Learning Application Models for Utility Resource Planning

Piyush Shivam
Duke University, Durham NC 27708
shivam@cs.duke.edu

Shivnath Babu
Duke University, Durham NC 27708
shivnath@cs.duke.edu

Jeffrey S. Chase
Duke University, Durham NC 27708
chase@cs.duke.edu

***Abstract*— Shared computing utilities allocate compute, network, and storage resources to competing applications on demand. An awareness of the demands and behaviors of the hosted applications can help the system to manage its resources more effectively. This paper proposes an active learning approach that analyzes performance histories to build predictive models of frequently used applications; the histories consist of measures gathered from noninvasive instrumentation on previous runs with varying assignments of compute, network, and storage resources. An initial prototype uses linear regression to predict application interactions with candidate resources, and combines them to forecast completion time for a candidate resource assignment. Experimental results from the prototype show that the mean forecasting errors range from 1% to 11% for a set of batch tasks captured from a production cluster. Examples illustrate how a system can use the learned models to guide task placement and data staging.**

I. Introduction

Resource provisioning and placement is a central challenge for computing utilities—systems that allocate compute, network, and storage resources on demand from a heterogeneous resource pool. Examples of networked utilities that can benefit from automated policy-driven resource management include computational grids, utility data centers, network testbeds, and outsourced storage services [1]. Solutions for automated resource management must balance simplicity, generality, cost, and performance.

Application performance can vary significantly across different resource assignments. To construct an effective assignment, the system must predict the interaction of application characteristics (e.g., compute-to-communication ratio) with resource attributes (e.g., CPU speed, cache, and I/O system behaviors). In networked environments, application deployment can involve a multi-step plan with distributed task workflows and staging of data between sites. Modeling the alternatives accurately and efficiently can help the system select the right plan.

This paper explores an approach to construct simple and efficient predictive models that combine limited *a priori* structure with statistical learning. This study is part of our research on NIMO (NonInvasive Modeling for Optimization), a framework for model-guided utility resource planning.[1] 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.

We limit our initial focus to performance estimation for a class of batch compute applications (see Section II-B). A batch workflow G consists of one or more batch *tasks* linked in a directed acyclic graph representing task precedence and data flow (e.g., [2]). In this paper we focus on individual tasks, but the approach extends naturally to task workflows with known structure. NIMO builds a performance model that can predict G's completion time on a *resource assignment* $\vec{R}$ comprising a set of hardware resources (e.g., compute, network, and storage) assigned to run G. In essence, the models approximate the peak data throughput of G on $\vec{R}$ based on the time G "occupies" processing resources, averaged over all units of data processed by G.

We evaluate the effectiveness and accuracy of our approach on a set of biomedical applications that run frequently on a shared production cluster at Duke, called the *DSCR*. A goal of NIMO is to induce models that are sufficiently accurate to guide resource planning in several scenarios:

- *Task placement*. Mapping individual tasks to candidate resources involves balancing multiple factors that affect performance on different CPU, host, network, and storage configurations. A system could use the models to evaluate alternatives and select the best candidates to maximize some objective such as global value.
- *On-time computing and SLAs*. A utility often must meet service-level agreements (SLAs) and performance targets for hosted applications. In a computational setting, specific deadlines may exist for tasks that deal with real-world events such as storm forecasting [3] and response. The system must find resource assignments that meet the performance constraints.
- *Storage outsourcing and data staging*. Storage access delays can be a key barrier to harnessing remote computing resources [4] or outsourced storage. We show how a system can use models to estimate the performance impact of remote I/O and the benefits of task migration or local staging.

The experimental results illustrate the potential of model-

[1]This research is supported by IBM and Network Appliance. Some funding was provided by the National Science Foundation through ANI-01-26231, ANI-0330658, and CNS-0509408.

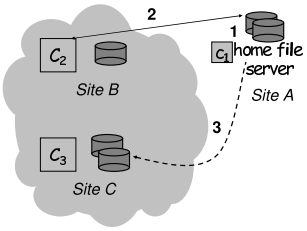

Fig. 1. Three choices for executing a compute application G in a wide-area utility: (1) run G locally at site A; (2) run G at site B and access data remotely from A; (3) stage the data at site C and run G locally at site C.

guided planning as a basis for automated management in these settings. The key contribution of this paper is to show that statistical learning enables the system to infer reasonably accurate predictive models from a modest amount of training data. The evaluation quantifies prediction accuracy using multiple metrics: absolute and percentage error, which are most relevant for meeting SLAs, and *ranking error*, which is important for selecting among competing candidate resource assignments. The current NIMO prototype uses simple models based on linear regression and with limited a priori structure. In our experiments, the mean percentage error from the model predictions is always below 11%. We also use synthetic workloads to stress the linear regression approach and demonstrate its limitations, measuring errors up to 30% when some resource assignments hide the I/O latency. We suggest enhancements to NIMO that have potential to further improve prediction accuracy and training cost.

This paper is organized as follows. Section II defines the problem, and gives an overview of NIMO. Section III surveys related work, and Section IV presents the initial approach to model learning used in the prototype. Section V presents experimental results from the prototype on a NIMO testbed, and Section VI concludes.

II. Overview

A. Motivating Scenario

Consider the scenario depicted in Figure 1, with three sites A, B, and C comprising a networked utility. Suppose a user at site A wants to run an application 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. The utility must choose a *plan* to execute G. Candidate plans 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

Previous studies have shown that plan performance can vary significantly depending on application characteristics and underlying resource characteristics [4], [5], [6], [7]. 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.

Figure 2 gives an overview of our approach to estimate the performance (completion time) of candidate plans. NIMO builds profiles of resources and frequently executed applications by analyzing instrumentation data gathered from previous runs using common and noninvasive tools (currently *sar*, *tcpdump*, and *nfsdump* [8]). A performance model M for an application G predicts the completion time of a plan for G given three inputs: (i) G's *application profile*, (ii) *resource profiles* of resources assigned to the plan, and (iii) *data profile* of the input dataset.

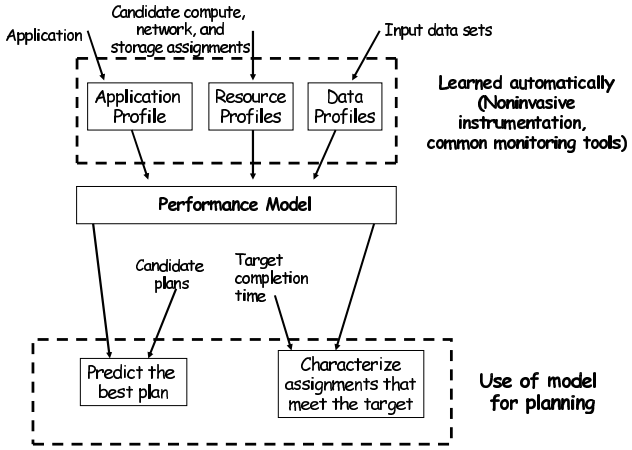

Fig. 2. Overview of the model-guided planning approach.

