Active Learning in Performance Analysis

Dmitry Duplyakin
Department of Computer Science
University of Colorado
Boulder, CO 80309, USA
dmitry.duplyakin@colorado.edu

Jed Brown
Department of Computer Science
University of Colorado
Boulder, CO 80309, USA
jed.brown@colorado.edu

Robert Ricci
School of Computing
University of Utah
Salt Lake City, UT 84112, USA
ricci@cs.utah.edu

***Abstract*—Active Learning (AL) is a methodology from machine learning in which the learner interacts with the data source. In this paper, we investigate application of AL techniques to a new domain: regression problems in performance analysis. For computational systems with many factors, each of which can take on many levels, fixed experiment designs can require many experiments, and can explore the problem space inefficiently. We address these problems with a dynamic, adaptive experiment design, using AL in conjunction with Gaussian Process Regression (GPR). The performance analysis process is "seeded" with a small number of initial experiments, then GPR provides estimates of regression confidence across the full input space. AL is used to suggest follow-up experiments to run; in general, it will suggest experiments in areas where the GRP model indicates low confidence, and through repeated experiments, the process eventually achieves high confidence throughout the input space. We apply this approach to the problem of estimating performance and energy usage of HPGMG-FE, and create good-quality predictive models for the quantities of interest, with low error and reduced cost, using only a modest number of experiments. Our analysis shows that the error reduction achieved from replacing the basic AL algorithm with a cost-aware algorithm can be significant, reaching up to 38% for the same computational cost of experiments.**

***Index Terms*—Active Learning, Performance Analysis, Gaussian Process Regression, Prediction Confidence**

I. Introduction

Active Learning (AL) is a set of Machine Learning techniques in which the learner interacts with the source of data being learned [1]. One use of AL is to query a human or run an "expensive" algorithm to label a datapoint for which correct labeling is difficult to predict. Intuitively, the learning algorithm makes decisions about which parts of the space it does not "know enough", and selectively invokes the expensive process to gather more supporting data to make predictions in that area. This type of learning is typically applied to classification problems, as described in [1]. The authors of [2] emphasize that only a limited number of studies have investigated AL applied to regression problems.

In this study, we apply techniques from AL to regression problems, creating regression models for performance and energy consumption data from parallel computing experiments. In this setting, our goal is to take data from existing empirical studies of the performance of HPC codes and perform regression analysis to discover the relationships between controlled variables (such as the size of the problem, the number and the speed of the used CPU, etc.) and the performance of the computation measured using such common metrics as runtime (in seconds) and total amount of consumed energy (in Joules). The constructed regression models can be used to predict the amount of time, energy, etc. that will be needed to run future computations, or as part of a performance model for a larger system. Depending on the complexity of the computation, each experiment might take seconds, minutes, or hours. Therefore, it might be infeasible to use experiment designs that run all or most of the combinations of variables (which can number in the hundreds, thousands, or more), making iterative designs that selectively run experiments that will yield the most information attractive. This is particularly true when the benchmarking budget is limited: for instance, there exists a fixed allocation on an HPC machine or a fixed maximum budget in a cloud environment.

To apply AL to regression analysis in performance studies, we need a process that both (a) produces regression models for datasets with multiple controlled and response variables and (b) produces estimates of the uncertainty associated with any point in the input space. We use Gaussian Progress Regression (GPR), also known as *kriging* in literature on geostatistics, for this purpose since it comes with a posterior probability distribution for predictions. With GPR, we can obtain the mean and the variance of the predictive distribution at every input point. The former is used to assess the quality of the investigated computational algorithm, while the latter is used in AL to select a sequence of points where the code associated with algorithm needs to run (we also investigate an AL algorithm which uses both for point selection).

We apply the selected AL techniques to a specific performance analysis problem where the performance of the HPGMG-FE benchmark [3], [4] is investigated. In this analysis, an impact of several controlled variables on performance metrics is studied based on measurements of performance and energy usage across a number of CPU and problem-size settings. We look at two different algorithms for guiding experiment selection: one that seeks to run experiments that are expected to purely reduce prediction uncertainty, and another one that weights experiments by the predicted cost (completion time) in order to learn the most information in the shortest amount of time. In the process of running these AL algorithms on a fixed dataset (which we collected by running a series of HPGMG-FE runs on available testbed hardware), we analyze the properties of the involved mathematical and

algorithmic components to ensure that the resulting system demonstrates reliability and optimality.

The main contributions of this study are:

- We propose a new framework for performance analysis based on Active Learning and Gaussian Process Regressions. This framework helps identify optimal sequences of experiments for reducing uncertainty about various quantities of interest.
- Under this framework, we develop a prototype which can be used to construct a number of diverse performance models, including models for application runtime, energy consumption, memory usage, and many others. We show how one can efficiently learn relationships between these metrics and multiple controlled variables.
- Using a dataset of measurements from over 3K parallel jobs, we compare the performance of two AL strategies, as well as their ability to reduce the prediction uncertainty for a given cumulative execution time. We confirm that the basic AL algorithm with no adjustment for the experiment cost can help efficiently construct regression models using only a modest number of experiments. However, the proposed cost-aware AL algorithm can provide significant advantages: for a range of total cost values, it will greatly reduce prediction errors, with the maximum reduction of 38% observed on the selected datasets.

The remainder of the paper is organized as follows: In Section II, we provide a brief summary of the related work. Section III introduces the model that we use to construct GPRs over sets of measurement points, and Section IV describes the particular technologies that we leverage to implement the model and also collect experimental performance and energy consumption data. Section V presents our application of several compelling AL algorithms to the collected datasets. We pay special attention to the AL "progress", and describe several metrics which help track how AL algorithms converge as the number of experiments increases. We conclude the paper and propose directions future work in Section VI.

II. Related Work

A. Active Learning and Regression Training

AL is also sometimes referred to as optimal experimental design. In [1], the author summarizes existing approaches in AL, primarily in the context of classification problems. The existing AL methods are referred to as *Query Synthesis*, *Stream-based Sampling*, and *Pool-based Sampling*.

In [5], the authors develop an AL system for learning regressions in studies where remote sensing data is analyzed to provide insights about biophysical parameters.

Regression analysis is typically used to fit a curve – linear, quadratic, cubic, etc. – through a dataset to predict future data. It is often assumed that the dependent variable, also known as *reponse*, behaves roughly according to one of these types of relationships with respect to controlled variables (sometimes called *features*). GPR provides a finer approach to regression learning where no such assumption is made and this relationship might be more complex (e.g., linear in one region of the input space and non-linear in adjacent regions). In GPR theory, the learning of a regressor is formulated in terms of a Bayesian estimation problem (summarized in Section III).

