Making Speculative Scheduling Robust to Incomplete Data

Ana Gainaru
Vanderbilt University,
Nashville, TN, USA
ana.gainaru@vanderbilt.edu

Guillaume Pallez (Aupy)
Inria & Labri, Univ. of Bordeaux
Talence, France
guillaume.pallez@inria.fr

***Abstract*—In this work, we study the robustness of Speculative Scheduling to data incompleteness. Speculative scheduling has allowed to incorporate future types of applications into the design of HPC schedulers, specifically applications whose runtime is not perfectly known but can be modeled with probability distributions. Preliminary studies show the importance of speculative scheduling in dealing with stochastic applications when the application runtime model is completely known. In this work we show how one can extract enough information even from incomplete behavioral data for a given HPC applications so that speculative scheduling still performs well. Specifically, we show that for synthetic runtimes who follow usual probability distributions such as truncated normal or exponential, we can extract enough data from as little as 10 previous runs, to be within 5% of the solution which has exact information. For real traces of applications, the performance with 10 data points varies with the applications (within 20% of the full-knowledge solution), but converges fast (5% with 100 previous samples).**

Finally a side effect of this study is to show the importance of the theoretical results obtained on continuous probability distributions for speculative scheduling. Indeed, we observe that the solutions for such distributions are more robust to incomplete data than the solutions for discrete distributions.

***Keywords*—HPC scheduling, stochastic applications, performance modeling, discrete and continuous estimators, sampling**

I. Motivation

With the drive to incorporate big data and machine learning applications to HPC, new types of applications need to be considered when submitting jobs using HPC schedulers. These applications have heterogeneous, dynamic and data-intensive requirements and their performance is widely dependent on the input data. The convergent nature of the code or the continuous updates to the used algorithms makes the prediction of their execution time extremely hard. Currently, the best solution to understanding the behavioral patterns of such applications is to use a model of their execution time (for instance a probability distribution) constructed based on historical data.

In our previous work [12], we have shown that using the current HPC schedulers with applications whose execution time is not precisely known can lead to poor response time and low utilization. Indeed, many of such software systems are *reservation-based* (for instance Slurm [24], Torque [18] and Moab [6]), meaning that the schedule is computed thanks to an estimate (often an overestimate) given by the users. The possible gaps generated by the applications finishing early are then filled using backfilling. However, if one overestimates consistently the execution time of applications, the backfilling work is not enough to cover for the gaps generated. In addition, platform administrators tend to penalize overestimation by decreasing user priority for future submissions.

While one natural direction could be to develop more precise models to try to obtain precise performance estimation [8], [23], we have chosen a different complementary approach.

In our work [12], we have developed a proof of concept for a novel HPC scheduler that would integrate the uncertainty in the execution time of those new applications. The key idea is the following: in order to deal with uncertainty in the execution time, instead of doing a single reservation, the scheduler should schedule a series of increasing-size reservations (called a *strategy*): $\mathcal{S} = (t_1, t_2, \cdots, t_i, \cdots)$. For a given batch of jobs, it starts by scheduling the first reservations of all the jobs of the local batch. For each job of the batch, either the first reservation was sufficient to finish the job execution (in which case this job is considered executed), or it was not (in this case the execution is failed, the time considered wasted). For all the jobs where the first reservation was not large enough, the scheduler schedules the second reservation, and so on until all the jobs of the batch are executed. We have shown in a previous work [2] how to derive the optimal sequence of reservations to minimize the expected cost of a job, given the knowledge of the its execution profile (modeled by a probability distribution). Using a modified version of this algorithm that incorporates backfilling, we also showed [12] that the batch scheduler algorithm derived above was more efficient for both total system throughput, and average job response time!

All these previous results rely on the assumption that we know the correct distribution of possible walltimes for a job. An example of such a profile could be: *with an input of size 50GB, on 2048 nodes, the job's execution time follows a lognormal distribution of mean 10h and variance 4h*. However, in practice, getting the correct application profile (even if it is a probability distribution) is complicated. One way is to use historical data (the last M runs with similar parameters) as a basis for this profile [23]. Figure 1 shows the historic execution times for two stochastic and big data applications from the neuroscience field for the 2017 year together with two possible fits (polynomial or using distributions) of the data.

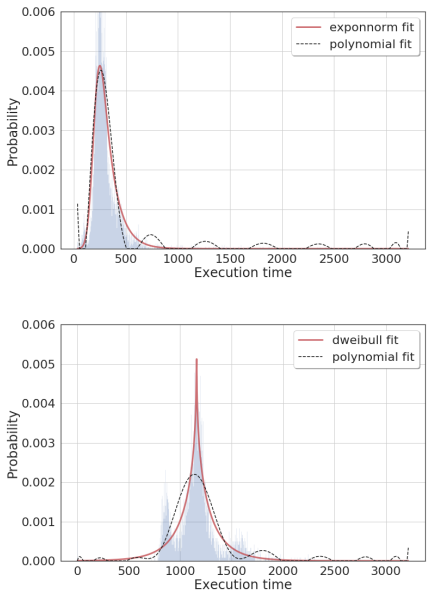


Fig. 1. Performance behavior for two stochastic functions together with two transformations into the continuous domain (top: Functional QA and preprocessing application, bottom: Deep brain structure segmentation application). Runs done during 2017 by the neuroscience department at the Vanderbilt University.

In this work we study the robustness of our solutions under various considerations. We show how one could design a *performance estimator* in a scheduler to derive efficient reservation strategies, solely based on historical data. We show that the natural strategy of using the data as is (discrete model) and computing the optimal solution is not efficient, but that with a simple transformation of the data to a continuous model, the strategies are often much more efficient.

In this work we make the following contributions:

- We study different applications both synthetic and real to study the impact of incomplete data.
- We show that even with minimal application knowledge, the strategies developed in our previous work still perform well.
- We show the importance of the theoretical study for continuous distributions, even though most of the data used to compute an application profile is in discrete format.
- Finally, we give a high level intuition on how one would include this into a scheduler.