Intuitively, the application profile captures how an application uses compute, network, and storage resources. Resource profiles specify attributes that characterize the function and power of those resources in an application-independent way. For example, a resource profile might represent a compute server with a fixed number of CPUs defined by attributes such as clock rate and cache sizes, with an attached memory of a given size. Similarly, storage resources can be approximated by attributes such as capacity, spindle count, seek time, and transfer speed. These attributes determine the performance that a given application (defined by its profile) can expect from them. The data profile comprises the data characteristics of G's input dataset, e.g., the input data size.

The performance model M uses these profiles as inputs to support the following target uses as shown in Figure 2:

- Given a set of candidate plans for G, M ranks the plans in order of (estimated) completion time;
- Given a target completion time t for G, M characterizes plans whose completion time is $\leq t$.

B. Problem Setting and Assumptions

In this paper we focus on the behavior of batch compute applications that combine processing and file I/O. The completion time of a batch task is limited only by the resources assigned to its plan. Our model inference makes several limiting assumptions; these are consistent with the workloads observed in the DSCR cluster at Duke.

- *Data-independent behavior.* We consider applications whose resource demands are regular across different input datasets. Specifically, the models consider only the characteristics of the input dataset that are represented in the

data profile (e.g., input size); if other data dependencies exist, then the predictions may be inaccurate. The current NIMO prototype does not consider data dependencies.

- *Commodity architectures.* We consider heterogeneity in the compute resources across CPU speeds and memory sizes within the same CPU architecture. A recent survey of grid resources [9] suggests that at any given point of time, there are only a few dominant architectures. Thus our resource profiles capture CPU type as a separate attribute; the analysis algorithm considers the impact of CPU type, but does not directly track correlations across different CPU types.
- *Sequential batch applications.* A survey of $280,000$ jobs submitted in a recent month to the DSCR cluster from a diverse set of user groups (computational biology, physics, chemistry, biochemistry, biomedical, statistics, etc.) show that almost 90% of the jobs are sequential batch jobs. Our techniques are applicable to parallel applications with a fixed degree of parallelism, but we do not consider them in this paper. All applications that we analyze in this paper run on a single CPU.

III. Related Work

Much of the previous work on model-based provisioning and placement focuses on online Internet services, which are driven by the arrival patterns of requests and queuing behavior [10], [11], [12], [13]. In contrast, we focus on compute batch tasks that run to completion at machine speed, so we do not need to model request arrivals. Hippodrome [14] uses detailed performance models and an optimizing planner to assign storage resources in a shared utility. Hippodrome emphasizes storage performance and modeling of contention; our approach is complementary in that it addresses the interaction of computation and storage access and their impact on end-to-end application performance.

Several previous studies support the potential of statistical techniques to capture application performance behavior accurately. For example, two early surveys ([15], [16]) found that a large class of scientific applications exhibit marked regularity in CPU usage and I/O activity over the execution interval. Some applications show data-dependent behavior—their resource usage depends on parameters or the contents of the input data—and several groups are exploring how to map such dependencies when they exist, e.g., [17], [18], [19]. Although this paper does not consider data-dependent behavior, NIMO's data profiles can represent input data characteristics that affect application resource demands; these complementary projects may help to understand the most relevant characteristics and how to capture them.

Accurate prediction of completion time is a prerequisite for many provisioning and scheduling strategies, e.g., [5], [7]. AppLeS [5] is a grid scheduling framework that uses estimates of computation and file transfer times to decide task placement. Menasce et al. [20] use queuing analysis to predict the throughput of a stream of batch tasks competing for homogeneous shared resources, assuming the behavior of individual tasks is known. A number of systems approximate NP-hard optimal scheduling assignments given estimates of performance or utility, e.g., [21]. All these systems can benefit from our work in predicting the completion times of individual tasks on heterogeneous resources.

Some recent prediction work instruments the program source or its binary (e.g., [22]), or assumes knowledge of the program internals. For example, Rosti et al. [23] instrument the source code of parallel applications to derive stochastic prediction models. In contrast, we rely only on noninvasive instrumentation, so our model predictions may be less accurate. [24] provides a black-box approach to predict completion time on heterogeneous platforms, but the approach requires a partial execution of each candidate application on each target system.

Remote storage access is a key barrier to harnessing remote computing resources effectively [1], [4]. Our approach models the impact of storage placement and access costs on end-to-end performance, as a basis for intelligent placement of tasks and data. Wolski et al. [6] evaluate the impact of various data-staging strategies on wide-area task farming, and show that staging overhead can often be overlapped with computation. The models in this paper are conservative with respect to this phenomenon, but could be extended to account for it.

Various mechanisms exist to realize a range of choices for data staging and task placement in networked systems. BAD-FS [2] assigns work to compute servers and storage servers to maximize the benefit of local caching and buffering. Logistical Networking [25] addresses the global scheduling of data movement and computation. Stork [26] is a batch scheduler that schedules data migration tasks as first-class citizens alongside computation. Abacus [27] uses analytical models to drive dynamic task placement for data-intensive cluster applications. Our work gives a model-guided approach to select among candidate task and data placements in such systems, and to induce the models automatically.

IV. Learning Performance Models

In general, a *batch workflow* G consists of one or more *tasks* linked in a directed acyclic graph (DAG) representing task precedence and data flow [2]. A *plan* makes a resource assignment $\vec{R}$ to each task $G_i \in G$. Suppose that each task G_i accesses its inputs and outputs from the storage resources specified in $\vec{R}$, and that G_i *fires* only when its predecessors have produced its inputs, as is common [2]. Then the completion time (execution time) of the workflow is the sum of the completion times of the tasks in its *critical path*, which can be found using existing techniques [28]. A plan may also interpose additional tasks for data migration and staging into the workflow.

This paper focuses on forecasting the completion time of individual tasks G_i, or (equivalently) graphs G consisting of a single task. Section IV-A gives an overview of the modeling problem and the structure of the application, resource, and data profiles. Section IV-B explains the performance model that combines the profiles to forecast the completion time of

a task on a candidate assignment $\vec{R}$. Sections IV-C and IV-D summarize how NIMO learns different profiles.

A. Profiles

Consider a task G's execution as an interleaving of *compute phases*, in which the compute resource is doing useful work, and *stall phases*, in which the compute resource is stalled on I/O. For the execution of task G with input dataset I on the resource assignment $\vec{R}$, we define:

- The *compute occupancy* of G on $\vec{R}$, denoted o_a, is the average time spent computing per unit of data flow processed by G.
- The *stall occupancy* of G on $\vec{R}$, denoted o_s, is the average time for which the compute resource is idle per unit of data flow. Stall occupancy $o_s = o_n + o_d$, where o_n and o_d capture the portion of occupancy caused by delays in the network and storage (disk) resources respectively. Note that the "stall occupancy" components for network and storage do not represent the service demands or utilization at those resources; rather, they represent the compute stalls caused by delays at those stages, given the overall resource assignment.