B. Performance Analysis

Raj Jain in [6] lays the foundation for experimental design in computer and network performance studies. The following classes of designs are described: simple designs (vary one factor at a time), 2^k full factorial design, and 2^{k-p} fractional factorial designs. Fractional factorial designs present a way to run less than the full number of possible experiments, but, unlike our work, do so in a *static* way: the set of desired experiments is determined *a prioi* from the factors and their levels. These designs do not change as measurements become available. The advantage to this type of design is that experiments can be arranged to specifically test for the effects (or lack of effect) of individual factors and combinations of factors, and multiple experiments can run in parallel. The downside is that the number of experiments does not typically represent the variation present in empirical data, and the selected experiment points do not necessarily represent the most "interesting" parts of the input space in terms of deviating from expected behavior. It is also difficult to create designs for factors that have more than two or three levels, and especially difficult when different factors have different numbers of level. The approach described in this paper is much easier to apply to experiments with many factor levels and is naturally adapted to running fewer experiments in situations where there is less variation in the results.

C. Response Surface Method

In [7], the authors introduce the Response Surface Method (RSM), which is also a type of "optimal experiment design" that is related to our work in that it helps choose future experiments based on the measurements recorded so far. However, it fundamentally differs from our work: we seek to characterize the entire problem space with reasonably high accuracy, while RSM is designed to search for combinations of factors that allow reaching specified goals. In this respect, it resembles an optimization process, where "sampling" from the space corresponds to running more (possibly expensive) experiments.

III. General Formulation for Non-deterministic Active Learning

In [2], the authors provide a foundation for AL in regression problems. They introduce the Expected Model Change Maximization (EMCM) scheme which is based on the following selection criterion:

$$x^* = \arg\max_{x \in pool} \left(\frac{1}{K} \sum_{k=1}^{K} \|(f(x) - f_k(x))x\| \right) \quad (1)$$

This formula uses differences between $f(x)$, the values predicted using the primary regression model learned from the data, and $f_k(x)$, the predictions obtained using k "weak"

learners, at points x. Functions $f(x)$ and $f_k(x)$ differ in the amount of training data they utilize: $f(x)$ is trained using all of the available data, while the ensemble $\{f_k(x)\}_{k=1..K}$ is trained with K subsets of the data generated via random sampling (with replacement). That study shows that the expression being maximized in (1) approximates the "model change" – the change of the regression parameters θ which occurs after a new datapoint x is added to the dataset and the model is retrained (where the model is fit to the data by minimizing the sum of squared differences between measurements and predictions).

Below we provide our analysis of EMCM and list the potential shortcomings of the method. It is difficult to use this method, as it is described, in performance studies for the following reasons:

- The quality of the learned regression models is evaluated using the widely-used Root Mean Squared Error (RMSE) metric:

$$RMSE = \sqrt{\frac{1}{|T|}\sum_{k=1}^{|T|}(f(x_i) - y_i)^2} \tag{2}$$

  where $|T|$ denotes the size of the test set and y_i is the ground truth of the example x_i. This evaluation, in principle, implies a single prediction $f(x_i)$ and a single ground truth value y_i for every value of x_i. A more general approach is required for problems where the regression learning algorithm produces a distribution of predictions (i.e. predicts the probability density function p_{x_i}) and where the algorithm needs to be compatible with datasets with multiple y values for the same x, representing repeated measurements of a noisy function.
- Once a point (x_i, y_i) is added to the training set, it is excluded from the pool of potential future candidates for AL. In contrast, when noisy functions are studied, AL algorithms need to be able to return to points already included in the dataset and recommend obtaining additional measurements if the variance observed or predicted at those points is high.

These limitations make it difficult to run EMCM on a wide range of datasets with measurements of performance, energy consumption, memory usage, and similar quantites. In many cases, the non-deterministic nature of computer and network performance requires obtaining repeated, often numerous, measurements in order to draw reliable conclusions. Moreover, the EMCM's K weak learners effectively provide a Monte Carlo estimate of variance, which is especially noisy when the training set is small.

An attractive alternative is documented in [5]. The authors leverage Gaussian Process Regression (GPR) for selecting the most "difficult" samples. That study relies on the Bayesian principle for estimating the posterior distribution of predictions given a simple prior for regression parameters. In that process, the authors pay special attention to *hyperparameters* – the parameters that define the prior distribution and characterize the noise in the data. Those hyperparameters have a significant impact on how GPR-based models are created, and influence what experiments are selected for learning in AL algorithms. However, that paper does not explain what algorithms can be used to find optimal values of GPR hyperparameters.

Chapter 5 in [8] provides a detailed description of optimization techniques for "model selection", which refers to the process of fitting hyperparameters. Two approaches are considered: the Bayesian inference with marginal likelihood and the cross-validation (CV) with pseudo-likelihood. While the former attempts to maximize the probability of the observations given the assumptions of the model, the latter typically uses the leave-one-out scheme (LOO-CV) and the squared error loss function. In its approach, the latter greatly resembles the EMCM method which constructs an ensemble of weak learners. Below we summarize the Bayesian hyperparameter fitting, the method we choose to focus on in this study (we leave the empirical comparison of the two methods for our future work).

Let X and y be the design matrix (where the columns are the vectors of input values) and the vector of the response measurements, respectively. The objective is to an find the underlying function $f(x)$ which best "explaines" the measurements which are assumed to include Gaussian noise:

$$y = f(X) + \mathcal{N}\left(0, \sigma_n^2\right), \tag{3}$$

where σ_n^2 is the variance of the noise. For an unknown input vector $x_\star$, the posterior distribution for individual predictions $f(x_\star)$ takes form of a multivariate Gaussian distribution:

$$p(f(x_\star)|x_\star, X, y) \sim \mathcal{N}\left(\mu_\star, \sigma^2 x_\star\right), \tag{4}$$

where

$$\mu_\star = k_\star^\mathsf{T} K_y^{-1} y, \tag{5}$$

$$\sigma_\star^2 = k_{\star\star} - k_\star^\mathsf{T} K_y^{-1} k_\star. \tag{6}$$

$$K_y = K + \sigma_n^2 I. \tag{7}$$

The critical role in the estimation of these parameters is played by the covariance function $k(x_p, x_q)$. A covariance matrix:

$$[K]_{ij} = k(x_i, x_j), \text{ for all columns } x_i \text{ and } x_j \text{ in } X, \tag{8}$$

a covariance vector:

$$[k_\star]_i = k(x_\star, x_i), \text{ for all columns } x_i \text{ in } X, \tag{9}$$

and a scalar,

$$k_{\star\star} = k(x_\star, x_\star) \tag{10}$$

are calculated using the selected function $k(x_p, x_q)$.

