Active Sampling for Accelerated Learning of Performance Models

Piyush Shivam, Shivnath Babu, Jeffrey S. Chase
Duke University, Durham NC 27708
{shivam,shivnath,chase}@cs.duke.edu

Abstract

Models of system behavior are useful for prediction, diagnosis, and optimization in self-managing systems. Statistical learning approaches have an important role in part because they can infer system models automatically from instrumentation data collected as the system operates. Work in this domain often takes a "given the right data, we learn the right model" approach, and leaves the issue of acquiring the "right data" unaddressed. This gap is problematic for two reasons: (i) the accuracy of the models depends on adequate coverage of the system operating range in the observations; and (ii) it may take a long time to obtain adequate coverage with passive observations. This paper describes our approach to bridging this gap in the NIMO system, which incorporates *active* learning of performance models in a self-managing computing utility.

1 Introduction

Statistical learning techniques (SLT) have potential to simplify a variety of system management tasks. Several recent systems have used SLT to infer quantitative models of application performance to enable effective resource allocation for repeated workloads. For example, [3] exposes potential causes of degraded transaction response time by identifying system-level metrics correlated with response time. In databases, [8] uses SLT to cost query-execution plans, enabling a query optimizer to adapt automatically to changes in workloads and system environments. *NIMO* [6] learns models to predict performance of frequently used applications (initially batch compute tasks) on heterogeneous compute and storage resources.

The general approach taken in these systems and many other related works is to transform the specific system-management problem to a problem of learning a statistical model that fits a set of m *sample* data points collected as the system operates. Each sample x_i ($1 \leq i \leq m$) is a point in a high-dimensional space with n dimensions (attributes) $X_1, X_2, \ldots, X_n$. For example, a sample s in NIMO represents a complete run of a task graph G on a set $\vec{R}$ of resources assigned to run G; s has the general form $\langle \rho_1, \rho_2, \ldots, \rho_k, t \rangle$, where each ρ_i is a hardware attribute of $\vec{R}$ (e.g., CPU speed or disk seek time), and t is the completion time of G on $\vec{R}$.

Given the set of m samples $x_1, \ldots, x_m$, an appropriate model can be fitted to the data. For example, NIMO learns multi-variate regression models that can predict the completion time t from the hardware attributes $\langle \rho_1, \rho_2, \ldots, \rho_k \rangle$ of the resources assigned to run G. Some challenges arise in this setting:

- *Limitations of passive sampling.* Samples collected only by passive observations of system behavior may not be representative of the full system *operating range*—e.g., system behavior on a flash crowd may never be observed—limiting the accuracy of models learned from such samples.
- *Cost of sample acquisition.* Acquiring a sample may have a nonnegligible cost. For example, a sample $\langle \rho_1, \rho_2, \ldots, \rho_k, t \rangle$ in NIMO "costs" time t to acquire. High costs—e.g., for long-running batch tasks—limit the rate at which samples can be acquired for learning.[1]
- *Curse of dimensionality.* As the *dimensionality* (n) of the samples increases, the number (m) of samples needed to maintain the accuracy of the learned model can increase exponentially.

In NIMO, we seek to address these challenges through an *active sampling* of candidate resource assignments chosen to *accelerate* convergence to an accurate model.

Active (or proactive) sampling acquires samples of system behavior by planning *experiments* that perturb the target system to expose the relevant range of behavior for each application G. In NIMO, each experiment deploys G on a sample candidate assignment $\vec{R}$, either to serve a real request, or proactively to use idle or dedicated resources (a "workbench") to refine the model for G.

Active sampling with acceleration seeks to reduce the time before a reasonably accurate model is available, as depicted in Figure 1. The x-axis shows the progress of time for collecting samples and learning models, and the y-axis shows the accuracy of the best model learned so far. While passive sampling may never collect a fully representative set of samples, active sampling without acceleration may take longer to converge to accurate models.

This paper outlines NIMO's approach to active sampling for accelerated learning.

