3Sigma: distribution-based cluster scheduling for runtime uncertainty

Jun Woo Park[*], Alexey Tumanov[‡], Angela Jiang[*]
Michael A. Kozuch[†], Gregory R. Ganger[*]
Carnegie Mellon University[*], UC Berkeley[‡], Intel Labs[†]

CMU-PDL-17-107

November 2017

Parallel Data Laboratory
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Abstract

The 3Sigma cluster scheduling system uses job runtime histories in a new way. Knowing how long each job will run allows 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 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 most job runtimes 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 history, and explicitly creates plans that mitigate the effects of anticipated runtime uncertainty. Experiments with workloads derived from the same traces show that 3Sigma approaches the end-to-end performance of a hypothetical perfect predictor, and greatly outperforms a state-of-the-art scheduler using point estimates from a state-of-the-art predictor. 3Sigma reduces SLO miss rate, increases cluster goodput, and improves or matches latency for best effort jobs.

Acknowledgements: We thank the member companies of the PDL Consortium (Broadcom, Dell EMC, Facebook, Google, Hewlett-Packard Labs, Hitachi, Intel, Microsoft Research, MongoDB, NetApp, Oracle, Salesforce, Samsung, Seagate Technology, Two Sigma, Toshiba, Veritas, Western Digital) for their interest, insights, feedback, and support. This research is supported in part by Intel as part of the Intel Science and Technology Center for Cloud Computing (ISTC-CC), by an NSERC Postgraduate Fellowship, by a Samsung Scholarship, and by National Science Foundation under awards CSR-1116282, 0946825 and CNS-1042537, CNS-1042543 (PRObE).

Keywords: cluster scheduling, cloud systems

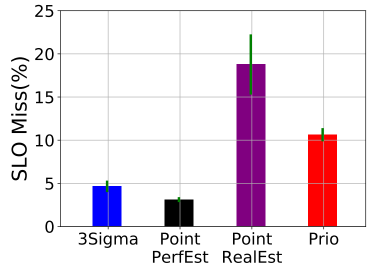


Figure 1: Comparison of 3Sigma with three other scheduling approaches w.r.t. SLO (deadline) miss rate, for a mix of SLO and best effort jobs derived from the Google cluster trace [18] on a 256-node cluster. (Details in Sec. 5) 3Sigma, despite estimating runtime distributions online with imperfect knowledge 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 Sec. 2.2). The value of exploiting runtime information, when done well, is confirmed by comparison to a conventional priority-based approach (`Prio`).

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 [18]. 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 [3, 12, 26]. With it, a scheduler can pack jobs more aggressively in a cluster's resource assignment plan [3, 12, 26, 29], 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 [26, 2]. Exploiting job runtime knowledge leads to better, more robust scheduler decisions than relying on hard-coded assumptions.

In most cases, the job runtime estimates come from 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 the estimates are accurate, schedulers relying on them outperform those using other approaches. Previous research [26] has also shown that these schedulers are robust to a reasonable degree of runtime variation (e.g., up to 50%).

Surprisingly, we find that the estimate errors, albeit expected in large, multi-use clusters, cover an unexpectedly larger range. Applying a state-of-the-art ML-based predictor [25] to three real-world traces, including the well-studied Google cluster trace [18] 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 actual runtime and most much closer). But, 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.[1]

This paper describes the 3Sigma cluster scheduling system, which uses all of the relevant runtime

[1] We also obtained a copy of the recent TetriSched system [26] and confirmed that it suffers the same negative effects from the estimate error profiles we observe with the real traces.

history for each job instead of just a point estimate derived from it. It associates and 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 indicates that it might be better served by more conservative start times (e.g., relative to a deadline). 3Sigma similarly exploits the runtime distribution to adaptively address a significant problem with point over-estimates, by deciding when to give up on 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 a state-of-the-art point-estimate-based scheduler and runtime-unaware priority scheduling, especially for mixes of deadline-oriented jobs and latency-sensitive jobs on heterogeneous resources. 3Sigma *simultaneously* provides higher SLO attainment for deadline-oriented jobs and increases 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 esimates including some large outliers. Second, it describes a solution based on using full runtime distributions and a scheduling system (3Sigma) that does so successfully. Third, it describes new core scheduler mechanisms, implemented in 3Sigma, needed to make distribution-based scheduling efficient and scalable enough 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 [18, 29, 21]. One result 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.

