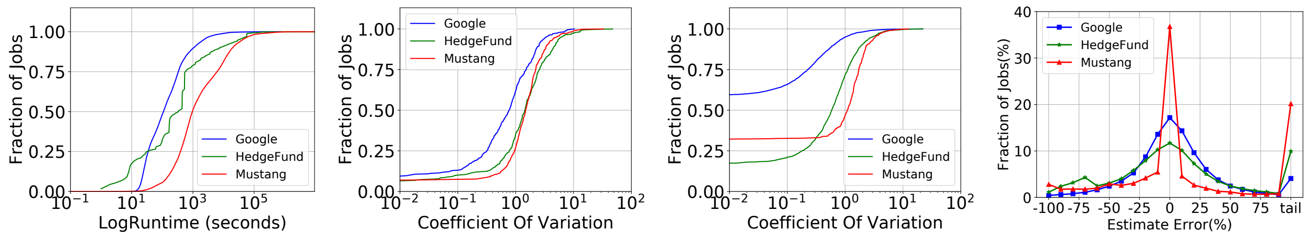
Fig. 2(d) is the histogram of percent estimate error. For all workloads, most job runtimes are estimated reasonably (e.g., ±25% error), but few are perfect. Worse, in each workload, a substantial fraction of jobs are over- or under-estimated by a large margin, well outside the range of errors considered in previous works [9, 28]. Even for the Mustang workload, which has large proportion of jobs with very accurate (±5% error) estimates, at least 23% jobs have estimate error larger than 95% and substantial amount of jobs have estimate error less than -55%. The HedgeFund trace has the fewest jobs with very accurate estimates and many jobs in both tails of the distribution. The Google cluster trace has fewer jobs in the tails of the distribution, but still has 8% of jobs mis-estimated by a factor of two or more.

Overall, we conclude that multi-purpose cluster workloads exhibit enough variability that even very effective predictors will have more and larger mis-estimates than has been assumed in previous research on schedulers that use information about job runtimes.

2.2 Mis-estimate mitigation strategies

The scheduling research community has explored techniques to mitigate the effects of job runtime mis-estimates, which can significantly hamper a scheduler's performance.

Some environments (e.g. [13, 33]) use conservative over-provisioning to tolerate mis-estimates by providing the scheduler more flexibility. Naturally, this results in lower cluster utilization, but does reduce problems. Morpheus [13] re-assigns resources to jobs that require more resources at runtime. Not all applications are designed to be *elastic*, though, and some cannot make use of additional resources.



(a) Runtime CDF (b) CoV - User Id (c) CoV - Resources Requested (d) Estimate Error

Figure 2: Analyses of cluster workloads from three different environments: (a) Distribution of job runtimes (b) Distribution of Coefficient of Variation for each subset grouped by user id (c) Distribution of Coefficient of Variation for each subset grouped by amount of resources requested (d) Histogram of Estimate Errors comparing runtime estimates from the state-of-the-art JVuPredict predictor and actual job runtimes. Estimate Error values computed by $\frac{\text{estimate}-\text{actual}}{\text{actual}} \times 100$. Each datapoint is a bucket representing values within 5% of the nearest decile. The “tail” datapoint includes all estimate errors $> 95\%$. Cluster:SC. Workload:Google_E2E, DFT_E2E, MUSTANG_E2E

Preemption can be applied to address some issues arising from mis-estimates, like it is used in many systems to reassign resources to new high-priority jobs, either by killing (e.g., in container-based clusters [33]) or migrating (e.g., in VM-based systems [35]) jobs.

Various other heuristics have been used to mitigate the effects of mis-estimates. [26] addresses mis-estimations of runtimes for HPC workloads by exponentially increasing under-estimated runtimes and then reconsidering scheduling decisions. Other systems [3, 15] use the full runtime distribution to compare the expected benefits of scheduling jobs. The “stochastic scheduler” [22] uses a conservative runtime estimate by padding the observed mean by one or more standard deviations. Such heuristics help (3Sigma borrows the first two), but do not eliminate the problem.

2.3 Distribution-based scheduling

Are estimates of job runtime distributions more valuable than point estimates (e.g., estimates of the average job runtime) for cluster scheduling? Intuitively, the distribution provides strictly more information to the scheduler than the point estimate. A simple example below illustrates the point. Suppose two jobs arrive to be scheduled on a toy cluster, and the resources are sufficient to execute only one job at a time. Further, one job is an SLO job with a 15 minute deadline, and the other is a best-effort (BE) job. The objective of the scheduler is to minimize SLO violations, while also minimizing BE job latency. The key question is which job should be executed first?

To answer that question, the scheduler naturally needs more information. Let’s start by assuming a point-estimate based scheduler. In our example, imagine that the average runtime of jobs like each of these is known to be 5 minutes. Because the deadline window of 15 minutes is 50% longer the sum of the two point estimates (10 minutes), one might assume that scheduling the BE job first would be relatively safe, which would allow the BE job to start early while still respecting the deadline of the SLO job.

Consider, instead, a distribution-based scheduler, and let’s imagine two cases: A and B. In case A, the runtime distribution of each job (SLO and BE) is uniform over the interval 0 to 10 minutes. The average runtime is still 5 minutes, but the scheduler is able to calculate that the probability of the SLO job missing its deadline would be 12.5% if the BE job were scheduled first. Hence, scheduling the SLO job first may be desirable. For case B, in contrast, imagine that the distributions are uniform over the interval 2.5 to 7.5 minutes. Again, the average runtime is 5 minutes, but now the scheduler may safely schedule the BE job first, because even if both jobs execute with worst-case runtimes, the SLO job will finish in the allotted 15 minute window.

The key observation is that the distributions enable the scheduler to make better-informed decisions; knowing just the average job runtime is not nearly as valuable as knowing whether jobs are drawn from distribution A or B. Two caveats should be mentioned here. First, we implied in this discussion that the SLO deadline is strict; in some environments, this may not be true, and some weighting between BE job start time and SLO miss rate may be desirable. Second, the discussion assumed that the distribution supplied to the scheduler is accurate. In practice, the distribution will have to be estimated in some way—likely from historical job runtime data—and may differ from observed behavior. We address both of these topics later and show that estimated distributions are effective for the workloads studied.

3 DISTRIBUTION-BASED SCHEDULING

In this section, we describe the mechanisms that enable schedulers to use the full runtime distribution, as opposed to point estimates. Any scheduler wanting to take advantage of runtime information can use the following generic scheduling algorithm. The scheduler first generates all possible placement options ($resource\_type$, $start\_time$), each of which has an associated utility. The scheduler chooses to run the set of jobs which both maximize overall sum of utility and fit within the available resources.

Using point runtime estimates, we can find the best schedule using basic optimization techniques (e.g. MILP). However, with runtime distributions, we have a much larger state-space to consider. For each running job, there are many possible outcomes. Naively considering all scenarios easily makes this problem intractable. Instead of considering each option, we use the *expected utility* per job and *expected resource consumption* over time. This section describes how both of these values are calculated.

3.1 Valuation of scheduling options

For each job, there is a set of possible placement options. The placement of the job dictates the job's final completion time, and consequently, it's usefulness or *utility.* A scheduler needs to place jobs in a way that maximizes overall utility. This section describes how to associate job placement options with the utility of the job.

Utility. To make informed placement decisions, a scheduler must quantify its options relative to the success metric the job cares about. We use utility functions to represent a mapping from the domain of possible job placement options and completion times to the potential utility of the job. We assume that a cluster administrator or an expert user will be able to define the utility function on a job-by-job basis. However, in this work, we model the utility of SLO and latency sensitive jobs separately. The utility curve used for SLO jobs is shown in Fig. 3(a). This curve models a job with constant utility if completed within the deadline, and zero utility if completed after the deadline. On the other hand, we represent latency sensitive jobs as having a linearly decreasing function over time to declare preference to complete faster.

Expected utility. For each placement option ($resource\_type$, $start\_time$), a scheduler computes the expected utility of a job using the runtime distribution. The expected utility is calculated as the sum of utilities for each runtime t, weighted by the probability that the job runs for t:

$$E[U(\text{startTime})] = \int_0^{\max(runtime)} U(\text{startTime} + t) PDF(t) dt \quad (1)$$

where $U(t)$ is utility function for placement in terms of completion time, and PDF is the probability density function for the job runtime. Fig. 3 provides a simple example.

3.2 Expected resource consumption

To calculate the set of available resources over time, we need to estimate the resource usage of currently running jobs over time. The use of point estimates for runtimes makes an implicit assumption that resource consumption is deterministic. In contrast, using full distributions acknowledges that resource consumption is, in fact, probabilistic for jobs with uncertain duration. Thus, we calculate the *expected* resource consumption, similarly to expected utility (§3.1).

The expected resource consumption of a job at time-slice t is dependent on the probability that the job still uses those resources at (i.e., hasn't completed by) time t. Given PDF(t)—the probability density function of a job's runtime, CDF(t) captures the probability with which the job will complete in at most t time units. The inverse CDF, or $1 - CDF(t)$ then captures the probability with which the job will complete in at least t time units, which is also the probability the job still uses the resources at time t. Thus, expected resource consumption at time t equals to the job's resource demand multiplied by $1 - CDF(t)$.

