Inductive Transfer Learning for Incremental Modeling and Optimization of Cloud Systems Performance

by

Francesco Iorio

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

Inductive Transfer Learning for Incremental

Modeling and Optimization of Cloud Systems Performance

Francesco Iorio

Doctor of Philosophy

Graduate Department of Computer Science

University of Toronto

2020

Due to the cost of sampling system performance, it is expensive to obtain performance characteristics of a complex computer system in different configurations. As an alternative, in this dissertation, we propose to reuse existing partial and full performance models and explicitly model the effects of configuration variations, with a goal of substantially reducing the time and effort required to perform performance reasoning and optimization on cloud-based systems, applications and services.

We introduce Model Mapping, a novel inductive transfer learning technique for incremental performance modeling of highly configurable systems. Model Mapping captures many explicit and latent types of dynamic system evolution, including configuration changes, scaling and hardware upgrades, by deriving and modeling these kinds of incremental transformations between system and/or application instances, over time. Modeling these transformations allows us to build accurate models for new configuration instances with just a few samples. We experimentally test our method on a variety of system performance modeling and optimization scenarios, using a carefully designed experimental testbed and realistic benchmarks, to obtain insight on the method's applicability in real-world cloud computing environments. Among other examples, we show how our method can be used to quickly derive an accurate resource allocation split that optimizes a given overall performance goal for co-hosted applications in a virtualized environment.

Compared to using conventional direct and incremental modeling techniques, our method achieves higher accuracy by up to an order of magnitude when the sampling budget is extremely limited, in particular when samples are limited to between 0% to 5% of an exhaustive sampling budget.

Dedication

To my parents Aldo and Resy.

Acknowledgements

Having the chance to investigate in depth and to model the characteristics of complex computer systems is a real pleasure for someone interested in what happens right at the hardware/software interface. I was fortunate enough to meet some fantastic people during this journey, and I am truly grateful to all of them for making this an experience I will treasure for the rest of my life.

My first thought goes to my parents, who allowed me to stray from the path they had in mind for me early on, and to follow my passion and curiosity. I also thank my wife Francesca and my daughter Arianna for supporting me throughout the years.

I want to deeply thank my co-advisors, Professor Cristiana Amza and Professor J. Christopher Beck, who supported and encouraged me since we first met, and incessantly provided invaluable feedback all throughout my journey. I am also deeply thankful to the members of my committee: Professor Angela Demke Brown patiently helped me over the years to narrow down my overly broad ideas into a focused topic, and Professor Bianca Schroeder provided excellent perspectives and recommendations on application scenarios for my research. I am also grateful to Professor Bettina Kemme as my external examiner, and to Professor Baochun Li for being a member of my final examination committee, your questions and comments gave me extra motivation to further pursue this line of research in the future.

Such an experience would not have been nearly as memorable, had it not been shared with friends and colleagues. I had two truly amazing collaborators in Ali Hashemi and Michael Tao, this research would probably not have happened without your help. Friends from UofT, Alex Tessier, Stelios Sotiriadis, Jin Chen, Madalin Mihailescu, Lawson Fulton, Seyed Ali Jokar, Arnamoy Bhattacharyya, all contributed with their knowledge, advice and support. I want to also acknowledge family and friends from outside the university, and in particular Andrew Delong, Ian Ameline, Max Meneghin, Aaron Szymanski, Max Moruzzi, Adrian Butscher and Nigel Morris for enduring their share of my ramblings over the years.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Public cloud platforms, such as Amazon Web Services (AWS) [1], Microsoft Azure [66] and Google Cloud [42], and private cloud systems deployed in large organizations' datacenters, are increasingly used for hosting a large variety of applications in a highly dynamic, large-scale environment, leveraging economies of scale to reduce the cost of serving these applications to clients. Competition among cloud service providers creates a constant need to use resources efficiently, in order to lower the cost of operating datacenters. Public cloud providers and organizations operating their own systems are also interested in minimizing the cost of maintaining these large IT infrastructures.

Datacenters are extensive IT server and storage infrastructures that are operated at scale, and the largest individual cost component of operating them is the electricity required to power the servers themselves and the supporting environmental and cooling systems [80]. For this reason it is common for cloud infrastructure and service providers to attempt to lower the operating costs by reducing the number of servers active at any given time, and multiplexing the collective, heterogeneous resources of a cloud datacenter across the many hosted applications. Cloud providers are required to respect a contractually agreed Quality of Service (QoS) for each application or service, and applications' performance is affected by sharing hardware resources. Therefore, significant care is placed on the choice of resource allocation. Finding

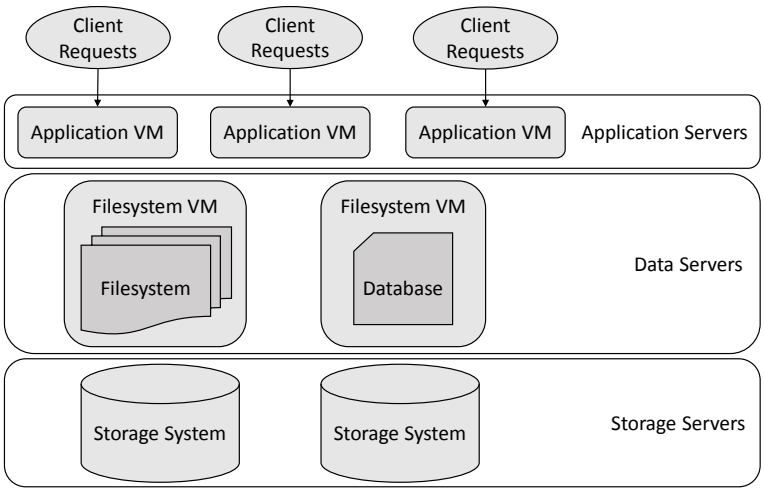


Figure 1.1: Example virtual infrastructure for multi-tenant filesystem and database service operation.

the amount and allocation of system resources necessary to satisfy QoS with a minimum cost is called *application consolidation* or *resource consolidation* [95].

Systems of the scale of a datacenter are complex, multi-tiered, interconnected and highly configurable, and are therefore difficult to model, configure and optimize. Figure 1.1 shows a simplified example of the internal components of a storage subsystem for a large IT infrastructure, composed of layers of interconnected services and subsystems, most of which have numerous settings, parameters and large configuration spaces. Furthermore, applications and services are generally highly configurable themselves, which further complicates the challenge of finding appropriate combinations of resource quotas and application configurations to efficiently use the underlying hardware. Even expert system administrators cannot hold a complete mental model of the effects individual resources and application configuration parameters have on the overall efficiency, and therefore rely on a variety of models to predict and verify opportunities to tune specific subsystems.

As an example. Figure 1.2 shows the results of measuring the latency of a database system

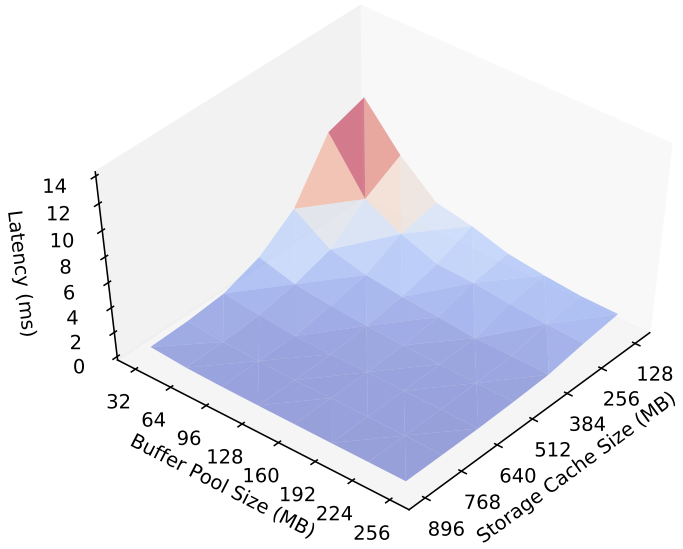


Figure 1.2: Graph of database performance *TPC-C*.

using the TPC-C benchmark, while varying the quotas of two system resources available to the application.

The space of all possible configurations grows exponentially with every resource or configuration parameter considered, and it very quickly becomes impractical to enumerate all the possible configurations. Generally, the task of modeling computer systems behavior is therefore handled by traditional mathematical modeling methods [10, 29, 30, 45, 114], which perform well for systems with a relatively limited number of configuration options.

Building empirical models for systems of a higher level of complexity requires adopting different forms of automated, data-driven modeling, which rely on obtaining a large number of samples by actuating a system in a series of controlled configurations to produce measurements. Machine learning methods such as Support Vector Machines [89] can be applied to obtain fast, low-dimensional models from the sets of sparsely sampled measured data, but still require many samples to approximate the system behavior with reasonable accuracy. The cost and time required to obtain sufficient samples to model a large system can be prohibitive, and

the natural tendency of systems behavior to vary over time as a side effect of maintenance and operation makes constantly rebuilding these models a daunting and expensive task.

In this dissertation, we observe that the challenges faced by system administrators to appropriately operate systems at scale are becoming increasingly insurmountable when using conventional performance modeling techniques. These systems' extensive configurability and constant evolution make it infeasible to rely on human intuition to ascertain the viability of performance models over time, or to frequently rebuild these models from scratch.

1.1 Problem Statement

The fundamental thesis in this dissertation is that:

The key to preserving the accuracy of an application or a system performance model throughout its lifetime is to model the configuration transformations that a system experiences, leveraging existing information in the form of partial or full legacy models. A model of this kind can therefore be quickly updated, and continuously leveraged for efficient resource provisioning and consolidation.

We introduce Model Mapping, a homogeneous incremental inductive transfer learning technique [72], for deriving new performance models for dynamic variations in a system and its hosted applications. These variations may be due to changes in system configuration, such as scaling, and upgrades, changes in data sets, workloads and so on.

Our technique aims to directly represent the transformations that systems may experience. The modeled transformation, called a *map*, relates one or more legacy models of the original system and a new model of the modified system. This *map* is itself a mathematical function. Intuitively, we expect the *map* to be a linear or nonlinear scaling transformation, and hence be of lower cardinality than the performance model itself. The *map* will also be simpler and faster to build with just a few new samples compared to building a new model for the new configuration instance from scratch.

Our solution leverages pre-existing knowledge of a system's behavior to quickly and efficiently model extensions and variations of its configuration (both physical and parametric), to preserve the accuracy of performance models throughout a system's lifetime. This approach recognizes the value of knowledge encoded in data and/or legacy models, and efficiently represents the evolution of a system's behavior through that data.

We are particularly interested in understanding the opportunity of obtaining viable models in scenarios where the sampling budget is very low (e.g. < 10 samples). Our experiments indicate that configuration changes affect system performance globally across the entire configuration space, rather than in small portions of the configuration space. This observation suggests that these transformations are best modeled as a global transformation function when the sampling budget is very low, rather than leveraging standard transfer learning methods.