The occupancies are determined by the interaction of the application and the resources, given the behavior of the application on the input dataset I. $\vec{R}$'s *resource profile* characterizes the resource properties, and the *data profile* represents the significant characteristics of I. G's *application profile* captures the interaction and the resulting application behavior in terms of the properties of the resource and data profiles. The profiles are defined as follows:

Resource Profile: A resource profile $\vec{\rho}$ is a vector $\langle \rho_1, \rho_2, \ldots, \rho_j \rangle$ where each ρ_i measures the value of some performance attribute of $\vec{R}$. The performance attributes are properties of the resources, and are quantifiable independently of any specific task. For example, for a compute resource, the attributes may include processor speed, cache size, memory size, memory latency, and memory bandwidth. In the single-task cases considered in this paper, the resource profile characterizes a resource assignment $\vec{R} = \langle C, N, S \rangle$ representing the compute (C), network (N), and storage (S) resources assigned to run a task.

Data Profile: The data profile $\vec{\lambda}$ of an input dataset I captures any characteristics of I that may correlate with resource demands, such as size, format type metadata, or histograms capturing data distribution. In this paper, we consider a single attribute for $\vec{\lambda}$, namely, the total size of the input dataset.

Application Profile: G's application profile includes four *predictor functions* $f_a(\vec{\rho}, \vec{\lambda})$, $f_n(\vec{\rho}, \vec{\lambda})$, $f_d(\vec{\rho}, \vec{\lambda})$, and $f_D(\vec{\rho}, \vec{\lambda})$. f_a, f_n, and f_d are occupancy predictor functions that predict G's occupancies o_a, o_n, and o_d respectively on a resource assignment $\vec{R}$ and input dataset I, as a function of $\vec{R}$'s resource profile $\vec{\rho}$ and I's data profile $\vec{\lambda}$. The data flow predictor function $f_D(\vec{\rho}, \vec{\lambda})$ estimates D, the total data flow processed by G (i.e., the total number of units of data read and written by G) for a given $\vec{R}$ and I.

B. Performance Model

Given task G's application profile $\langle f_a, f_n, f_d, f_D \rangle$, the resource profile $\vec{\rho}$ of a candidate resource assignment $\vec{R}$, and input dataset I's data profile $\vec{\lambda}$, NIMO predicts G's completion time on $\vec{R}$ by the performance model $M(G, I, \vec{R})$:

$$Completion\,Time = f_D(\vec{\rho}, \vec{\lambda}) \times (f_a(\vec{\rho}, \vec{\lambda}) + f_n(\vec{\rho}, \vec{\lambda}) + f_d(\vec{\rho}, \vec{\lambda})) \quad (1)$$

In this paper, we make a crucial choice to capture significant performance effects implicitly in the predictor functions, rather than explicitly in the structure of an a priori analytical model. The complexity arising from factors such as CPU caching, file caching, I/O patterns, latency hiding, and queuing behavior at storage servers is captured *implicitly* in the training data for the predictor functions. As long as the effects of these factors show in the training data, the statistical learning techniques can capture them (Section V-B). However, to build an accurate model, it is important to choose training sample runs (or experiments) with good coverage over the entire operating range. We discuss this problem briefly in Section V-B.4.

We also considered the alternative of using a closed-loop queuing network model parameterized by analysis of the training data. An explicit model would capture factors such as concurrency, latency hiding behavior, and queuing effects in the structure of the analytical model, so it would require less training data to learn an accurate task model (Section IV-A). However, this approach is less general, and it may be difficult to obtain parameters such as service demands (utilization) from noninvasive instrumentation data, particularly for resources such as storage servers. In addition, this approach requires NIMO to infer the degree of concurrency in the closed-loop system, e.g., resulting from the depth of operating system prefetching, or the threading structure of the application. Section V-B.4 revisits the tradeoff between the two performance modeling approaches.

C. Learning Application Profiles

To infer G's application profile, NIMO must learn G's occupancy predictor functions $\langle f_a(\vec{\rho}, \vec{\lambda}), f_n(\vec{\rho}, \vec{\lambda}), f_d(\vec{\rho}, \vec{\lambda}) \rangle$ and the data flow predictor function $f_D(\vec{\rho}, \vec{\lambda})$. NIMO uses Algorithm 1 to derive the occupancies and total data flow from measurements of G running on an assignment of compute, network, and storage resources. From each run of G, NIMO obtains a *training* sample of the form $\langle \vec{\rho}, \vec{\lambda}, o_a, o_n, o_d, D \rangle$, where the attributes $\rho_1, \rho_2, \ldots, \rho_j$ and $\lambda_1, \lambda_2, \ldots, \lambda_k$ corresponding to the resource profile $\vec{\rho}$ and the data profile $\vec{\lambda}$ respectively are the *independent* variables, and $\langle o_a, o_n, o_d, D \rangle$ are the *dependent* variables. If NIMO has a reasonably large and representative set of such samples, then the problem of learning accurate predictor functions $\langle f_a(\vec{\rho}, \vec{\lambda}), f_n(\vec{\rho}, \vec{\lambda}), f_d(\vec{\rho}, \vec{\lambda}), f_D(\vec{\rho}, \vec{\lambda}) \rangle$ reduces to a statistical learning problem of fitting accurate regression functions to predict each of o_a, o_n, o_d, and D using attributes $\rho_1, \rho_2, \ldots, \rho_j, \lambda_1, \lambda_2, \ldots, \lambda_k$.

Algorithm 2 depicts the learning process for f_a; f_n, f_d, and f_D can be learned similarly. Section V uses an example to illustrate how the predictor functions are constructed. Note

Algorithm 1: Computing G's occupancies and total data flow to process dataset I on assignment $\vec{R} = \langle C, N, S \rangle$

1) Run G on $\vec{R}$, and measure the compute resource C's average utilization U, G's completion time t, and the total data flow D (using network I/O traces);
2) Solve for o_a and o_s from $U = \frac{o_a}{o_a+o_s}$, $\frac{D}{t} = \frac{1}{o_a+o_s}$;
3) Use network I/O traces to derive the average time spent per I/O in the network resource N and in the storage resource S;
4) Split $o_s = o_n + o_d$ into o_n and o_d in proportion to the ratio of network and storage components of the average I/O time from Step 3, to obtain $\langle o_a, o_n, o_d, D \rangle$;

Algorithm 2: Learning G's compute occupancy predictor function $f_a(\vec{\rho}, \vec{\lambda})$

