

Active and Accelerated Learning of Cost Models for Optimizing Scientific Applications

Piyush Shivam
Duke University
shivam@cs.duke.edu

Shivnath Babu
Duke University
shivnath@cs.duke.edu

Jeff Chase
Duke University
chase@cs.duke.edu

ABSTRACT

We present the *NIMO* system that automatically learns cost models for predicting the execution time of computational-science applications running on large-scale networked utilities such as computational grids. Accurate cost models are important for selecting efficient plans for executing these applications on the utility. Computational-science applications are often scripts (written, e.g., in languages like Perl or Matlab) connected using a workflow-description language, and therefore, pose different challenges compared to modeling the execution of plans for declarative queries with well-understood semantics. *NIMO* generates appropriate training samples for these applications to learn fairly-accurate cost models quickly using statistical learning techniques. *NIMO*'s approach is *active* and *noninvasive*: it actively deploys and monitors the application under varying conditions, and obtains its training data from passive instrumentation streams that require no changes to the operating system or applications. Our experiments with real scientific applications demonstrate that *NIMO* significantly reduces the number of training samples and the time to learn fairly-accurate cost models.

1. INTRODUCTION

High-performance computing has become a key requirement for rapid advances in a range of sciences including astrophysics, bioinformatics, systems biology, and climate modeling [15, 31]. This new area of *computational science* has given rise to many resource-intensive scientific applications. For example, modern high-energy particle detectors generate up to 10^{15} bytes of data for analysis per year [4]. Other sources of important scientific applications include BIRN [6], GEON [14], and SDSS [30].

The typical scientific application can be represented as a *workflow* consisting of one or more *batch tasks* linked in a directed acyclic graph (DAG) representing task precedence and data flow (e.g., [5]). Complex scientific workflows are often run on networked computing utilities—systems that

allocate compute, network, and storage resources on demand from a large heterogeneous resource pool. Examples of networked utilities include clusters of machines [8], computational grids [13], utility data centers, PlanetLab, and outsourced storage services.

A number of researchers have recently pointed out the critical need for automated systems to manage scientific workflows [15, 31]. Database technology is well-suited to handle many aspects of workflow management as evident from the number of *Workflow Management Systems (WFMSs)* [31]—e.g., Griddb [23], GriPhyn [16], Kepler [2], and Zoo [20]—that exist to provide functionality such as modeling, execution, provenance, auditing, and visualization for workflows.

One aspect of workflow management where database technology can help significantly is *workflow planning* that involves finding an efficient *plan* for executing a workflow on a networked utility. Workflow planning is both important and challenging. Many scientific workflows perform complex computations, process very large amounts of data, or both. The difference in completion time can be on the order of days between a good execution plan for a workflow and a poor one [5]. These differences get magnified when workflows run on networked utilities composed of highly heterogeneous pools of geographically-distributed resources.

Example 1. Consider a motivating scenario where three sites A , B , and C comprise a networked utility. Suppose a user at site A wants to run a workflow composed of a single task G on the utility. The input data for G is stored at A . Site B has the fastest compute resources, but insufficient storage to store G 's input data locally. Site C has faster compute resources than A and sufficient local storage for G 's data. Candidate plans to execute G include:

- P_1 : Run G locally at A
- P_2 : Run G at B , so G gets the best compute resource available, but incurs remote I/O to A for data access
- P_3 : Stage G 's data to C from A , and run G locally at C

The performance of these plans can vary significantly depending on G 's characteristics and the underlying resource characteristics. For example, plan P_2 can be much more efficient than plans P_1 and P_3 if G does a lot of computation, but relatively little I/O. \square

A plan for a workflow G specifies a *resource assignment* \vec{R} for each batch task in the workflow. A task may be a batch application or a *data-staging task* interposed between a pair

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permission from the publisher, ACM.

VLDB '06, September 12-15, 2006, Seoul, Korea.

Copyright 2006 VLDB Endowment, ACM 1-59593-385-9/06/09.

of application tasks. \vec{R} comprises the hardware resources—compute, network, and disk storage—that are assigned simultaneously to run G . G 's performance can vary significantly across different resource assignments. To construct an effective resource assignment for G , the WFMS must predict the interaction of G 's application-level characteristics (e.g., compute-to-communication ratio) with resource attributes (e.g., processor speed, cache size, and I/O system behaviors). Specifically, the system needs a cost model that can predict G 's total execution time on \vec{R} , which is the most common performance metric for scientific workflows. Accurate cost models are important for selecting efficient resource assignments.