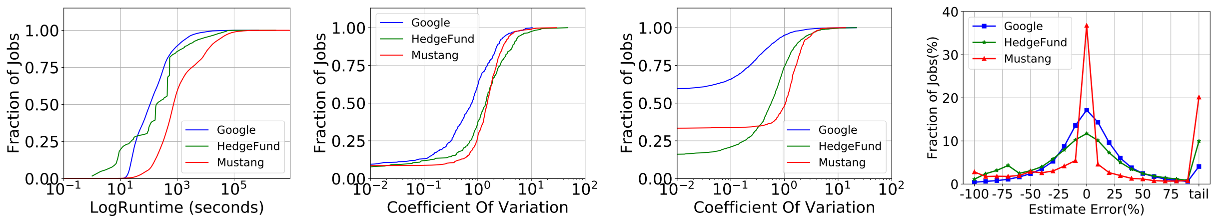
1) Cluster workloads are increasingly a mixture of business-critical production jobs and best-effort engineering/analysis jobs. The production jobs, often submitted by automated systems [10, 23], tend to be resource-heavy and to have strict completion deadlines [3, 12]. The best-effort jobs, such as 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 [3, 12, 26].

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 [26, 2, 34].

3) Many parallel computations can only run when all tasks comprising them are initiated and executed simultaneously (Gang-scheduling) [16, 15]. Maximizing resource utilization while arranging for such bulk resource assignments is easier when job runtimes are known.

Thus, many recent systems [3, 26, 7, 12, 8] 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 are outside a relatively small error range. Worse, we find that this should be expected in many environments.

2.1 Inherent variation and uncertainty in runtimes



(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

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 Sec. 5): (1) 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 well-known Google cluster trace[19] 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 un-predictability 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 obtained a copy of and use JVuPredict, the runtime predictor module from the recent JamaisVu [25] 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.[2]

Fig. 2(d) is the histogram of percent estimate error. For all workloads, most job runtimes are estimated reasonably (e.g., $\pm 25\%$ 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

[2] Smith et al. [22] describe a similar scheme and its effectiveness for parallel computations.

works [26, 8]. Even for the Mustang workload, which has large proportion of jobs with very accurate ($\pm 5\%$ 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. [31, 12]) 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 [12] 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.

Preemption can be applied to address some issues arising from mis-estimates, like it is used in many systems to re-assign resources to new high-priority jobs, either by killing (e.g., in container-based clusters [31]) or migrating (e.g., in VM-based systems [33]) jobs.

Various other heuristics have been used to mitigate the effects of mis-estimates. [24] addresses mis-estimations of runtimes for HPC workloads by exponentially increasing under-estimated runtimes and then reconsidering scheduling decisions. The "stochastic scheduler" [20] uses a conservative runtime estimate by padding the observed mean by one or more standard deviations. Such heuristics help, and 3Sigma borrows the first, but do not eliminate the problem.

2.3 A case for distribution-based scheduling

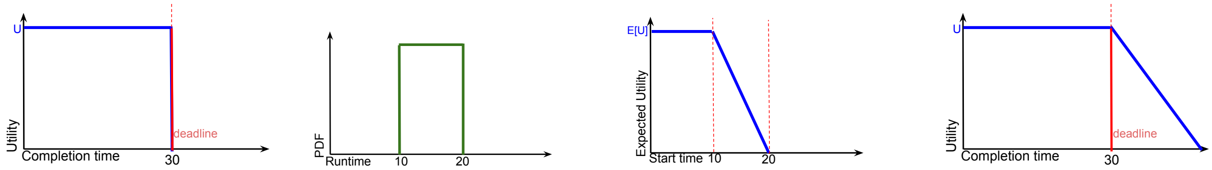
Are estimates of job runtime distributions more valuable than point estimates (e.g., estimates of the average job runtime) for cluster scheduling?

One's intuition may be to say "yes," if for no other reason than that the distribution provides strictly more information to the scheduler than the point estimate. However, a simple thought experiment might deepen our confidence in this response. Imagine a very simple case: two jobs arrive to be scheduled on the cluster, and the resources available are such that only one job may be scheduled at a time. Further, one job is an SLO job with a deadline 15 minutes into the future and the other is a best-effort (BE) job. The objective of the scheduler is to never miss the deadline of a SLO job, while also minimizing the latency of BE jobs. The key question is then: which job should be run 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;



(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 sec. 4.2, the overestimate handling technique involves modifying the utility function (a) associated with the job with an extended version illustrated in (d).