For running jobs, 3σSched updates the runtime distribution, as it has additional information, namely the fact that the job has been running for some elapsed_time. This enables us to dynamically compute a *conditional* probability density function for the job's expected runtime $P(t|t \geq elapsed\_time)$. This probability update simply renormalizes the original $CDF_{\text{original}}(t)$ and computes the updated probability distribution as follows:

$$1 - CDF_{\text{updated}}(t) = \frac{1 - CDF_{\text{original}}(t)}{1 - CDF_{\text{original}}(\text{elapsed\_time})} \quad (2)$$

The amount of available resources in the cluster at time t is then computed by subtracting the aggregate expected resource consumption at time t from the full cluster capacity.

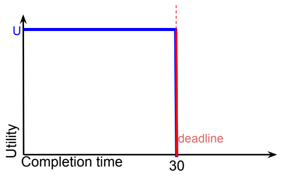
4 DESIGN AND IMPLEMENTATION

This section describes the architecture of 3Sigma (Fig. 4). 3Sigma replaces the scheduling component of a cluster manager (e.g. YARN). The cluster manager remains responsible for job and resource life-cycle management.

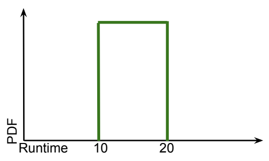
Job requests are received asynchronously by 3Sigma from the cluster manager (Step 1 of Fig. 4). As is typical for such systems, the specification of the request includes a number of attributes, such as (1) the name of the job to be run, (2) the type of job to be run (e.g. MapReduce), (3) the user submitting the job, and (4) a specification of the resources requested.

The role of the predictor component, 3σPredict, is to provide the core scheduler with a probability distribution of the execution time of the submitted job. 3σPredict (§4.1) does this by maintaining a history of previously executed jobs, identifying a set of jobs that, based on their attributes, are similar to the current job and deriving the runtime distribution the selected jobs' historical runtimes (Step 2 of Fig. 4).

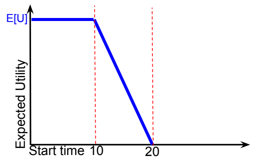
Given a distribution of expected job runtimes and request specifications, the core scheduler, 3σSched decides *which jobs*



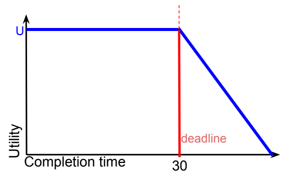
(a) Job's Utility Function



(b) Estimated PDF of Runtime



(c) Expected Utility Function



(d) Job's Utility Function with Overestimate Handling

Figure 3: Example curves for estimating utility for a given job. Each job is associated with a utility function (a) describing its value as a function of completion time. 3σPredict produces a PDF (b) describing potential runtimes for the job. 3σSched combines them to compute expected utility (c) for the job as a function of its start time. [Note the different x-axes for (a), (b), and (c).] As described in §4.2, the overestimate handling technique involves modifying the utility function (a) associated with the job with an extended version illustrated in (d).

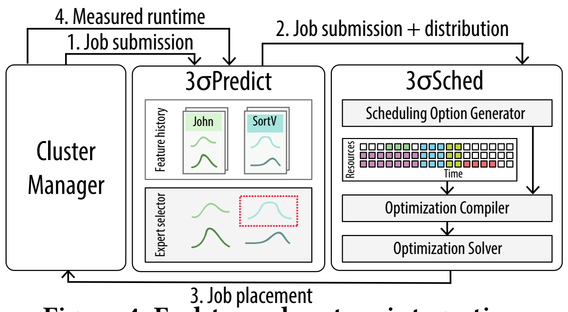


Figure 4: End-to-end system integration.

to place on which resources and when. The scheduler evaluates the expected utility of each option (§3.1) and the expected resource consumption and availability over the scheduling horizon (§3.2). Valuations and computed resource capacity are then compiled into an optimization problem (§4.3), which is solved by an external solver. 3σSched translates the solution into an updated schedule and submits the schedule to the cluster manager (Step 3 of Fig. 4). On completion, the job's actual runtime is recorded by 3σPredict (along with the attribute information from the job) and incorporated into the job history for future predictions (Step 4 of Fig. 4).

In this section, we detail how 3σPredict estimates runtime distributions (§4.1), how 3σSched handles mis-estimation (§4.2), and the details of the core scheduling algorithm (§4.3).

4.1 Generating runtime distributions

For each incoming job, 3σPredict provides 3σSched with an estimated runtime distribution. 3σPredict generates this distribution using a black-box approach for prediction. It does not require user-provided runtime estimates, knowledge of job structures, or explicit declarations of similarity to specific previous jobs. However, it does assume that, even in multi-purpose clusters used for a diverse array of activities, most jobs will be similar to some subset of previous jobs.

3σPredict associates each job with set of features. A feature corresponds to an attribute of the job (e.g., user, program name, submission time, priority, resources requested, and etc.). Attributes can be combined to form a single feature as well (e.g., user and submission time). 3σPredict tracks job runtime history for each of multiple features, because no single feature is sufficiently predictive for all jobs.

3σPredict associates the new job with historical job runtimes with the same features. Because no single feature is always predictive, 3σPredict generates multiple *candidate distributions* for each job. For example, one candidate distribution may consist of runtimes of jobs submitted a single user. A second candidate distribution may consist of runtimes of jobs submitted with the same job name.

3σPredict selects one candidate distribution to send to 3σSched. To make this decision, 3σPredict compares each distribution's ability to make accurate point estimates. For a given candidate distribution, 3σPredict makes point estimates in multiple ways as different estimation techniques will be more predictive for different distributions. Specifically, 3σPredict uses four estimation techniques: (a) average, (b) median, (c) rolling (exponentially weighted decay with $\alpha = 0.6$), (d) average of X recent job runtimes. 3σPredict tracks the accuracy of each feature-value:estimator pair, which we refer to as an "expert", using the normalized mean absolute error (NMAE) of past estimates. It designates the runtime distribution from the expert with the lowest NMAE as the distribution estimate of the job.

3σPredict does not make any assumption about the shape of the distribution. Instead, we use empirical distributions, stored as a histogram of the runtimes for each group. Runtimes often exhibit uneven distributions (e.g. heavy-tailed, multi-modal), so we use varying bucket widths to ensure that the shape of the distribution is accurately modeled. We dynamically configure bin sizes using a stream histogram algorithm [1] with a maximum of 80 bins.

Scalability. Storing and querying the entire history of runtimes of a datacenter is not scalable. 3σPredict employs several sketching techniques to greatly reduce the memory footprint. 3σPredict 1) uses a stream histogram algorithm [1] to maintain an approximate histogram of runtimes, 2) computes the average and rolling estimates and NMAE metric for each expert in a streaming manner, and 3) computes the median using recent values as a proxy for the actual median. Using these techniques, 3σPredict provides effective runtime distributions using constant memory, per feature-value.

4.2 Handling imperfect distributions

3σPredict estimates the empirical distribution of a job using the history of previously executed jobs. In practice, the estimated runtime distribution is imperfect. Not all jobs have sufficient history to produce a representative distribution. The runtimes of recurring jobs will also evolve over time (e.g. different input data, program updates). 3σSched uses the following mitigation strategies to tolerate error in the estimated runtime distribution.

4.2.1 Under-estimate handling. Distribution schedulers encounter under-estimates when a job runs longer than all historical job runtimes provided in the distribution. An under-estimate can cause a queued job waiting for the busy resource to starve or miss its deadline. To mitigate this, when the elapsed time of the job reaches the maximum observed runtime from the distribution, 3σSched exponentially increments the estimated finish time by 2^t cycles, starting with $t = 0$ in similar fashion to [26]. Exponential incrementing (exp-inc) avoids over-correcting for minor mis-predictions. As 3σSched learns that the under-estimate is more significant, it updates the runtime estimate by progressively longer increments. Note that under-estimates in 3σSched are much more rare compared to using a single point estimate. Point estimate schedulers encounter under-estimates when a job runs longer than the point estimate, whereas 3σSched encounters under-estimates when a job runs longer than all historical job runtimes.

4.2.2 Over-estimate handling. 3σSched encounters over-estimates when all historical runtimes are greater than the time to deadline. In this case, the expected utility is zero, leading the scheduler to not see any benefit from spending resources on the job. 3σSched would prefer to keep resources idle, rather than scheduling a job with zero utility. To mitigate the effects of over-estimates, 3σSched proactively changes the utility functions of SLO jobs to degrade gracefully. Instead of a sharp drop to zero utility (Fig. 3(a)), 3σSched uses a linearly decaying slope past the deadline (Fig. 3(d)). This way, the estimated utility of the job will be non-zero, even if all possible completion times exceed the deadline. The post-deadline utility will be lower than other SLO jobs submitted with the same initial utility. 3σSched will therefore only schedule seemingly impossible jobs when there are available resources in the cluster.

4.2.3 Adaptive over-estimate handling. Enabling 3σSched's over-estimate handling comes at a cost. It increases the number of SLO jobs being tried in favor of completing lower priority jobs. For jobs that were not over-estimated, resources are wasted. Ideally, we should only enable over-estimate handling for jobs which have a reasonable probability of being over-estimates.