Both [5] and [8] describe the squared exponential (also referred to as the radial basis function or RBF) as a common choice of the covariance function:

$$k(x_p, x_q) = \sigma_f^2 \mathsf{exp}\left(-\frac{|x_p - x_q|^2}{2l^2}\right), \tag{11}$$

where the length scale l and the amplitude σ_f^2 are hyperparameters, along with the noise level σ_n^2. Appropriate values need to be selected for all three hyperparameters in order for

the regression to match the given dataset (X, y) and be able to produce reliable predictions.

According to the Bayesian inference with marginal likelihood, the following quantity, referred to as the *log marginal likelihood* (LML), is minimized with respect to the hyperparameters:

$$\log p(y|X, l, \sigma_f^2, \sigma_n^2) = -\frac{1}{2}\left(y^{\mathsf{T}} K_y^{-1} y + \log|K_y|\right) + \mathsf{C}, \quad (12)$$

$$(l, \sigma_f^2, \sigma_n^2) = \underset{l,\sigma_f^2,\sigma_n^2}{\arg\min}(\log p(y|X, l, \sigma_f^2, \sigma_n^2)). \quad (13)$$

After solving this optimization problem and fitting the hyperparameters, we can use the constructed GPR model to obtain the mean and the variance of predictions for every input point x. Instead of the Monte Carlo criterion (1), at every iteration of AL we propose choosing x for which the variance of predictions is the highest, with or without adjusting for the expected cost. In Section V, we present two AL algorithms which we develop based on this idea and describe their performance. Our analysis shows that the proposed AL algorithms run as expected even when the number of data points is small, for instance, only a single measurement of a quantity of interest is available at the beginning (the situation where EMCM is unlikely to perform well).

IV. Implementation

We developed a prototype capable of choosing experiment candidates based on AL with GPR for noisy performance and power data. The prototype, which is implemented in Python, allows us to analyze characteristics of large collections of computational jobs, as well as provides insights into properties of the implemented AL algorithms. Although a detailed analysis of computational requirements and the scalability of these algorithms is out of the scope of this paper (planned for further research), we successfully analyzed numerous sequences with hundreds and thousands of experiments.

The prototype, given a dataset with the design matrix X and the vector of response values y, partitions it into 3 sets: Initial (for initial regression training), Active (for one-at-a-time experiment selection with AL), and Test (for prediction quality analysis). In our evaluation, we use random partitioning with selected ratios between these sets. Thus, we typically used the Initial set with a single experiment, which corresponds to realistic scenarios where in many performance studies an application is first run on a new platform to verity correctness and the subsequent runs are used to evaluate the performance. The Active and Test sets in our analysis split the remaining experiments roughly with the 8:2 ratio.

In addition to single realizations of AL, our prototype is capable of running batches of random partitions of the same dataset. The aggregate results, such as the average error and the average cumulative cost of experiments, provide insights into how the AL process behaves independent of the initial state (i.e. the partition properties) and allows us to compare alternative algorithms for candidate selection.

To test the prototype and investigate the available tuning mechanisms in the developed AL algorithms, we ran the prototype on two datasets which we collected from running parallel computational jobs on the hardware available for short-term dedicated experiments and provisioned via the CloudLab portal [9]. Below we describe in detail the leveraged technologies; in Section V-A we characterize the collected datasets and provide our intuitions about them.

A. Leveraged Technologies

In order to gather practical empirical data for our analysis, we leveraged several existing technologies:

- **CloudLab** [10]: Performance and power data were gathered on CloudLab, a facility that gives users exclusive bare-metal access to compute, storage, and network resources. Because access is exclusive, we could be certain that the data collected from our experiments was not significantly impacted by other users of the facility. CloudLab also provides power measurements at the server-level, a critical feature for the energy consumption modeling in this study. We collect power traces with frequent recordings of the instantaneous power draw (in Watts) from the on-board IPMI sensors and infer per-job energy consumption estimates (in Joules) using the recorded timestamps.
- **HPGMG** [3]: A robust and diverse benchmark that is designed to be representative of modern HPC applications. This benchmark is used to rank supercomputers and HPC systems based on geometric multigrid methods. We ran HPGMG-FE, the compute- and cache-intensive component which solves constant- and variable-coefficient elliptic problems on deformed meshes using Full Multigrid.
- **scikit-learn** [11]: A Python module with efficient tools for Machine Learning and data analysis. Specifically, we leverage the code for Gaussian Processes [12] in the latest development version (0.18.dev0).

We used the Wisconsin cluster on CloudLab to instantiate a 4-node homogeneous environment where every physical machine is equipped with 2 8-core Intel E5-2630 v3 Haswell CPUs, 128GB of RAM, and 10Gb NICs (full hardware description is provided at [13]).

We constructed a fully functional compute cluster out of the provisioned individual machines by installing and configuring the following software stack: NFS for sharing job input and output between the nodes, OpenMPI (ver. 1.10.0) for message passing, and SLURM (ver. 15.08) for resource management and scheduling. The latest stable versions of the PETSc library [14] and the HPGMG code were built and executed. In this configuration process, we relied on the infrastructure related to the Chef configuration management system [15] – the Chef-Cluster profile and the relevant cookbooks (bundles of installation and configuration scripts) developed in the previous work and documented in [16].

HPGMG-FE jobs with different parameters (e.g., operator type, problem size, etc.) were organized into batches and submitted to the job queue, after which SLURM managed

TABLE I: The Parameters of the Analyzed Datasets.

	Dataset: **Performance**	Dataset: **Power**
# Jobs	3246	640
Responses	Runtime (S)	Runtime (S), Energy (J)
Runtime, S	0.005 - 458.436	0.005 - 458.436
Energy, J	-	6.4e3 - 1.1e5
Variables	Operator: poisson1,poisson2,poisson2affine Global Problem Size: 1.7e3 - 1.1e9 NP: 1,2,4,8,16,24,32,48,64,96,128 CPU Frequency (GHz): 1.2,1.5,1.8,2.1,2.4	

their execution on the available nodes. After completion, we collected all relevant information, including benchmark output, error logs, SLURM accounting information, power consumption traces, and system information. Then, we transferred that data from the cluster to a local workstation to plot it and analyze it using the developed AL algorithms.