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 is 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—probably from historical job runtime data—and may differ from observed behavior. We address both of these topics later in the paper.

3 Distribution-based Scheduling

In this section, we describe the mechanisms that enable schedulers to use the full runtime distribution, as opposed to runtime 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*, *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 job runtimes estimates, we can find the best schedule using basic optimization techniques (e.g. MILP). However, with runtimes 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. 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*, *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:

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

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 Estimating resource consumption of running jobs

To calculate the set of available resources over time, we need to estimate the resource usage of currently running jobs. Using point estimates, we assume that the usage is deterministic. Using full distributions, we acknowledge that the usage is probabilistic. Analogous to utility, we calculate and use expected resource consumption.

Similarly to expected utility, the expected resource consumption of a job at time t is the weighted average of resource consumption observed in the runtime distribution. It is defined as $1-CDF(t)$ per each unit of resource, where the CDF is the cumulative density function of the job runtime.

To calculate the available resources of the cluster, 3σSched tracks the expected "remaining resource consumption" (RRC) for each running job in the cluster. This metric is an up-to-date estimate of the remaining resource consumption of running jobs. The RRC provides more information than the original expected resource consumption by accounting for the fact that the job has run at least *elapsed* time so far. It is calculated by recomputing the weighted average of resource consumption using a distribution that only includes runtimes longer than $elapsed$.

$$E[RRC(\text{t},\text{elapsed})] = \frac{1-CDF(\text{t})}{1-CDF(\text{elapsed})}$$

The remaining capacity of cluster at time t is computed by subtracting E[RRC] at time t for all jobs from full cluster capacity.

4 3Sigma Design and Implementation

This section describes the architecture and key components of 3Sigma (Fig. 4). 3Sigma replaces the scheduling component of a cluster manager, such as Hadoop 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 amount of resources requested.

The role of the predictor component, 3σPredict, is to provide the core scheduler with the probability distribution of the execution time of the submitted job. 3σPredict (Sec. 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 from that set of completed jobs (Step 2 of Fig. 4).

Given such a distribution of expected job runtimes and request specifications, the core scheduler, 3σSched decides *on which resources to place which jobs and when*. To achieve this, the scheduler first evaluates the expected utility of each option (described in Sec. 3.1) and computes the amount of resources that will become available in the future (Sec. 3.2). Job valuations and computed resource capacity are then compiled into an optimization problem (Sec. 4.3), which is sent to an external solver to be solved. 3σSched translates the solution into an updated schedule and submits the schedule to the cluster manager (Step 3 of Fig. 4).

When a job completes, its actual runtime is recorded by 3σPredict (along with the attribute information from the job) to be incorporated into the job history for future predictions (Step 4 of Fig. 4).

In this section, we detail how 3σPredict estimates runtime distributions and how 3σSched copes with mis-estimation. We also describe how 3σSched automatically generates the optimization problem used for scheduling.