3σSched leverages the user provided deadline for SLO jobs in predicting the probability that a job is over-estimated. The deadlines for high priority SLO jobs in production systems are known to be correlated with its actual runtime, since they are usually the result of profiled test runs or previous executions of the same jobs. Thus, 3σSched treats the time from submission to deadline as a reasonable proxy for the upper-bound of the runtime. It compares this upper-bound with the runtime distribution and enables over-estimate handling only if the likelihood of running for less than the upper-bound is below a configured threshold. If the historical runtime distribution implies that the job has no chance of meeting its deadline, even if started immediately upon submission, it is likely that the runtime distribution is skewed toward over-estimation.

4.3 Scheduling algorithm

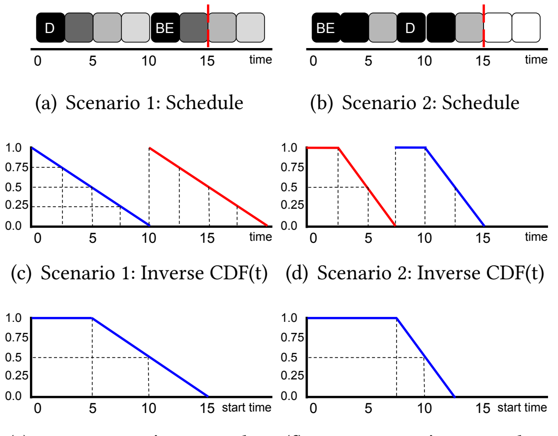
This section describes the details of the core scheduling algorithm used by 3σSched. The discussion includes how we adapt the generalized scheduling algorithm (§3) to cope with approximate runtime distributions, the formulation of the optimization problem, and algorithm extensions to support preemption. We conclude the section by examining scalability issues arising from the complexity of solving MILP.

4.3.1 Intuition. The high level intuition behind the scheduling algorithm is to bin-pack jobs, each represented as a space-time rectangle in cluster resource space-time, where the x-axis represents time and the y-axis enumerates the resources available. Current time is represented as a point on the x-axis. Placing a job further along the x-axis, away from current time, is equivalent to deferring it for future execution. This is useful when a job's preferred resources are busy, but are expected to free up in time to meet the job's deadline. Placing a job on different types of resources (moving its position along the y-axis) changes the shape of the rectangles, as the resource type affects its required resources and completion time. Each job can then be thought of as an enumeration of candidate space-time rectangles, each corresponding to a utility value. The job of the scheduler is to maximize the overall utility of the placement decision by making an instantaneous decision on (a) which jobs to execute and which to defer, and (b) which placement options to pick for those jobs. To achieve this, the scheduler must have a way to formulate all jobs' resource requests so that all pending requests may be considered in aggregate. 3σSched achieves this by formulating jobs' resource requests as Mixed Integer Linear Programming instances.

The scheduler operates on a periodic cycle (at the granularity of seconds, e.g., 1-2s), making a placement decision at each cycle for all pending jobs. The schedule for all pending jobs is re-evaluated every cycle to provide a basic level of robustness to runtime mis-estimation [28]. The sketch of the scheduling algorithm is as follows.
(1) Translate each job's resource request to its MILP representation.
(2) Aggregate jobs' demand constraints.
(3) Construct resource capacity constraints.
(4) Construct the aggregate objective function as the sum of jobs' individual objective functions, modulated by binary indicator variables.
(5) Solve the MILP (using an external MILP solver).
(6) Extract job placement results from MILP solution.
(7) Report the scheduling decision to the resource manager.
(8) Dequeue scheduled jobs from the pending queue.

4.3.2 *Example.* Fig. 5 illustrates the example introduced in §2.3, where two jobs simultaneously arrive to a single-node cluster: an SLO job (D) with a 15min deadline and a best effort (BE) job. Here, we highlight the mechanics of leveraging the distribution information to achieve the best job schedule. In the left column of Fig. 5, we focus on a first scenario, in which both jobs' expected runtimes are drawn from a uniform distribution $U(0, 10)$. The right column focuses on a second scenario, where the jobs' expected runtimes are drawn from a uniform distribution $U(2.5, 7.5)$. In scenario 1, the scheduler picks a schedule that schedules the SLO job, because it recognizes the risk of it not completing in time otherwise, while in scenario 2, it schedules the BE job first since the SLO job is expected to finish in time regardless of where the runtimes fall within the full distribution. In both cases, it realizes the right decision by maximizing the overall expected utility offered by the two pending jobs.

As sketched in §4.3.1, 3σSched first constructs and aggregates the jobs' expected demand. The key insight is that this draw on resources over time is probabilistic (Figs. 5(a) and 5(b)). E.g., with the SLO job scheduled to start at $t = 0$, it is *expected* to consume the cluster node with 100% probability in the first time slice, and monotonically decreasing probability < 1 in subsequent time slices. Fig. 5 plots these probabilities on gray scale from 0 (white: resource not used) to 1 (black: resource expected to be used with 100% probability). We use this grayscale to illustrate the best job schedule in each of the two scenarios in Fig. 5. Note that the SLO job meets its 15min deadline in both cases, but is scheduled first in scenario 1 and second in scenario 2. Expected resource consumption is calculated by referring to the inverse



(a) Scenario 1: Schedule (b) Scenario 2: Schedule
(c) Scenario 1: Inverse CDF(t) (d) Scenario 2: Inverse CDF(t)
(e) Scenario 1: D's Exp. Utility (f) Scenario 2: D's Exp. Utility

Figure 5: Job D is an SLO job with a 15min deadline. Job BE is a BE job. Left column: job runtimes $\sim U(0, 10)$ (scenario1). Right column: job runtimes $\sim U(2.5, 7.5)$ with the same $\mu = 5$ (scenario 2). (a) and (b): The final order that yields maximal utility. The intensity of the black represents the expected resource consumption at the start of each slot (from 100% certainty (darkest) to 25% certainty (lightest), in 25% decrements for the scenario 1 and a 50% decrement for the scenario 2. (c) and (d): Inverse CDF ($1 - CDF(t)$), the probability of D (blue) and BE (red) jobs completing before t, which is also the probability of still using the resource at that time. (e) and (f): SLO job's expected utility, set to the probability of the job's completion by the deadline at each start time in this example (note: x-axis is different from other subfigs).

CDF (Figs. 5(c) and 5(d)) of each job's conditional runtime distribution (§3.2).

As the scheduler constructs and aggregates resource demands from both jobs, it ensures that their sum does not exceed the expected resource capacity at any given time t in the plan-ahead window $\in [0; 20)$. These declarative constraints are automatically generated and added to the MILP problem instance (as described below). The aggregate utility for each job, derived from the expected utility curves, forms the overall objective function to maximize. Figs. 5(e) and 5(f) show the expected utility curves for the SLO job ("D") in the two scenarios, respectively, and the BE job's utility curves (not shown) are a linearly decaying function with a significantly lower maximum value. Figs. 5(a) and 5(b) show the best outcome for each of the scenarios. Concretely, we observe that awareness of the runtime distribution allows the scheduler to determine the likelihood of it being safe to delay an SLO job (Fig. 5(b)) to minimize the BE job's latency while still meeting the SLO job's deadline.

4.3.3 *MILP Formulation.* The first step of the algorithm is to convert all jobs to their MILP representation. This is done by generating all possible placement options, both over different types of resources (space) and over time. 3σSched minimizes the set of possibilities by adopting the notion of *equivalence sets*, introduced in [28]. Equivalence sets are sets of resources equivalent from the perspective of a given job, e.g. all nodes with a GPU. 3σSched reasons about these sets of resources instead of enumerating all possible node combinations. As a result, the complexity of MILP depends on the number of equivalence sets rather than the cluster size. Thus, equivalence sets help manage the size of generated MILP in the space dimension. MILP size in the time dimension is controlled by the plan-ahead window *sched*.

A given placement option includes a specification of the equivalence set, the starting time $s \in [now; now + sched]$, the estimated runtime *distribution*, and how many nodes are requested k. The estimated runtime distribution for a running job is reconsidered at every scheduling event, based on how long the job has run so far as described in Eq. 2. Updates to the runtime distribution changes the scheduler's expectation of the jobs' future resource consumption. This allows 3σSched to react to mis-estimates, e.g., by re-planning pending jobs waiting for preferred resources to a different set of nodes, or preempting lower priority jobs.

Each placement option is associated with a *const* utility value obtained by using Eq. 1. The corresponding objective function for this job becomes a sum of these values modulated by binary indicator decision variables. Namely, given job j and placement option o, the MILP generator associates an indicator variable I_{jo}, adding a constraint that at most one option is selected for each job: $\forall j \sum_o I_{jo} \le 1$. Thus, the aggregate objective function to maximize is $\sum_j \sum_o U_{jo} I_{jo}$. A solution that maximizes this function effectively selects (a) which jobs to run now, and (b) which placement option o to pick for selected job j.

This objective function is maximized subject to a set of auto-generated constraints: capacity and demand constraints. Demand constraints ensure that (a) the sum of allocations from different resource partitions [28] is equal to the requested quantity of resources k, and (b) at most one placement option is selected: $\forall j \sum_o I_{jo} \le 1$. Capacity constraints provide the invariant that

$$\forall t \in [now; now + sched] \sum_{jo} k \cdot RC_j(t - s) I_{jo} \le C(t), \quad (3)$$

where $RC_j(t)$ is the expected resource consumption of job j at time t (§3.2). This ensures that aggregate allocations do not exceed the expected available capacity $C(t)$ at time t.