V. Evaluation

A. Understanding the Dataset

For experimental evaluation, we run our prototype "offline", consulting a database with the collected data. We demonstrate the advantages which AL algorithms can provide on these or similar datasets, while the target use case for practical applications is the "online" operation, where every iteration of AL includes selecting an experiment, running it, and using the experiment outcome to update the underlying GPR model.

Table I provides a high-level summary of the collected datasets. It lists all considered responses and controlled variables available in the datasets. It is worth mentioning that the intervals in which Runtime, Global Problem Size, and NP (number of processes) change are relatively large, allowing us to explore the behavior of the application across a domain where the growth along some of the dimensions is significant. For instance, the runtime increases from its smallest recorded value to the largest by growing 5 orders of magnitude. Also, CPU Frequency varies from 1.2 to 2.4 GHz, which are the low and the high limits on the selected machines' CPUs. The datasets include up to 3 repeated experiments for every combination of the controlled variables. The complete datasets, with up to 46 attributes for each job – controlled variables, job execution properties reported by SLURM (e.g., memory usage on every node), and the listed responses – are available in the CSV format at [17].

In order to validate our intuitions about this data, we fix the Operator variable (choose "poisson1"), select several levels of NP, and show these subsets on 3D plots in Fig. 1. These plots demonstrate that the variance in the Power dataset is much higher comparing to the Performance dataset. There are also fewer points in the Power dataset due to the fact that the collected power traces from the utilized machines had gaps and we excluded a number of jobs with insufficient number of corresponding power draw records (less than 10 for 60 seconds of computation). The per-job Total Energy estimates, obtained from the traces of instantaneous power draw via numerical integration, can be interpreted as approximation of the actual per-job energy consumption. Calibration of these estimates using measurements from a set of physical electrical power meters falls beyond the scope of the current analysis; it constitutes a context for an appealing future study where, in the AL terms, experiments for which physical measurements are available can be used in the modeling with higher confidence, while the derived IPMI-based estimates are marked with lower confidence and the corresponding experiments are recommended for repeated execution.

Fig. 2 shows the same subsets with log-transformed responses. The plot for the Performance dataset confirms the linear growth of Runtime along the problem size dimension, for which the plot also uses the log-transformed scale. One downside of this log-transformed representation is that it becomes difficult to see the growth along the frequency dimension, therefore, the original, non-log plot needs to be always referenced when AL decisions and GPR predictions in this space are analyzed. As we can see, log-transformation does not significantly change the structure of the Power dataset.

In the remainder of the paper, we describe how we use these datasets with log-transformed Runtime, Energy, and Global Problem Size for GPR and AL with one and two controlled variables.

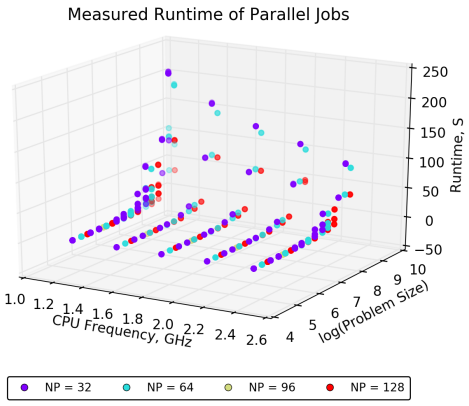
B. Regression Training and Active Learning

We aim to gain confidence about the reliability of the proposed AL algorithms by running our prototype on simple problems. We ensure that it works in 1D and 2D scenarios: the prototype receives subsets of the datasets where all but one and two controlled variables are fixed. Initially, we gain intuitions about the created one-variable GPR models by inspecting their visualizations (shown in Fig. 3). Then, we switch to the two-variable GPRs (depicted in Fig. 5) and describe the behavior of AL for pure variance reduction (without adjusting for the expected cost). An alternative strategy, which uses a cost efficient selection algorithm is developed and compared to the original strategy. Throughout this section we document the outlined progression and summarize our observations.

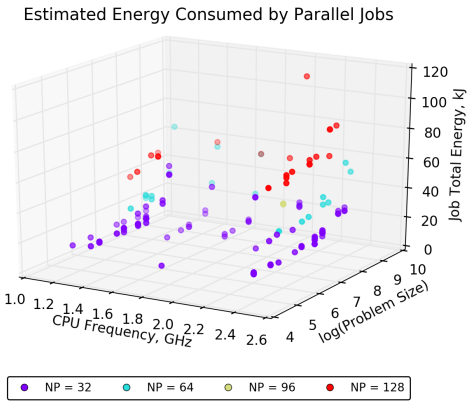
1) 1D – Varying Problem Size: In 1D case, we choose to model how Runtime and Energy respond to changes in the Global Problem Size. We fix the rest of the variables (NP=32, Freq=2.4, Operator='poisson1'), and use GPR to produce a regression with associated prediction mean and variance as functions of the problem size.

Fig. 3(a) shows four GPRs with different values of two hyperparameters, the length scale l and the amplitude σ_f, which define the shape of the squared exponential defined in (11). The difference between the predictive mean curves is negligible. In contrast, the 95% confidence intervals (shown as: mean $\pm$ 2*standard deviation) are greatly impacted by the value of l: a decrease of l leads to a significant increase in the uncertainty, characterized by the standard deviation of predictions, in areas between measurement points.

In Fig. 3(b), we can see that the growth of uncertainty can be exaggerated at the edge of the domain if there is no measurement nearby. "Clamped" at the existing points, the predictive distribution is so wide at the maximum problem size that we

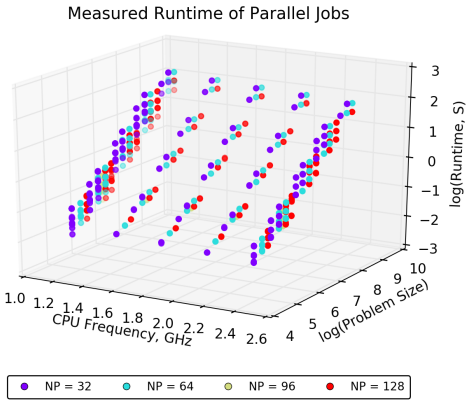


(a) Subset of jobs in *Performance*

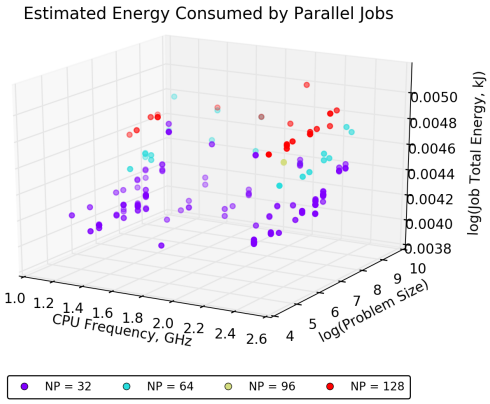


(b) Subset of jobs in *Power*

Fig. 1: Visualization of subsets from the analyzed datasets.