The rest of the paper is organized as follows. Section II presents related works and an overview of the set of applications that show stochastic behavior and would benefit from the presented strategy. In Section III we remind the Speculative Scheduling model, and describe the optimization problem with incomplete knowledge of the application profile. In Section IV, we propose two different algorithms to derive strategies to solve the optimization problem. We then study it more on synthetic data in Section V, before experimenting on actual traces of applications in Section VI. Finally, we give some pointers on how to best practices for running on HPC systems before giving some concluding remarks.

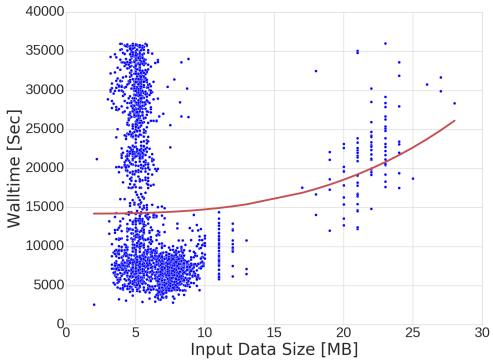


Fig. 2. The relation between the input data size and total execution time for a cortical surface modeling application

II. Background and Related Work

Emerging applications normally have a stochastic nature due to two reasons: (i) users not having a full understanding of the runtime behavior and resource requirements prior to execution and (ii) applications changing their runtime behavior and resource requirements during execution [22]. In our experience many big data and emerging applications fit both criteria. They are driven by the dynamic nature of each emerging field, using adaptive algorithms that can have different resource requirements depending on input data characteristics (e.g. the proximity of the input data to the convergence value) or on input parameters. (e.g. a specific area of interest which triggers an in-depth analysis algorithms). Their exploratory codes are in continuous change depending on previous results and thus so is the dependency of their resource requirements to the input data. There are currently a few studies to look at the performance of specific variable applications (e.g. for seismic wave propagation [7], or for shock hydrodynamics [3]). There are also studies that analyze platform workload's performance (e.g. for systems at TACC [11] or at Sandia National Laboratory [1]). The study in [20] characterizes application variation due to resource contention and software-related problems. However, there is nothing in literature on understanding application behavioral patterns for applications like the ones developed by emerging fields that focus on productivity and not performance.

Emerging applications are typically composed of multiple stages that could use multiple programming languages and with different performance characteristics. The total walltime of an application could depend on complex properties inside the input data and do not have a correlation with the input data size (typical for current scientific large-scale applications). Figure II shows the normal pattern in emerging applications, where the execution time can vary several orders of magnitude for the same input data size.

Current HPC schedulers extensively used on today's large-scale systems rely on accurate estimates for the requested walltimes and memory requirements. Enforcing static limits for resources leads to two unfavorable scenarios: (i) underestimation when the users request resource requirements lower than the actual needs of their application in which case the runtime system kills the execution the moment the needs of the application exceed the requested amount; or (ii) overestimation in which case the utilization of the system is being wasted and users are being penalized by the cluster policies (by larger wait times in the queue due to either the system having to schedule larger jobs than necessary or by system administrator decreasing the user priority). Datacenter platforms like UC Berkeley's Apache Mesos [14] and Google's Borg [21] alleviate some of the constraints of current HPC platforms for emerging applications. However they are built for Cloud type workflows, are disruptive in nature and do not allow for an evolution in their design. In addition, emerging applications, often have data which is tied down to a particular site and therefore the only viable option is to run the application on local institutional clusters.

Considerable research has been conducted to improve job runtime estimates. In [13], the authors used a polynomial model to enhance the accuracy of runtime estimates; [17] uses regression models to alleviate the problem of underestimation; [19] uses the average of the last two runtimes as a prediction. While this last method is simple, it is capable of doubling the accuracy. Based on current research, the Medical-image Analysis and Statistical Interpretation (MASI) [16] laboratory at Vanderbilt is using a similar method, averaging the last 10 runs of an application for determining the request time, and doubling it each time it underestimates the walltime. This method adapts to the changes in behavior while decreasing the overestimation caused by asking for the largest estimation. We compare our estimators with this method as well as the HPC classical model.

To conclude this section, we discuss the problematics of learning an unknown probability distribution from its samples. This problematic is one of the most natural and important questions in statistics and has been widely studied [4], [5], [9], [15]. The typical strategy is to derive a good estimator (i.e. a function that transforms the samples) and to evalute it on a given loss function (hence the estimators are specific to the loss function). Kamath et al. [15] provide a good overview of the different work of the field. They describe how a theoretical evaluation should be performed. This is orthogonal to our work, and future studies could be dedicated to finding better and proved estimators for our loss function which would improve the performance of speculative scheduling.

III. Model and Estimators

In this section we formalize the problem at hand. The core idea is the following: when we do not know the theoretical performance model of an application, can we use the sample of data available to find a *good* reservation strategy? Hence the underlying problem is the following: if we only have samples from the distribution of execution times of an application, can we *estimate* a reservation strategy that minimizes the expected cost for this distribution? We can also characterize how far are we using this strategy from the *best* one that has complete information about the application behavior.

A. Stochastic jobs and Reservation strategy

In this work, we use the model from our previous work [2], [12]. We consider *stochastic jobs*, that is jobs whose execution time X is unknown but (i) deterministic: two successive execution on the same input have the same duration; and (ii) randomly and uniformly sampled from a probability distribution law $\mathcal{D}$ (PDF: f_X). In this work, we consider that $\mathcal{D}$ is not known except for its domain $[a, b]$ (essentially we use $a = 0$, and consider that the users can give an upper bound on the execution time). In addition, we have a sample of size n: $X_1, \ldots, X_n$ also iid (independent and identically distributed) from $\mathcal{D}$.