4.3.4 MILP Example. Let's examine 3σSched's MILP formulation in detail on the same two job example (Fig. 5).

Scenario 1 ($U(0, 10)$). First, for each job j, the scheduler generates indicator variables, I_{jt} where $t \in \{0, 2.5, ..., 17.5\}$, that represent whether the job should be scheduled or deferred to each t. The expected utilities of each placement option is also computed. For the SLO job, the expected utility is 1 for all placement options with start times $t \le 5$. The options that start later will have gradually decreasing values, 0.75, 0.5, 0.25, and 0, as the probability of missing the deadline increases as a function of time. The expected utilities for the BE job will be linearly decreasing values over time.

Second, demand and capacity constraints are generated. A demand constraint $\sum_t I_{jt} \le 1$ is constructed to ensure at most one option is selected for each job j. Capacity constraints are generated by calculating expected resource consumption for each job. For a job that starts at $t = s$, the expected resource consumption at elapsed_time $= 0, 2.5, 5, 7.5$ is the probability of the job still running and equals 1.0, 0.75, 0.5, 0.25 respectively, zero thereafter.

Third, 3σSched constructs the overall MILP problem by aggregating per-job MILP contributions. Demand constraints are aggregated into the constraints of the problem. Resource capacity constraints are constructed by aggregating the expected resource consumption of placement options across all jobs for each time slot. For example, it is $0.75 I_{SLO;0} + 1.0 I_{SLO;2.5} + 0.75 I_{BE;0} + 1.0 I_{BE;2.5} \le 1$ for $t = 2.5$.

The aggregate objective function is constructed by adding all $U_{jt} I_{jt}$ terms, where U_{jt} is the expected utility of the placement option for job j that starts at t. Examining the objective function while satisfying the demand and capacity constraints, 3σSched decides to schedule the SLO job first. Starting the SLO job at $t = 10$ would only yield an expected utility of 0.5, which corresponds to a 50% probability of meeting the deadline. The BE job utility gain would be insufficient to offset the SLO utility loss.

Scenario 2 ($U(2.5, 7.5)$). First, all the decision variables are generated similarly to the scenario 1. The expected utility is calculated for both jobs. For the SLO job, the expected utility is 1 for all placement options with start times $t \le 7.5$. The expected utility of the BE job and the demand constraints are same as before.

For a job that starts at $t = s$, the expected resource consumption at elapsed_time $= 0, 2.5, 5$ is $1.0, 1.0, 0.5$ respectively, and zero otherwise.

The aggregate MILP problem is constructed, aggregating demand constraints, objective functions, and generating capacity constraints that ensure the sum of aggregate resource demand does not exceed total resource capacity. This ensures the second job is deferred to start at $t = 7.5$.

Examining the objective function while satisfying the demand and capacity constraints, 3σSched decides to schedule the BE job first and postpones the SLO job, as it is possible to complete both jobs before the deadline. The aggregate expected utility reflects that, as it yields the SLO job utility of 1 when started by $t = 7.5$ and the highest value for the BE job when started at $t = 0$.

4.3.5 *Preemption.*

In rare cases, 3σSched needs to reconsider scheduling decisions for currently running jobs. For example, due to under-estimates, the scheduler may incorrectly choose to aggressively postpone SLO jobs to complete more BE jobs. The scheduler might be able to reschedule if there are enough resources. However, sometimes the only way to meet the deadline is to preempt lower-priority jobs running in the cluster.

Preemption is naturally supported in the existing MILP generation framework, as it is able to simultaneously consider pending jobs for placement and running jobs for preemption. Thus, the goal of the scheduler is to maximize the aggregate value of placed jobs, while incurring a cost for preempting jobs. The latter is a $\sum_r P_r I_r^p$, where I_r^p is an indicator variable tracking whether to preempt a running job r. P_r is the preemption cost for the running job r and is configured by the preemption policy. The overall objective function then becomes $\sum_{j,o} U_{jo} I_{jo} - \sum_r P_r I_r^p$.

The capacity constraint extension, intuitively, credits back resources associated with preempted jobs: $\sum_{jo} k_o RC_j(t - s) I_{jo} \leq C(t) + \sum_r k_r RC_r(t-e) I_r^p$. RC_r is the up-to-date expected resource consumption of the running job r, and e is the elapsed time of r.

4.3.6 *Scalability.*

Solving MILP is known to be an NP-hard problem. To minimize the excessive latency caused by the solver, we apply a number of optimizations. The primary optimization we perform is seeding each new cycle's MILP problem with the solution from the previous cycle. Intuitively, the previous cycle's solution corresponds to leaving the cluster state unchanged. As such, it represents a feasible solution. Second, we have empirically found that the solver spends most of the time validating optimality for the solution it otherwise quickly finds. Thus, we get near-optimal performance by querying the solver for the best solution found within a configurable fraction of its scheduling interval. Third, the plan-ahead window bounds the complexity of the MILP problem by adjusting the range of time over which job placements are considered. Fourth, 3σSched performs some internal pruning of generated MILP expressions, which include eliminating terms with zero constant.

5 EXPERIMENTAL SETUP

We conduct a series of end-to-end experiments and microbenchmarks to evaluate 3Sigma, integrated with Hadoop YARN [29]– a popular open source cluster scheduling framework. We find YARN's support for time-aware reservations and placement decisions and its popularity in enterprise a good fit for our needs. We implement a proxy scheduler wrapper that plugs into YARN's ResourceManager and forwards job resource requests asynchronously to 3Sigma. Jobs are modeled as Mapper-only jobs. We use a synthetic generator based on Gridmix 3 to generate Mapper-only jobs that respect the runtime parameters for arrival time, job count, size, deadline, and task runtime from the pre-generated trace.

System	Runtime Estimation	Overestimate Handling
3Sigma	Real Distributions	ADAPTIVE
`PointPerfEst`	Perfect Point Estimates	NO
`PointRealEst`	Real Point Estimates	NO
`Prio`	N/A	N/A

Table 1: Scheduler approaches compared.

Cluster configurations. We conduct experiments on two cluster configurations: a 256-node real cluster (RC256) and a simulated 256-node cluster (SC256). RC256 consists of 257 physical nodes (1 master + 256 slaves in 8 equal racks), each equipped with 16GB of RAM and a quad-core processor. The simulations complete in $\frac{1}{5}^{th}$ the time on a single node, allowing us to evaluate more configurations and longer workloads. We also conduct an experiment with a simulated 12,583-node cluster (GOOGLE) to evaluate 3Sigma's scalability.

Systems compared. We compare the four scheduler approaches in Table 1. 3Sigma is our system in which 3σSched is given real runtime distributions provided by 3σPredict and uses adaptive overestimate handling. Both `PointPerfEst` and `PointRealEst` use an enhanced version of [28] with underestimate handling (§4.2) and preemption (§4.3). It represents the state-of-the-art in schedulers that rely on point estimates. This includes Rayon, Morpheus, and TetriSched [4, 13, 28], enhanced with the state-of-the-art in techniques for handling imperfect estimates. `PointPerfEst` is a hypothetical system in which the scheduler is given a correct runtime for every incoming job. `PointRealEst` uses point runtime estimates from 3σPredict. `Prio` is a priority scheduler, giving SLO jobs strict priority over BE jobs rather than leveraging runtime information, which represent schedulers like Borg [33].

Workloads. The bulk of our experiments use workloads derived from the Google cluster trace [20]. We use a Google trace-derived workload (termed "E2E") for overall comparisons among schedulers as well as workloads that vary individual workload characteristics (e.g., runtime variation or cluster load) to explore sensitivities. All workloads are 5 hours in length (~1500 jobs) except for the 2hr E2E (~600 jobs), used to expedite the experiment in RC256. The E2E workload is synthetically generated from Google trace characteristics. We evaluated the quality of estimates (as in §2.1) and confirmed that the runtime predictability of the generated workload was similar to the original Google trace. In simulation, we have also obtained similar experimental results by drawing random trace samples from the original instead of using the E2E workload. To generate a workload, all jobs larger than 256 nodes were filtered out. The remaining jobs — clustered using k-means clustering on their runtimes. We derive parameters for the distributions of the job attributes (e.g., runtime and number of tasks) and the probability mass function of features in each job class. The arrival process used was exponential with a coefficient of variance of 4 (c_a^2=4). We draw jobs from each job class proportionally to the empirical job-class distribution. We also pick job attributes and features for each job according to the empirical distribution of attributes and features from the job-class. Each workload consists of an even mixture of SLO jobs with deadlines and latency sensitive best effort (BE) jobs. SLO jobs have soft placement constraints (preferred resources set to a random 75% of the cluster, as observed in the original trace). SLO jobs run 1.5x longer if scheduled on non-preferred resources.

For the experiment in §6.1, we also use workload HEDGEFUND_E2E and MUSTANG_E2E derived from the HedgeFund and Mustang cluster, respectively. For these workloads we filtered out jobs larger than 256 nodes, but took a 5 hour segment of the original workload instead of deriving parameters and regenerating based on the distribution. The segment was randomly selected among many segments that have a similar load to the E2E workload.