(a) Subset of jobs in *Performance*



(b) Subset of jobs in *Power*

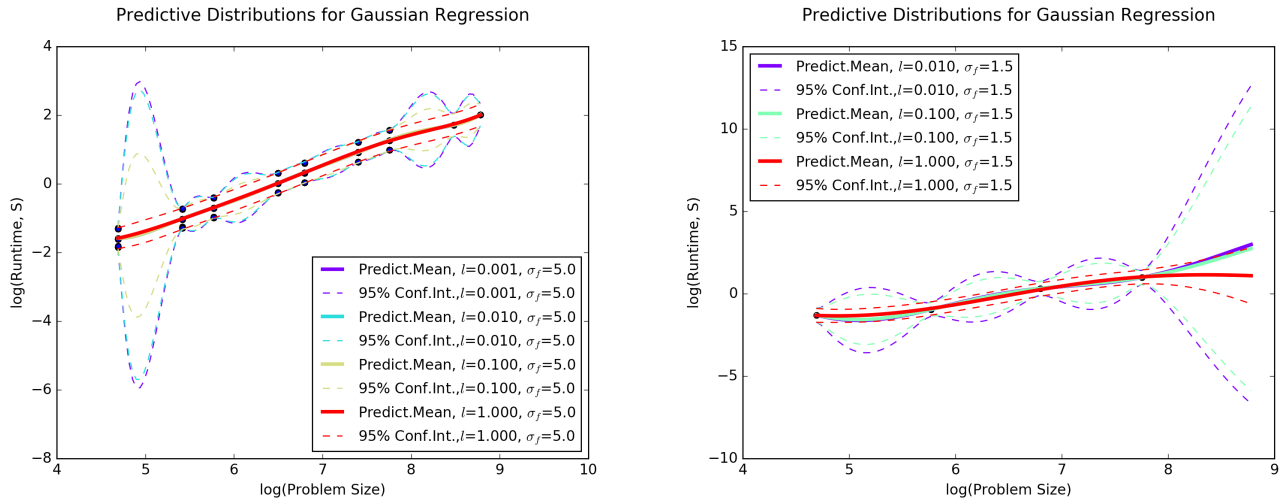
Fig. 2: Jobs from Fig. 1 shown with log-transformed Runtime and Energy.

observe the difference not only for the confidence intervals but also for the predictive means with different hyperparamters. In choosing values of hyperparameters to best fit the datasets, we rely on the scikit-learn's implementation which runs gradient ascent on the LML as described at [12]. This implementation finds hyperparameter values from a domain with specified boundaries and, in order to increase reliability, repeats this search multiple times, each time starting from a random point.

It is worth noting that when GPR is created using a small number of points, it may be erroneously assumed that the measurements are noise-free because the dataset does not include multiple y values for the same x. The produced GPR will optimistically consider its predictions to be exact, and the selected noise level may approach the machine precision. Further discussion of this point is provided in Section V-B4.

In general, GPRs constructed on datasets with many points should be more reliable. As described in [8], LML becomes more peaked with the growth of the dataset size. We can see that finding the peak in the landscape shown in Fig. 4, which corresponds to the Performance subset from Fig. 3(a), is a straightforward optimization problem with a unique global optimum. This peak can be found using gradient ascend with a single randomly selected starting point.

2) 2D – Varying Problem Size and Frequency: Instead of curves, we obtain GPR surfaces when we vary two factors. For each GPR solution, defined by a set of specific hyperparameters, we obtain 3 surfaces: the high bound for the confidence interval, the predictive mean, and the low bound for the confidence interval. Fig. 5(a) shows the bounds with green wireframes and the mean with the red wireframe between the two bounds. In this particular case, the four randomly selected training points are aligned with each other well enough to define a tight predictive distribution. Nowhere in this space the confidence interval is particularly wide. However, further away from the training points, e.g., where both Frequency and Problem Size are near their maximum values, the confidence



(a) All Selected Measurements

(b) Random Subset of 4 Points

Fig. 3: Predictive distribution for 1D cross section of *Performance* dataset

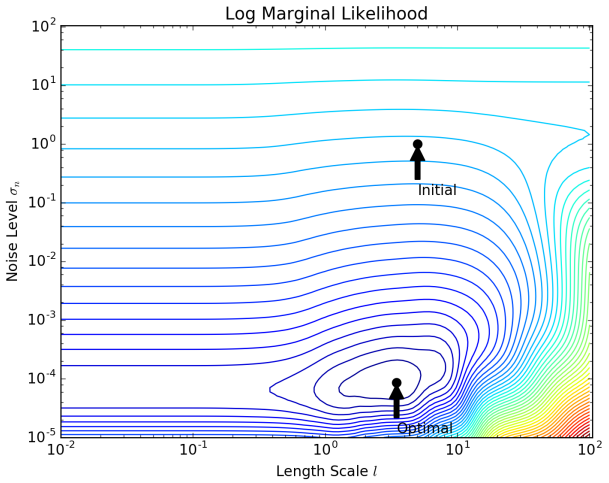


Fig. 4: Contour plot of LML as a function of hyperparameters l and σ_n.

interval bounds are further apart. These are the areas where AL should select candidates for subsequent experiments. All experiment candidates are depicted on this plot with vertical magenta lines connecting the confidence interval surfaces.