A reservation strategy $\mathcal{S} = (t_1, t_2, \ldots, t_i, t_{i+1}, \ldots)$ for X is a set of increasing reservation sizes that are submitted to the machine in increasing order following the algorithm:

- A reservation of size t_1 is made.
- If $X \leq t_1$, then the first reservation was enough for the successful execution of X and we are done.
- If $X > t_1$, then the reservation failed and the work is considered lost. A second reservation of size $t_2 > t_1$ is performed.
- This process is repeated until the reservation is enough for the successful execution of X.

Finally, for a job of actual length t, we define the *cost* of a reservation t_1 as:

$$\alpha t_1 + \beta \min(t, t_1) + \gamma.$$

where the α part of this cost is the reservation cost, and the β part is the utilization cost. We have shown [2] that this cost function could indeed cover many different cost models (from execution time on an HPC platform to cost of usage in the cloud). The studies in this work are done for $\alpha = 1$ and $\beta = \gamma = 0$. For example, when simulating HPC systems, depending on how α is chosen, the cost will be determined by the total makespan (or response time) of an application by including the wait time in the queue until the scheduler submits the job for execution on the machine and/or the reservation time in the system.

Overall, the total cost for job t with strategy $\mathcal{S}$ is:

$$C(k, t) = \sum_{i=1}^{k-1} (\alpha t_i + \beta t_i + \gamma) + \alpha t_k + \beta t + \gamma$$

where k is the smallest index in the sequence such that $t \leq t_k$.

The overall expected cost is then given by the equation:

$$Cost(\mathcal{S}, X) = \sum_{k=1}^{\infty} \int_{t_{k-1}}^{t_k} C(k, t) f_X(t) dt$$

B. Model, Optimization problem

In our previous work, we have shown that the general cost model could be written as:

Theorem 1 (Expected Cost [2, Theorem 1])**.** *Given a random variable X and a strategy $\mathcal{S} = (t_1, t_2, \ldots, t_i, t_{i+1}, \ldots)$, we define the Expected Cost of $\mathcal{S}$ on X:*

$$Cost(\mathcal{S}, X) = \beta \cdot \mathbb{E}[X] + \sum_{i=0}^{\infty} (\alpha t_{i+1} + \beta t_i + \gamma)\, \mathbb{P}(X > t_i) \quad (1)$$

In this work, we use $\alpha = 1$, $\beta = 0$, $\gamma = 0$ (RESERVATIONONLY model [2]). In this case, the HPC system creates rigid reservations which are not adapted depending on how long the application is running (typical for several current large-scale systems). This scenario also assumes that large jobs have higher priority in being scheduled to be executed and thus do not wait longer in the queue.

Given a random variable X, we can compute $\mathcal{S}_{\text{opt}}(X)$ the strategy that minimizes the Expected Cost on X. In statistics, estimators are often used to estimate an unknown parameter of a distribution. For example, if one knows that the distribution is a gaussian distribution, then given $X_1, \cdots, X_n$ the following are unbiased estimators

$$\bar{X} = \frac{1}{n} \sum_{i=1}^{n} X_i \qquad \text{for the mean} \qquad (2)$$

$$S^2_{n-1} = \frac{1}{n-1} \sum_{i=1}^{n} \left(X_i - \bar{X}\right)^2 \qquad \text{for the variance} \qquad (3)$$

Their *Mean-Square Error* is $O(1/n)$.

Using this we can now define a loss function for the estimators that we consider:

Definition 1 (Expected Loss (EL))**.** Given a random variable X. We define the *Expected Loss* of a strategy $\mathcal{S}$ as:

$$d_{\text{EL}}(\mathcal{S}) = \frac{Cost(\mathcal{S}, X) - Cost(\mathcal{S}_{\text{opt}}(X), X)}{Cost(\mathcal{S}_{\text{opt}}(X), X)} \quad (4)$$

Essentially, this loss function is the information about the proximity to the optimal algorithm. It is a distance to evaluate the quality of the estimators.

We can now define formally the problem that we consider here:

Definition 2 (Sample-based strategy)**.** Given a set of n iid samples $X_1, \cdots, X_n$ for a random variable X following an unknown distribution. Find an estimator $\mathcal{S}_n$ for $\mathcal{S}_{\text{opt}}(X)$ that minimizes d_{EL}.

IV. ESTIMATORS FOR RESERVATION STRATEGIES

In this section we provide different solutions to the *Sample-Based strategy* problem introduced in the previous section. We consider that we are given a sample data of size n: $X_1, \cdots, X_n$.

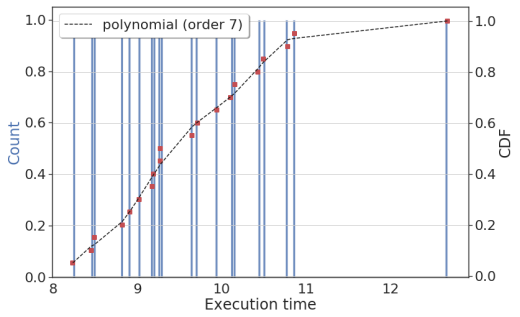


Fig. 3. From a discrete cumulative to a continuous cumulative function, for $n = 20$ data points, sampled from a truncated normal distribution (mean=8, variance=2, domain=12 to 14).

a) Discrete-Method: The most natural way to compute an estimate for the reservation strategy is to use the data as is: the n iid samples can be seen as a discrete distribution with n values, each of probability $1/n$ (assuming there is no redundant value, otherwise k/n if a value appears k times). We know that we can compute in polynomial time the optimal strategy for this distribution [2, Theorem 5]. We denote $\mathcal{S}^n_{\text{disc}}(X_1, \cdots, X_n)$ this strategy.

b) Continuous-Method: The second way is to transform the discreet data to a continuous domain and compute the optimal sequence of reservations in this domain. In order to do so, the idea is to transform the discrete cumulative function based on the n sample values into a continuous cumulative function (see the dashed line from Figure 3).