HedgeFund: This workload is collected from two private computing clusters of a quantitative hedge fund firm. Each cluster uses an instance of an internally developed scheduler, run on top of a Mesos cluster manager. The workload consists of 3.2 million jobs submitted to two clusters over a nine month period. The majority of jobs analyze financial data and there are no long-running services.

Mustang: This workload includes the entire operating history of the Mustang HPC cluster used for *capacity computing* at Los Alamos National Laboratory. Entire machines are allocated to users, in similar fashion to Emulab[10, 34]. The workload consists of 2.1 million jobs submitted within a period of 61 months (2011 ∼ 2016).

Estimates. Because the experiments are 5-hour windows, we pre-train 3σPredict before running them to produce steady-state estimates for 3Sigma and `PointRealEst`. For the Google workload, we use a subset of the generated trace for pre-training and use the rest for our experiments. Only the features present in the original trace were used to generate point and distribution estimates (e.g., job class, the runtime class membership feature not present in the original trace, was never used in order to maintain a fair experimental setup). For other workloads, we pre-train on jobs completed before the selected 5 hour segment begins.

Workload configurations. For SLO jobs, the deadline slack is an important consideration. Since the original workloads do not include deadline information, we generate deadlines for each SLO job as follows. Deadline slack is defined as $(deadline - submissiontime - runtime)/runtime * 100$ (i.e., a slack of 60% indicates that the scheduler has a window 60% longer than the runtime in which to complete the job). Tighter deadlines are more challenging for schedulers. By default, we select each job's deadline slack randomly from a set of 4 options: 20%, 40%, 60%, and 80%. These default values are much smaller than experimented in [28] (which used slacks of 250% and 300%), matching the finding in [13] that tighter deadlines are also possible.

Load is a measure of offered work ($machine \times hours$) submitted to the cluster scheduler as a proportion of cluster capacity. The nominal offered load is 1.4 (unless specified otherwise). We first chose the load for SLO jobs as 0.7, approximating the load offered by production jobs in [21]. We added equal proportion of BE jobs as to not unfairly bias the scheduling problem towards SLO jobs and to demonstrate the behavior of system under stressful conditions.

Note our definition of load is different from effective load, a ratio of actual resources allocated for all jobs (successful and not successful) to the cluster capacity. Effective load is different for each scheduling approach as they make different allocation decisions, even if the same jobs are injected to the system. In all experiments and for all scheduling approaches, the cluster was run close to its space-time capacity.

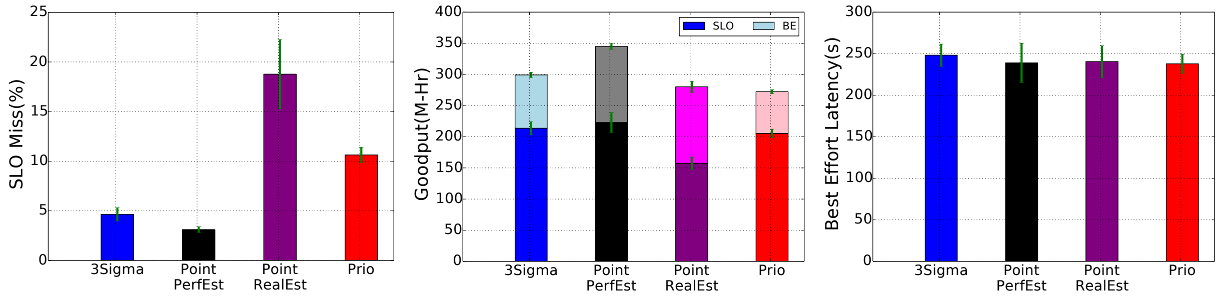
Success metrics. We use the following goodness metrics when comparing schedulers. Our primary goal is to minimize SLO miss rate: the percentage of SLO jobs that miss their deadline. We also want to measure the total work completed in machine-hours (goodput), showing how much aggregate work is completed, since BE goodput and the goodput of incomplete SLO jobs is not represented by the SLO miss rate. Finally, we measure mean BE latency—the mean response time for BE jobs.

6 EXPERIMENTAL RESULTS

This section evaluates 3Sigma, yielding five key takeaways. First, 3Sigma achieves significant improvement over the state-of-the-art in SLO miss rate, best-effort job goodput, and best-effort latency in a fully-integrated real cluster deployment, approaching the performance of the unrealistic `PointPerfEst` in SLO miss rate and BE latency. Second, all of the 3σSched component features are important, as seen via a piecewise benefit attribution. Third, estimated distributions are beneficial in scheduling even if they are somewhat inaccurate, and such inaccuracies are better handled by distribution-based scheduling than point-estimate-based scheduling. Fourth, 3Sigma performs well (i.e., comparably to `PointPerfEst`) under a variety of conditions, such as varying cluster load, relative SLO job deadlines, and prediction inaccuracy. Fifth, we show that the 3Sigma components (3σPredict and 3σSched) can scale to >10000 nodes.

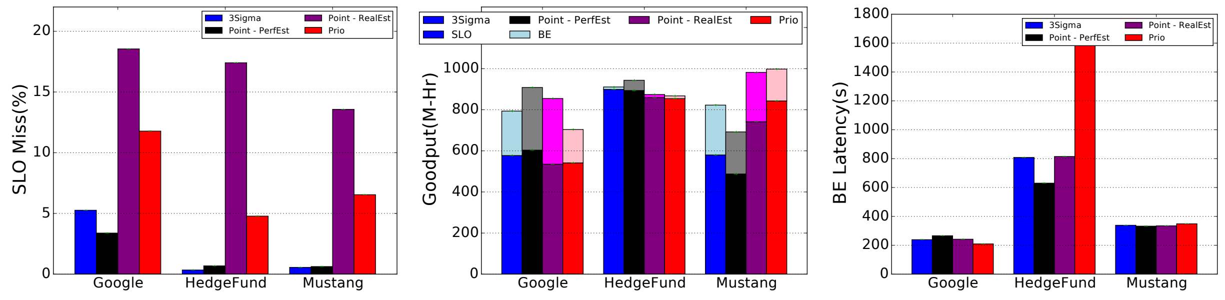
6.1 End-to-end performance

Fig. 6 shows performance results for the four scheduling systems running on the real cluster (RC256).



(a) SLO Miss rate (b) Goodput (c) BE latency

Figure 6: Compares the performance of 3Sigma with other systems in the real cluster. 3Sigma constantly outperforms `PointRealEst` **and** `Prio` **on SLO miss-rate and Goodput while nearly matching** `PointPerfEst`**. Cluster:RC256. Workload:E2E**



(a) SLO Miss rate (b) Goodput (c) BE latency

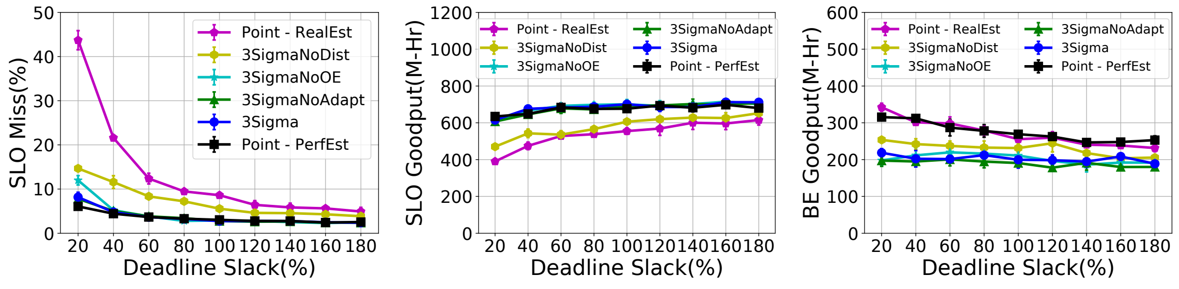
Figure 7: Compares the performance of 3Sigma with other systems under workloads from different environments in simulated cluster. 3Sigma constantly outperforms `PointRealEst` **and** `Prio` **on SLO miss rate and Goodput while nearly matching** `PointPerfEst`**. The Google workload is 5hr variant of E2E. Cluster:SC256. Workload:E2E, HEDGEFUND_E2E, MUSTANG_E2E**

3Sigma is particularly adept at minimizing SLO misses, our primary objective, and completing more useful work, approaching `PointPerfEst` and significantly outperforming the non-hypothetical systems. 3Sigma performs well, despite not having the luxury of perfect job runtime knowledge afforded to `PointPerfEst`. It uses historical runtime distributions to make informed decisions, such as whether to start a job early to give ample time for it to complete before its deadline, or to be optimistic and schedule the job closer to the deadline. However, 3Sigma is not perfect. It misses a few more SLO job deadlines than `PointPerfEst`, and it completes fewer best-effort jobs because 3σSched preempts more best-effort jobs to make additional room for SLO jobs for which the distribution indicates a wider range of possible runtimes for a job . BE latency is similar across all system.

`PointRealEst` exhibits much higher SLO miss rates (18%, or 4.0X higher than 3Sigma), and lower goodput (5.4% lower than 3Sigma), because previous approaches struggle with realistic prediction error profiles. Because `PointRealEst` schedules based on only point estimates (instead of complete runtime distributions) and lacks an explicit overestimate handling policy, it makes less informed decisions and struggles to handle difficult-to-estimate runtimes (e.g., due to greater variance for a job type). For underestimated SLO jobs (that ran shorter in the past on average), `PointRealEst` is often too optimistic and starts the job later than it should. For overestimated SLO jobs, `PointRealEst` is often too conservative, neglecting to schedule SLO jobs which are predicted to not finish in time, even if cluster resources are available.