[1] Acquiring samples corresponding to a mere 1% of a 10-dimensional space with 10 distinct values per dimension and average sample-acquisition time of 5 minutes, takes around 951 years!

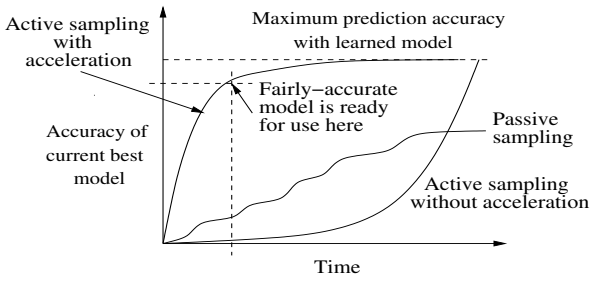

Figure 1: Active and accelerated learning

2 Overview of NIMO

The *NIMO (NonInvasive Modeling for Optimization)* system generates effective resource assignments for *batch workflows* running on networked utilities. Figure 2 shows NIMO's overall architecture consisting of: (i) a *workbench* for doing experiments and collecting samples; (ii) a *modeling engine* composed of a *resource profiler* and an *application profiler* that drives sample collection to learn performance models for batch workflows; and (iii) a *scheduler* that uses learned performance models to find good resource assignments for batch workflows. NIMO is *active*: it deploys and monitors applications on heterogeneous resource assignments so that it can collect sufficient training data to learn accurate models. NIMO is also *noninvasive*: it gathers the training data from passive instrumentation streams, with no changes to operating systems or application software.

A batch workflow G input to NIMO consists of one or more batch *tasks* linked in a directed acyclic graph representing task precedence and data flow (e.g., [1]). In this paper, we speak of G as an individual task, although the approach generalizes to build models of workflows composed of multiple tasks. NIMO builds a performance model $M(G, \vec{R})$ that can predict G's completion time on a *resource assignment* $\vec{R}$ comprising the hardware resources (e.g., compute, network, and storage) assigned simultaneously to run G. Note that accurate prediction of completion time is a prerequisite for many current provisioning and scheduling systems, e.g., [2].

NIMO models the execution of a task G on a resource assignment $\vec{R}$ as a multi-stage system where each *stage* corresponds to a resource $R_i \in \vec{R}$. We characterize the work done by G in the ith stage as G's "occupancy" o_i of R_i defined as the average time spent by G on R_i *per unit of data*. Intuitively, o_i captures the average delay incurred on R_i by each unit of data flow in the multi-stage system. For a known volume of data flow D, G's completion time t is given as $D \times \sum_{i=1}^{i=n} o_i$ for an n-stage system. G's *application profile* captures the application behavior resulting from G's interaction with its resources $\vec{R}$, whose properties are characterized by $\vec{R}$'s *resource profile*.

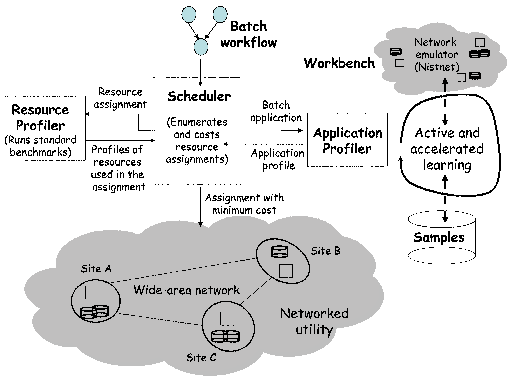

Figure 2: Architecture of NIMO

Resource Profile. The resource profile $\vec{\rho}$ is a vector of hardware attributes $\langle \rho_1, \rho_2, \ldots, \rho_k \rangle$ of $\vec{R}$ that are measurable independently of any application. For example, the resource profile of a compute resource may contain the processor speed and memory latency as attributes. NIMO generates resource profiles by running standard benchmark suites on the resources.