Given this transformation, we now have a cumulative function on a continuous domain. We then interpolate it with a continuous function using two different methods:

- **Polynomial** (P_{10}-$\mathcal{S}^n_{\text{cont}}$ $(X_1, \cdots, X_n)$): we find the polynomial of maximum degree 10^1 that minimizes the mean square error to the data. Figure 1 (lower) gives an example where a polynomial interpolation is a good fit.
- **Usual-Distribution** (UD-$\mathcal{S}^n_{\text{cont}}$ $(X_1, \cdots, X_n)$): we use the set of distribution available in scipy stats package (alpha, beta, cosine, dgamma, dweibull, exponnorm, exponweib, exponpow, genpareto, gamma, halfnorm, invgauss, invweibull, laplace, loggamma, lognorm, lomax, maxwell, norm, pareto, pearson3, rayleigh, rice, truncexpon, truncnorm, uniform, weibull) to find the best interpolation to the data with respect to the mean square error. Figure 1 (upper) gives an example of an application where a distribution interpolation gives the best fit.

For those two interpolation strategies, we can then compute reservation strategies using the different algorithms provided in [2] (in this work, we use the discretization scheme: EQUAL-TIME and divide the support in 500 equal length steps).

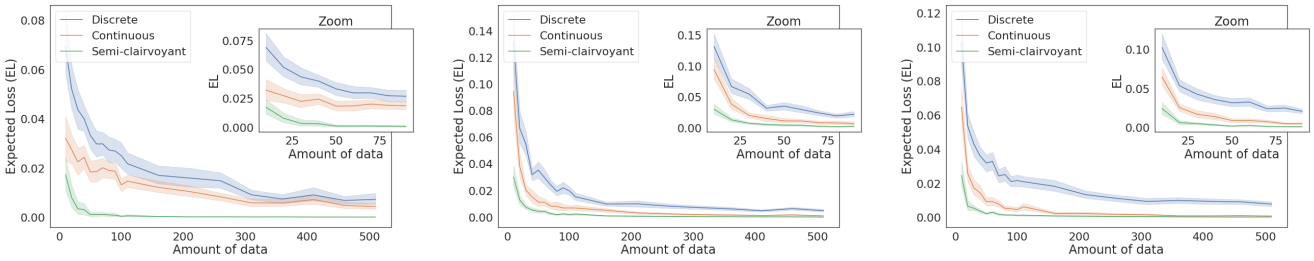


Fig. 4. Results for the truncated normal distributions (left: low variance, middle: high variance) and the truncated exponential distribution (right)

V. Simulation on synthetic applications

In this first set of simulations we evaluate the quality of our estimators on known distributions. This allows to measure exactly the value of d_{EL} since knowing the shape of the data allows us to compute the optimal cost. We do not consider *in this section* actual traces of applications. We compare our estimators to estimators from the literature.

We are particularly interested whether our estimators are efficient estimators.

In the following we compare different estimators on several probability distributions:

1) Truncated gaussian (mean: 8h, domain: 0 to 20h) with low (2h) and wide (4h) variance;
2) An exponential distribution (mean: 8h);
3) A distribution whose density is the average of the density of two truncated gaussian (domain: 0 to 20h respectively, variance: 2) with different means (4h and 10h), representing an application with two different behaviors.

We compare the estimators $\mathcal{S}^n_{\text{disc}}$, [$\text{P}_{10}$/UD]-$\mathcal{S}^n_{\text{cont}}$ as introduced in Section IV to a *semi-clairvoyant* estimator which is expected to perform better and which can give us information on a *good* behavior for our estimators.

Specifically, the *Semi-clairvoyant* estimator is defined as a version of UD where the set of functions for the interpolation is restricted to the actual distribution of the data.

A. Speed of convergence

In the first set of evaluation, we are interested at the volume of samples needed to obtain a good solution. Hence we vary the number of data elements available (representing the number of previous runs) from 10 to 500 and measure the expected loss. Each evaluation is performed 100 times to evaluate the variation of our results (Figure 4).

To understand the results: an estimated loss of 0.05 with k sample points for estimator E essentially means that with a history of k runs, the strategy given by E gives a cost which is only 5% larger than that of the optimal strategy!

Here we observe that if the distribution follows a truncated gaussian distribution with a history of 10 runs the cost of $\mathcal{S}^n_{\text{cont}}$ is only 6% larger than that of the optimal. With 60 runs we are down to 3%. It is interesting to observe the difference depending on the variance of the distribution: with low variance, the number of samples needed to be below 3% is small, but it is hard to get below 1%. With high variance, we need more runs to get below 3%, but we can get very close to optimal when the number of samples increases. The other observation is that $\mathcal{S}^n_{\text{disc}}$ has poorer performance than $\mathcal{S}^n_{\text{cont}}$. $\mathcal{S}^n_{\text{disc}}$ over-fits the current data and is a inferior predictor of future data points. The results are similar for the exponential distribution (Figures 4 right).

[1]Note that the number 10 for the maximum degree of the polynomial has been chosen to give some leniency while not overfitting the data.

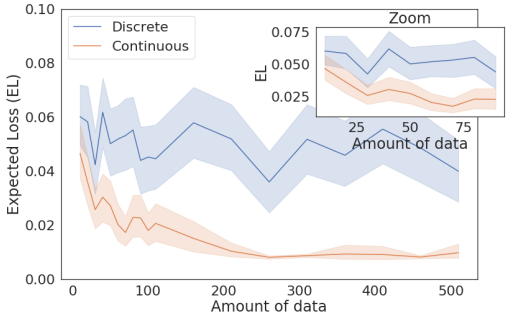


Fig. 5. Results for the sum of truncated normal distributions. For this one we do not plot the semi-clairvoyant algorithm since the cumulative function is not part of the available regular distributions.