1) Conduct m runs of G where run i is conducted on $\langle \vec{R}_i, I_i \rangle$ with resource profile $\vec{\rho_i} = \langle \rho_{1_i}, \ldots, \rho_{j_i} \rangle$ and data profile $\vec{\lambda_i} = \langle \lambda_{1_i}, \ldots, \lambda_{k_i} \rangle$;
2) Use Algorithm 1 to generate m training data points where the ith point is $\langle \rho_{1_i}, \ldots, \rho_{j_i}, \lambda_{1_i}, \ldots, \lambda_{k_i}, o_{a_i} \rangle$;
3) Normalize the training points using a reference configuration $\langle \vec{R}_{ref}, I_{ref} \rangle$ with resource profile $\vec{\rho}_{ref}$ and data profile $\vec{\lambda}_{ref}$. Let G's compute occupancy on $\langle \vec{R}_{ref}, I_{ref} \rangle$ be $o_{a_{ref}}$, so the ith normalized training data point is $\langle \frac{\rho_{1_i}}{\rho_{1_{ref}}}, \ldots, \frac{\rho_{j_i}}{\rho_{j_{ref}}}, \frac{\lambda_{1_i}}{\lambda_{1_{ref}}}, \ldots, \frac{\lambda_{k_i}}{\lambda_{k_{ref}}}, \frac{o_{a_i}}{o_{a_{ref}}} \rangle$;
4) Use regression on the training data to learn a function $F(\vec{\rho}, \vec{\lambda})$ that predicts the value of $\frac{o_a}{o_{a_{ref}}}$ from the normalized values of $\rho_1, \ldots, \rho_j$ and $\lambda_1, \ldots, \lambda_k$. Set $f_a(\vec{\rho}, \vec{\lambda}) = o_{a_{ref}} \times F(\vec{\rho}, \vec{\lambda})$;

that all samples are normalized with respect to a reference configuration $(\vec{R}_{ref}, I_{ref})$ since the dependent and independent variables have different units and ranges.

Currently, our NIMO prototype uses multi-variate linear regression to learn the predictor functions. Linear regression gave good prediction accuracy for almost all the applications and resource assignments considered in the paper. Section V-B.4 discusses more sophisticated techniques for capturing non-linearity in performance effects. Another current limitation of our prototype is that it does not consider data profiles while learning the predictor functions, so it cannot, e.g., predict completion time accurately for applications whose resource demands *per unit of data* depend on characteristics of the input dataset.

D. Learning Resource Profiles

NIMO automatically obtains resource profiles by running standard benchmark suites that are designed to expose the differences that are most significant for the performance of real applications. Currently NIMO uses *whetstone* [29] to calibrate CPU speeds, *lmbench* [30] to calibrate memory latency and bandwidth, and *netperf* to calibrate the network latency and bandwidth between the compute servers and storage resources. Our approach is independent of the specific benchmarks as long as they capture the underlying resource characteristics; we found that these benchmarks are sufficient for making accurate predictions in our environment. Other researchers have confirmed that simple benchmarks can be used in profiling HPC platforms [22], studied benchmark selection for comprehensive coverage [31], and devised strategies for robust resource profiling in the presence of competition for shared resources [32].

Application	Description
fMRI [33]	Statistical parametric mapping to normalize functional MRI images of human brains
CardioWave [34]	Cardiac electrophysiology simulation (MPI)
BLAST [35]	Search genomic dataset for matches on proteins and nucleotides
NAMD [36]	Simulation of large biomolecular systems (MPI)
GAMUT [37]	An application emulation tool for generating workmixes; generate 6 synthetic tasks with equivalent compute costs but different access patterns: rand./seq. read/write, and rand./seq. read and write in 1:1 ratio

TABLE I
APPLICATIONS USED IN NIMO EXPERIMENTS.

V. EXPERIMENTAL EVALUATION

We experimented with a range of applications and resource assignments in a NIMO testbed. We used a standard cross-validation methodology to evaluate the effectiveness of model induction with various training sets, and the accuracy of the resulting performance predictions. Section V-A uses multiple accuracy metrics to profile the prediction accuracy, and Section V-B explores the sensitivity of the results to training set size and selection. Section V-C illustrates the potential role of the induced performance models to guide resource planning in several scenarios for a network utility: task placement, on-time task completion, and storage outsourcing or data staging.

Workloads. Table I lists the applications used in the experiments. The *fMRI, CardioWave*, *BLAST*, and *NAMD* applications are used in biomedical research at Duke and elsewhere. We consider one representative input dataset for each application. GAMUT is a tool for emulating cluster application workloads with controlled degrees of CPU, I/O (including I/O patterns), and memory usage. GAMUT is useful for generating synthetic workloads to explore the space of possible application behaviors comprehensively. We focus on the *fMRI* workloads for the in-depth application studies because their performance is the most sensitive to the combination of CPU and I/O resources.

Candidate resources. NIMO's testbed contains five compute nodes—451 MHz, 797 MHz, 930 MHz, 996 MHz, 1396 MHz—with Intel PIII architecture, CPU cache size ranging from 256 KB to 512 KB, memory size ranging from 512 MB to 2 GB, and Linux 2.4.25 kernel. We used `NISTnet` [38] to impose varying network round-trip delays: $0, 2, \ldots, 16, 18$ ms. For the experiments reported in this paper we held the storage system, CPU architecture (Pentium III), and network

bandwidth constant. The five different CPU configurations and ten network latencies yield a total of 50 candidate resource assignments for a task in the testbed.

Model Construction. The accuracy experiments use a 50-way cross-validation methodology as follows. For each application in Table I:

1) We fix one of the 50 candidate assignments as the reference assignment $\vec{R}_{ref}$.
2) NIMO learns the predictor functions for the application using a *training set* of 14 (28%) of the 50 candidate assignments containing at most one non-reference resource.
3) The resource assignments not in the training set constitute the *test set* on which we compare model-predicted completion times with measured completion times to evaluate prediction accuracy.
4) For cross-validation and sensitivity analysis, we repeat Steps 1–2 50 times with different training and test sets. A different $\vec{R}_{ref}$ is selected for each trial.

As an illustration, Figure 3 shows two projections of a subset of training data collected for *fMRI*. These projections show that the dominant effect on compute occupancy in this case comes from the speed of the compute resource (as characterized by *whetstone* performance, Section IV-D). Similarly, in these assignments, changes in network latency affect only network occupancy, but have little effect on compute occupancy. Based on this training data, NIMO learns the following occupancy predictor functions for *fMRI*, with occupancy measured in microseconds per byte.

$$f_a = \frac{4.40}{\rho_{cpu\ speed}} - 0.2, f_n = 4.46\ \rho_{net\ lat} + 0.7, f_d = 0.32$$

Here, $f_D = 0.1064 \times 10^9$ bytes. $\rho_{cpu\ speed}$ and $\rho_{net\ lat}$ are attributes of the resource profile of the candidate assignment normalized to the reference assignment. Depending on the heterogeneity in the hardware resources, the predictor functions may depend on more attributes, e.g., architecture type, memory size, number of disk spindles, and network bandwidth. Correlation analysis can identify the most relevant attributes. When we tested the above model on the test assignments, the predicted completion times were within 5% of the measured completion times.

A. Model Accuracy

Table II summarizes the accuracy of the completion times predicted by the induced performance models. It reports the following metrics:

- **Absolute Error (AE) and Percentage Absolute Error (PE):** AE measures the absolute difference between predicted completion time and measured completion time. PE measures the corresponding percentage error. AE and PE are particularly relevant to the use of models to identify candidate resource assignments to meet a completion time target.