In some cases, it is possible that the system or application change is of such a nature that little or no similarity exists between the original system model and the new model. In such cases, the cost of building the map may be the same or higher than directly building a new model from scratch. Therefore, to get the best of both worlds, we may use new samples for building both the map and the new model, in parallel, until the accuracy of either modelling approach converges to within a desired threshold.

An important benefit of tracking system and application evolution through a library of historical mapping functions is the ability to derive trends and interpolate across existing models, even when these models are partial or of different cardinality. Besides obtaining new models with a low sampling budget, in principle we may derive a new model or answer what-if scenarios even with no new samples.

When little or no sampling data exists, our method provides higher modeling accuracy, more flexibility, or both, compared to regression methods based on direct modeling. In our experiments, we observe that modeling the transfer function explicitly has specific advantages in scenarios with very limited sampling budgets over conventional modeling methods and other transfer learning methods. Thus, we find our technique to be uniquely suitable to the dynamic,

long running and constantly evolving nature of Cloud platforms and their typical applications.

In this dissertation we provide details of Model Mapping and its application to a variety of performance modeling scenarios. In all scenarios, we find that with a sampling budget between 0% and 5% of the required samples needed for exhaustive sampling, our method achieves higher accuracy by up to an order of magnitude, relative to direct modeling, incremental modeling and other forms of transfer learning.

We also introduce *ModelMap*, a flexible toolkit designed to provide support for experimentation and deployment of Model Mapping and other direct and incremental modeling methods. *ModelMap* allows rigorous and systematic comparison of the accuracy of multiple modeling methods in the presence of varying sampling budgets and partial models. Additionally, *ModelMap* supports a customizable representation of the transfer maps.

This research is the result of a series of progressively deeper investigations in automated system performance modeling and optimization in our group at the University of Toronto. Our work builds on the previous contributions and experiences in our group to apply combinations of heuristic, semantic and data-driven models towards the goal of obtaining efficient methods and tools for the management of highly configurable systems [40, 92, 93].

In recent years, incremental modeling and transfer learning have had some initial success in adjacent fields, although very limited exploration has been conducted in their application to systems performance. In our investigation, we could find only very limited published work on the subject, including our own. updatedIn particular, our work on the application of Gaussian Processes for incremental modeling [24] was an early attempt to improve model reuse in cloud systems performance modeling.

The main point of reference for baseline comparison is therefore our group's *Chorus* framework [25], a powerful system for storage and retrieval of legacy models, which allows for incremental modeling.

We now provide a summary of the most salient contributions, which constitute the foundations of this dissertation.

1.2 Contributions

In this dissertation we show that while a system's behavior changes over time by means of configuration modifications, it retains some similarities across these changes, and that it is possible to systematically leverage these similarities to achieve substantial savings in the time and cost to obtain performance models of transient systems, and therefore to optimize their configuration.

We introduce Model Mapping, an inductive transfer learning technique for systems performance modeling that embodies this intuition. Given an existing performance model of a system, we transfer this existing knowledge to an incremental variation of the system or to a novel system by learning a simple, low-dimensional relationship. This process is used to improve the accuracy of our approximation for system performance, given a limited sampling budget.

We also present *ModelMap*, a toolkit designed to leverage Model Mapping and other forms of incremental and transfer learning for performance modeling.

Finally, we present three case studies that represent realistic application scenarios in modern virtualized datacenters. All the scenarios involve datasets that have been systematically collected from measuring actual physical systems, using realistic benchmarks that are faithful representation of application load on production systems. The first represents a process for allocating resources to database systems and their applications in a multi-tenant cloud environment. The second represents performance modeling for the configuration of a file storage system. The third represents performance modeling for a set of configurable hosted applications and services. All case studies demonstrate the effectiveness of incremental modeling, and of our technique in particular.

Our contributions are in summary:

1. **Model Mapping incremental modeling technique.** We study the fundamental characteristics of system performance modeling. We propose a novel inductive transfer learning

technique to increase the accuracy of new performance models with small sampling budgets. Our technique uses existing information about a system, embodied in legacy models, to efficiently capture the variations a system may experience by representing their effects in relation to the legacy models using transfer functions. We observe in some basic scenarios how the lower dimensionality of a function representing this relation may provide an advantage over different incremental learning techniques. We then compare and contrast our technique with a selection of direct and incremental modeling methods, and find that it is specifically advantageous in situations where the incremental variations to a system result in non-linear, but not overly complex behavioral modifications.

2. ***ModelMap* toolkit.** We design and implement a flexible software toolkit to experiment and compare Model Mapping with other direct and incremental modeling techniques. The toolkit is designed to be easily extended and integrated with resource configuration automation systems, by implementing application interfaces compatible with widely available mathematical modeling libraries such as scikit-learn [85]. *ModelMap* has been used to conduct all the experiments for this research, and is publicly available as an open-source project [67].

3. **Application of incremental modeling to transient systems.** We apply our technique and verify its effectiveness on a variety of system performance modeling scenarios, designed to represent situations commonly found in real-world cloud systems. As part of this effort, we designed and instrumented an experimental testbed to obtain an extensive collection of performance measurements for a representative virtualized database system. These samples were collected to model the performance of databases in the presence of resource quota configurations. The resulting dataset [109] is publicly released as part of *ModelMap*. In particular, we examine scenarios representing the performance effects of configuring a multi-tier virtualized database system, a file storage system, and a set of hosted applications. In these scenarios we identify three opportunities to leverage incre-

mental modeling, where we find our technique demonstrates substantial effectiveness:

- Extension of a performance model's configuration space. We analyze a variety of scenarios where a performance model built to represent a system or an application/service requires extensions to its domain to cover additional resources and/or parameters that were not previously modeled. We demonstrate how in these scenarios, our technique can leverage legacy information and substantially increase the accuracy of modeling additional resources and parameters with a limited need of measurements.

- Modeling explicit and latent incremental changes to a system. We observe the effects of modifications to systems configuration, as a result of direct configuration alterations or indirect, latent effects. Unsurprisingly, direct configuration variations, such as changing a specific resource quota, upgrading a hard disk to an SSD or varying a filesystem scheduler parameter, may have profound, highly non-linear effects on a system's performance. Latent effects on performance, such as the size of a database growing over time, also affects the ability of a service to satisfy the necessary QoS, although the variations may be less abrupt. In both situations, we find that applying our technique results in substantial improvement in the accuracy of the obtained models, which gives system administrators the ability to react quickly to these variations.

- Resource optimization and application consolidation in virtual systems. Given the importance of consolidation to the general operations of large-scale cloud systems, we observe the ability of incremental models built using our technique of capturing a system's behavior with sufficient accuracy to be used for the purpose of the optimization of virtual resource quotas. We show how our technique requires very few samples to build effective incremental models for this usage scenario, and as part of the comparison of our technique with other direct and incremental modeling

methods, we also observe the adverse effects of using inaccurate models.

1.3 Organization

The outline of this dissertation is as follows: Chapter 2 introduces the background and motivation for this research. Chapter 3 presents Model Mapping, our incremental modeling technique, and *ModelMap*, our modeling and transfer learning toolkit. Chapter 4 describes the results of our experiments on database system performance scenarios. Chapter 5 describes the results of our experiments on filesystem performance scenarios. Chapter 6 describes the results of our experiments on hosted applications and services performance scenarios. Chapter 7 presents related work. Chapter 8 concludes the dissertation and outlines avenues for future work.

Chapter 2

Background and Motivation

Modeling the behavior of complex systems is key to many scientific fields and is often necessary to make well-informed decisions. Our work aims to reduce the effort and improve the timeliness involved in obtaining accurate models of a computer system's performance behavior, as it varies due to explicit and latent changes to its configuration. This chapter provides an overview of the characteristics and challenges of operating cloud systems efficiently, and the critical importance that accurate performance models have for the task of selecting appropriate system configurations.

2.1 Performance Modeling and Resource Consolidation

Efficiently operating large computing systems to host numerous applications concurrently is a delicate balancing act. The vast majority of the cost of operating such systems is directly related to the energy consumption of the computing and storage servers themselves, plus the energy consumption of the mechanical systems designed to maintain the appropriate environment (temperature, humidity) for the servers to operate [12, 76].

However, applications are generally required to satisfy desired levels of responsiveness, which in cloud environments are defined as Quality of Service (QoS) or Service Level Agreements (SLAs). These SLAs define thresholds of performance that a cloud service provider's

operations team is required to ensure for all applications hosted by the system.

If cost were not a factor, each application could be hosted by one or more dedicated servers, in a solution designed to ensure the desired SLA is met in a worst-case demand scenario. In reality, the requirement to reduce the operating cost, and the realization that systems designed for worst-case scenarios generally run at a fraction of their capacity [9], pushed operators to aggregate multiple applications on a shared pool of servers.

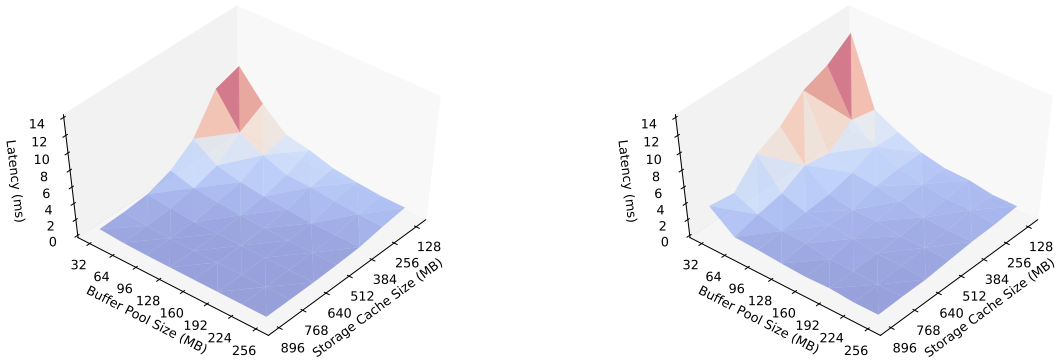
Applications have different behaviors and resource demands, so it would be detrimental to share a server across multiple applications that use the same resource (e.g. the CPU) heavily, while leaving other resources (e.g. system memory) under-utilized. While most modern operating systems include sophisticated features to handle the management and sharing of individual system resources among applications, increased pressure to host multiple applications on the same system required additional levels of application isolation. Additionally, dealing with issues of precise control of resource utilization and incompatibility of different applications with different operating systems and security requirements, favored the emergence of methods to further abstract the hardware resources present on generic servers [18].

Virtualization systems such as VMware ESX Server [122], Xen [8] and KVM [46] introduced a layer of abstraction between hardware and the operating systems and applications, allowing a greater isolation of applications and a substantially finer-grained control over the allocation of system resources to applications. However, while virtualization technologies underpin most efforts in performance isolation on individual servers and large distributed systems, they do not directly assist in the resource allocation. Different applications tolerate resource starvation differently, so the assignment of Virtual Machines to servers, and the choice of appropriate resource quotas require a thorough understanding of how their performance reacts to resource availability.