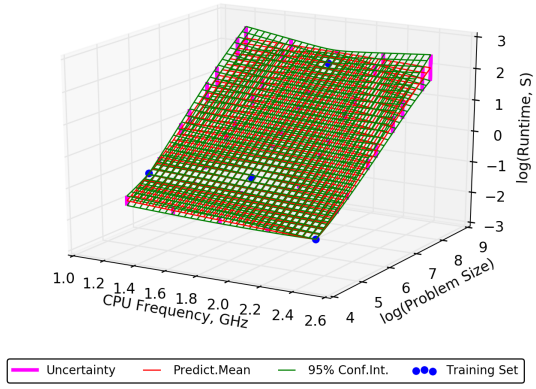
Fig. 5(b) depicts LML for this GPR. Comparing to the aforementioned landscape for the case with abundant data, this one is significantly more shallow. However, the identified peak yields a GPR model which behaves as expected: it has a reasonable predictive distribution, and, we are confident that its standard deviation can be used as a selection criterion in AL.

3) Monitoring Active Learning Progress: When we implement an Active Learning algorithm it is important to confirm that it indeed suggests an "interesting" sequence of experiments for learning. In other words, we need to monitor the steps taken by the algorithm and, at least in simple cases, compare its behavior with our expectations. In more difficult cases, e.g., when the number of varied variables is greater than two and we cannot easily visualize AL trajectories, we need to track metrics that characterize the learning progress. Such metrics will be used in making practical decisions such as whether to keep AL running or terminate it because the desired prediction accuracy is achieved.

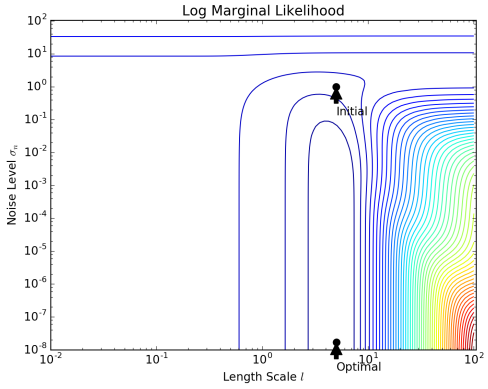
We start by visualizing how AL chooses points in the 2D input space where Problem Size and Frequency are varied. Unlike the previous example, we consider a much larger dataset: a subset of points from the Performance dataset for which NP = 32 and Operator='poisson1'. This selection yields 251 jobs, which we randomly split into the Initial, Active, and Test sets. Fig. 6 depicts these sets and visualizes how AL suggests exploring this domain for 10 and 100 iterations. Each transition from one point to another in these sequences is shown with a colored arrow, starting with the blue-colored and progressing to the green-colored arrows. We can see that initially points from the middle of the domain are not selected. In a star-like pattern, AL chooses experiments at the edges and, only after exhausting all edge points, progresses toward the middle. This is exactly the type of exploration that is intuitively employed by human experimenters, typically when there is no understanding of the relationships captured in the studied data, and often described in performance studies.

4) Interpretation of Results and Discussion: To understand AL's progress, we monitor 3 selected quantities as follows:

- $\sigma_{f(x)}$ – Standard Deviation (SD) of the prediction distribution at selected AL candidates (should not be confused with σ_f, one of the GPR hyperparameters). In the AL algorithm discussed so far, these are the highest values found across all candidate points.
- $\frac{1}{|L|}\sum_{i=0}^{|L|}\sigma_{f(x)}$ – Arithmetic Mean of the Standard Deviation (AMSD) calculated across all points in the Active Set. Our initial analysis shows that a geometric mean can