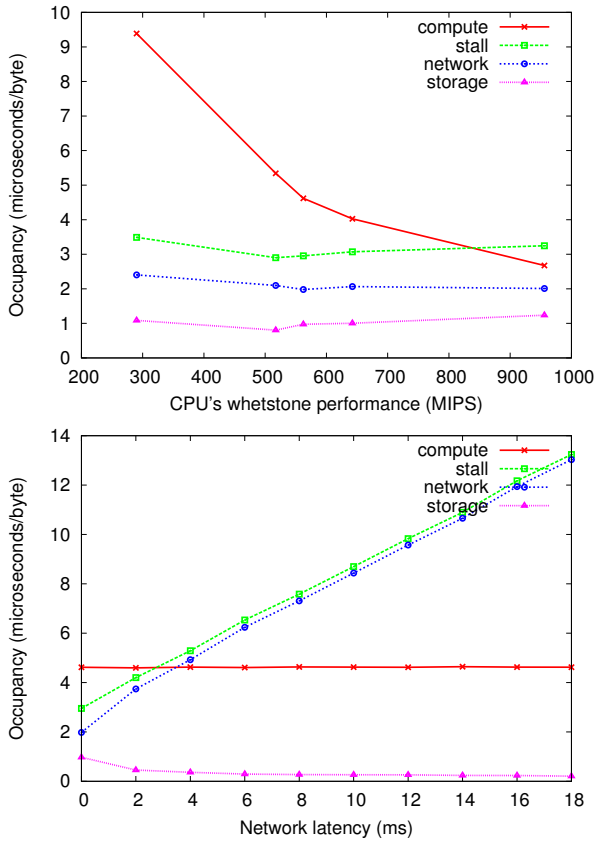

Fig. 3. Impact of CPU speed and network latency on *fMRI*'s occupancies

- **Top-k Ranking Distance (RD(k)):** Consider n resource assignments that are ranked in the order $\vec{R}_1, \ldots, \vec{R}_n$ based on increasing completion time for application G. Let p_i be the model-predicted rank of $\vec{R}_i$. The normalized top-k ranking distance *RD(k)* for model predictions is $\frac{\sum_{i=1}^{i=k} |p_i - i|}{\sum_{i=1}^{i=k} n-i}$. When the model is used to select resource assignments in a utility setting, a low value of *RD(k)* indicates good performance. $0 \leq$ *RD(k)* ≤ 1, with *RD(k)* $= 0$ indicating accurate ranking of the k best assignments.
- **Order Preserving Degree (OPD) [39]:** A model preserves the relative order of resource assignments i and j iff $t_{p_i}(<,=,>)t_{p_j}$ holds when $t_{o_i}(<,=,>)t_{o_j}$ holds, where t_p and t_o respectively represent model-predicted and measured completion times. Given n candidate assignments, *OPD* $= \frac{|OPP|}{n^2}$ where *OPP* is the set of order-preserving pairs. $0 \leq$ *OPD* ≤ 1, with *OPD* $= 1$ indicating accurate ranking.

For each metric, Table II reports the mean, standard deviation, worst-case value (100th percentile), and 90th percentile value from a 50-way cross-validation. Note that mean percentage error from model predictions is under 11%, and ranking-related errors are low (high OPD and low RD). The worst reported PE occurs for a synthetic application doing sequential file reads (GAMUT seq. read in the table). Even in this case, the induced models were accurate to within 23% for 90% of

	AE (mins)				PE				OPD				RD(1)			
Application	μ	σ	Wst	90 pc	μ	σ	Wst	90 pc	μ	σ	Wst	90 pc	μ	σ	Wst	90 pc
fMRI	.9	.6	4.2	1.7	3	2	15	6	.98	.00	.98	.98	.02	.02	.05	.05
CardioWave	.9	.7	2.9	1.2	11	7	27	22	.87	.02	.83	.85	.07	.03	.09	.09
BLAST	.1	.08	.36	1.1	1	.7	4	2	1	0	1	1	0	0	0	0
NAMD	.6	.4	1.5	1	8	6	19	13	.92	.02	.89	.90	.15	.09	.21	.21
GAMUT (seq. read)	.7	1.2	4.6	3.2	6	8	30	23	.92	.02	.89	.90	.15	.09	.21	.21
GAMUT (rand. read)	.1	.09	.4	.2	3	2	13	6	.98	.01	.97	.97	.02	.02	.04	.04
GAMUT (seq. write)	.1	.1	.5	.2	2	2	7	4	.98	.01	.97	.97	.03	.03	.09	.09
GAMUT (rand. write)	.4	.3	1.6	.9	5	3	15	10	.95	.02	.93	.93	.01	.03	.09	.04
GAMUT (rand. r/w)	.1	.07	.35	.19	1	1	6	2.3	.99	.01	.99	.99	.01	.02	.04	.04
GAMUT (seq. r/w)	.3	.3	1.6	.8	5	5	23	12	.90	.02	.88	.88	.09	.02	.16	.12

TABLE II

SUMMARY OF ACCURACY METRICS FOR 4 REAL AND 6 SYNTHETIC APPLICATIONS USING 50-WAY CROSS VALIDATION. μ = MEAN, σ = STANDARD DEVIATION, *Wst* = WORST CASE ERROR, *90 pc* = 90TH PERCENTILE

the trials. Even so, this sequential I/O workload illustrates the limits of the linearity restriction in the current NIMO prototype (see Section IV-C). Section V-B.4 explores this issue in more detail below.

B. Sensitivity Analysis

1) Choice of reference assignment: As noted, the induced predictor functions normalize resource occupancy as relative to occupancy observed on a designated reference assignment $\vec{R}_{ref}$ (see Algorithm 2). The 50-way cross-validation experiments in Table II select a different reference assignment $\vec{R}_{ref}$ for each trial. The low variances in Table II suggest that model induction is robust to the choice of reference assignment.

Training Set Size	AE (mins) (mean)	AE (mins) (SD)	PE (mean)	PE (SD)	OPD (mean)	RD(1) (SD)
4	3.5	2.7	13.12	8.22	.96	.03
6	1.8	1.3	6.8	4.1	.97	.03
8	1	.8	4	2.5	.98	.02
10	.9	.65	3	2	.98	.02

TABLE III

VARYING TRAINING SET SIZE FOR *fMRI*.

2) Size of training set: Table III shows the impact of different training set sizes on the accuracy metrics for *fMRI* across 50 training and test sets. Larger training sets yield more accurate models, but even training sets as small as 6 have good accuracy.

3) Queuing delays and concurrency: Although we do not investigate parallel applications in this paper, our approach appears to be robust across degrees of concurrency in storage access. The key assumption is that the internal degree of concurrency is fixed in the application and the OS it runs on, i.e., the CPU initiates a bounded number of pending I/Os. Given this assumption, the parameterization captures the impact of queuing (e.g., in the storage system) implicitly.