So substantial are the time and cost involved in directly testing the effects of different resource allocations, that the ability to model and predict the performance of each application running in a system, based on how its resource allocation is configured, becomes one of the

most useful tools for application and resource consolidation [54]. A performance model is a mathematical function which maps resource configurations to application performance. Specifically, each point in the model represents the predicted application performance, given a set of system resource quotas e.g., memory, CPU, disk bandwidth. When sufficient knowledge of a system's behavior is available in advance, a performance model can take a purely analytical form, and it is therefore representable by closed-form expressions. Conversely, when the complexity of a system's behavior is higher and the available information is insufficient to study the performance function directly, models are built using observations (samples) of the system's performance. Covering a system's configuration space may require performing an extensive set of measurements, and therefore the most significant challenge is the sampling time and cost.

In particular, in the context of system performance modeling for resource allocation, the sampling time depends on the number of resources modeled and the number of resource quotas sampled for each resource. Moreover, specific systems require additional care to ensure the statistical significance of the measurements taken. Measuring the performance of database systems, for example, needs to allow for cache warm-up time, which increases the time required to perform a single measurement.



(a) *Benchmark* = *TPC-C* (b) *Benchmark* = *TPC-W*

Figure 2.1: Graph of database performance (a) *TPC-C* and (b) *TPC-W*.

As an example, Figure 2.1a and Figure 2.1b represent performance models for the TPC-

C transactional benchmark [106] and the TPC-W e-commerce benchmark [111], respectively, when running in a virtualized environment. In this scenario, each application is allocated various quotas of two system resources: *storage cache* and *buffer pool size*. These performance models represent the page access latency at the database buffer pool as a function of the *storage cache* and *buffer pool* configuration parameters, varied between *128*MB and *896*MB, and between *32*MB and *256*MB, respectively. Building this model by combinatorial sampling of the configuration space requires a significant amount of time. Each performance measurement requires the following steps:

1. Initializing the storage system to a predetermined, deterministic state.
2. Configuring the virtualization system to provide the desired resource quotas to the application.
3. Starting the database system's VMs and waiting for all the services to become available.
4. Waiting for the database cache to warm-up.
5. Running the benchmark application repeatedly to ensure statistical significance.
6. Shutting down the database system's VMs.

Sampling time for building each of the two models was approximately 10 days. The higher the number of resources modeled, the higher the sampling cost for building the model.

For many real-world applications with much larger configuration spaces, a sampling time of weeks or months is prohibitively expensive and infeasible, and therefore exhaustive sampling is replaced with training regression models (black-box models [47]) using smaller training sets. The larger the available set of samples, the more accurate the model that can be obtained.

The work by Soundararajan et al. [92] shows the benefits of online resource allocation adaptation for a database system, studying the effects of applying quotas to two system resources: the database buffer pool and storage cache. The work presents the challenges in

finding good configurations in a reasonable amount of time, and the related need for application performance models to retrieve these configurations efficiently. In this case, modeling the application performance as a function of resource quotas used Support Vector Machine Regression [89]. Van Aken et al. [118] also apply a machine learning method (Gaussian Process Regression) for the purpose of automatically tuning the configuration of a database system, using a smart selection of training samples.

Over time, complex systems such as a cloud infrastructure undergo a multitude of software and hardware replacements, extensions, upgrades and general maintenance. Any incremental, but significant, variations of a system's configuration may render previously obtained performance models obsolete. Specifically, situations exist where one may wish to incorporate new resource types into a system or application, or situations that would require changing the number of available system resources or configurable parameters. For example, one may wish to extend an existing performance model of the latency of a storage hierarchy to cover the influence of imposing CPU quotas.

Furthermore, systems may exhibit latent performance variations, which produce an additional source of inaccuracy that negatively effects performance reasoning and optimization efforts. For example, it has been observed that similar systems deployed in similar environments, e.g. web servers, suffer from tuning-related delays [6, 27, 70], availability, or performance problems [129, 130]. To compensate for these sources of inaccuracy, performance models need to be periodically rebuilt, further amplifying the modeling costs. In all these circumstances, the traditional approach is to build a new model from scratch in any new configuration. This approach yields a model of the application or system in the new configuration, based on all the sampling data available, using black-box [37] or grey-box models [103], and may incur in a substantial latency between a configuration change and the availability of a new model.

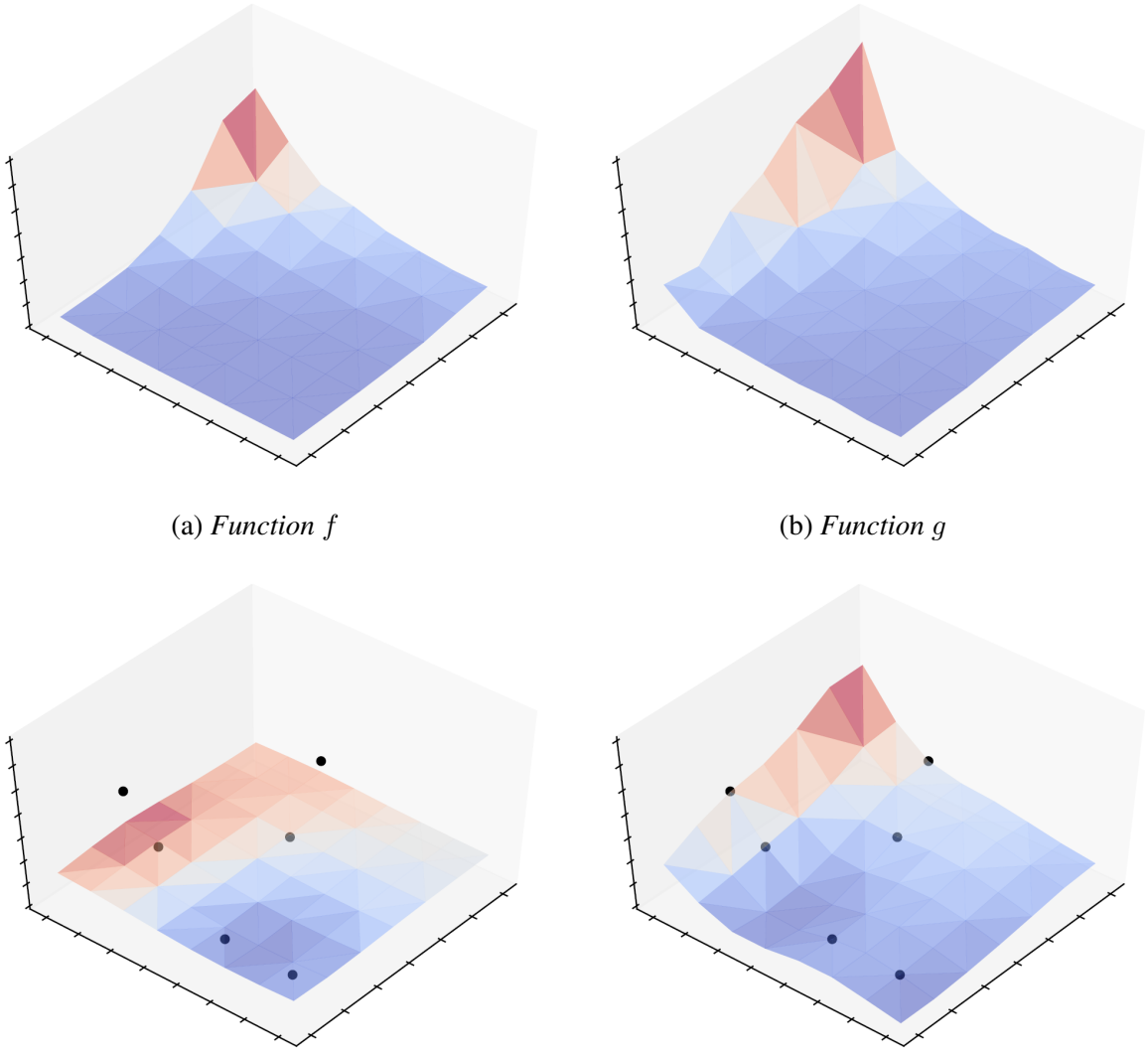
2.2 Transfer Learning

Transfer learning is a branch of research in computer science which aims to reuse knowledge for the purpose of solving related problems/tasks [73, 105]. This approach mimics the way humans extract useful experience from different activities and abstract some notions to improve their ability to solve previously unseen, but related challenges. In particular, transfer learning leverages partial structural similarity between tasks, expressed as mathematical models, to improve modeling accuracy and to reduce the need for a large number of samples in a model's training set.

Let $A \in R^n$ and $B \in R^m$ and $f : A \rightarrow B$ be a function we are interested in modeling. With a sufficient training set of pairs $\{x \in A, y \in B\}$, f can be approximated by a mathematical model σ, such that $\sigma \approx f$. Transfer learning methods use knowledge encoded in σ to reduce the effort in modeling a different function g, where $g : A \rightarrow D, D \neq B$ or $g : C \rightarrow B, C \neq A$.

An example of transfer learning is presented in Figure 2.2. Figures 2.2a and 2.2b represent two functions, f and g. Figure 2.2c represents a model of the function g built with only six samples, shown as black dots. Clearly, the model captures very few of the characteristics of function g, and therefore exhibits a large error. Using transfer learning and the same set of six samples, and reusing knowledge of the entire function f, we obtain the model in Figure 2.2d. Transfer learning effectively leverages the similarities in the two models to produce a much more accurate model from limited samples.

At the time of writing, transfer learning is a very active field of research, as it is considered one of the fundamental avenues for the generalization of machine learning models used in multiple scientific fields, as they get increasingly complex and time consuming to train [31, 43, 51, 127]. Transfer learning has therefore emerged as a viable set of techniques to approach at the same time model complexity and scarcity of available training data for supervised learning. Many examples of application of transfer learning are present in literature, to approach problems such as text classification [28], defects analysis and classification in software applications [69] and image classification [55].



(a) *Function* f

(b) *Function* g

(c) *Direct model of* g

(d) *Transfer learning* $f \rightarrow g$

Figure 2.2: Comparison between direct modeling and transfer learning.

Based on the information we gathered in this chapter, we acknowledge the importance and complexity of performance modeling for the goal of successfully and economically operating cloud systems, and the theoretical advantages transfer learning presents in improving training speed and accuracy in multiple mathematical modeling scenarios. In the following chapters we tackle the challenge of incremental modeling of transient systems by introducing a novel transfer learning approach, and then proceed to experimentally evaluate its effectiveness in a set of scenarios designed to represent realistic circumstances in cloud systems operations.