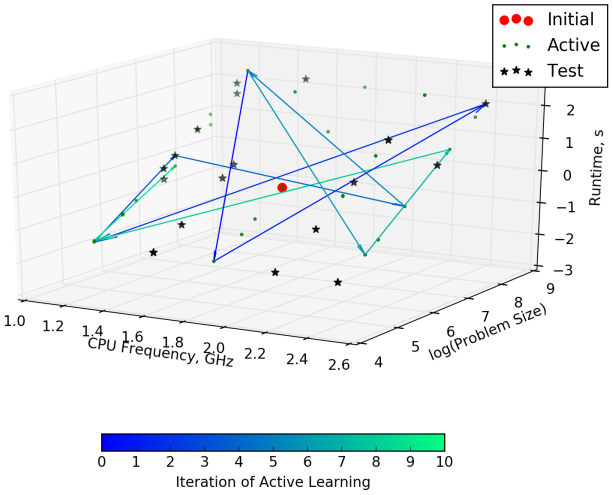


(a) Regression trained on a small dataset

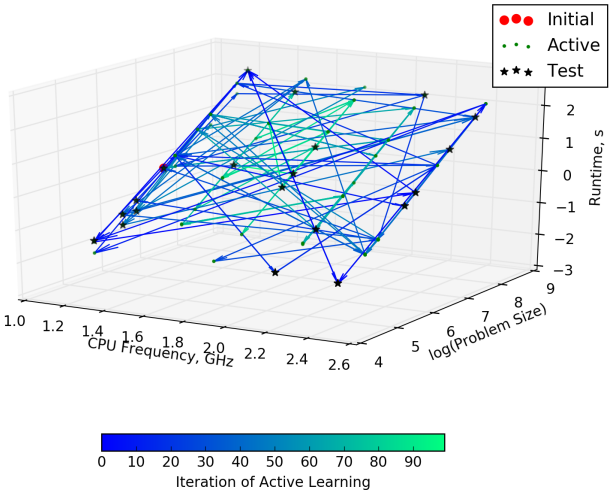


(b) Contour plot of a shallow LML function

Fig. 5: GPR for a small dataset with two controlled variables



(a) 10 Iterations



(b) 100 Iterations

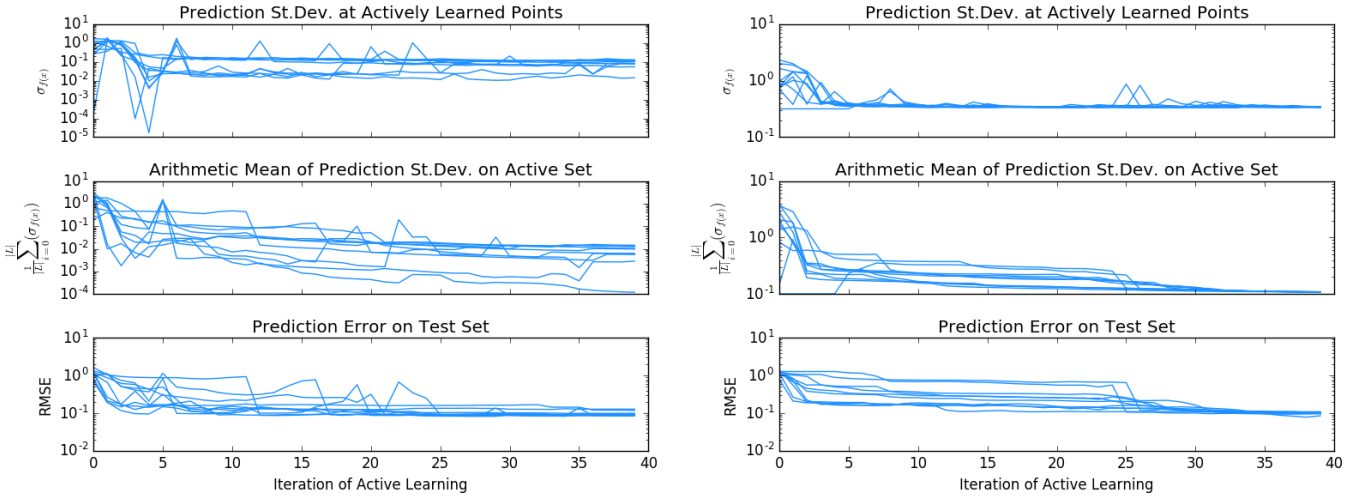
Fig. 6: AL with Variance Reduction.

be used as an alternative metric, but it does not provide any significant advantages.

- RMSE: Aggregate error on the Test set defined by (2).

All three metrics are evaluated for 10 repetitions of AL for different random partitions of the same Performance subset, and the results are depicted in Fig. 7(a). We immediately notice that all three quantities seem to converge to their stable values after about 25 iterations (with small subsequent variations). However, we consider inadequate the behavior where $\sigma_{f(x)}$ drops to negligible values before the 5th iteration. Also, AMSD in many of the shown trajectories decreases significantly below its stable value of approximately 10^{-2}. We explain this effect by the model overfitting: from the GPR's perspective, a regression with a very small variance is produced. This happens because with fewer than 5 points the measurements often align well and the mean and the confidence interval bounds are close to each other. Even with more points, it is still possible to obtain random partitions containing no outliers or particularly skewed measurements, which can results in non-optimal AL decisions.

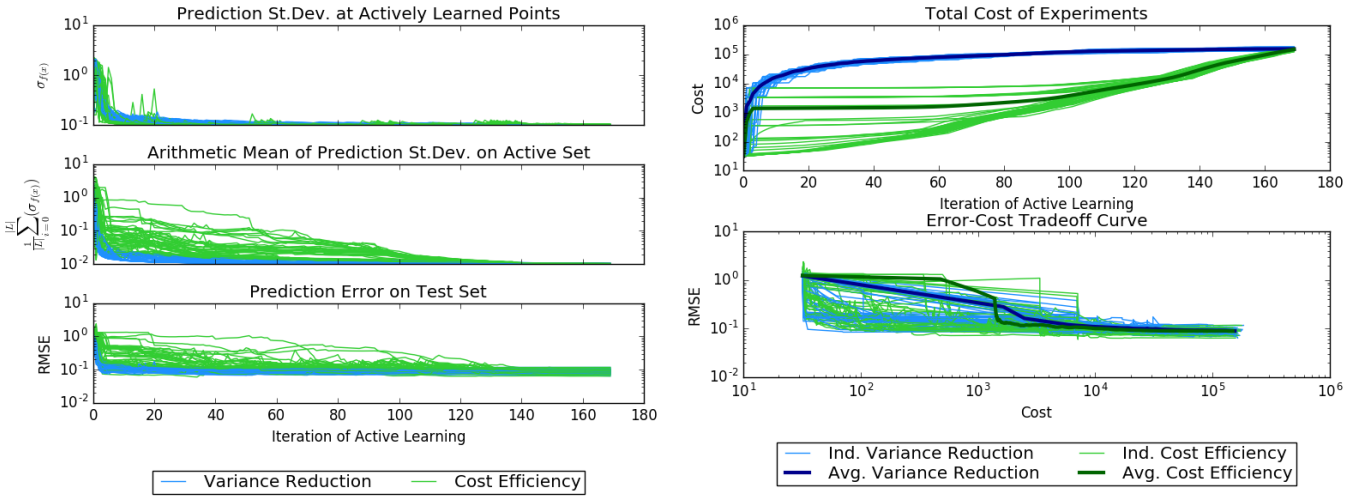
To eliminate the overfitting issue, we restrict the search space for the hyperparameter σ_n in GPR. Specifically, we increase the lowest bound for the values which can be considered in gradient ascent. In contrast with Fig. 7(a), which shows properties of GPR with $\sigma_n \geq 10^{-8}$, we generate 10 new trajectories with this limit increased to 10^{-1} and show them in Fig. 7(b). The new trajectories do not demonstrate the aforementioned downsides. In this setting, AMSD becomes an important practical measure: when it converges (i.e. the average does not change significantly with additional AL



(a) $\sigma_n \geq 10^{-8}$

(b) $\sigma_n \geq 10^{-1}$

Fig. 7: Strong influence of the limit on the noise-level σ_n on the quality of AL. The increased limit helps eliminate the overfitting problem.



(a) Reduction of Error and Uncertainty

(b) Growth of Cumulative Cost and Cost-Error Tradeoff Curves

Fig. 8: Comparing AL strategies: Variance Reduction and Cost Efficiency.

iterations), AL can be terminated. The plots confirm that at that point RMSE will also converge to its stable value, and subsequent experiments may be considered excessive.

While the described fixed limit for σ_n results in the desired behavior in our experiments, we believe that a more general solution should involve a limit that dynamically adjusts. For instance, we expect that the restriction: $\sigma_n \geq 1/\sqrt{(N)}$, where N is the iteration counter, is a viable choice. In future work, we will investigate such limits and their implications.

AL which selects points with the highest SD is the approach which we refer to as Variance Reduction. Below we discuss our experience with an alternative cost-aware AL approach. In the cost-aware algorithm, rather than minimizing the variance we attempt to minimize the variance/cost ratio. While it may not be entirely clear how to define the cost in many other application domains, in performance analysis and computational science, certainly, longer experiments are more "expensive". Therefore, Runtime and Total Energy, the same quantities we are predicting, can be interpreted as cost approximations, as well as different measures of the resource utilitzation.