For applications that shift their behavior due to configuration parameters (like in the case of profiles consisting of aggregated distributions), $\mathcal{S}^n_{\text{cont}}$ needs more samples to converge but it is much better than using the discrete data on it's own (Figure 5). These results confirm the intuition that going through a continuous interpolation of data makes the scheduling strategy more robust to variation.

From the results there are two key observations:

- The $\mathcal{S}^n_{\text{cont}}$ strategies are clearly dominating over the $\mathcal{S}^n_{\text{disc}}$ strategy;
- The number of samples needed for good performance is relatively low.

B. Precision of polynomial approximation

In the second set of evaluation, we are interested in the impact of the maximum degree k of the polynomial in P_k-$\mathcal{S}^n_{\text{cont}}$ on the performance of the fit. When the degree of the

TABLE I
IMPACT ON d_{EL} OF THE MAX-DEGREE k OF THE POLYNOMIAL IN P_k-$\mathcal{S}^n_{\text{CONT}}$ FOR DIFFERENT SAMPLE SIZES.

	Max Degree k			
Sample size = 10	2	5	10	15
TNormal ($\mu = 8$, $\sigma = 2$)	0.253	0.104	0.186	not fitted
Exponential	0.203	0.091	0.092	0.241
Sum of TNormal	0.198	0.024	0.021	0.021
Multi Atlas	2.175	1.405	1.181	1.413
Sample size = 60	2	5	10	15
TNormal ($\mu = 8$, $\sigma = 2$)	0.108	$8.64e^{-3}$	$5.52e^{-3}$	$9.62e^{-3}$
Exponential	0.182	$2.41e^{-3}$	$3.12e^{-3}$	$9.39e^{-3}$
Sum of TNormal	0.163	0.018	0.019	0.017
Multi Atlas	1.274	1.036	0.811	1.13
Sample size = 110	2	5	10	15
TNormal ($\mu = 8$, $\sigma = 2$)	0.121	$9.61e^{-3}$	$6.49e^{-3}$	$9.82e^{-3}$
Exponential	0.179	$5.39e^{-3}$	$8.12e^{-3}$	$1.53e^{-2}$
Sum of TNormal	0.211	0.031	0.034	0.032
Multi Atlas	0.829	0.636	0.615	0.721
Sample size = 510	2	5	10	15
TNormal ($\mu = 8$, $\sigma = 2$)	0.203	$8.95e^{-2}$	0.103	$9.22e^{-2}$
Exponential	0.191	0.015	0.017	0.015
Sum of TNormal	0.391	0.062	0.052	0.103
Multi Atlas	1.061	1.26	0.955	0.821

polynomial increases, more dimensions of freedom open up for the fitting function. While underfitting is expected to give poor performance, one can expect the same from overfitting the data: the ultimate overfitting of a function is the discrete data, which we showed was not as good as a continuous fit of the data to compute an efficient schedule. We write the result in Table I.

Note that in this set of experiments, we have also added results from Multi-Atlas: traces from a real applications where we found that polynomial fit gave specifically poor results. In this case d_{EL} is computed slightly differently (see Section VI).

From these results we make the following observations.

- Under-fitting is a problem for the performance of the algorithm, thus we need to fit polynomials of high enough order to capture the behavior of applications
- It seems that over-fitting is less of a problem. However in some cases it shows poor results so it is a good idea to limit the order of the polynomial interpolation. Moreover, we observed that the fitting error of polynomial is not always correlated with the total cost.

Overall, while the difference between a fitting with a maximum degree of 2 and 5 are notable, there is never a significant difference between results on a maximum degree of 5 and 15. This is true both for distributions where a polynomial fit gives poor results (Multi-Atlas), and distributions where is gives a good fit.

In future work, it would be interesting to show theoretically the observations here, or to find even better estimators.

VI. EVALUATIONS ON REAL APPLICATIONS

In this section we focus on real applications and study the applicability of our previous observations. For this purpose, we analyze the highly stochastic applications used by the The Medical-image Analysis and Statistical Interpretation laboratory at Vanderbilt. The dataset consists of over 30 neuroscience and medical applications for which we have execution logs of their runs on the ACCRE cluster (local to the Vanderbilt university). To compute the strategy, for each application we pick a number of samples (from 50 to 500) either consecutive or randomly chosen from history log and use the two strategies: $\mathcal{S}^n_{\text{cont}}$ and $\mathcal{S}^n_{\text{disc}}$. We then evaluate their cost on the entire year of data by summing the cost of these strategies over all samples. Hence, the cost for application A is given by the following formula:

$$Cost(\mathcal{S}, A) = \sum_{t \in A} C(k_t, t) = \sum_{j=0}^{N} (\sum_{i=0}^{k_j} t_i),$$

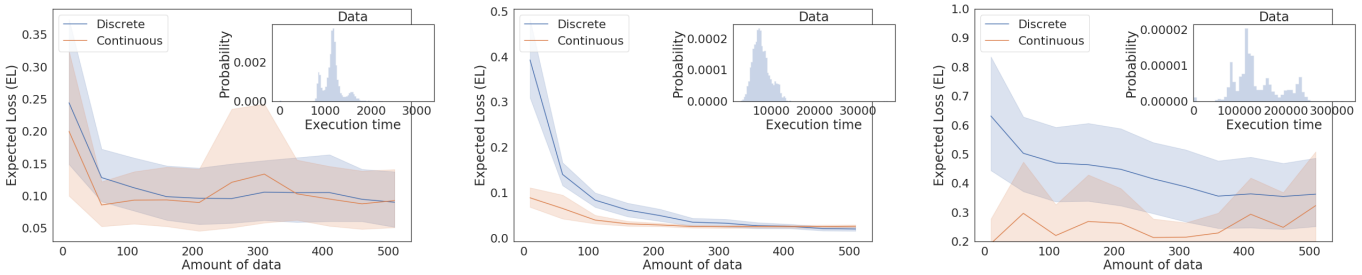
where N is the total number of runs of application A and k is chosen so that t_{k_j} is the first reservation larger than the walltime of the j-th run (i.e. $t_{k_j} > X_k$ and $t_i < X_k \forall i < k_j$). These evaluations are done 100 time to study the variation of the performance.