Application Profile. G's application profile is a vector of *predictors* $\langle f_1(\vec{\rho}), f_2(\vec{\rho}), \ldots f_n(\vec{\rho}) \rangle$ where $f_i(\vec{\rho})$ predicts G's occupancy o_i of resource $R_i \in \vec{R}$ as a function of the resource profile $\vec{\rho}$ of assignment $\vec{R}$, $o_i = f_i(\vec{\rho})$. Section 3 explains how NIMO builds each predictor automatically through active and accelerated learning.

3 Active Learning of Predictors

NIMO's application profiler uses active sampling for accelerated learning of predictors comprising a task G's application profile. The application profiler uses a workbench consisting of heterogeneous compute and storage resources and a network link emulator (*nistnet*). Learning a predictor $f_i(\vec{\rho})$ from training data is nontrivial because of several reasons: (i) the curse of dimensionality—resource profile $\vec{\rho} = \langle \rho_1, \ldots, \rho_k \rangle$ may contain many attributes; (ii) the training data must expose the full system operating range; and (iii) the high cost of data acquisition—collecting a sample $\langle \rho_1, \ldots, \rho_k, o_1, \ldots, o_n \rangle$, where $\langle o_1, \ldots, o_n \rangle$ represents G's occupancies on assignment $\vec{R}$ with profile $\langle \rho_1, \ldots, \rho_k \rangle$, involves running G to completion on $\vec{R}$. (Recall that f_i predicts o_i as a function of a subset of attributes in $\rho_1, \ldots, \rho_k$.)

Algorithm 1 summarizes the main steps to learn the predictors. (Details of these steps are discussed in Sections 3.1–3.5.) The algorithm consists of an initialization step and a loop that continuously refines predictor accuracy. It acquires new samples by deploying and monitoring G on selected resource assignments instantiated in the workbench.

3.1 Sequence of Exploring Predictors

In each iteration of Algorithm 1, Step 2.1 picks a specific predictor to refine. NIMO chooses the predictor whose refinement contributes the maximum to the

Algorithm 1: Active and accelerated learning of predictors $f_1(\rho_1, \ldots, \rho_k), \cdots, f_n(\rho_1, \ldots, \rho_k)$ for task G

1) **Initialize:** $f_1 = f_2 = \ldots f_n = 1$;
2) **Design the next experiment:** (Sections 3.1–3.3)
   - 2.1 Select a predictor for refinement, denoted f;
   - 2.2 Should an attribute from $\rho_1, \ldots, \rho_k$ be added to the set of attributes already used in f? If yes, then pick the attribute to be added;
   - 2.3 Select new assignment(s) to refine f using the set of attributes from Step 2.2;
3) **Conduct chosen experiment:** (Section 3.4)
   - 3.1 In the workbench run G on the assignment(s) picked in Step 2.3;
   - 3.2 After each run, generate the corresponding sample $\langle \rho_1, \ldots, \rho_k, o_1, \ldots, o_n \rangle$, where $o_1, \ldots, o_n$ are the observed occupancies;
   - 3.3 Refine f based on the new sample set;
4) **Compute current accuracy:**(Section 3.5) Compute the current accuracy of each predictor. If the overall accuracy of predicting completion time is above a threshold, and a minimum number of samples have been collected, then stop, else go to Step 2;

model's overall accuracy. We are exploring both static and dynamic schemes for guiding the choice of predictors.

- Static scheme: In a static scheme, NIMO first decides the *order* in which to refine the predictors, and then defines the *traversal plan* for choosing the predictor to refine in each iteration.

  Order: NIMO obtains the refinement order either from a domain expert, or from a relevance-based design (e.g., *Plackett-Burman* [7]). The relevance-based techniques determine the relevance of independent variables (predictors) on a dependent variable (completion time).

  Traversal: NIMO uses two traversal techniques: (i) round-robin, where it picks the predictor for refinement in a round-robin fashion from the given total order; and (ii) improvement-based, where NIMO traverses the predictors sequentially in order, and refines each predictor until the improvement in accuracy for that predictor drops below a threshold. Section 3.5 explains how NIMO obtains the current accuracy of a predictor.