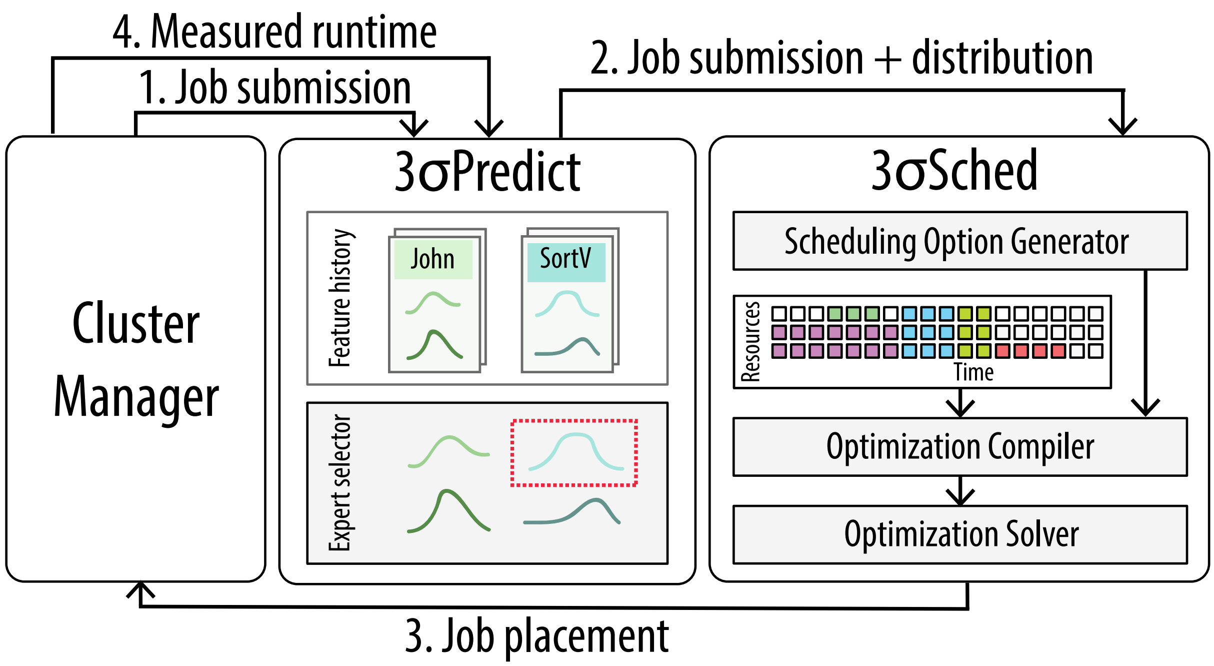


Figure 4: End-to-end system integration

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 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 feature. 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 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.

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 [24]. 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.

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.

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 Optimization problem formulation

3σSched solves an MILP problem to decide where and when a job will be run. The MILP is automatically generated by 3σSched. Its inputs are the job submission, runtime distribution and utility curve of both pending and currently running jobs. This section describes how 3σSched uses this information to generate the optimization problem.

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

The objective of the optimization problem is to maximize utility. There are multiple placement options o, per pending job j, each of which is associated with a expected utility U_{jo}. The solution is subject to demand constraints and capacity constraints. The demand constraints ensure that only one option of each job is chosen. The capacity constraints ensure that all scheduled jobs fit within the cluster capacity.

At the start of each scheduling cycle, 3σSched computes the estimated cluster capacity C_t at each time t. This is computed using the $RRC(t)$ of each running job in the cluster. The terms of the objective function and constraints are added iteratively with each pending job. The fully formulated problem is sent to the external solver to be solved. 3σSched retrieves the solution and sends the placement decisions to the cluster manager for jobs due to be started immediately. 3σSched repeats this process every scheduling interval.

Formulation. For completeness, we detail the formulation of the objective function and constraints. 3σSched uses one indicator variable I_{jo}, for each placement option o of each pending job j, to indicate whether the option is chosen. $I_{jo} = 1$ when the option o is selected for job j.

Objective function: The objective function $\sum_{j,o} U_{jo} I_{jo}$ maximizes the overall benefit across all selected options.

Demand constraint: The series of demand constraints for each job $\forall j \sum_o I_{jo} \leq 1$ ensures at most one option is selected for each job j.

Capacity constraint: The capacity constraint $\sum_{jo} kRC_j(t-s) I_{jo} \leq C_t$ ensures that allocations do not exceed the available capacity at time t. k is the amount of resources needed by this placement option, and s is the start time of this option.

Scalability. MILP is known to be an NP-hard problem. To minimize the excessive latency caused by solver, we apply a few optimizations. 1) 3σSched does not include terms with a zero constant, 2) only considers a bounded range of time into the future and 3) uses the best solution achieved within a timeout of 75% of its scheduling interval.

5 Experimental Setup

We conduct a series of end-to-end experiments and microbenchmarks to evaluate 3Sigma, integrated with Hadoop YARN [27]—a popular open source cluster scheduling framework. We find YARN's support for time-aware reservations and placement decisions and its popularity in the 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.

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 many 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 uses enhanced version of [26] and is representative of schedulers which rely on point estimates and techniques to handle imperfect estimates. This includes Rayon, TetriSched and Morpheus [26, 3, 12].

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 best-effort jobs rather than leveraging runtime information, which represent schedulers like Borg [31].

Workloads. The bulk of our experiments use workloads derived from the Google cluster trace [18]. We use a base Google-trace-derived workload (termed "E2E") for overall comparisons among schedulers as well as several workloads that vary individual workload characteristics (e.g., runtime variation for a job type or cluster load) to explore sensitivities. All workloads are 5 hours in length (1500 jobs) except for E2E which is 2 hours (600 jobs), used to expedite the experiment in RC256.