We create a synthetic task (GAMUT rand. write in Table II) that saturates the storage resource to introduce a bottleneck. Figure 4 shows the impact of queuing delays on the occupancies. As seen in the figure, faster network or faster CPU increases the occupancy at the storage resource and vice versa. Such an effect is also created if concurrency at the compute resource introduces an I/O rate that cannot be handled by the storage resource, e.g., in the case of a parallel application with high degree of parallelism.

NIMO learns the following predictor functions for this synthetic task using the approach outlined in Section IV-C. The mean percentage error is within 10% as shown in Table II. This suggests that our approach is reasonably robust with respect to queuing delays as well.

$$f_a = 0.019 + 0.59/\rho_{cpu\ speed} + 0.007 * \rho_{net\ lat}$$
$$f_n = -0.184 - 0.16/\rho_{cpu\ speed} + 0.42 * \rho_{net\ lat}$$
$$f_d = 1.04 + 0.34/\rho_{cpu\ speed} - 0.19 * \rho_{net\ lat}$$

4) Latency hiding and operating range: Figure 5 shows the impact of latency hiding in the case of a synthetic task doing pure sequential reads (GAMUT seq. read in Table II). The worst case percentage error for this task is 30%. We found that this error occurs because prefetching behavior of the file system hides the network latency up to a certain point, causing a non-linear impact on the network occupancy.

The predictor function for predicting network occupancy in this case is a piece-wise linear function, and we are currently exploring two fundamental alternatives for capturing it more accurately. One approach is to replace our linearity assumption with a more sophisticated learning algorithm such as regression splines [40]. Another approach is to incorporate an a priori model of the closed-loop queuing structure to capture the effect of concurrency, queuing, and latency hiding explicitly; see Section IV-B. The first approach is more general, but the second requires fewer samples to parameterize the models, and may yield accurate predictions even outside of the range sampled in the training data. Figure 5 illustrates that the learned predictor functions may be inaccurate outside of the sampled range; in addition to considering structural models, we are also exploring active sampling approaches to training set selection to maximize coverage of the operating range.

C. Model-Guided Planning

The primary goal of our work with NIMO is to guide autonomic resource management from analysis of instrumentation data gathered as the system operates. This section illustrates how an autonomic utility can use the induced models to select from candidate resource assignments and strategies for

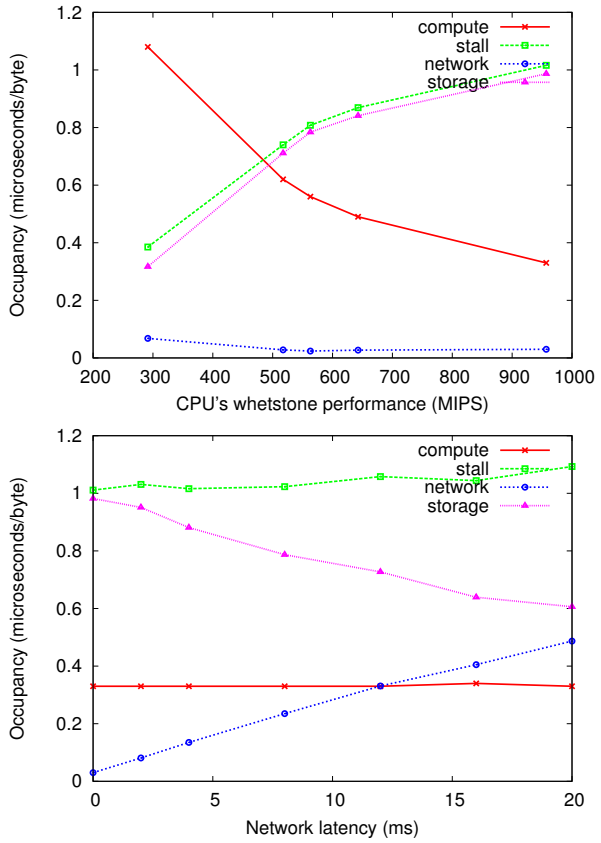

Fig. 4. Impact of CPU speed and network latency on occupancies of GAMUT doing random file writes. Note that the storage resource is saturated and hence the storage occupancy changes with CPU speed as well as network latency.

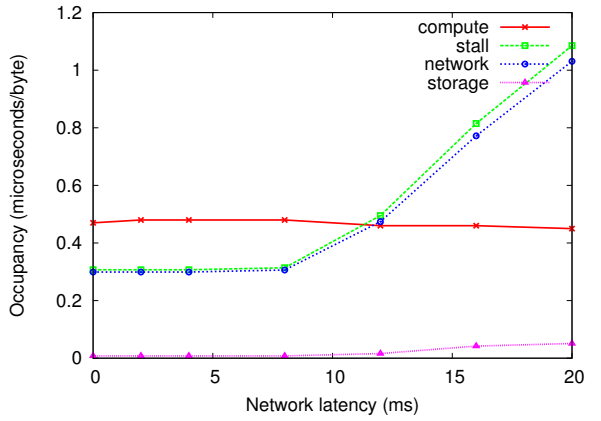

Fig. 5. Impact of network latency on occupancies of GAMUT doing sequential file reads. Prefetching hides I/O latency up to a point, making a single linear function insufficient to predict network occupancy throughout the operating range.

task placement and data staging. We defer a comprehensive analysis to future work.

1) Selecting task placement: A utility resource manager can evaluate the models directly to compare the predicted performance of a set of competing candidate resource assignments. For example, the utility can predict which combination of CPU and I/O resources will yield the shortest completion time for a batch task.

Table V compares the measured performance of *fMRI* runs on four candidate assignments A_1–A_4 shown in Table IV. It shows the predicted and measured completion times for the candidate (A_1) preferred by an induced model, and compares them to the candidates chosen by two simple alternative strategies: (i) select the assignment with the fastest CPU clock; and (ii) select the assignment with the lowest latency to network storage. Since the delivered performance depends on the combination of CPU and I/O resources, the naive approaches cannot identify the best candidate. The model captures the relative importance of the different factors alone and in combination for each application, so it can guide the choice of the best candidate.

Candidate Assgs.	CPU speed	Network latency (ms)
A_1	996 MHz	4
A_2	797 MHz	4
A_3	1396 MHz	10
A_4	451 MHz	2

TABLE IV
CANDIDATE ASSIGNMENTS FOR *fMRI*

Choice of Assg.	Performance
Actual best = A_1	Completion time = 16.67 min
Model-predicted = A_1	Predicted time = 17.46 min
Fastest CPU = A_3	23% slower than A_1
Fastest network = A_4	50% slower than A_1

TABLE V
COMPARING ASSIGNMENT CHOICES

2) On-time computing: A resource manager can also use the models to search for a candidate assignment that meets a specified completion time target t for a task or a sequence of tasks, such as the critical path of a complex workflow graph. For a chain of n tasks, eligible candidates must satisfy the inequality:

$$\sum_{i=1}^{n} D_i(o_{a_i} + o_{n_i} + o_{d_i}) \leq t$$

Identifying eligible candidates is a heuristic search problem in the general case, but it is often possible to solve directly for attribute values that yield target occupancies for each resource, i.e., when the occupancy is driven by continuously valued attributes such as CPU clock speed. Figure 6 plots occupancies for candidate assignments for an *fMRI* instance with a completion time target of $t = 20$ minutes. The figure shows the subset of candidates that meet the target completion time, the set that does not meet the target, and also the model-predicted "boundary" between these two. For each occupancy value of o_a and o_n, the resource manager can obtain the profile of the corresponding resources using the predictor functions f_a and f_n (Section IV-C).