Figure 6 shows the cost of each strategy relative to the optimal for 6 applications when varying the number of samples used for computing the sequence. The optimal strategy is computed by considering the discrete algorithm on the entire dataset.

As the number of samples increases, the $\mathcal{S}^n_{\text{disc}}$ strategy converges to $\mathcal{S}^n_{\text{cont}}$. Note that in some cases, $\mathcal{S}^n_{\text{disc}}$ exceeds the performance of $\mathcal{S}^n_{\text{cont}}$ (e.g. Figure 6a). This is due to the fact that the number of data points available for the evaluation is actually very small, hence $\mathcal{S}^n_{\text{disc}}$ with many samples is almost the same as the computation of the optimal solution. However, in the majority of cases $\mathcal{S}^n_{\text{cont}}$ shows better results after only a few samples being used in the training phase (for example for the cortical surface modeling code, using only 10 runs for training gives a cost only 10% over the optimal). For applications with complex behaviors where the pattern of the application changes dramatically (for example Figure 6f shows execution times between 3 to 78 hours) both $\mathcal{S}^n_{\text{disc}}$ and $\mathcal{S}^n_{\text{cont}}$ have a high performance variation. We believe allowing polynomials of higher degree or more complex interpolations will be beneficial for these applications and continue to improve the results of $\mathcal{S}^n_{\text{cont}}$.

Further we analyzed the impact of how to chose the training samples. For all results we used consecutive runs in the logs in order to simulate a user having access to x previous runs and trying to predict the behavior of future ones. This might create a bias in the results since consecutive runs might share similar patterns. We made the same experiments again, this time randomly selecting the x sample runs (from 50 to 500) from the log. The results show that randomly choosing the samples can decrease the cost by 50% in some cases (a typical example presented in Figure 7). Depending on the number of samples in the execution log, it might be preferable to select a smaller random sample in order to compute the sequence for future runs.

(a) Cerebellum segmentation ($N = 718$) (b) Functional connectivity analysis ($N = 1151$) (c) Functional QA and pre-processing ($N = 17416$)



(d) Deep brain structure segmentation ($N = 3774$) (e) Cortical surface modeling ($N = 2411$) (f) Whole brain segmentation ($N = 17416$)

Fig. 6.

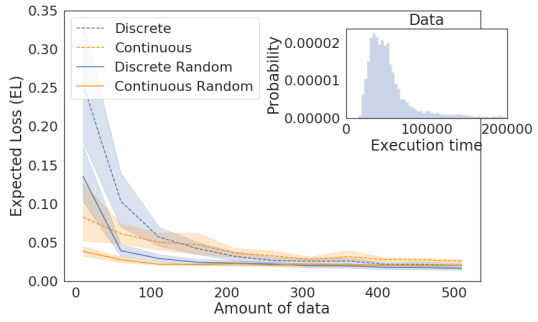


Fig. 7. Results for a structural identification of orbital anatomy application when using consecutive and random sampling

A. *One week in a life of an HPC scheduler*

The final question that remains, is: in practice, how useful is it to be 20, 10, or even 2% far from the optimal strategy.

In this final evaluation section we try to answer this question by simulating a production system (like ACCRE at Vanderbilt) executing the 6 applications characterized in the previous section. For this purpose, we chose a week of execution from the history logs and extracting the application submission information. We used this data to feed it in an HPC scheduler simulator (ScheduleFlow [10] used in our previous studies) and compute the average job response time, system utilization and average job stretch (the ratio between the job response time and walltime) for different reservation strategies:

- *HPC* model in which successful completion is guaranteed by using the largest past execution as the estimate.
- The model used by the MASI group, by doing the *Average of previous runs*
- Our *Continuous fit* model, by using random previous runs to compute the reservations using the $\mathcal{S}^n_{\text{cont}}$ strategy.

In some cases, the past largest execution is smallest than a future submission for the same application. In those cases, we extend all strategies by multiplying the last reservation by 1.5 until the reservation is large enough to complete the job.

Figure 8 presents the results. Using only 10 previous runs, the $\mathcal{S}^n_{\text{cont}}$ strategy decreases the average stretch by over 25%. Due to Multi Atlas runs that require a larger training sets, the average makespan and utilization do not see the same effect. However, including 110 samples into the training, the average job makespan decreases by 33% while the total system utilization increases by almost 50%.

The results show that even with little data one can use linear interpolation to obtain efficient scheduling strategies. While the performance of each individual application is different depending on the fitting functions and the number of samples used, randomly sampling previous runs seems to be a good optimization in all cases. In this conditions, speculative scheduling becomes feasible either integrated into a current HPC scheduler or as an additional layer at submission used to adjust request estimates. Our method can easily be adapted to the characteristics of any system by tunning the α, β and γ parameters.

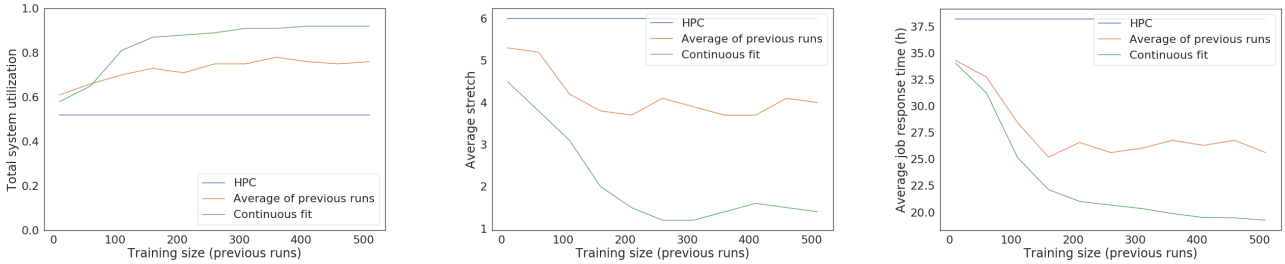


Fig. 8. Results for one week of execution (from top to bottom: system utilization; average job stretch and average job makespan)

VII. Conclusion

The goal of this work is to show that speculative scheduling can be used, even in the presence of incomplete information. We do not claim to have designed a good predictor for the performance of applications. Additional study on more applications need to be done. For instance, one can expect that the performance of some applications is also time dependent: updates to the software can be made for instance which intrinsically changes the code. Again, more research should be done in this regard.