Workflow planning is similar to query optimization in database systems, but it poses an entirely new set of challenges. A task in a workflow G is typically a script in a programming language like Perl or Matlab. Hence, a WFMS usually has no prior knowledge about G 's resource usage characteristics, or its performance sensitivity to the diverse hardware platforms comprising the underlying networked utility. Studies indicate that it is almost impractical to ask scientists to use a declarative language like SQL, or a single programming language, or even to add instrumentation code to tasks to help with modeling and workflow planning [15, 31]. Consequently, G is a *black-box* to the WFMS, making it challenging to generate an accurate cost model for G .

In previous work [32] we showed that accurate cost models can be learned using statistical techniques if the right training data is given. We transformed the problem of generating a cost model for a task G to that of learning a regression model that fits a set of m sample data points collected by running G on different resource assignments. Each sample s_i ($1 \leq i \leq m$) is a point in a high-dimensional space. s_i represents a complete run of G on a resource-assignment \vec{R}_i , and has the general form $\langle \rho_1, \rho_2, \dots, \rho_k, T \rangle$, where each ρ_j is a hardware attribute of \vec{R} (e.g., processor speed or disk seek time), and T is the total execution time of G on \vec{R}_i . Given the set of m samples s_1, \dots, s_m , an appropriate regression model can be fitted to the training data to predict the execution time T from the values of attributes $\rho_1, \rho_2, \dots, \rho_k$.

However, the challenge of acquiring the right training data remains unresolved. Three challenges arise in this setting:

- *Cost of sample acquisition:* Acquiring each sample may have high overhead. For example, a sample $\langle \rho_1, \dots, \rho_k, T \rangle$ ‘costs’ time T to acquire, which may be on the order of hours or days for long-running scientific tasks.
- *Curse of dimensionality:* As the dimensionality of the data increases, the number of samples needed to attain a given level of accuracy can increase exponentially.
- *Operating range of samples:* The training sample set must represent the entire operating range of the system. A model learned from samples that cover only a limited range may not give accurate predictions across the entire system operating range.

Example 2. Acquiring samples corresponding to a mere 1% of a 5-dimensional space with 10 distinct values per dimension and average sample-acquisition time of 10 minutes, takes around 7 days. If the space becomes 8-dimensional, then the overall time becomes 19 years! However, if the system knows or learns, e.g., that the task is CPU-intensive for most resource assignments because it performs complex

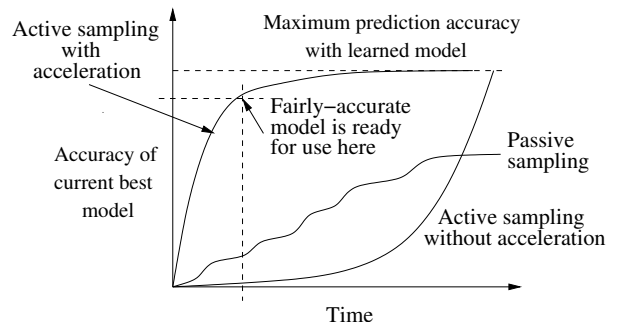


Figure 1: Active and accelerated learning

computations per unit of input data, then the dimensionality and the learning time can be reduced significantly. \square

1.1 Contributions

- We present the *NIMO* system that learns cost models automatically for predicting task execution time on heterogeneous resources. NIMO performs *active sampling* of resource assignments to *accelerate* convergence to an accurate cost model for a task G . Active sampling acquires data to expose the relevant range of G 's behavior by planning *experiments*. Each experiment runs G on a candidate resource assignment deployed in a *workbench* composed of heterogeneous resources. Active sampling with acceleration seeks to reduce the time before a reasonably-accurate cost model is available, as depicted in Figure 1. (The x -axis in Figure 1 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 NIMO actively deploys and monitors the task under varying conditions, NIMO is noninvasive in that it obtains its training data from passive instrumentation streams that require no changes to systems or applications. Specifically, NIMO can be applied to applications without changing application source or binary.
- We present experimental results that demonstrate how NIMO reduces the time to learn fairly-accurate cost models for real scientific applications.

2. NIMO

NIMO (NonInvasive Modeling for Optimization) is a workflow planning system that generates effective resource assignments for scientific workflows running on large-scale networked utilities. Figure 2 shows NIMO's overall architecture consisting of: (i) a *scheduler* that enumerates, selects, and executes plans for workflows; (ii) a modeling engine that consists of an *application profiler*, a *resource profiler*, and a *data profiler* that learns cost models for plans; and (iii) a workbench where NIMO proactively runs plans to collect samples for learning cost models. We describe each component in turn.

2.1 Scheduler

NIMO's scheduler is responsible for generating and executing a plan for a given workflow G . The scheduler enumerates candidate plans for G , estimates the cost of each plan, and chooses the execution plan with the minimum total execution time. A *plan* P for workflow G is an execution