Chapter 3

Modeling Systems Performance with Model Mapping

In this chapter we introduce our proposed Model Mapping technique, we present its fundamental characteristics and limitations, and discuss the considerations behind the design and implementation of *ModelMap*, our incremental learning toolkit.

Our work builds upon many years of intense experimentation and observation of the behavior of computer systems in our group [24, 25, 40, 92, 93], which led us to choose an approach to transfer learning that uses the results of these observations. As observed in Section 2.2, adopting transfer learning for the purpose of performance modeling and optimization has numerous advantages, as it addresses the need to systematically retain information about systems behavior, and at the same time provide an avenue to reuse this information to quickly react to explicit and implicit variations.

Model Mapping is a transfer learning technique based around the inherent characteristics of systems performance modeling, and the goal of reducing the dimensionality of performance modeling problems in transient systems, by focusing on the essential features that describe the transformations a system may experience. In particular, it is meant to represent relatively simple, non-linear, incremental system variations, and it focuses on scenarios where sampling

budgets are extremely low.

Other transfer learning approaches such as layer substitution in deep transfer learning [100] leverage similarities and differences between models using more extensive features (e.g. a full covariance matrix or complex non-linear models like Artificial Neural Networks), and often require larger training sets to be effective.

3.1 Model Mapping

System performance modeling is the task of obtaining a function that represents how a system's configuration parameters affect its performance characteristics. The space of possible configurations, denoted by $\mathcal{C} \subset \mathbb{R}^n$, is defined by a set of continuous or discrete parameters. At the same time, each performance characteristic (the result of measuring performances), is encoded as a value in a space $\mathcal{M} \subset \mathbb{R}^m$. The relationship between system configurations and performance characteristics is therefore encoded by a function $\rho : \mathcal{C} \to \mathcal{M}$.

Given a sufficient sampling of how different configurations perform, we can directly build a performance model for a new state of a system. Let us assume that, as one may expect in reality, there already exists a pre-built model ρ^0 of a system, which we deem the *legacy model* and we take to be trustworthy throughout $\mathcal{C}$. Rather than requiring a costly sampling procedure in order to build a performance model for the new system state ρ^1, it would seem prudent to utilize information of the system's behavior encoded in ρ^0.

For instance, when a system has a large number of configuration parameters, the curse of dimensionality [112] makes it infeasible to create a single performance model over the entire configuration space. Multiple models may be required to capture the performance of different sub-spaces of configuration space, where a subset of parameters is held constant. After a trustworthy model is built with a subset of static parameters, it would be prudent to generalize the model to more parameters rather than exhaustively sampling a different subspace.

Furthermore, multiple performance models would be required to continuously optimize

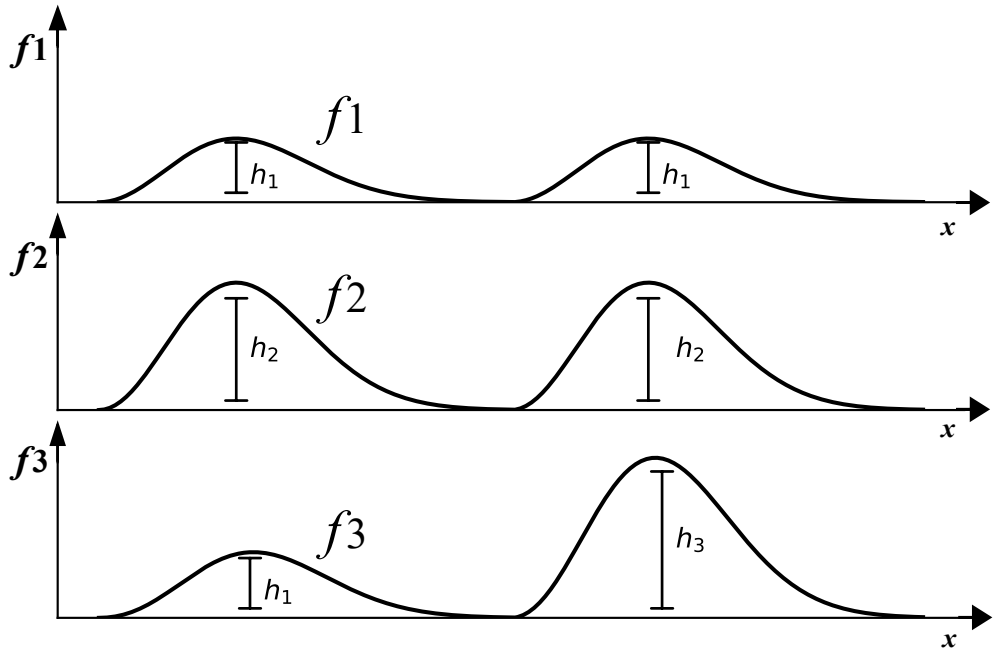


Figure 3.1: Graph of the performance function $f1$, with two incremental variations $f2$ and $f3$.

configurations of an evolving system. We claim that, rather than exhaustively sample and build a model for each passing system, a more effective and efficient strategy would be to model the system's variations from historically built models.

That is precisely what our proposed technique prescribes, by building a *map* (transformation) σ such that

$$\rho^1 \approx \sigma \circ \rho^0, \tag{3.1}$$

As an example of our technique, let us assume that we have a hypothetical system with a single configuration parameter x. Let graph $f1$ in Figure 3.1 be a legacy model for our system, a function which represents our system's response to all possible configurations of the parameter x. Say that an incremental change in the overall system mutates the performance function to become $f2$ in Figure 3.1. In such situations, $f1$ may cease to be sufficiently accurate, so building a new model of the system may become necessary. Intuitively, we can see that the optimal transformation of the model from $f1$ to $f2$ is in fact the scaling of $f1$ by a constant. This is in fact confirmed by plotting a few samples from $f1$ vs. $f2$, as can be seen in Figure 3.2.

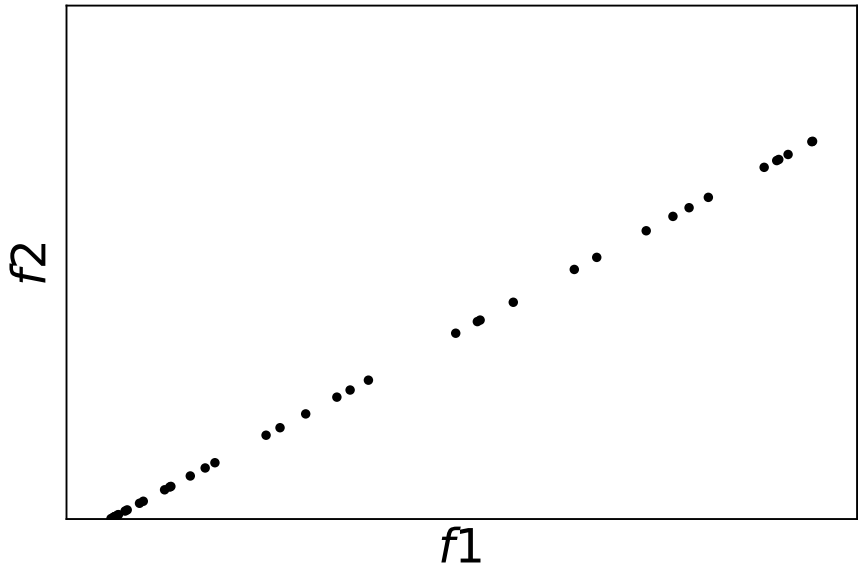


Figure 3.2: Graph of the functions $f1$ vs. $f2$.

Building a model for $f2$ directly would likely involve more sampling effort than building a model of the map between models $f1$ and $f2$ in performance space (from $\mathcal{M}$ to $\mathcal{M}$), independent of any configuration parameters. Such a map could be resolved from as few as a single sample from $f2$:

$$\sigma(y) = 2 \times y \qquad y \in \mathcal{M} \tag{3.2}$$

$$f2 = \sigma \circ f1 = 2 \times f1. \tag{3.3}$$

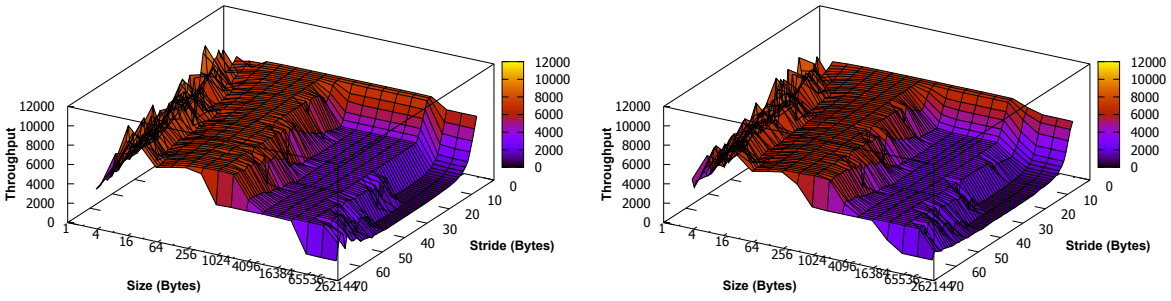
These maps are expressive, but simple. In some situations, the system under consideration may vary its performance characteristics due to a combination of configuration parameters, as depicted by $f3$ in Figure 3.1. In this case the map between $f1$ and $f3$ requires some awareness of x and therefore requires a feature from the configuration space. Even though $f3$ and the map have the same dimension, we hypothesize that a generalized map may still be easier to model than $f3$.

There are, of course, circumstances where a simple map is insufficient to appropriately represent the relation between two models. In such situations higher order maps are required, but we hypothesize that even in these circumstances, modeling a generalized higher order map may be simpler than modeling the system's performance function from scratch.

3.2 A Simple Example

The performance function we intend to model is the memory throughput of a computer system, as reported by the Memory Mountain micro-benchmark [17]. In this example, the system configuration parameters are **memory access size** and **memory access stride**, and the performance metric being modeled is the overall memory throughput in MB/s. We assume that we have obtained a trustworthy legacy model which represents the performance function of the system when an Intel Xeon E5-2650 microprocessor is used (Figure 3.3a).

The goal is to model the entire performance space of the same system, when the microprocessor is replaced with a different one: an Intel Xeon E5-4620 (Figure 3.3b).



(a) *Xeon E5-2650* (b) *Xeon E5-4620*

Figure 3.3: Performance graph of the Memory Mountain microbenchmark for two CPU architectures.

While the average throughput of the Xeon 4620 is nearly 15% higher than the cheaper Xeon 2650, at low and high memory access sizes the throughput is actually 30% lower. Despite these differences, the structural similarity between the two configurations' performance spaces can

be observed in Figure 3.4.