We use the following process to generate a workload representing the Google cluster trace that will fit on our available clusters (E2E). We first filter out all jobs larger than 256 nodes. We use k-means clustering on the remaining jobs' runtimes, associating each resulting cluster with a job class. We derive parameters for the distributions of the job attributes (e.g., runtime and number of tasks) and the probability mass function of the features in each job class. When generating the workload, we set the arrival time using an inter-arrival distribution (exponential distribution with c^2a=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 job-class distribution. Each workload consists of an even mixture of Service Level Objective (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 Sec. 6.1.1, 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[32, 9]. The workload consists of 2.1 million jobs submitted within a period of 61 months (2011 $\sim$ 2016).

Estimates. We pre-train 3σPredict before running experiments to produce steady-state estimates for 3Sigma and PointRealEst. For the Google workload, we use the original trace for pre-training. (In our experiments, we use the generated workload representing the Google trace, rather than the original trace.) 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. Deadline slack is computed by $(deadline - submissiontime - runtime)/runtime * 100$ (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%. The default values are much smaller than experimented in [26] (250% and 300%), matching the finding in [12] that tighter deadlines are also possible.

SLO miss rate	Fraction of SLO jobs that miss their deadline
Goodput	Total useful work measured in machine-hours consumed (M-hrs)
BE latency	Mean response time for best effort jobs

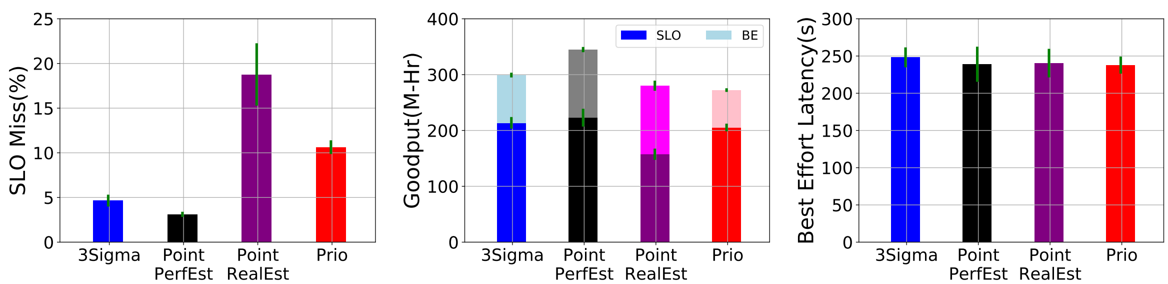
Table 2: Key goodness metrics measured.

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 [19]. 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 goodness metrics of Table 2 when comparing schedulers. Our primary goal is to minimize SLO miss rate: the percentage of SLO jobs that fail to complete before their deadline. We also want to measure the total work completed (BE goodput and the goodput of incomplete SLO jobs is not represented by the SLO miss rate). Therefore, we measure the goodput in machine-hours, showing how much aggregate work is being completed. Finally, we also measure mean BE latency.

6 Experimental Results



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

Figure 5: Compares the performance of 3Sigma with other systems in the real cluster. 3Sigma constantly outperforms `PointRealEst` strict priority scheduler on SLO miss rate and Goodput while nearly matching `PointPerfEst`. Cluster:RC256. Workload:E2E

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

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 3: Absolute performance deviation between RC256 real experiment and SC256 simulation. Workload: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, potentially 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.

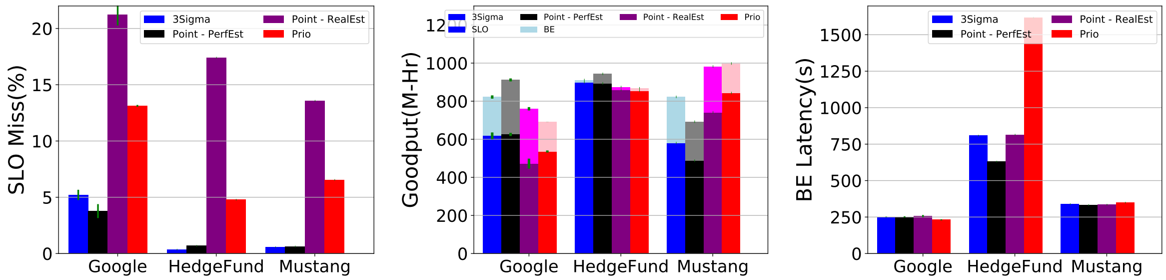
Simulator experiments. We validate our simulation setup (SC256) by running the identical workload to that in experiment in Fig. 5. Similar trends are observed across all our systems and success metrics. Table 3 shows the small differences observed for the 12 bars shown in Fig. 5.