We believe that we are opening a new direction in performance prediction research, where performance predictors do not need to necessarily predict the exact performance profile of an application, but can describe it with random variable and probability distributions. This may be an easier problem to solve for the field of performance prediction.

In addition, we have shown that even if the performance profile is not correct, we can still find benefits to it in speculative scheduling. Finally, an important takeaway from this work is the importance of the theoretical results for continuous distributions as well as those for discrete. This was not straightforward since historical data is more often given out as discrete samples. However, as we have seen in this work, interpolating the data makes for more robust solutions.

In the future, we plan to use this study as a basis to verify the upcoming theoretical results (e.g. using checkpoints so that the work is not lost; integrating multi-dimension such as memory needs, machine needs and performance models such as malleability).

Acknowledgements

We thank the VUIIS Center for Computational Imaging for sharing de-identified logs without patient or investigator identifiable data.

References

[1] A. Agelastos, B. Allan, J. Brandt, A. Gentile, S. Lefantzi, S. Monk, J. Ogden, M. Rajan, and J. Stevenson. Toward rapid understanding of production hpc applications and systems. In *2015 IEEE International Conference on Cluster Computing*, pages 464–473, Sep. 2015.

[2] Guillaume Aupy, Ana Gainaru, Valentin Honoré, Padma Raghavan, Yves Robert, and Hongyang Sun. Reservation strategies for stochastic jobs. In *IEEE IPDPS*, 2019.

[3] M. J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *J. Comput. Phys.*, 82(1):64–84, May 1989.

[4] Dietrich Braess, Jürgen Forster, Tomas Sauer, and Hans U Simon. How to achieve minimax expected kullback-leibler distance from an unknown finite distribution. In *International Conference on Algorithmic Learning Theory*, pages 380–394. Springer, 2002.

[5] Dietrich Braess and Thomas Sauer. Bernstein polynomials and learning theory. *Journal of Approximation Theory*, 128(2):187–206, 2004.

[6] N. Capit, G. Da Costa, Y. Georgiou, G. Huard, C. Martin, G. Mounie, P. Neyron, and O. Richard. A batch scheduler with high level components. In *CCGrid*, pages 776–783, 2005.

[7] L. Carrington, D. Komatitsch, M. Laurenzano, M. M. Tikir, D. Michea, N. Le Goff, A. Snavely, and J. Tromp. High-frequency simulations of global seismic wave propagation using specfem3d_globe on 62k processors. In *SC '08*, 2008.

[8] Laura Carrington, Allan Snavely, and Nicole Wolter. A performance prediction framework for scientific applications. *Future Generation Computer Systems*, 22(3):336–346, 2006.

[9] Siu-On Chan, Ilias Diakonikolas, Rocco A Servedio, and Xiaorui Sun. Learning mixtures of structured distributions over discrete domains. In *ACM-SIAM SODA*, pages 1380–1394. SIAM, 2013.

[10] Ana Gainaru et al. ScheduleFlow: A simulator for HPC schedulers. https://github.com/anagainaru/SchedulerSimulator, 2019. [Online; accessed 19-April-2019].

[11] T. Evans, W. L. Barth, J. C. Browne, R. L. DeLeon, T. R. Furlani, S. M. Gallo, M. D. Jones, and A. K. Patra. Comprehensive resource use monitoring for hpc systems with tacc stats. In *2014 First International Workshop on HPC User Support Tools*, pages 13–21, Nov 2014.

[12] Ana Gainaru, Guillaume Pallez, Hongyang Sun, and Padma Raghavan. Speculative scheduling for stochastic hpc applications. In *ICPP*, 2019.

[13] E. Gaussier, D. Glesser, V. Reis, and D. Trystram. Improving backfilling by using machine learning to predict running times. In *SC'15*, 2015.

[14] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *the 8th USENIX Conference on Networked Systems Design and Implementation*, pages 295–308, 2011.

[15] Sudeep Kamath, Alon Orlitsky, Dheeraj Pichapati, and Ananda Theertha Suresh. On learning distributions from their samples. In *Conference on Learning Theory*, pages 1066–1100, 2015.

[16] MASI Lab. Medical-image Analysis and Statistical Interpretation (MASI) Lab. https://my.vanderbilt.edu/masi/.

[17] Sara Mustafa, Iman Elghandour, and Mohamed A. Ismail. A machine learning approach for predicting execution time of spark jobs. *Alexandria Engineering Journal*, 57(4):3767 – 3778, 2018.

[18] Garrick Staples. Torque resource manager. In *SC'06*, SC '06, 2006.

[19] D. Tsafrir, Y. Etsion, and D. G. Feitelson. Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Transactions on Parallel and Distributed Systems*, 18(6):789–803, June 2007.

[20] Ozan Tuncer, Emre Ates, Yijia Zhang, Ata Turk, Jim Brandt, Vitus J. Leung, Manuel Egele, and Ayse K. Coskun. Diagnosing performance variations in hpc applications using machine learning. In *High Performance Computing*, pages 355–373, Cham, 2017. Springer International Publishing.

[21] 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.