- Dynamic scheme: In a dynamic scheme, NIMO does not use any predefined order for refining the predictors. In each iteration, it picks the predictor with the maximum current prediction error.

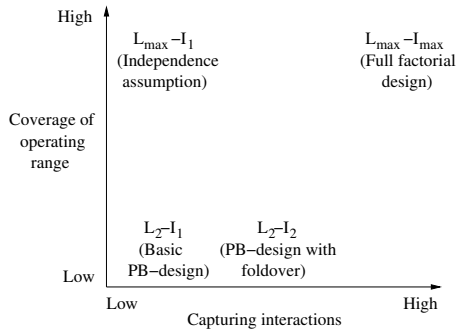

Figure 3: Techniques for selecting new sample assignments (PB = Plackett-Burman [7])

3.2 Adding New Attributes to Predictors

Step 2.2 of Algorithm 1 decides when to add a new attribute in $\vec{\rho}$ to a predictor $f(\vec{\rho})$, and if so, which of the k attributes to add. As in Section 3.1, NIMO's twofold strategy is to first define a total order over the $\rho_1, \ldots, \rho_k$ attributes, and then to define a traversal plan based on this order to select attributes for inclusion in $f(\vec{\rho})$.

NIMO obtains the total ordering of attributes as described in Section 3.1. Once NIMO decides the order in which to add attributes, it considers two approaches for deciding when to add the next attribute:

1. Add the next attribute when the marginal improvement in a predictor's accuracy (with the current set of attributes) per new added sample drops below a predetermined threshold.
2. At each point of time, maintain two competing attribute sets for each predictor. When the accuracy of one candidate dominates the other, add the next attribute (in the order) to the losing attribute set.

3.3 Selecting New Sample Assignment(s)

Step 2.3 of Algorithm 1 chooses new assignments to run task G and collect new samples for learning. For each new assignment $\vec{R}$ we need to choose the values for each attribute ρ_i in $\vec{R}$'s resource profile, while accounting for:

1. The *coverage of the operating range* of each attribute to improve accuracy over a wider range of attribute values. For example, limited variation of CPU speeds and network latencies may fail to expose latency-hiding behavior due to prefetching [6].
2. The important *interactions* among attributes. For example, changing CPU speed or network latency affects the occupancy of the storage resource [6].

Figure 3 shows some of sample selection techniques in terms of their general performance and tradeoff on the two metrics above. We use an L_α-I_β naming format, where (i) α represents the number of significant distinct values, or *levels*, in the attribute's operating range covered by the technique, and (ii) β represents the largest

degree of interactions among attributes captured by the technique.

- L_{max}-I_1: This technique systematically explores all levels of a newly-added attribute using a binary-search-like approach. However, it assumes that the effects of attributes are independent of each other, so it chooses values for attributes independently.
- L_2-I_2: The classic *Plackett-Burman design with foldover* technique [7] captures two levels (*low* and *high*) per attribute and up to pair-wise interactions among attributes. This technique is an example of the common approach of *fractional factorial design*.

3.4 Performing the Selected Experiment

In Step 3 of Algorithm 1, NIMO instantiates the assignment selected in Step 2.3 in the workbench and runs the task. The noninvasive instrumentation data collected from the run and its analysis to derive occupancy measures is described in [6].

3.5 Evaluating Accuracy

NIMO uses two techniques for computing the current prediction error of a predictor.

- *Cross-Validation:* We use leave-one-out cross-validation to estimate the current accuracy of each predictor. For each sample s of the m samples, we learn predictor f_i using all samples other than s. The learned predictors are used to predict occupancies for s, and the prediction accuracy is noted. The individual accuracy of each predictor and the overall prediction accuracy are averaged over the m samples.
- *Fixed test set*: We designate a small subset of resource assignments in the workbench as a *test set*, e.g., a random subset of the possible assignments in the workbench. The current prediction error of $f(\vec{\rho})$ is computed as the average of $f(\vec{\rho})$'s percentage error in predicting occupancy on each assignment in the test set. Note that the samples collected from the test set are never used as training samples for $f(\vec{\rho})$.