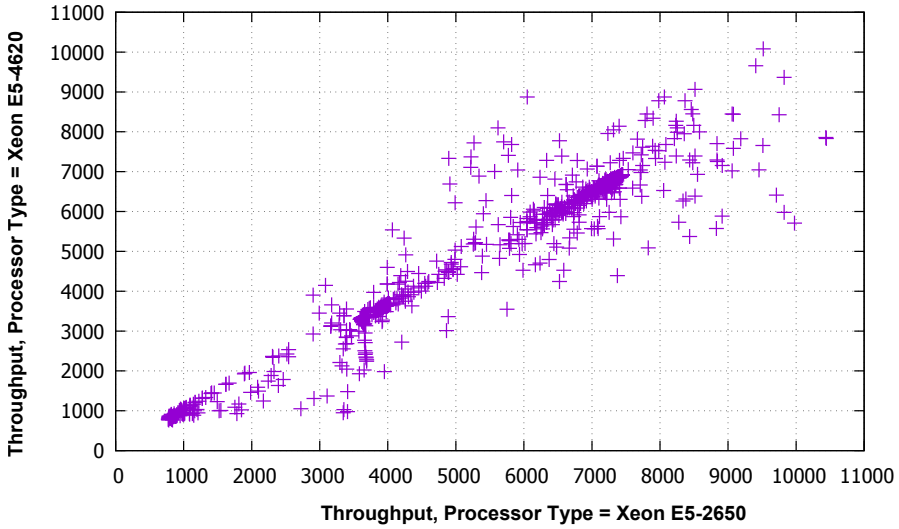


Figure 3.4: Graph of the relative performance spaces shown in Figures 3.3a and 3.3b.

Our technique exploits this similarity, and the results depicted in Table 3.1 show that it outperforms direct modeling for small sampling budgets. All the tested modeling methods (ref. Section 4.3), when used to find the map, exhibit better accuracy than the best performing method used to model the performance function directly. With a sampling budget of only 10 samples and a 1D map, the Root Mean Squared Error of our technique is up to three times lower than with direct modeling.

The results show that We see that even a simple 1D map, relating the two performance spaces independently from the configuration parameters, is sufficient in this case to transfer information from the legacy model to the new model.

3.3 Generalization of Model Mapping

Recall from Section 3.1 that we aim to model a function $\rho^1 : \mathcal{C} \rightarrow \mathcal{M}$, for which the only way to evaluate it is to perform an experiment/evaluation that is presumably costly. We introduced the concept of modeling the performance function ρ^1 indirectly, through modeling a simple

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	1,833.85±84.91	710.09±68.08
	Polynomial regression	2,134.64±73.67	1,677.65±123.61
	Linear SVR	1,731.43±69.69	**650.22±47.17**
	Gaussian Process	1,734.27±74.78	1,117.49±91.71
10	Linear regression	1,500.07±34.80	582.29±13.03
	Polynomial regression	1,794.36±59.99	1,478.58±119.04
	Linear SVR	1,495.99±35.81	**535.47±3.78**
	Gaussian Process	1,355.41±42.92	872.99±75.16

Table 3.1: Root Mean Squared Error of model mapping and direct modeling. Legacy model = Xeon E5-2650, unknown model = Xeon E5-4620.

relationship between two functions: a previously validated legacy model and ρ^1.

The relationship between a previously modeled performance function ρ^0 and ρ^1, for which no prior information is available is encoded by a *map* of the form $\sigma : \mathbb{R} \to \mathbb{R}$ so that

$$\rho^1(c) \approx \sigma \circ \rho^0(c) \qquad c \in \mathcal{C}. \tag{3.4}$$

3.3.1 Definitions

Our aim is to define the variations of a system's performance function based on changes in its configuration parameters or other latent variables as maps. We therefore introduce and discuss a taxonomy of such maps according to their input parameters. This taxonomy allows us to view the simple maps introduced in Section 3.1 as a basic case and to provide a clear direction for subsequent development.

We assume that, in addition to a given system configuration space $\mathcal{C} \subset \mathbb{R}^n$, we have already obtained a space of performance measurements $\mathcal{M} \subset \mathbb{R}^m$ representing the evaluation of the system's performance function $\rho^1 : \mathcal{C} \to \mathcal{M}$. The combination of a particular configuration and its performance measurements forms a unique set of features for that configuration. From this perspective, we can reinterpret the task of modeling a performance function by using some function from the aforementioned features to the desired objective, $\sigma : \mathcal{C} \times \mathcal{M} \to \mathbb{R}$. We

therefore must find some σ such that

$$\rho^1(c) \approx \sigma(c, \rho^0(c)) \qquad c \in \mathcal{C}. \tag{3.5}$$

At first glance, adding more parameters to σ may seem counterproductive, but the hope is that the available, pre-computed values of ρ^0, which constitute a subset of $\mathcal{M}$, may be more useful than simply the features from $\mathcal{C}$. However, it is only when we write σ in this general form that we can begin to remove features that are *either* configurations or performance measurements by forcing σ to ignore those features. This is equivalent to the processes of feature selection and extraction in machine learning [13], where more relevant features are derived to reduce modeling complexity.

In this context, we can organize our maps according to the kinds of features they utilize. In particular, to identify a map we use two sets of features $\mathcal{L} \subseteq \mathcal{C}$ and $\mathcal{K} \subseteq \mathcal{M}$. For generality we choose to classify maps using the cardinality of each set, and we therefore introduce the concept of a *class* to identify any particular σ type by the cardinality of the sets $\mathcal{L}$ and $\mathcal{K}$ it uses, which we will denote as a tuple $(\ell, k) = (|\mathcal{L}|, |\mathcal{K}|)$. This taxonomy provides an alternative perspective as to how we may want to model ρ^1. In the context we have established, using a map of class $(|\mathcal{C}|, 0)$ is equivalent to performing direct modeling.

Transformation functions different from simple composition may also be used. For example, one could model the difference between ρ^1 and ρ^0, as follows:

$$\sigma = \rho^1 - \rho^0 \tag{3.6}$$

$$\rho^1(c) \approx \sigma(c, \rho^0(c)) + \rho^0(c) \qquad c \in \mathcal{C} \tag{3.7}$$

3.3.2 Challenges and Limitations

Given a sufficiently general and expressive modeling method, a map of class $(|\mathcal{C}|, 0)$ represents the choice of modeling the performance function directly, assuming the model could approxi-

mate ρ^1 itself, but this is presumably prohibitively expensive, and therefore a map of different class ($\ell \leq |\mathcal{C}|, 0 < k \leq |\mathcal{M}|$) should be selected when legacy information is available. We assume that the cost of modeling is correlated positively with the number of features used.

In this context we can see the simple mapping technique introduced in Section 3.1 is a case where the transformation map relating the legacy model and new model was built using only features from the legacy function's codomain. The map uses $|\mathcal{M}| = 1$ and σ makes no use of any features from $\mathcal{C}$, therefore it is of class $(0, |\mathcal{M}|) = (0, 1)$.

In general, when using this simplest class of map, it is improbable that the model of ρ^1 will converge to a reasonable approximation due to the strong limitation in the features being used. For example, this class of 1-dimensional maps imposes a necessary condition that whenever the condition $\rho^0(x_1) = \rho^0(x_2)$ is encountered, then $\rho^1(x_1) \approx \rho^1(x_2)$ must hold as well. A more general way of stating this limitation is that the existence of a σ of class $(0, 1)$ implies that the level sets of ρ^0 and ρ^1 must coincide, and therefore the values of these level sets can be put into correspondence under σ.

This condition may not hold in general; though in a simple, 1-dimensional case one can assert that if ρ^1 is monotone, then the desired map of class $(0, 1)$ exists because we can have $\sigma = \rho^1 \circ {\rho^0}^{-1}$ where ${\rho^0}^{-1}$ is the inverse of ρ^0. In several scenarios of systems performance, the postulated relationship is approximately true when x_1 and x_2 are sufficiently similar.

We can see this assumption verified when analyzing the plot relating the codomains of the two performance functions reported in Figure 2.1a and Figure 2.1b in Section 2.1. The plot shows a mostly linear relationship between the two codomains.

Example with 1D function

We can now revisit the simple example from Section 3.1, where we have a hypothetical system with a single configuration parameter x, represented by graph $f1$ in Figure 3.1. When the system under consideration experiences a variation due to a combination of one or more configuration parameters, as depicted by $f3$ in Figure 3.1, a map of class $(0, 1)$ is insufficient to

model the transformation between $f1$ and $f3$. In this case the transformation function between functions $f1$ and $f3$ is a 2D function, which is depicted in Figure 3.5.

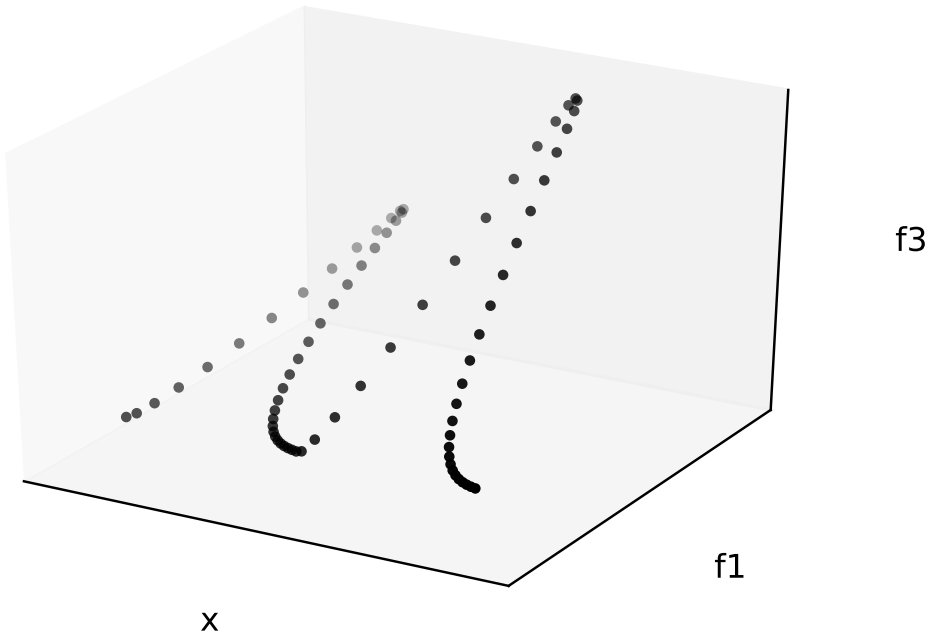


Figure 3.5: 2-dimensional transformation map between $f1$ and $f3$ in Figure 3.1.

The map requires awareness of the configuration parameter x to appropriately capture the variation, therefore we add x, the single feature from $\mathcal{C}$ to the map, increasing its dimensions to two. This is therefore a map of class $(1, 1)$, as it depends on x, hence it uses one feature from $\mathcal{C}$ and one from $\mathcal{M}$.