6.1.1 Performance comparison varying workload

Fig. 6 summarizes the performance of all scheduling systems under three different workloads.

We observe that the overall behavior of the systems in comparison is similar to our observations in Sec. 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 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.

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 much a 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 large margin, adversely affecting the ability of the scheduler to make informed decision. However, many of the mis-estimated

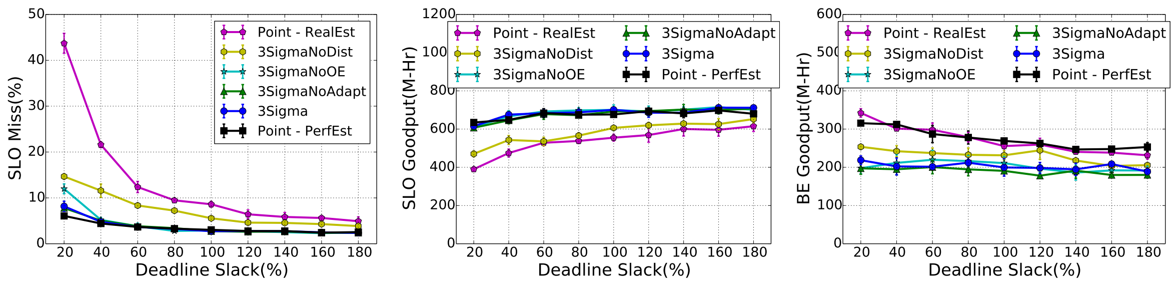


(a) SLO Miss rate (b) Goodput (c) BE latency

Figure 6: Compares the performance of 3Sigma with other systems under workloads from different environments in simulated cluster. 3Sigma constantly outperforms `PointRealEst` and strict priority scheduler 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

runtimes are associated with small jobs; consequently, `PointRealEst` and `Prio` are able to perform well on the goodput metric despite having high SLO miss rates.

6.2 Attribution of benefit



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

Figure 7: **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 $n \in [20, 40, 60, 80, 100, 120, 140, 160, 180]$

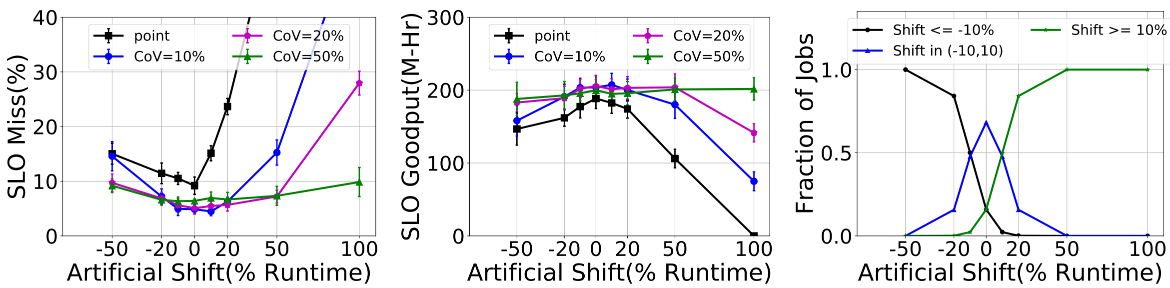
3σSched introduces distribution-based scheduling and adaptive overestimate handling to robustly address the effects of runtime uncertainty. This section evaluates the individual contributions of these techniques. Fig. 7 shows performance as a function of deadline slack for 3Sigma, `PointPerfEst`, `PointRealEst`, 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 (e.g., 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 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 Benefits of distribution-based scheduling



(a) SLO Miss rate (b) SLO Goodput (c) Shift Profile

Figure 8: 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

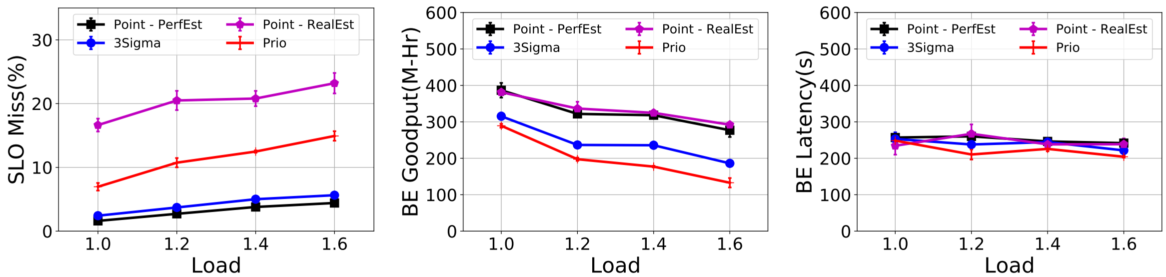
This section explores the robustness of 3Sigma to perturbations of the runtime distribution. In this study, for each job drawn from the E2E workload, we provide 3σSched with a synthetically generated runtime distribution instead of the distribution produced by 3σPredict.