4 Discussion

4.1 Curse of Dimensionality

The exploration strategies discussed in Sections 3.1 and 3.2 help NIMO cope with the curse of dimensionality by considering predictors and attributes iteratively. For example, the relevance-based ordering guides NIMO to include only the predictors and attributes that are most relevant to the model's overall accuracy. For a batch task NIMO may quickly identify that it is compute-intensive for the range of attribute values of interest, and guide sample acquisition to refine the predictor for the compute resource only.

4.2 Vertical Vs. Horizontal Compression

Algorithm 1 attempts to reduce the overall learning time by acquiring only the relevant samples. We term this methodology *vertical compression* since it reduces the total data points in the training set table. An alternative is to reduce the run time for a task G to acquire a sample. For example, if G has very periodic behavior, then we can learn G's occupancies on a resource assignment based on a few of G's initial periods. (Recall that occupancies are averages per unit of data flow.) This methodology, termed *horizontal compression*, can be applied in conjunction with vertical compression to further reduce the overall learning time.

4.3 Workbench

The configuration of the workbench where NIMO conducts experiments poses some interesting questions:

- Does the workbench replicate actual resources from the target system or emulate them? While replicating actual resources leads to accurate empirical data, it may restrict the range of resource assignments. One alternative is to emulate a variety of hardware resources over real hardware through virtualization.
- Should the system learn from production runs instead of (or in addition to) runs on a separate workbench? For example, NIMO can harness underutilized resources in the utility, or NIMO's scheduler can perform online experiments through controlled resource assignments to production runs of tasks. However, having a workbench as a separate entity allows NIMO to circumvent the traditional *exploitation versus exploration* tradeoff by allowing both to occur in parallel.

4.4 Extended Iterative Learning

When NIMO's scheduler actually uses a task G's performance model learned on the workbench, it may discover significant prediction errors for some resource assignment $\vec{R}$. Such errors may arise because Algorithm 1 failed to capture (i) the effect of a resource attribute, (ii) an interaction among attributes, or (iii) the correct operating range of an attribute. This scenario is similar to one where Step 4 of Algorithm 1 detects low prediction accuracy on an assignment. In such an event, NIMO can bring G back to the workbench and update G's predictors by continuing Algorithm 1 where it left off at Step 4, with $\vec{R}$ now included in the set of samples.

5 Preliminary Experimental Results

We are currently exploring accelerated learning using active sampling for several production batch tasks. We provide preliminary results for one such task, *fMRI*, a statistical parametric application used in biomedical imaging. We consider the execution of this task on three resources—a compute node, network, and a storage array, corresponding to learning compute, network, and

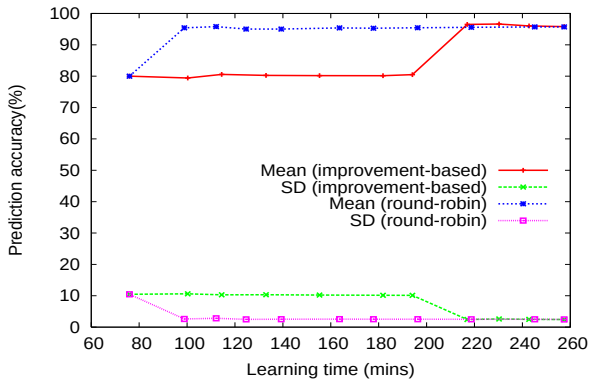

Figure 4: This graph compares round-robin active sampling strategy to an improvement-based active sampling strategy to learn predictors for *fMRI*. Prediction accuracy increases immediately under the improvement-based strategy.

storage predictors respectively. Currently, our NIMO prototype uses multi-variate linear regression to learn the predictors. We are also exploring the potential of more sophisticated learning algorithms such as regression splines [5].