[22] Ole Weidner, Malcolm Atkinson, Adam Barker, and Rosa Filgueira Vicente. Rethinking high performance computing platforms: Challenges, opportunities and recommendations. In *ACM DIDC '16*, pages 19–26, New York, NY, USA, 2016. ACM.
[23] Leo T Yang, Xiaosong Ma, and Frank Mueller. Cross-platform performance prediction of parallel applications using partial execution. In *SC'05*, page 40. IEEE Computer Society, 2005.
[24] Andy B. Yoo, Morris A. Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In *JSSPP*, pages 44–60, 2003.

APPENDIX

ARTIFACT DESCRIPTION: MAKING SPECULATIVE SCHEDULING ROBUST TO INCOMPLETE DATA

A. Abstract

This artifact contains the code for computing the sequence of requests for an application given its past behavior (together with the corresponding cost as described in Section III), as well as instructions on how to use the code and print the data to reproduce the results from Section V. Section VI uses neuroscience applications that are open source, however the execution logs as well as the input MRI data for each application is not available outside the Vanderbilt University.

B. Description

1) Check-list (artifact meta information):

- **Algorithm:**
- **Program: Python code**
- **Data set: Synthetic application runs created from the truncated normal and truncated exponential distributions**
- **Run-time environment: the code should execute successfully on any environment capable of running python, scipy and numpy**
- **Hardware: any configuration**
- **Execution: the compute_sequence_cost.py script can receive as input either a scipy distribution (between truncnorm and expon) or a file with a list of execution times (either one per line or separated by space)**
- **Output: the output of the script is a csv file containing the optimal cost as well as (Best Fit, Parameters for the best fit, Cost, EML) for the discrete and continuous fit of the input data (when using 10 to 500 entries for training and all entries when computing the cost)**
- **Experiment workflow: download the code, run compute_sequence_cost.py script on different distributions; obtain output csv files, run the jupyter notebook to print the data, analyze the output figure**
- **Publicly available?: Yes**

2) How software can be obtained (if available): The code used is available on github at https://github.com/anagainaru in the ReproducibilityInitiative repository (link in footnote)[2].

3) Software dependencies: Python3 is required. The code is using scipy functions for fitting the data to a distribution. The pearson3 and rice have been removed from the list of distributions for which fitting is performed since often the CDF values given by scipy does not correspond with the PDF (Figure 9). Our results do not include these distributions. They can be added in the WorkloadFit.py file in the DistInterpolation class (in which case the outliers need to be removed from the results file).

[2]https://github.com/anagainaru/ReproducibilityInitiative/tree/master/2019_scala

The classes used to compute the sequence of requests are part of the ScheduleFlow simulator (available for download at: https://github.com/anagainaru/ScheduleFlow).

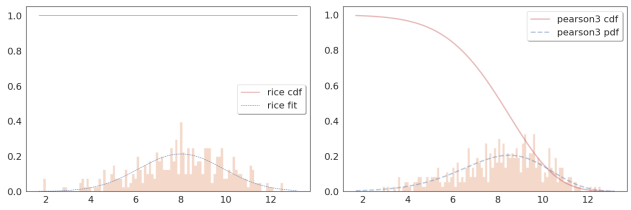


Fig. 9. Behavior of the CDF and PDF for the rice and pearson3 distributions, which best fit the datasets presented.

4) Datasets: All datasets are generated during runtime by the script (to follow either truncated normal or truncated exponential distributions).

C. Installation

```
git clone \
git@github.com:anagainaru\
/ReproducibilityInitiative.git

cd ReproducibilityInitiative\
/2019_scala
```

D. Experiment workflow

Download or clone from git the source code.

To execute the code using a log file of past executions for an application:

```
python compute_sequence_cost.py\
{folder/dataset_file}
```

Dataset file needs to contain a list of execution times

To execute the code using synthetic data for the execution times:

```
python compute_sequence_cost.py\
{distribution}
```

Supported distributions: [truncnorm, expon]

The truncated normal distribution uses μ=8, σ=2 with limits $[a, b]$=[0,20]. The truncated exponential distribution uses μ=1, σ=1.5 with limits $[a, b]$=[0,9] These values can be changed inside the compute_sequence_cost.py script.

After the successful execution of the script, a csv file will be created (wither dataset_file_cost.csv or distribution_cost.csv. This file holds the results of one experiments per line where each line contains the fields:

```
Function, Fit, Parameters,\
Cost, Trainset, EML
```

Function is either Optimal, Discrete or Continuous. The fit column contains values only for the Continuous function and can either be a distribution or a order for the polynomial.

The parameters hold the parameters for the best interpolation function, wither the distribution parameters or the k-1 parameters for a k degree polynomial function. The Cost represents the cost for the sequence obtain using the continuous or interpolation fits, EML is the relative cost to the cost of the optimal. The trainset defines the number of samples used for computing the sequence (from 10 to 500).

E. Evaluation and expected result

The experiments ran 100 times each for a training set of 50 to 500 elements (in steps of 50).

```
for i in range{1..100}; do\
python compute_sequence_cost.py truncnorm;\
done
```

After running the experiments script, the results will be printed out in the corresponding csv file. This jupyter notebook print_sequence_cost.ipynb can be used to plot the figures from Section V.

We do not expect the results to be exactly the same as the ones presented in this paper but the observed trend should be similar.

F. Experiment customization

There are four possible customizations:

1) The truncated normal and exponential parameters can be changed from the compute_sequence_cost.py script by changing the values from the ExponentialRun and TruncNormRun classes
2) The maximum degree of the polynomial interpolation can be changed in the compute_sequence_cost.py script when initializing

   ```
   WorkloadFit.PolyInterpolation(\
   max_order=10)
   ```

3) New distributions can be added in the compute_sequence_cost.py script by adding a class with the distribution parameter similar to ExponentialRun and TruncNormRun. The new class needs to inherit the DistributionRuns class.
4) The distributions used for the distribution interpolation can be changed from the DistInterpolation class in the WorkloadFit file.