3) Storage outsourcing and data staging: As described in Section II, a networked utility with multiple compute and storage sites must consider tradeoffs between availability of compute resources and access costs to dataset storage, which may be remote. To evaluate the potential of model-guided

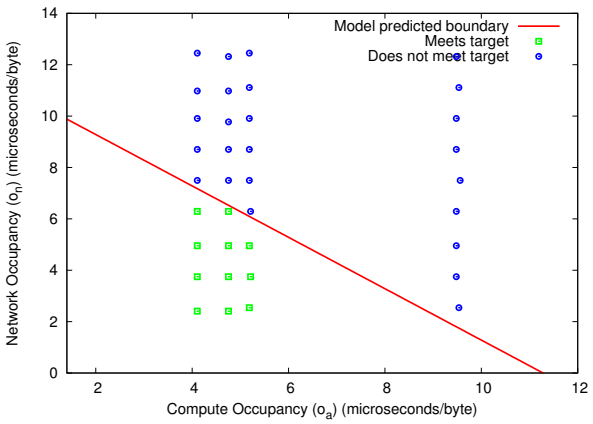

Fig. 6. Model-predicted boundary separating assignments that meet and those that do not meet a target runtime for *fMRI*. Higher values of o_a and o_n indicate slower CPU and farther storage respectively.

planning in this setting, we applied our approach to data published in a recent empirical study of storage outsourcing [1]. The study investigates the viability of storage outsourcing by measuring the impact of remote data placement, caching, and local data staging under various workloads with varying network latencies to the storage site. We induce performance models for two synthetic benchmarks—PostMark and SPEC-SDET—by taking data published for three remote storage configurations as training data, and then use the models to predict the results for the other configurations. The worst case percentage error (PE) was 6% and 10% for PostMark and SPEC-SDET respectively. Figure 7 shows the results for SPEC-SDET. We conclude that our approach has excellent potential to capture the phenomena explored in this comprehensive empirical study.

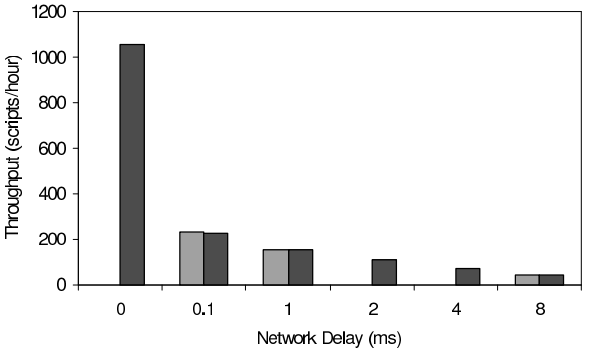

Fig. 7. Accurate prediction of results from an empirical study of storage outsourcing. We parameterize the model using three configurations (single bars), and predict the throughput for the remaining ones (double bars). The maximum error in prediction is 10%.

A utility may use the models to evaluate remote storage and data staging alternatives in conjunction with task placement. Data staging options—in which the candidate plan copies an input or output data set between sites before or after task execution—are modeled as one or more additional stages inserted into a task graph. The predicted completion time is the sum of the predicted times for the data staging steps and task execution.

Table VI shows sample candidate assignments for *fMRI* involving both remote I/O and local I/O with data staging for the input dataset. The table shows that neither data staging nor remote I/O is always preferred. The models identify the best alternative and predict the overall completion time fairly accurately even when data staging is involved.

Candidate Assg.	CPU speed (MHz)	Network latency (ms)	Data staging done?	Actual time (mins)	Model Predicted (mins)
1.1	451	14	Yes	54.47	53.48
1.2	451	14	**No**	35.16	37.65
2.1	996	14	**Yes**	23.43	22.78
2.2	996	14	No	28.25	26.71
3.1	996	2	Yes	24.63	24.44
3.2	996	2	**No**	14.58	15.40
4.1	451	16	**Yes**	28.21	25.51
4.2	451	16	No	37.43	39.81

TABLE VI

CANDIDATE ASSIGNMENTS FOR *fMRI* WITH AND WITHOUT INPUT DATA STAGING; THE PREFERRED CHOICE OF EACH PAIR IS SHOWN IN BOLD.

VI. CONCLUSIONS AND FUTURE WORK

This paper outlines our approach to active learning of simple performance models for batch tasks, using instrumentation data gathered at the resource level, without invasive changes to system or application software. The mean percentage error from model predictions is under 11% for a set of batch tasks captured from a production cluster.

The ultimate goal of the NIMO project is to optimize resource assignments for complex workflows across a networked utility. Performance models are useful to guide assignments of shared resources to competing workloads. For example, a utility resource manager could use the models to select from among heterogeneous hardware alternatives, or to size resource reservations for virtualized resources (e.g., virtual machines). This paper provides several examples illustrating the promise of simple application models for resource planning in a networked utility.

However, further steps are needed to determine if this approach is practical in real deployments. One limitation of our work to date is that it does not capture the impact of different input data sets on application behavior; many applications show regular behavior across different input data sets, but not all. Another ongoing focus is active sampling strategies to select training sets automatically, to maximize coverage of the operating range of resources and capture complexity in the training data. On an alternate path, we are investigating learning of queuing models that capture concurrency-related behavior explicitly in the model structure.

ACKNOWLEDGEMENTS

Lavanya Ramakrishnan, Emma Buneci, John Pormann, and David Becker assisted with the biomedical applications and testbed configuration. Adriana Iamnitchi and Aydan

Yumerefendi contributed to this work at an early stage. We thank the anonymous reviewers whose comments and insights greatly improved the paper.

REFERENCES

[1] W. T. Ng, B. Hillyer, E. Shriver, E. Gabber, and B. Ozden, "Obtaining High Performance for Storage Outsourcing," in *Proceedings of the Conference on File and Storage Technologies*, Jan 2002.

[2] 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.

[3] D. A. Reed, L. Ramakrishnan, *et al.*, "Service-Oriented Environments in Research and Education for Dynamically Interacting with Mesoscale Weather," in *Proceedings of Computing in Science and Engineering*, Nov-Dec 2005.

[4] J. Gray, "Distributed Computing Economics," IEEE TFCC Newsletter http://www.clustercomputing.org/content/tfcc-5-1-gray.html, Mar 2003.