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. 8 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 artificial shift$= 0.0$, where $\approx 70\%$ of estimates are generally accurate (within $\pm 10\%$ estimate error), using distribution estimates 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 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



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

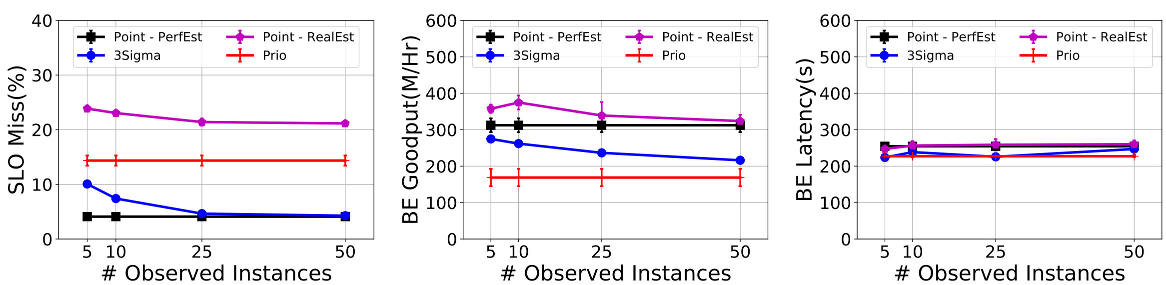
Figure 9: 3Sigma outperforms others on SLO misses for a range of loads, matching `PointPerfEst` closely. All systems prioritize SLO jobs by sacrificing BE jobs when load spikes. Cluster:SC256, Workload: E2E-LOAD-ℓ where $\ell \in [1.0, 1.2, 1.4, 1.6]$

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. As noted above, Fig. 7 shows performance as a function of deadline slack. We make two additional observations. First, smaller slack makes it harder to meet SLOs, for all policies, because of 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 it tries do succeed, though.

Sensitivity to load. Fig. 9 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.



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

Figure 10: 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 $n \in [5, 10, 25, 50, 75, 100]$

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. 10, 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.

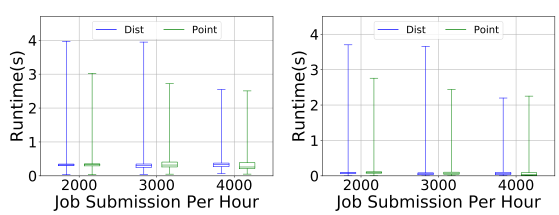
6.5 Scalability

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 [3, 26] 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. 11. Fig. 11(a) depicts the runtime of each scheduling cycle, including generation of scheduling options, evaluation, formulation of optimization problem, and execution of the solver. Fig. 11(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 Sec.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.



(a) Scheduling Cycle (b) Solver

Figure 11: 3Sigma scalability as a function of job submission per hour. Cluster: GOOGLE, Workload: SCALABILITY-*n* where $n \in [2000, 3000, 4000]$

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.

References

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

[2] Eric Boutin, Jaliya Ekanayake, Wei Lin, Bing Shi, Jingren Zhou, Zhengping Qian, Ming Wu, and Lidong Zhou. Apollo: Scalable and coordinated scheduling for cloud-scale computing. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 285–300, Broomfield, CO, October 2014. USENIX Association.

[3] Carlo Curino, Djellel E. Difallah, Chris Douglas, Subru Krishnan, Raghu Ramakrishnan, and Sriram Rao. Reservation-based scheduling: If you're late don't blame us! In *Proceedings of the ACM Symposium on Cloud Computing*, SOCC '14, pages 2:1–2:14, New York, NY, USA, 2014. ACM.

[4] Pamela Delgado, Florin Dinu, Anne-Marie Kermarrec, and Willy Zwaenepoel. Hawk: Hybrid datacenter scheduling. In *Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '15, pages 499–510, Berkeley, CA, USA, 2015. USENIX Association.

[5] Christina Delimitrou and Christos Kozyrakis. 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, pages 127–144, New York, NY, USA, 2014. ACM.