Examples with higher dimension functions

The same reasoning applies to the benefits of the mapping transformation for both of the following cases: when the functions in question, i.e., $f1$, $f2$, $f3$, are i) models of the evolution of the whole system, or ii) models of the evolution of a certain subset of the system configuration space e.g., the configuration obtained by keeping one system parameter fixed and varying all

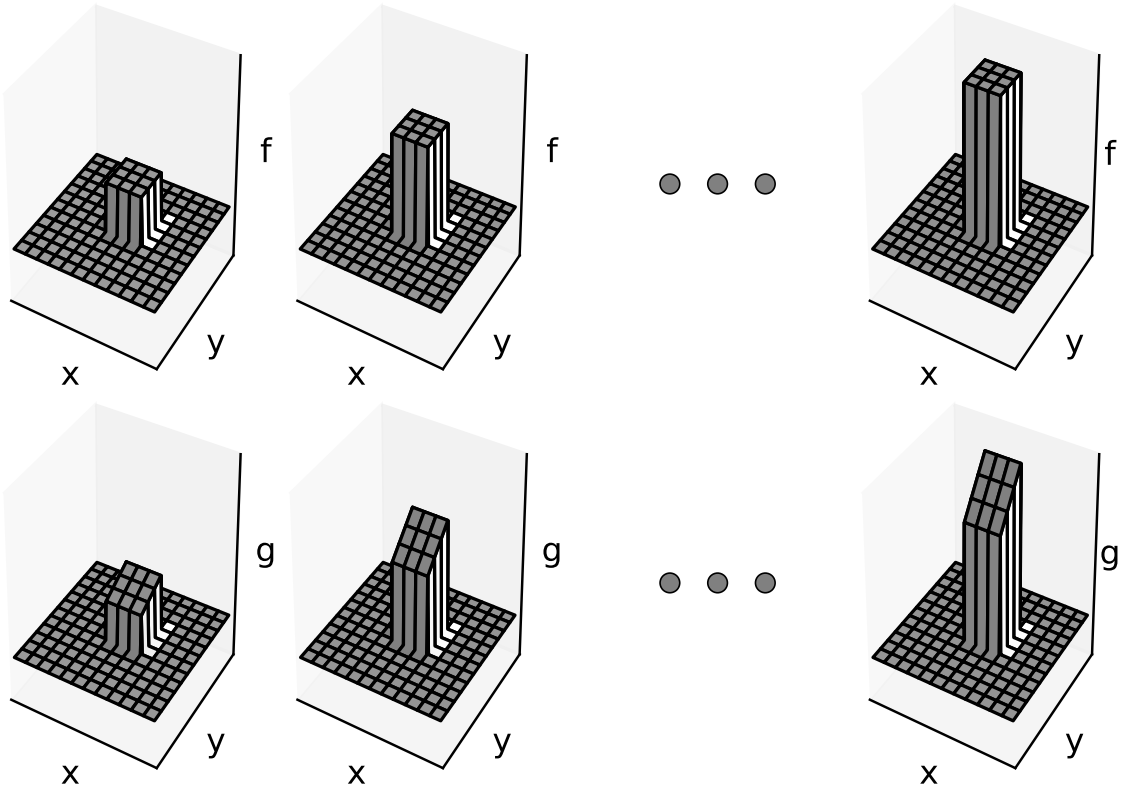


Figure 3.6: Graph of performance functions f and g incrementally modified by a configuration parameter

the others.

Figure 3.6 shows a series of 3D models pertaining to several possible successive evolutions of a system that has two configuration parameters x and y. We can see that, a simple scaling function, similar to the scenario presented in Section 3.1, suffices to represent all the evolutions in the first row. This is also the case for all the evolutions in the second row. Therefore, for building any new 3D model in any of the two sequences represented on each of the two rows, only a simple 1D transformation function suffices that relates the two functions' codomains, and therefore a map of class $(0, 1)$.

However, for any transformation of a model in the first row to the corresponding model in the second row, we see that we need a more complex transformation function. We denote this type of transformation a transformation of class $(2, 1)$ because we need two features (x and y), both pertaining to the function domain, as well as the single codomain feature, to build the new

model.

The choice of an appropriate map dimension and the selection of features can be performed using information contained in the legacy model. While it may be impossible to select the best dimensions and features prior to sampling the system in the unknown configuration, feature selection may be performed using variance-based sensitivity analysis [84] of the legacy model's output, or other related techniques. Ultimately, run-time model ranking and selection by cross-validation of multiple trained models with different combinations of dimensions and features, is a valid option when the analysis of the legacy model does not offer useful information.

An additional consideration is that in principle our mapping technique may also be applied to interpolate or extrapolate across models, even with no new sample points at all. For example, if the first two or three models in one row of Figure 3.6 are available as legacy information, we could derive the next one(s) in the sequence without the need for any sampling. The reason is that we could capture the trend of the transformations the model experiences, as system performance evolves. Using multiple models as features corresponds to using multiple codomain features, and we classify maps of this type as $(\ell >= 0, k > 1)$.

As we will see, our technique is orthogonal to the choice of modeling method used to create the map, as it aims to reuse knowledge from previously available models and model the relation between models.

3.4 ModelMap

Studying and verifying the feasibility and effectiveness of our technique, and comparing it with others, requires the ability to systematically apply it in a variety of controlled and reproducible experiments.

Several open source mathematical modeling frameworks and programming libraries such as *scikit-learn* [85] allow application and comparison of modeling methods. These tools generally focus on providing easy access to a library of standard modeling methods, and an API to extend methods, implement new ones, and test them. While offering powerful features for fitting mathematical models and verifying their accuracy, none of these tools is specific to the investigation of incremental modeling. Our research seeks to understand how incremental changes that occur to a system affect modeling accuracy, in relation to the amount of available information. The goal requires a flexible toolkit, allowing us to run large numbers of experiments programmatically,

We designed and implemented *ModelMap* in support of this research, a software toolkit that combines widely used open-source libraries with dedicated incremental modeling features, and offers a flexible API to:

- Combine legacy performance data with data from unknown systems configurations
- Compare modeling and transfer learning techniques using multiple error metrics
- Conduct experiments on incremental modeling and track error trends

To obtain the required flexibility, fast experimentation, and wide applicability, we implemented the toolkit using the Python 3 programming language [79], which gained wide support both in the data science and in the computer engineering communities. The *ModelMap* toolkit was written to conform to the PEP-8 coding standards [77].

To ensure ease of use and integration with other applications, *ModelMap* leverages scikit-learn, a very popular open-source library for mathematical modeling and data analysis. Scikit-learn is used as the baseline implementation of most basic mathematical modeling techniques,

instances of which are used in *ModelMap* to represent both legacy performance models and transfer maps models. *ModelMap* adheres to the scikit-learn general API principles, and the fundamental interfaces have been retained in our base transfer learning model class, to give users a familiar construct when using it for a variety of modeling tasks. For dataset manipulation and general input/output, our toolkit leverages *pandas* [74], a powerful data analysis library.

The implementation of the *ModelMap* modeling classes includes functionality to simultaneously train models using two different approaches: direct modeling (including incremental modeling and transfer learning) and Model Mapping, and to experimentally compare the techniques.

The toolkit is designed to allow complete reproducibility of the experiments, therefore each class performing operations that involve randomness (e.g. sampling selection) uses its own pre-initialized random number generator.

A repository was created on the Cloud-based code hosting site GitHub [41] to contain all the source code and datasets that will be used for the experiments required by the current and future research directions [67].

We now review the fundamental design principles behind the implementation of the toolkit classes.

ModelMap class

The ModelMap class (*ModelMap.py*) is the fundamental interface to perform model fitting and regression using both direct modeling and Model Mapping at the same time. The class can be initialized to simultaneously use a variety of modeling methods to fit a given dataset and to model a transfer map that leverages legacy data. It fully implements the base scikit-learn interfaces *BaseEstimator* and *RegressorMixin*, and it is therefore ready to be used in any application that already makes use of a different scikit-learn regressor, including model selection.

The class constructor allows specifying all the necessary parameters for initialization and the simultaneous runtime training and evaluation of all the specified mathematical models. The main configuration parameters are as follows:

- *authoritative_models*: a list of one or more instances of pre-trained models compatible with the scikit-learn Regressor interface, to represent the legacy model the system will use as the legacy information available to build the transfer map.
- *map*: an instance of the ModelMapMap class, which is used to represent and train the map model.
- *direct_modeling_methods*: a list of one or more instances of untrained models compatible with the scikit-learn Regressor interface. These are used as the point of comparison for the experiments, and may be used as a fallback mechanism in circumstances where the map model is deemed to perform poorly.

ModelMapMap class

The ModelMapMap class (*ModelMapMap.py*) implements the abstract interface to define the type of function composition for a transfer map. Instances of this class store and manipulate all required data involved in training and querying the transfer map models. The main configuration parameters are as follows:

- *authoritative_models*: see the same parameter in the ModelMap class.
- *map_modeling_methods*: a list of one or more instances of untrained models compatible with the scikit-learn Regressor interface. These instances are used to train and store multiple representations of the transfer map, that the system can track and compare.

ModelMapExperiment class

The ModelMapExperiment class (*ModelMapExperiment.py*) implements the testing framework used to perform experiments in a controlled, repeatable way. All the experiments in

this work are expressed as instances of the ModelMapExperiment class, allowing the appropriate parameters for each experiment to be expressed concisely and effectively. The class uses a collection of pseudo-random number generators to initialize the model instances, in such a way that the experiments are completely reproducible. Reproducibility is guaranteed in all circumstances that require stochastic selection, for example in the random sequence of sampling locations used to determine the training set for the models. The main configuration parameters are as follows:

- *authoritative_dataset_filename*: the filename of the .CSV file containing the dataset representing the legacy model.
- *unknown_dataset_filename*: the filename of the .CSV file containing the dataset representing the ground truth for training and measuring the accuracy of the unknown model.
- *domain_columns, codomain_columns*: labels of the performance function domain and codomain columns in the dataset files.
- *num_runs*: number of repetitions for the experiment, to obtain statistics on the behavior of the models under different random selections of sample points.
- *authoritative_models*: see above.
- *map*: see above.
- *direct_modeling_methods*: see above.
- *sampling_budgets*: an array of values representing multiple sampling budgets (number of randomly picked samples from the dataset for the model training set) the experiment uses to produce a trend of accuracy.

Regressor classes

The Regressor classes implement mathematical modeling methods used by the experiments in addition to the ones present in the scikit-learn framework. These have been designed and

implemented to expose the same interface for training and evaluation as their scikit-learn counterparts. The modeling methods implemented in *ModelMap* are the following:

- Database regressor (*DatabaseRegressor.py*): a non-interpolating model that simply stores the samples passed to it at training time and returns them upon evaluation. This model has been developed for the purpose of analyzing categorical datasets, in which continuous interpolation across the parameter space is meaningless, but instead model comparisons are performed against an exhaustive dataset, where all the combinations of parameters have been sampled.