`Prio` misses 12% of SLO job deadlines (2.3x more than 3Sigma). It does not take advantage of any runtime information, thereby missing opportunities to wait for preferred resources or exploit one job's large deadline slack to start a tighter deadline job sooner. `Prio` is better than `PointRealEst` in terms of SLO misses but much worse in BE goodput, as it always prioritizes SLO jobs at the expense of increased preemption of BE jobs, even when deadline slack makes preemption unnecessary. When the runtime is over-estimated, `PointRealEst` may not even attempt to run a job thinking that it would not complete in time, while `Prio` will always attempt to schedule any SLO jobs if there are enough resources.

Simulator experiments. We validate our simulation setup (SC256) by running the identical workload to that in experiment in Fig. 6. Similar trends are observed across all our systems and success metrics. Table 2 shows the small differences observed for the 12 bars shown in Fig. 6.



(a) SLO Miss (b) SLO Goodput (c) BE Goodput

Figure 8: Attribution of Benefit. The lines representing 3Sigma with individual techniques disabled— demonstrating that all are needed to achieve the best performance. The workload is E2E with a constant deadline slack. Cluster:SC256 Workload:DEADLINE-*n* where $n \in [20, 40, 60, 80, 100, 120, 140, 160, 180]$

Performance comparison varying workload. Fig. 7 summarizes the performance of the scheduling systems under three different workloads. We observe that the overall behavior of the schedulers is similar to our observations in §6.1. For all workloads, 3Sigma outperforms PointRealEst and Prio, while approximately matching the performance of PointPerfEst. Surprisingly, for the HedgeFund and Mustang workloads, 3Sigma slightly outperforms PointPerfEst. This is possible because, while PointPerfEst does receive perfect runtime knowledge as jobs arrive, it does not possess knowledge of future job arrivals (nor do any of the other systems). Consequently, it may make sub-optimal scheduling decisions, such as starting a SLO job late and not leaving sufficient resources for future arrivals. 3Sigma also does not possess knowledge of future job arrivals, but it tends to start SLO jobs earlier than PointPerfEst when the distribution suggests likelihood of a runtime longer than the actual runtime.

We also observe that PointRealEst performs poorly on SLO miss rate across different workloads. Further, miss-rate is only slightly better for Mustang. This is surprising, as a much larger portion (compared to other workloads) of jobs in Mustang have very accurate point estimates (Fig. 2(a)). We believe PointRealEst still performs poorly as a small number of the estimates are off by a large margin, adversely affecting the ability of the scheduler to make informed decision. But, many of the mis-estimates are associated with small jobs; consequently, PointRealEst and Prio are able to provide high goodput despite having high SLO miss-rates.

6.2 Attribution of benefit

3σSched introduces distribution-based scheduling and adaptive overestimate handling to robustly address the effects of runtime uncertainty. This section evaluates the individual contributions of these techniques. Fig. 8 shows performance as a function of deadline slack for 3Sigma, PointPerfEst, PointRealEst, and three versions of 3Sigma, each with a single technique disabled: 3SigmaNoDist uses point estimates instead of distributions, 3SigmaNoOE turns off the overestimate handling policy, and 3SigmaNoAdapt turns off just the adaptive aspect of the policy and uses maximum overestimate handling for every job.

When the scheduler explicitly handles overestimates (compare 3SigmaNoDist to PointRealEst), SLO miss rate decreases because over-estimated SLO jobs are optimistically allowed to run, rather than discarding them as soon as they appear to not have enough time to finish before the deadline. However, SLO miss rate for 3SigmaNoDist is still high, because the lack of distribution awareness obscures which jobs are more likely to succeed if tried; therefore, 3SigmaNoDist wastes resources on SLO jobs that won't finish in time.

Simply using distribution-based scheduling (see, e.g., 3SigmaNoOE) drops SLO miss rate to the level of PointPerfEst for most deadline slacks. By considering the variance of job runtimes, the scheduler can conservatively schedule jobs with uncertain runtimes and optimistically attempt jobs that are estimated to have a non-zero probability of completion.

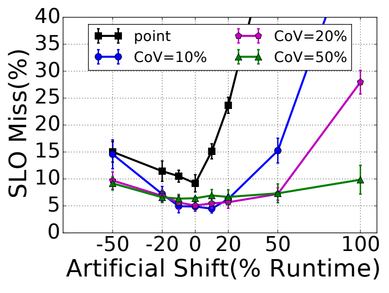
Blindly turning on overestimate handling decreases SLO miss rates at the lowest deadline slacks (3SigmaNoAdapt). However, 3SigmaNoAdapt is overly optimistic— even attempting jobs that would seem impossible given their historical runtimes— provided there are enough resources for SLO jobs in the cluster. This over-optimism results in lower BE goodput relative to 3Sigma's adaptive approach of enabling overestimate handling only for a small proportion of the jobs whose distributions indicate likely success.

6.3 Distribution-based scheduling benefits

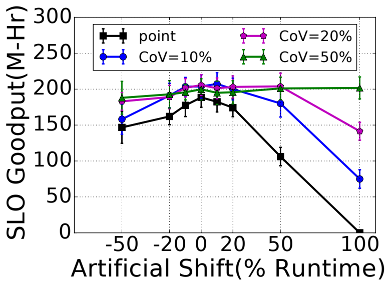
This section explores the robustness of 3Sigma to perturbations of the runtime distribution. In this study, for each job drawn from the E2E workload, we provide 3σSched with a

Metric(unit)	Δ SLO miss(%)	Δ goodput(M-Hr)	Δ BE latency(s)
PointPerfEst	0.6784	25.27	7.282
3Sigma	0.2875	27.10	11.08
PointRealEst	2.025	22.83	2.383
Prio	1.853	19.83	12.07

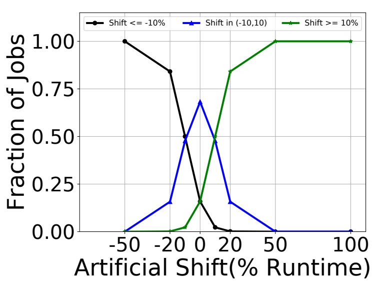
Table 2: Absolute performance difference between real and simulation experiments. Workload:E2E.



(a) SLO Miss rate

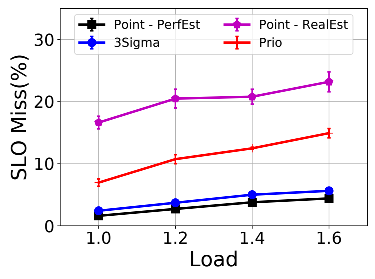


(b) SLO Goodput

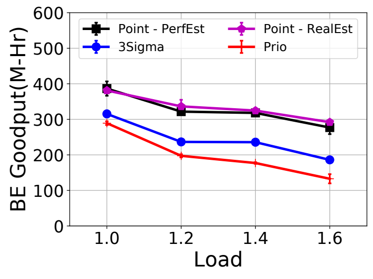


(c) Shift Profile

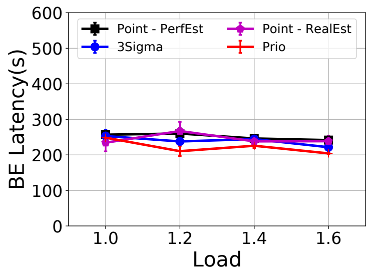
Figure 9: 3Sigma's performance when artificially varying runtime distribution shift (x-axis) and width (Coefficient of Variation curves). The runtime distribution provided to the scheduler is $\sim \mathcal{N}(\mu = job\_runtime * (1 + \frac{x}{100}), \sigma = job\_runtime * CoV)$. Each trace consists of jobs that are either within 10% accuracy or under- or over-estimates jobs. The group of jobs achieves a target average artifical shift. (c) shows the breakdown of these job types for each artificial shift value. Distribution-based schedulers always outperforms the point estimate-based scheduler. Tighter distributions perform better than wider distribution with a smaller artificial shift, but wider distributions are better with a larger artificial shift. The workload is 2 hrs in length. Cluster:SC256. Workload:E2E



(a) SLO Miss-rate



(b) BE Goodput



(c) Best Effort latency

Figure 10: 3Sigma outperforms others on SLO misses for a range of loads, matching `PointPerfEst` closely. All systems prioritize SLO jobs by sacrificing BE jobs when load spikes. Cluster:SC256, Workload: E2E-LOAD-ℓ where $\ell \in [1.0, 1.2, 1.4, 1.6]$

synthetically generated distribution instead of the distribution produced by 3σPredict.

We adjust the synthetic distributions in two dimensions, corresponding to an off-center mean and different variances. The former is realized by artificially shifting the entire distribution by an amount equal to a selected percent difference between the mean of the distribution and the actual runtime. The latter is represented by the CoV, which refers to the ratio of standard deviation to the actual runtime of the job. For each job, the artificial distribution is $\sim \mathcal{N}(\mu = job\_runtime * (1 + shift), \sigma = job\_runtime * CoV)$, where the shift itself is $\sim \mathcal{N}(\mu = shift, \sigma = 0.1)$.