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

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

[8] Robert Grandl, Mosharaf Chowdhury, Aditya Akella, and Ganesh Ananthanarayanan. Altruistic scheduling in multi-resource clusters. In *OSDI*, pages 65–80, 2016.

[9] Mike Hibler, Robert Ricci, Leigh Stoller, Jonathon Duerig, Shashi Guruprasad, Tim Stack, Kirk Webb, and Jay Lepreau. Large-scale virtualization in the emulab network testbed. In *USENIX 2008 Annual Technical Conference*, ATC'08, pages 113–128, Berkeley, CA, USA, 2008. USENIX Association.

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

[11] Virajith Jalaparti, Peter Bodik, Ishai Menache, Sriram Rao, Konstantin Makarychev, and Matthew Caesar. 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, pages 407–420, New York, NY, USA, 2015. ACM.

[12] Sangeetha Abdu Jyothi, Carlo Curino, Ishai Menache, Shravan Matthur Narayanamurthy, Alexey Tumanov, Jonathan Yaniv, Ruslan Mavlyutov, Íñigo Goiri, Subru Krishnan, Janardhan Kulkarni, et al. Morpheus: towards automated slos for enterprise clusters. In *Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation*, pages 117–134. USENIX Association, 2016.

[13] S. Krishnaswamy, S. W. Loke, and A. Zaslavsky. Estimating computation times of data-intensive applications. *IEEE Distributed Systems Online*, 5(4), April 2004.

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

[15] I. A. Moschakis and H. D. Karatza. 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*, pages 418–423, June 2011.

[16] John K Ousterhout. Scheduling techniques for concurrent systems. In *International Conference on Distributed Computing Systems (ICDCS)*, volume 82, pages 22–30, 1982.

[17] Kaushik Rajan, Dharmesh Kakadia, Carlo Curino, and Subru Krishnan. Perforator: Eloquent performance models for resource optimization. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, SoCC '16, pages 415–427, New York, NY, USA, 2016. ACM.

[18] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proc. of the 3nd ACM Symposium on Cloud Computing*, SOCC '12, 2012.

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

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

[21] Bikash Sharma, Victor Chudnovsky, Joseph L. Hellerstein, Rasekh Rifaat, and Chita R. Das. Modeling and synthesizing task placement constraints in Google compute clusters. In *Proc. of the 2nd ACM Symposium on Cloud Computing*, SOCC '11, pages 3:1–3:14. ACM, 2011.

[22] Warren Smith, Ian T. Foster, and Valerie E. Taylor. Predicting application run times using historical information. In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*. IEEE, 1998.

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

[24] Dan Tsafrir, Yoav Etsion, and Dror G. Feitelson. Backfilling using system-generated predictions rather than user runtime estimates. In *IEEE Transactions on Parallel and Distributed Systems*. IEEE, 2007.

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

[26] Alexey Tumanov, Timothy Zhu, Jun Woo Park, Michael A. Kozuch, Mor Harchol-Balter, and Gregory R. Ganger. Tetrisched: Global rescheduling with adaptive plan-ahead in dynamic heterogeneous clusters. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, pages 35:1–35:16, New York, NY, USA, 2016. ACM.

[27] 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. Apache Hadoop YARN: Yet another resource negotiator. In *Proc. of the 4th ACM Symposium on Cloud Computing*, SOCC '13, 2013.

[28] S. Verboven, P. Hellinckx, F. Arickx, and J. Broeckhove. Runtime prediction based grid scheduling of parameter sweep jobs. In *Asia-Pacific Services Computing Conference, 2008. APSCC '08. IEEE*, pages 33–38, Dec 2008.

[29] A. Verma, M. Korupolu, and J. Wilkes. Evaluating job packing in warehouse-scale computing. In *2014 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 48–56, Sept 2014.

[30] Abhishek Verma, Ludmila Cherkasova, and Roy H. Campbell. ARIA: Automatic resource inference and allocation for mapreduce environments. In *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, pages 235–244, New York, NY, USA, 2011. ACM.

[31] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at Google with Borg. In *Proceedings of the Tenth European Conference on Computer Systems*, EuroSys '15, pages 18:1–18:17, New York, NY, USA, 2015. ACM.

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

[33] Timothy Wood, Prashant J Shenoy, Arun Venkataramani, and Mazin S Yousif. Black-box and gray-box strategies for virtual machine migration. In *NSDI*, volume 7, pages 17–17, 2007.

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