- Polynomial regressor (*PolynomialRegressor.py*): a model that interpolates the given data using a simple polynomial curve of selectable degree.

- Gaussian Process regressor (*GPyRegressor.py*): a model that uses the Gaussian Process method to fit the dataset. This class uses the GPy library [44] implementation of Gaussian Process Regression and wraps it into a scikit-learn compatible interface.

3.5 Summary

In this chapter we introduced Model Mapping, a transfer learning technique that leverages the characteristics of transient computer systems performance with the goal of improving modeling accuracy in the presence of small sampling budgets. We also presented *ModelMap*, an open source software toolkit that implements Model Mapping for the purpose of experimentation, ease of integration with existing performance modeling applications, and deployment in production systems. In the following chapters we present a set of scenarios, designed to experimentally evaluate the effectiveness of Model Mapping, in relation to other performance modeling approaches.

Chapter 4

Database System Performance Modeling

4.1 Introduction

Our database performance experiments are designed to reflect real-world scenarios, focusing on the typical process for allocating resources to database systems and their applications in a multi-tenant cloud environment. In these environments, user applications are allocated an amount of system resources that balance operating costs against the risk of violating service level agreements, through the following steps:

- **Obtain an accurate model of the application's performance characteristics:** finding the appropriate allocation of resources to a given application requires a model of the application's performance graph. This model is obtained by running the application under a set of system configurations in an instrumented system and measuring the application's behavior. The model represents the application's performance as a function of the resources allocated to it.

- **Search the optimal resource configuration using the obtained model:** when an accurate enough model is obtained or when the time budget for modeling is exhausted, optimization of the system configuration parameters can be performed. The optimization process uses the performance model to predict the application's behavior across the

entire system configuration space.

- **Apply the resource configuration:** based on the results of the optimization process and the service level agreement, resources are allocated to the application using hypervisor control in Virtual Machines or other means of resource throttling.

When a major system configuration change occurs, such as when a new generation of hardware becomes available, we want to cheaply obtain an accurate model of the application's performance characteristics under the new configuration.

In order to test the characteristics and limitations of the Model Mapping technique, we created five scenarios that are representative of a database system undergoing a variety of modifications to its resources. The scenarios evaluate our performance modeling technique by introducing increasingly significant modifications in the system configuration. The five scenarios are as follows:

- Scenario 1: incremental variations of a single resource quota. Increase in CPU resource allocation to improve performance.
- Scenario 2 and 3: different, increasingly significant system hardware downgrades to lower the operating cost.
- Scenario 4: incremental variations of latent factors (data accumulation over time).
- Scenario 5: incremental changes on optimal system resource allocation in a multi-tenant scenario.

4.2 The System Under Test

In order to ascertain the behavior of a representative database system when incrementally varying its resource allocation, we choose TPC-C, a standard database performance benchmark [108], as the application under scrutiny.

The TPC-C benchmark aims to simulate the running conditions of a database system underpinning a business operating in multiple sales districts and serving customers from a number of regional warehouses. Warehouses cover 10 sales districts each, sales districts serve 3000 customers each. This performance benchmark uses a combination of multiple types of database transactions performed by a traditional sales operation. Sales operators use terminals to enter transactions such as orders, payments, and inventory queries, which are relayed to the central database system. The types of transactions TPC-C simulates are the following: New Order, Delivery, Payment, Order Status and Stock Level. The transactions mix used by the benchmark is a combination of both read and write operations. The benchmark measures the transactions per minute (tpmC) executed within a given sampling time.

4.2.1 Data Collection Platform

To obtain a sufficiently complex dataset and properly evaluate the effectiveness of our technique, we set up an instrumented system. We established a complete environment to reproduce the behavior of a standard cloud application leveraging a database system, backed by a separate storage subsystem. The system was designed to reliably perform all the required performance measurements over long periods of time.

In order to obtain statistically reliable data, we executed the TPC-C benchmark multiple times, and instrumentation was used to detect and filter outliers caused by spurious system events, such as spikes in system CPU usage unrelated to the benchmark. The instrumentation consisted in a Unix daemon that logs CPU and memory usage during each execution of the benchmark. To reduce the chance of the instrumentation system affecting the performance behavior, we sampled the overall system usage once per second and stored the small traces ($\sim$1KB per experiment) in a pinned virtual memory page, to avoid incurring in virtual memory-related exception/interrupt delays. We compared the benchmark running with and without instrumentation and found the difference to be statistically insignificant.

We used a two-stage Z-score outlier detection [81] procedure, as follows:

- Stage 1 calculated the Z-score of the TPC-C throughput (tpmC) values of all benchmark runs, then removed the runs with $Z \leq -3$ or $Z \geq 3$.

- Stage 2 used the CPU and memory usage traces, by calculating their averages for each benchmark run, then calculating their Z-scores and removing the runs where $Z \leq -3$ or $Z \geq 3$.

The procedure discarded less than 1% of all data points, with all the outliers being detected by Stage 1. Inspection of the system traces for the outliers detected by Stage 1 showed sudden spikes in CPU usage, which we attributed to unrelated background system activity. Our system ran in a very controlled environment, which resulted in the small number of outliers. In similar, tightly controlled environments, the selection of an appropriate confidence interval can lead to fewer sampling repetitions to obtain similarly statistically reliable data. However, real production systems often operate in more unpredictable conditions and extensive repeated sampling may be necessary.

4.2.2 Hardware and Software Systems

As one of the key goals of our method is to efficiently predict an application's performance behavior in a changing environment, we setup three different hardware platforms, by selecting the components for each platform from different generations of server hardware. For each platform we obtained exhaustive measurements of the application's performance characteristics, which we subsequently processed using the outlier detection procedure described above to obtain the datasets for our modeling experiments.

The detailed hardware and software specifications of the servers we used to generate the data are provided in Table 4.1.

	Platform 1	Platform 2	Platform 3
CPU	2 × Intel Xeon CPU E5-2640 v3 @ 2.60GHz (Haswell) (Hyper-Threading enabled)	2 × Intel Xeon CPU E5-2640 v2 @ 2.00GHz (Ivy Bridge EP) (Hyper-Threading enabled)	2 × Intel Xeon CPU W5580 @ 3.20GHz (Nehalem EP) (Hyper-Threading enabled)
Chipset	Supermicro B10DRT-TP	Supermicro X9DRG-O(T)F	HP 0AECh
BIOS	Version 2.0a	Version 3.2	Version 3.57
Memory	8 × Hynix HMA84GL7AMR4N-TF 32GB DDR4-2133 ECC LRDIMM (total 256GB)	16 × Samsung M386B4G70DM0-YK04 32GB DDR3-1600 ECC LRDIMM (total 512GB)	6 × SAMSUNG M393B5170EH1-CH9 4GB DDR3-1333 ECC Registered (total 24GB)
Hard Disk	Seagate Constellation.2 ST9500620NS, 500GB, 7200 RPM, 64MB Cache, SATA 6.0Gb/s	Samsung SSD 850 Pro, 256GB, 2 GB Low Power DDR3 SDRAM Cache, SATA 6.0Gb/s	2 × Western Digital WD4003FZEX Black, 4TB, 7200 RPM, 128 MB Cache, SATA 6.0Gb/s (RAID 1)
Network Interface	MT27520, 10 Gbps	BCM57840 NetXtreme II, 10 Gbps	NetXtreme BCM5764M, 1 Gbps
Operating System	Base installation of CentOS Linux release 7.3.1611, Kernel 3.10.0-514.16.1.el7.x86_64	Same as Platform 1	Same as Platform 1

Table 4.1: Hardware and OS configuration details.

4.2.3 Virtualization and Performance Isolation Strategy

In order to simulate the effects of configuration and performance factors of cloud-based solutions, we created an isolated, virtualized environment. This choice allowed us to quickly and effectively deploy and reproduce the performance profiling environment on a variety of hardware, and reliably collect all the performance measurements necessary for our dataset.

The Oracle VirtualBox virtualization platform (version 5.1.22) was used. All the virtual machines were run on the same hardware server, with a bridged virtual network connecting them.

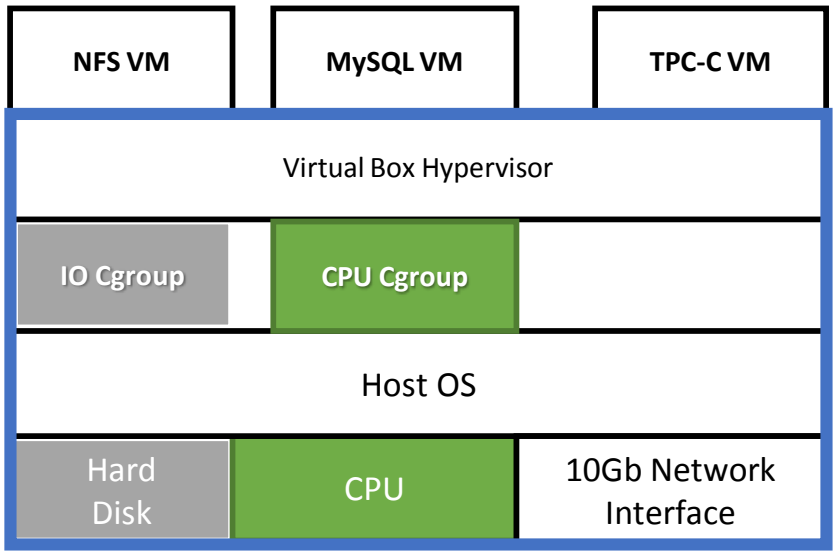


Figure 4.1: Data collection platform setup.

Three Virtual Machines were created to represent the main system components:

- TPC-C application
- Database server
- Storage system (hosting the database filesystem on a NFS server)

Figure 4.1 depicts the overall system layout.

This setup allows us to easily isolate and limit the resource usage of each service/system component, with levels of control that are similar to what a cloud environment provides. At the same time, the setup allows ease of management and fast turnaround time for the numerous configuration changes our data collection effort requires.

All VMs have been configured with the same virtual hardware configuration, as follows:

- **CPU**: 1 virtual core.
- **Memory**: 2048 MB
- **Network**: bridged network, i.e. direct access to the host's primary network interface

The specific software configuration of each Virtual Machine is as follows:

- **Application VM**
  - **OS**: CentOS Linux release 7.3.1611, kernel 3.10.0-514.21.1.el7.x86_64
  - **TPC-C application**: We modified a TPC-C implementation for MySQL [108] to output additional statistics, formatted for easier capture and further processing, and compiled it with gcc 4.8.5 20150623 (Red Hat 4.8.5-11), with the default compiler flags (-w -O3 -g).
- **Database VM**
  - **OS**: CentOS Linux release 7.3.1611, kernel 3.10.0-514.21.1.el7.x86_64
  - **Database**: MariaDB [62][1] 10.2.6-1.el7.centos with InnoDB: 5.7.14. According to MariaDB technical documentation [63], we do not expect to see any substantial difference between MariaDB 10.2, which we used in this work, and MySQL 5.7, required by our selected TPC-C implementation.
- **Storage VM**
  - **OS**: Ubuntu 14.04.5 LTS, kernel: 4.4.0-31 i686
  - **NFS server**: nfs-kernel-server and nfs-common packages, version 1:1.2.8-6ubuntu1.2 i386

4.2.4 Data Collection

The TPC-C benchmark tests the response of modern database software to mixed application requests when running in a cloud environment under a variety of configurations; we therefore measure and model the variation in the database response performance under varying database configuration parameters and system-level parameters.