Since we work primarily with the log-transformed responses, the proposed variance/cost ratio takes the form:

$$x^* = \underset{x \in pool}{\arg\max} \left(\sigma_{f(x)} - \mu_{f(x)} \right), \tag{14}$$

where $\mu_{f(x)}$ and $\sigma_{f(x)}$ are the mean and the SD of the predictive distribution at point x. This selection criterion defines the AL algorithm which we refer to as Cost Efficiency. We expect that this algorithm will be less aggressive about SD minimization and lean toward smaller experiments rather than larger ones where such choice is appropriate.

We integrated the described algorithm into our prototype and performed the initial analysis. We ran both Variance Reduction and Cost Efficiency on 50 random partitions of the Performance subset shown in Fig. 6(b). Fig. 8(a) illustrates different trajectories of individual learning sequences. As expected, RMSE for Cost Efficiency does not converge as quickly as for Reduction Variance, neither does AMSD (we again used $\sigma_n \geq 10^{-1}$), but both quantities converge after approximately the same number of iterations. However, the cost for these algorithms, shown in Fig. 8(b), confirms that Cost Efficiency can be the algorithm of choice in cost-sensitive and cost-limited studies. The tradeoff curves shown in the same figure are meant to act as a practical tool for selecting one algorithm over the other. In other words, it can help an experimenter who relies on AL suggestions to see how aggressive the available selection algorithms can minimize the error for a given cost. What we infer from the intersecting curves is that Cost Efficiency initially selects several points that on average do not efficiently reduce the error. In the longer term, by learning from a larger set of smaller experiments as opposed to a smaller set of larger experiments, Cost Efficiency becomes the superior algorithm (lower cost and lower error), at least for a subrange of cost values. In the shown example, the curves intersect at the value of the cumulative cost of $C = 1626$ (total compute time in seconds * number of cores). After that point, Cost Efficiency outperforms Variance Reduction, where the relative difference between the two algorithms for any given cost reaches up to 38%. White it is difficult to see it on the log-scaled plot, this advantage is also significant at many other points: at $2 * C$, $3 * C$, $5 * C$, and $10 * C$ it is 25%, 21%, 16%, and 13%, respectively. Eventually, the curves meet at the maximum cost, the point where all available experiments are used to create GPR models.

VI. Conclusions and Future Work

This paper has shown how Active Learning can be effectively applied to regression problems in performance analysis. By combining AL and GPR, we have created a method that allows us to get high-confidence predictions across a large input space without the need for a static, and possibly inefficient, experiment design. This means that performance regression against expensive computations or with larger parameter sets, become more feasible.

The experiments in this study have been run "offline" (consulting a database of precomputed performance samples), but the target use case is "online" where the next experiment must be scheduled. As future work, some experiments could reasonably be run in parallel which adds additional scheduling concerns and may indicate a less greedy selection strategy. Realistic simulations often involve continuous or near-continuous parameters, such that the active set cannot be treated as finite. We expect that this could be handled by choosing the best option within a finite subset or, preferably, by using continuous optimization. Gradient-based methods, which are available with GPR, would provide an important benefit for problems with high-dimensional parameter spaces. We expect that the techniques in this paper can also be generalized to perform cost-aware adaptive reduced order modeling for application-relevant response surfaces. In a separate study, we plan to investigate computational requirements of competing GPR and AL algorithms and consider available optimizations.

Acknowledgments

We would like to thank the CloudLab and Emulab staff at the University of Utah for their feedback on running experiments on CloudLab, configuring the provisioned resources into compute clusters, and the development of Chef profiles and cookbooks. We would also like to thank our collaborators at the University of Massachusetts Amherst, Prof. Michael Zink and Emmanuel Cecchet, for providing access to the power data and continually improving the logging infrastructure.

References

[1] B. Settles, "Active learning literature survey," *Computer Sciences Technical Report*, vol. 1648, 2010.

[2] W. Cai, Y. Zhang, and J. Zhou, "Maximizing expected model change for active learning in regression," in *2013 IEEE 13th International Conference on Data Mining*, Dec 2013, pp. 51–60.

[3] M. Adams, J. Brown, J. Shalf, B. van Straalen, E. Strohmaier, and S. Williams, "HPGMG: High-performance geometric multigrid." [Online]. Available: https://hpgmg.org

[4] M. Adams, J. Brown, J. Shalf, B. V. Straalen, E. Strohmaier, and S. Williams, *HPGMG 1.0: A Benchmark for Ranking High Performance Computing Systems*, May 2014. [Online]. Available: http://www.osti.gov/scitech/servlets/purl/1131029

[5] E. Pasolli and F. Melgani, "Gaussian process regression within an active learning scheme," in *Geoscience and Remote Sensing Symposium (IGARSS), 2011 IEEE International*, July 2011, pp. 3574–3577.

[6] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley, 1990. [Online]. Available: https://books.google.com/books?id=eOR0kJjgMqkC

[7] P. J. Whitcomb and M. J. Anderson, *RSM Simplified: Optimizing Processes Using Response Surface Methods for Design of Experiments*. Productivity Press, 2004.

[8] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[9] CloudLab, "Cloudlab portal," https://cloudlab.us/, accessed: 2016-08-01.

[10] R. Ricci, E. Eide, and The CloudLab Team, "Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications," *USENIX ;login:*, vol. 39, no. 6, Dec. 2014. [Online]. Available: https://www.usenix.org/publications/login/dec14/ricci

[11] "scikit-learn - machine learning in python." [Online]. Available: http://scikit-learn.org

[12] "scikit-learn-0.18.dev0 - gaussian processes." [Online]. Available: http://scikit-learn.org/dev/modules/gaussian_process.html

[13] "The cloudlab manual - hardware." [Online]. Available: http://docs.cloudlab.us/hardware.htm

[14] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, K. Rupp, B. F. Smith, S. Zampini, H. Zhang, and H. Zhang, "PETSc Web page," http://www.mcs.anl.gov/petsc, 2016. [Online]. Available: http://www.mcs.anl.gov/petsc

[15] Chef, "An overview of chef," https://docs.chef.io/chef_overview.html, accessed: 2016-08-01.

[16] D. Duplyakin and R. Ricci, "Introducing configuration management capabilities into CloudLab experiments," in *Proceedings of the International Workshop on Computer and Networking Experimental Research Using Testbeds (CNERT)*, Apr. 2016. [Online]. Available: http://www.flux.utah.edu/paper/duplyakin-cnert16

[17] D. Duplyakin, "Performance and power dataset for active learning." [Online]. Available: http://dmitry.duplyakin.org/al-performance-and-power-dataset.html