Fig. 9 shows the results. Comparing point estimates (`point`) and distribution estimates, we observe that it is strictly better to use distribution estimates (`CoV=x%`) than to use point estimates (`point`) for scheduling jobs. Even at an artificial shift= 0.0, where $\approx$ 70% of estimates are generally accurate (within $\pm$10% error), using a distribution yields 2X fewer SLO misses compared to the point estimates. Hence, even a small proportion of jobs with inaccurate estimates can cause the scheduler to make mistakes and miss the opportunity to finish more jobs on time. Comprehending entire distributions enables the scheduler to reason about uncertainty in runtimes.

Furthermore, for small artificial shifts (within $\pm$20%), it is better to have narrower distributions with a smaller CoV. This is because a wider distribution indicates greater likelihood of runtimes that are much shorter and much larger than the actual runtime. The scheduler is more likely to incorrectly make risky decision to start some jobs later than it should and make overly conservative decisions for other jobs.

However, if the actual runtime is far away from the center of the runtime distribution (larger artificial shift), wider distributions provide a benefit. As the distribution widens, the scheduler correctly assigns higher expected utility to scenarios that hedge the risk of runtimes being farther away from the mean. On the other hand, narrower distributions suffer more as the artificial shift deviates further from zero. The likelihood of the job running for the actual runtime decreases significantly, and causes the scheduler to discount the placement options that hedge the associated risks.

6.4 Sensitivity analyses

Sensitivity to deadline slack. Fig. 8 shows performance as a function of deadline slack. We make two additional observations. First, smaller slack makes it harder to meet SLOs across all policies, due to increased contention for cluster space-time, leading to higher SLO miss rates. Second, best effort goodput decreases for all systems, but for different reasons. As slack increases, `PointPerfEst` sees more wiggle room for placement and tries (and completes) more difficult larger SLO jobs. Since the schedule is optimally packed, it needs to bump best effort jobs in order to schedule more SLO jobs. BE goodput of `PointRealEst` shows similar trends; `PointRealEst` tries more over-estimated jobs, since increasing slack reduces the number of seemingly impossible jobs. 3Sigma, on the other hand, was already trying most completable overestimated jobs, so it sees the smallest decrease in BE goodput. More of the SLO jobs succeed though.

Sensitivity to load. Fig. 10 shows performance as a function of load. As load increases, we observe an increase in all systems' SLO miss rates due to increased contention for cluster resources. The relative effectiveness of `PointPerfEst`and the three realistic scheduling approaches is consistent across the range. We observe that as the load increases, all systems increasingly prioritize SLO jobs, decreasing BE goodput. The gap between the BE goodputs of `PointPerfEst` and `3Sigma` widens as `3Sigma` makes more room for each incoming SLO job to address its uncertainty about runtimes.

Sensitivity to sample size. Another concern may be: how is the performance of the scheduler affected by the number of samples observed per feature (user, job names, etc.)? To answer this question, we used another modified version of the E2E workload where we controlled the number of samples comprising the distributions used by 3Sigma, drawing those samples from the original distributions. We also created a version of `PointRealEst` where the point estimates were derived from the observed samples. In Fig. 11, we vary the number of samples used from 5 to 100. We observe that increasing the number of samples from 5 to 25 significantly improved performance (for both schedulers), but by 25 samples, the performance of 3Sigma converges to the performance of `PointPerfEst`. 3Sigma outperforms `PointRealEst`at each point and benefits more from additional instances, since it uses the distribution rather than just the mean. Naturally, `PointPerfEst` and `Prio` are not affected.

6.5 Scalability

This section shows that 3Sigma can handle the additional complexity from distribution-based scheduling even while managing more than 12500 nodes and a job submission rate comparable to the heaviest load observed in the Google cluster trace (3668 jobs per hour).

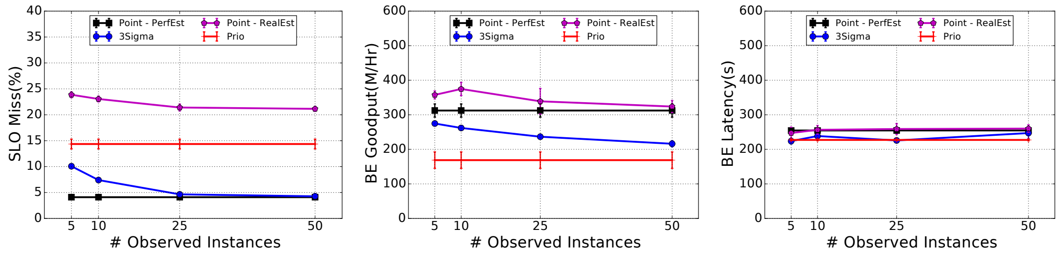
3Sigma requires more CPU time to make decisions than not using runtime estimates (e.g., `Prio`), which can affect scheduler scalability. Although previous work [4, 28] has shown that packing cluster space-time using runtime estimates can be sufficiently efficient for 100s to 1000s of nodes, 3Sigma adds sources of overhead not evaluated in such previous work: (1) latency of 3σPredict at Job Submission (I/O and computation for looking up the correct group of jobs in the runtime history database and generating distribution), (2) latency from additional computation (e.g. computing expected utility and expected resource consumption) to formulate the bin-packing problem, and (3) increased solver runtime due to increased complexity of the bin-packing problem at 3σSched.

In this experiment, 3Sigma schedules microbenchmark workloads, SCALABILITY-*n*. Each workload consists of *n* jobs per hour for 5 hours. The ratio of tasks to job matches those observed in the Google cluster trace. The load is set to 0.95. Even under these conditions, the latency of producing distributions at 3σPredict is negligible (maximum=14ms) compared to the job runtimes in the trace. 3σPredict maintains minimal state for each group of jobs, so the cost of data retrieval is low. Similar latency is observed for producing point estimates, since most of the work is the same (accessing histories and choosing among them).

We also compare the performance of `PointRealEst` and 3Sigma in Fig. 12. Fig. 12(a) depicts the runtime of each scheduling cycle, including generation of scheduling options, evaluation, formulation of the optimization problem, and execution of the solver. Fig. 12(b) reports the runtime of the solver. For both systems, the solver execution is a non-trivial fraction of the scheduling cycle runtime. We observe that distribution-based scheduling also results in a moderate increase in worst-case solver time. As noted in §4.3, distribution-based scheduling induces a moderate increase in the number of constraint terms but does not change the number of decision variables. Also note that the actual impact on the solver runtime is upper-bounded by a a solver timeout parameter, so the impact of solving on scheduling latency is bounded.

7 SUMMARY

3Sigma's use of distributions instead of point estimates allows it to exploit job runtime history robustly. Experiments with trace-derived workloads both on a real 256-node cluster and in simulation demonstrate that 3Sigma's distribution-based scheduling greatly outperforms a state-of-the-art point-estimate scheduler, approaching the performance of a hypothetical scheduler operating with perfect runtime estimates.



(a) SLO Miss-rate (b) BE Goodput (c) BE latency

Figure 11: 3Sigma outperforms others on SLO Misses for a range of runtime variability. 3Sigma matches `PointPerfEst` **in terms of SLO misses at the sacrifice of Best Effort goodput. Cluster:SC256. Workload:E2E-SAMPLE-*n* where** $n \in [5, 10, 25, 50, 75, 100]$

ACKNOWLEDGMENTS

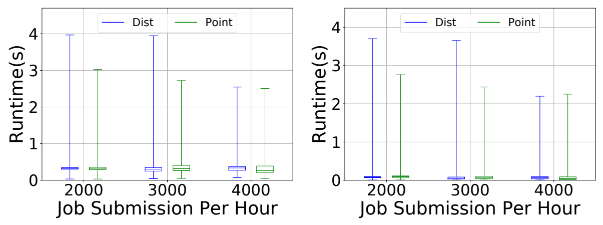
We thank the members and companies of the PDL Consortium: Broadcom, Ltd., Dell EMC, Facebook, Google, Hewlett-Packard Labs, Hitachi Ltd., Intel Corporation, IBM, Microsoft Research, MongoDB, NetApp, Inc., Oracle Corporation, Salesforce, Samsung Information Systems America, Seagate Technology, Toshiba, Two Sigma, Veritas and Western Digital for their interest, insights, feedback, and support. We also would like to thank Los Alamos National Laboratory for their data, feedback, and support. This research is supported in part by a Samsung Scholarship. This research is also supported in part by DHS Award HSHQDC-16-3-00083, NSF CISE Expeditions Award CCF-1139158, and gifts from Alibaba, Amazon Web Services, Ant Financial, CapitalOne, Ericsson, GE, Google, Huawei, Intel, IBM, Microsoft, Scotiabank, Splunk and VMware.

REFERENCES

[1] Yael Ben-Haim and Elad Tom-Tov. 2010. A streaming parallel decision tree algorithm. *Journal of Machine Learning Research* 11, Feb (2010), 849–872.

[2] Eric Boutin, Jaliya Ekanayake, Wei Lin, Bing Shi, Jingren Zhou, Zhengping Qian, Ming Wu, and Lidong Zhou. 2014. Apollo: Scalable and Coordinated Scheduling for Cloud-Scale Computing. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. USENIX Association, Broomfield, CO, 285–300. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/boutin

[3] Yun Chi, Hakan Hacígümüş, Wang-Pin Hsiung, and Jeffrey F Naughton. 2013. Distribution-based query scheduling. *Proceedings of the VLDB Endowment* 6, 9 (2013), 673–684.