The important resource-profile attributes in this experiment were found to be CPU speed, memory size, and network latency. NIMO's workbench comprises 5 CPUs varying from 451 MHz to 1396 MHz, 4 memory sizes ranging from 256 MB to 2GB, and 10 different network latencies ranging from 0, 2,..., 18 ms between the compute and storage nodes. Therefore, the total sample space is $5 \times 4 \times 10 = 200$ assignments.

We compare two active sampling strategies to illustrate the potential of active sampling for accelerated learning. Exhaustive evaluation of all strategies is beyond the scope of this paper. We use round-robin traversal of predictors in one case, and improvement-based traversal in other (Section 3.1). The latter strategy learns an accurate predictor for the network resource, then the predictor for the CPU resource, and finally the predictor for the storage resource. In both strategies we use static ordering of predictors and attributes, and L_{max}-I_1 technique for selecting samples (Section 3.3), where we vary the operating range of one resource at a time keeping the others constant.

Figure 4 shows the accelerated learning in NIMO using active sampling. The x-axis shows the progress of time for collecting samples and learning predictors, and the y-axis shows the mean and standard deviation of *fMRI*'s completion-time prediction accuracy on 15 test assignments chosen randomly from the 200 candidate assignments. The figure shows that the round-robin strategy learns an accurate function quickly using less than 5% of total sample assignments in less than 100 minutes. (The total time to run all fMRI on all 200 assignments is around 5 days.) The round-robin strategy converges faster to accurate predictions because it refines the CPU and I/O predictors together, without waiting for one predictor to be learned properly. This result shows the potential of accelerated learning; active sampling is able to expose the primary factors that affect performance quickly, and hence learn accurate predictors.

6 Related Work

The theory of *design of experiments (DOE)* is a branch of statistics that studies planned investigation of factors affecting system performance. *Active learning* [4] from machine learning deals with the issue of picking the next sample that provides the most information to maximize some objective function. NIMO uses both DOE and active learning in an iterative fashion to enable effective resource assignment in network utilities. Recently, the computer architecture community has also looked at DOE for improving simulation methodology [7].

7 Conclusion

Current work that applies SLT to system management often takes a "given the right data, we learn the right model" approach, and leaves the issue of acquiring the "right data" unaddressed. NIMO bridges this gap by using active sampling strategies for accelerated model learning. Our preliminary experiments with NIMO indicate its potential for significant reduction in the training data and learning time required to generate fairly accurate models.

References

[1] J. Bent, D. Thain, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, and M. Livny. Explicit Control in a Batch-Aware Distributed File System. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation*, Mar 2004.

[2] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid. In *Proceedings of the ACM/IEEE Conference on Supercomputing*, Nov 2000.

[3] I. Cohen, J. S. Chase, M. Goldszmidt, T. Kelly, and J. Symons. Correlating Instrumentation Data to System States: A Building Block for Automated Diagnosis and Control. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, Dec 2004.

[4] V. Fedorov. *Theory of Optimal Experiments*. Academic Press, 1972.

[5] J. Friedman. Multivariate Adaptive Regression Splines. *Annals of Statistics*, 19(1):1–141, 1991.

[6] P. Shivam, S. Babu, and J. Chase. Learning Application Models for Utility Resource Planning. In *Proceedings of International Conference on Autonomic Computing*, Jun 2006.

[7] J. J. Yi, D. J. Lilja, and D. M. Hawkins. A Statistically Rigorous Approach for Improving Simulation Methodology. In *Proceedings of International Symposium on High-Performance Computer Architecture*, Feb 2003.

[8] N. Zhang, P. Hass, V. Josifovski, G. Lohman, and C. Zhang. Statistical Learning Techniques for Costing XML Queries. In *Proceedings of International Conference on Very Large Data Bases*, Aug-Sep 2005.