[5] 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.

[6] W. Elwasif, J. Plank, and R. Wolski, "Data Staging Effects in Wide Area Task Farming Applications," in *IEEE International Symposium on Cluster Computing and the Grid*, May 2001.

[7] T. Phan, K. Ranganathan, and R. Sion, "Evolving Toward the Perfect Schedule: Co-scheduling Job Assignments and Data Replication in Wide-Area Systems Using a Genetic Algorithm," in *International Workshop on Job Scheduling Strategies for Parallel Processing*, vol. 3834, 2005, pp. 173–193.

[8] D. Ellard and M. Seltzer, "New NFS Tracing Tools and Techniques for System Analysis," in *Proceedings of the Annual Large Installation System Administration Conference*, Oct 2003.

[9] Y.-S. Kee, H. Casanova, and A. Chien, "Realistic Modeling and Synthesis of Resources for Computational Grids," in *Proceedings of the ACM/IEEE Conference on Supercomputing*, Nov 2004.

[10] R. Doyle, J. Chase, O. Asad, W. Jin, and A. Vahdat, "Model-Based Resource Provisioning in a Web Service Utility," in *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, Mar 2003.

[11] C. Stewart and K. Shen, "Performance Modeling and System Management for Multi-component Online Services," in *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation*, May 2005.

[12] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi, "An Analytical Model for Multi-tier Internet Services and its Applications," in *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, Jun 2005.

[13] B. Urgaonkar, P. Shenoy, and T. Roscoe, "Resource Overbooking and Application Profiling in Shared Hosting Platforms," in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, Dec 2002.

[14] E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, and A. Veitch, "Hippodrome: Running Circles Around Storage Administration," in *Proceedings of the Conference on File and Storage Technologies*, Jan 2001.

[15] E. L. Miller and R. H. Katz, "Input/Output Behavior of Supercomputing Applications," in *Proceedings of ACM/IEEE Conference on Supercomputing*, Nov 1991.

[16] B. K. Pasquale and G. C. Polyzos, "A Static Analysis of I/O Characteristics of Scientific Applications in a Production Workload," in *Proceedings of ACM/IEEE Conference on Supercomputing*, Nov 1993.

[17] V. Taylor, X. Wu, and R. Stevens, "Prophesy: An Infrastructure for Performance Analysis and Modeling of Parallel and Grid Applications," in *Proceedings of ACM Sigmetrics Performance Evaluation Review*, Mar 2003.

[18] J. Xu, S. Adabala, and J. A. B. Fortes, "Towards Autonomic Virtual Applications in the In-VIGO System," in *Proceedings of International Conference on Autonomic Computing*, May 2004.

[19] L. Eeckhout, H. Vandierendonck, and K. Bosschere, "Quantifying the Impact of Input Data Sets on Program Behavior and its Applications," *Journal of Instruction-Level Parallelism*, vol. 5, pp. 1–33, Feb 2003.

[20] M. Bennani and D. A. Menascé, "Resource Allocation for Autonomic Data Centers Using Analytic Performance Models," in *Proceedings of International Conference on Autonomic Computing*, May 2005.

[21] T. Kelly, "Utility-Directed Allocation," in *First Workshop on Algorithms and Architectures for Self-Managing Systems*, Jul 2003.

[22] L. Carrington, M. Laurenzano, A. Snavely, R. Campbell, and L. Davis, "How Well Can Simple Metrics Represent the Performance of HPC Applications?" in *Proceedings of the ACM/IEEE Conference on Supercomputing*, Nov 2005.

[23] E. Rosti, G. Serazzi, E. Smirni, and M. S. Squillante, "Models of Parallel Applications with Large Computation and I/O Requirements," *IEEE Transactions on Software Engineering*, vol. 28, no. 3, pp. 286–307, 2002.

[24] L. T. Yang, X. Ma, and F. Mueller, "Cross-Platform Performance Prediction of Parallel Applications Using Partial Execution," in *Proceedings of ACM/IEEE Conference on Supercomputing*, Nov 2005.

[25] M. Beck, T. Moore, J. S. Plank, and M. Swany, "Logistical Networking: Sharing More Than the Wires," in *Active Middleware Services*, C. A. L. S. Hariri and C. S. Raghavendra, Eds. Norwell, MA: Kluwer Academic, 2000.

[26] T. Kosar and M. Livny, "A Framework for Reliable and Efficient Data Placement in Distributed Computing Systems," *Journal of Parallel and Distributed Computing*, vol. 65, no. 10, pp. 1146–1157, 2005.

[27] K. Amiri, D. Petrou, G. R. Ganger, and G. A. Gibson, "Dynamic Function Placement for Data-intensive Cluster Computing," in *Proceedings of the USENIX Annual Technical Conference*, Jun 2000.

[28] B. A. Fields, S. Rubin, and R. Bodik, "Focusing Processor Policies via Critical-Path Prediction," in *Proceedings of International Symposium on Computer Architecture*, Jun 2001.

[29] H. J. Curnow and B. A. Wichmann, "A Synthetic Benchmark," *The Computer Journal*, vol. 19, no. 1, pp. 43–49, Feb 1976.

[30] L. McVoy and C. Staelin, "lmbench: Portable Tools for Performance Analysis," in *Proceedings of the USENIX Annual Technical Conference*, Jan 1996.

[31] 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.

[32] S. Vazhkudai and J. M. Schopf, "Using Regression Techniques to Predict Large Data Transfers," *International Journal of High Performance Computing Applications*, vol. cs.DC/0304037, 2003.

[33] S. A. Huettel, A. W. Song, and G. McCarthy, *Functional Magnetic Resonance Imaging*. Sinauer Associates, Inc., 2004.

[34] J. Pormann, J. Board, D. Rose, and C. Henriquez, "Large-Scale Modeling of Cardiac Electrophysiology," in *Proceedings of Computers in Cardiology*, Sep 2002.

[35] S. Altschul, T. Madden, A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. Lipman, "Gapped BLAST and PSI-BLAST: A New Generation of Protein Database Search Programs," *Nucleic Acids Research*, vol. 25, pp. 3389–3402, 1997.

[36] J. C. Phillips, R. Braun, *et al.*, "Scalable Molecular Dynamics with NAMD," *Journal of Computational Chemistry*, vol. 26, pp. 1781–1802, 2005.

[37] J. Moore, J. Chase, K. Farkas, and P. Ranganathan, "Data Center Workload Monitoring, Analysis, and Emulation," in *Proceedings of Computer Architecture Evaluation using Commercial Workloads*, Feb 2005.

[38] M. Carson and D. Santay, "NIST Net: A Linux-Based Network Emulation Tool," *SIGCOMM Computer Communication Review*, vol. 33, no. 3, pp. 111–126, 2003.

[39] 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.

[40] J. Friedman, "Multivariate Adaptive Regression Splines," *Annals of Statistics*, vol. 19, no. 1, pp. 1–141, 1991.