(a) Scheduling Cycle (b) Solver

Figure 12: 3Sigma scalability as a function of job submission per hour. Cluster: GOOGLE, Workload: SCALABILITY-*n* where $n \in [2000, 3000, 4000]$

[4] Carlo Curino, Djellel E. Difallah, Chris Douglas, Subru Krishnan, Raghu Ramakrishnan, and Sriram Rao. 2014. Reservation-based Scheduling: If You're Late Don't Blame Us!. In *Proceedings of the ACM Symposium on Cloud Computing (SOCC '14)*. ACM, New York, NY, USA, Article 2, 14 pages. https://doi.org/10.1145/2670979.2670981

[5] Pamela Delgado, Florin Dinu, Anne-Marie Kermarrec, and Willy Zwaenepoel. 2015. Hawk: Hybrid Datacenter Scheduling. In *Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC '15)*. USENIX Association, Berkeley, CA, USA, 499–510. http://dl.acm.org/citation.cfm?id=2813767.2813804

[6] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: Resource-efficient and QoS-aware Cluster Management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '14)*. ACM, New York, NY, USA, 127–144. https://doi.org/10.1145/2541940.2541941

[7] Andrew D. Ferguson, Peter Bodik, Srikanth Kandula, Eric Boutin, and Rodrigo Fonseca. 2012. Jockey: guaranteed job latency in data parallel clusters. In *Proc. of the 7th ACM european conference on Computer Systems (EuroSys '12)*. 99–112. https://doi.org/10.1145/2168836.2168847

[8] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. 2015. Multi-resource packing for cluster schedulers. *ACM SIGCOMM Computer Communication Review* 44, 4 (2015), 455–466.

[9] Robert Grandl, Mosharaf Chowdhury, Aditya Akella, and Ganesh Ananthanarayanan. 2016. Altruistic Scheduling in Multi-Resource Clusters.. In *OSDI*. 65–80.

[10] Mike Hibler, Robert Ricci, Leigh Stoller, Jonathon Duerig, Shashi Guruprasad, Tim Stack, Kirk Webb, and Jay Lepreau. 2008. Large-scale Virtualization in the Emulab Network Testbed. In *USENIX 2008 Annual Technical Conference (ATC'08)*. USENIX Association, Berkeley, CA, USA, 113–128. http://dl.acm.org/citation.cfm?id=1404014.1404023

[11] Mohammad Islam, Angelo K Huang, Mohamed Battisha, Michelle Chiang, Santhosh Srinivasan, Craig Peters, Andreas Neumann, and Alejandro Abdelnur. 2012. Oozie: Towards a Scalable Workflow Management System for Hadoop. In *SWEET Workshop*.

[12] Virajith Jalaparti, Peter Bodik, Ishai Menache, Sriram Rao, Konstantin Makarychev, and Matthew Caesar. 2015. Network-Aware Scheduling for Data-Parallel Jobs: Plan When You Can. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*. ACM, New York, NY, USA, 407–420. https://doi.org/10.1145/2785956.2787488

[13] Sangeetha Abdu Jyothi, Carlo Curino, Ishai Menache, Shravan Matthur Narayanamurthy, Alexey Tumanov, Jonathan Yaniv, Ruslan Mavlyutov, Íñigo Goiri, Subru Krishnan, Janardhan Kulkarni, et al. 2016. Morpheus: towards automated SLOs for enterprise clusters. In *Proceedings*

of the 12th USENIX conference on Operating Systems Design and Implementation. USENIX Association, 117–134.

[14] S. Krishnaswamy, S. W. Loke, and A. Zaslavsky. 2004. Estimating computation times of data-intensive applications. *IEEE Distributed Systems Online* 5, 4 (April 2004). https://doi.org/10.1109/MDSO.2004.1301253

[15] Shuo Liu, Gang Quan, and Shangping Ren. 2010. On-line Scheduling of Real-time Services for Cloud Computing. In *Services (SERVICES-1), 2010 6th World Congress on.* IEEE.

[16] Kristi Morton, Abram Friesen, Magdalena Balazinska, and Dan Grossman. 2010. Estimating the progress of MapReduce pipelines. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on.* IEEE, 681–684.

[17] I. A. Moschakis and H. D. Karatza. 2011. Performance and cost evaluation of Gang Scheduling in a Cloud Computing system with job migrations and starvation handling. In *Computers and Communications (ISCC), 2011 IEEE Symposium on.* 418–423. https://doi.org/10.1109/ISCC.2011.5983873

[18] John K Ousterhout. 1982. Scheduling Techniques for Concurrent Systems. In *International Conference on Distributed Computing Systems (ICDCS)*, Vol. 82. 22–30.

[19] Kaushik Rajan, Dharmesh Kakadia, Carlo Curino, and Subru Krishnan. 2016. PerfOrator: Eloquent Performance Models for Resource Optimization. In *Proceedings of the Seventh ACM Symposium on Cloud Computing (SoCC '16).* ACM, New York, NY, USA, 415–427. https://doi.org/10.1145/2987550.2987566

[20] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. 2012. Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis. In *Proc. of the 3nd ACM Symposium on Cloud Computing (SOCC '12).*

[21] Charles Reiss, Alexey Tumanov, Gregory R Ganger, Randy H Katz, and Michael A Kozuch. 2012. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing.* ACM, 7.

[22] Jennifer M. Schopf and Francine Berman. 1999. Stochastic scheduling. In *SC '99 Proceedings of the 1999 ACM/IEEE conference on Supercomputing.* ACM.

[23] Bikash Sharma, Victor Chudnovsky, Joseph L. Hellerstein, Rasekh Rifaat, and Chita R. Das. 2011. Modeling and synthesizing task placement constraints in Google compute clusters. In *Proc. of the 2nd ACM Symposium on Cloud Computing (SOCC '11).* ACM, Article 3, 14 pages. https://doi.org/10.1145/2038916.2038919

[24] Warren Smith, Ian T. Foster, and Valerie E. Taylor. 1998. Predicting Application Run Times Using Historical Information. In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing.* IEEE.

[25] Roshan Sumbaly, Jay Kreps, and Sam Shah. 2013. The Big Data Ecosystem at LinkedIn. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD).*

[26] Dan Tsafrir, Yoav Etsion, and Dror G. Feitelson. 2007. Backfilling Using System-Generated Predictions Rather than User Runtime Estimates. In *IEEE Transactions on Parallel and Distributed Systems.* IEEE.

[27] Alexey Tumanov, Angela Jiang, Jun Woo Park, Michael A. Kozuch, and Gregory R. Ganger. 2016. *JamaisVu: Robust Scheduling with Auto-Estimated Job Runtimes.* Technical Report CMU-PDL-16-104. Carnegie Mellon University.

[28] Alexey Tumanov, Timothy Zhu, Jun Woo Park, Michael A. Kozuch, Mor Harchol-Balter, and Gregory R. Ganger. 2016. TetriSched: Global Rescheduling with Adaptive Plan-ahead in Dynamic Heterogeneous Clusters. In *Proceedings of the Eleventh European Conference on Computer Systems (EuroSys '16).* ACM, New York, NY, USA, Article 35, 16 pages. https://doi.org/10.1145/2901318.2901355

[29] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, , Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. 2013. Apache Hadoop YARN: Yet Another Resource Negotiator. In *Proc. of the 4th ACM Symposium on Cloud Computing (SOCC '13).*

[30] S. Verboven, P. Hellinckx, F. Arickx, and J. Broeckhove. 2008. Runtime Prediction Based Grid Scheduling of Parameter Sweep Jobs. In *Asia-Pacific Services Computing Conference, 2008. APSCC '08. IEEE.* 33–38. https://doi.org/10.1109/APSCC.2008.189

[31] Abhishek Verma, Ludmila Cherkasova, and Roy H. Campbell. 2011. ARIA: Automatic Resource Inference and Allocation for Mapreduce Environments. In *Proceedings of the 8th ACM International Conference on Autonomic Computing (ICAC '11).* ACM, New York, NY, USA, 235–244. https://doi.org/10.1145/1998582.1998637

[32] A. Verma, M. Korupolu, and J. Wilkes. 2014. Evaluating job packing in warehouse-scale computing. In *2014 IEEE International Conference on Cluster Computing (CLUSTER).* 48–56. https://doi.org/10.1109/CLUSTER.2014.6968735

[33] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-scale Cluster Management at Google with Borg. In *Proceedings of the Tenth European Conference on Computer Systems (EuroSys '15).* ACM, New York, NY, USA, Article 18, 17 pages. https://doi.org/10.1145/2741948.2741964

[34] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. 2002. An Integrated Experimental Environment for Distributed Systems and Networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation.* USENIX Association, Boston, MA, 255–270.

[35] Timothy Wood, Prashant J Shenoy, Arun Venkataramani, and Mazin S Yousif. 2007. Black-box and Gray-box Strategies for Virtual Machine Migration.. In *NSDI*, Vol. 7. 17–17.

[36] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. 2010. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European conference on Computer systems (Eurosys).* ACM, 265–278.