[1]MariaDB is a community-developed fork of the MySQL RDBMS intended to remain free under the GNU GPL.

For data collection we initially configured the TPC-C benchmark on each of the three platforms to populate and exercise a database representing 10 warehouses, for a total database size on disk of approximately 1.2GB. Before starting the performance measurement activity, TPC-C initializes the database by creating the appropriate tables and populating them with an initial dataset. This operation takes a considerable amount of time. During the benchmark operations the database contents are modified by sequences of write operations, therefore the initialization procedure is required before each performance measurement to obtain consistent profiling data. In order to reduce the time taken by profiling the system in each configuration, we used the TPC-C application to pre-load the database once, then stopped the database, backed up the contents of its supporting filesystem and reused a clone of the obtained initial database state for all the subsequent experiments.

For each database configuration change and system configuration change, we allowed the database cache to warm up for 15 seconds. We then run the benchmark for 3 minutes with 10 simultaneous connections to the database, then gracefully shut down the database, change the configuration, replace database files with the initial database, and restart the database server. The above process to obtain a single configuration sample takes approximately 5 minutes.

The data collection involved executing the TPC-C benchmark while exhaustively varying three system configuration parameters across the identified domain. These parameters represent the quota of three shared system resources allocated to the database system: I/O, memory, and CPU. Details about the selected system resources and the choice of tools for controlling their allocation are reported below.

- **Disk I/O quota**: maximum allowed system I/O throughput for the storage system applied by assigning the Storage VM process to a dedicated cgroup and controlling the cgroup's I/O allocation by varying the configuration parameters `blkio.throttle.write_bps_device` and `blkio.throttle.read_bps_device`.

- **Buffer pool size**: size configuration of the MariaDB innodb database buffer pool ap-

plied by varying the `innodb_buffer_pool_size` system environment variable in the Database VM.

- **CPU quota**: maximum allowed percentage of the system's CPU allocation for the database applied by assigning the Database VM process to a dedicated cgroup and controlling the cgroup's CPU quota allocation [22] configuration parameter:
  `cpu.cfs_quota_us`=1000000
  and varying the configuration parameter:
  `cpu.cfs_period_us`.

We wrote a set of shell scripts to alter the three chosen system configuration parameters as follows:

- **Disk I/O quota**:
  `write_bps_device` = `read_bps_device` = {1, 2, 4, 8, 16, 32, 48} (MBps).
- **Buffer pool size**:
  `innodb_buffer_pool_size` = {1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024} (MB).
- **CPU quota**:
  `cfs_quota_us`=1000000
  `cfs_period_us`={100000 (10%), 250000 (25%), 500000 (50%), 1000000 (100%)}.

The total size of the system configurations space is therefore 7 x 11 x 4 = 308 different configurations.

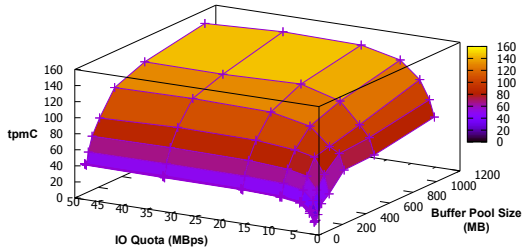
For the modeling experiments, we modified the TPC-C benchmark code to capture data directly in the form of the overall transactions per minute metric (tpmC[2]) rather than parsing the default output. We ran the benchmark 30 times for each combination of the system configuration parameters values to collect statistically reliable information and stored all the aggregated

[2]In TPC-C, throughput is defined as number of New-Order transactions per minute while the system is executing four other transactions types (Payment, Order-Status, Delivery, Stock-Level) [107].

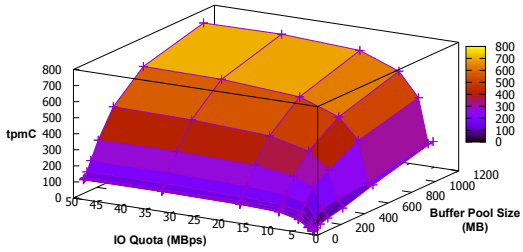
results to disk. In all the modeling experiments, for each configuration, we used the average value of the tpmC throughput as the sample of the performance function. The obtained dataset for each platform is four-dimensional: three varying domain parameters and a single scalar codomain (Table A.1, Table A.2, and Table A.3 for Platforms 1, 2, and 3, respectively).

We then proceeded to repeat the data collection process on Platform 1, configuring the TPC-C benchmark to populate and exercise a database representing, respectively, 20, 40 and 64 warehouses, for a total database size on disk of approximately 2.4GB to 6.4GB. The dataset is itself an independent contribution of this dissertation and is available for further research [109]. The total time required to obtain the dataset was approximately 4600 CPU-hours.

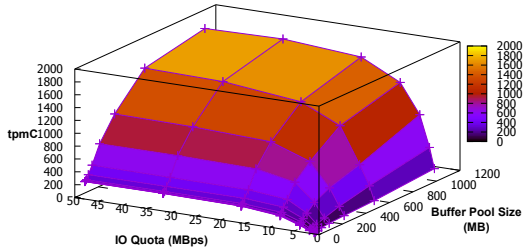
Visual examples of the gathered data are depicted in Figure 4.2 and Figure 4.3. Throughout this chapter, the scale of each graph has been selected to highlight the differences in behavior as the system parameters are varied, independent of scale. Versions of all the graphs with consistent scale can be found in Appendix A.



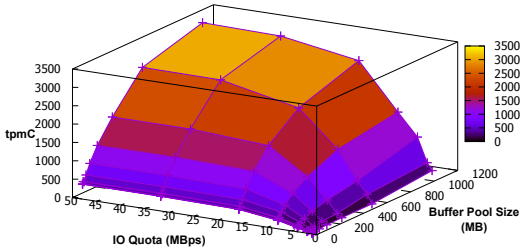
(a) **CPU quota**=10%



(b) **CPU quota**=25%

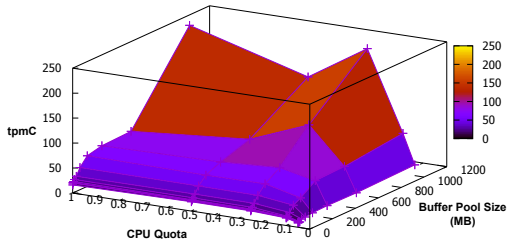


(c) **CPU quota**=50%

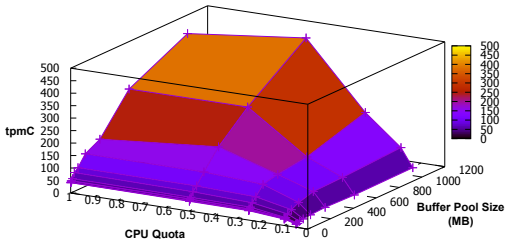


(d) **CPU quota**=100%

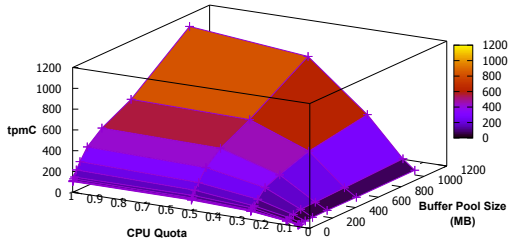
Figure 4.2: Graphs of TPC-C transactions per minute (tpmC) on **Platform 1** when the **IO quota** varies between 1 and 48 MBps and the **Buffer pool** varies between 1 and 1024 MB.



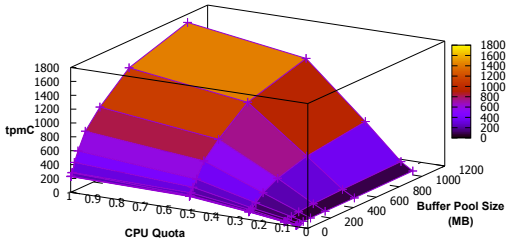
(a) **IO quota** = 1MBps



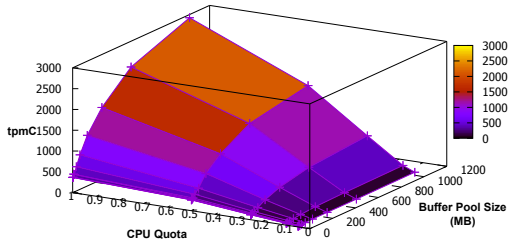
(b) **IO quota** = 2MBps



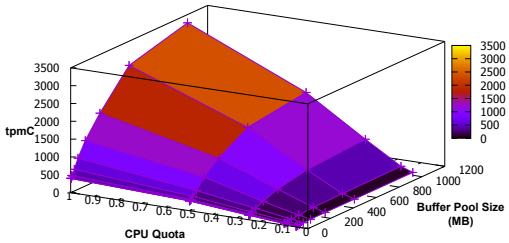
(c) **IO quota** = 4MBps



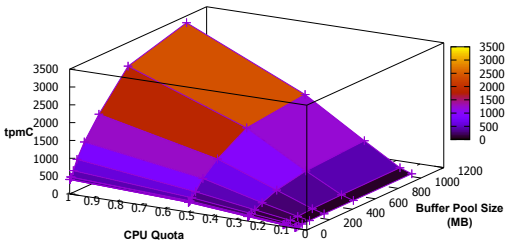
(d) **IO quota** = 8MBps



(e) **IO quota** = 16MBps



(f) **IO quota** = 32MBps



(g) **IO quota** = 48MBps

Figure 4.3: Graphs of Transactions per minute (tpmC) on **Platform 1**, when the **CPU quota** varies between 10% and 100% and the **Buffer pool** varies between 1 and 1024 MB.

4.3 Techniques Used as Baselines for Comparison

For all the experiments, we evaluate the effectiveness of our Model Mapping technique against a selection of direct and incremental modeling methods, which are often encountered in literature.

4.3.1 Direct Modeling Methods

Four methods have been selected for direct modeling, and the same four are also used for Model Mapping to represent the transfer functions.

Linear Regression

Linear Regression [13] is a method that fits a linear equation to sampled data in order to obtain a model of the relation between two variables. We used the scikit.learn [85] toolkit implementation.

Polynomial Regression

Polynomial Regression [13] is a method that fits a polynomial curve of degree n to sampled data, to model a function. We selected a polynomial of degree 4 after reviewing the characteristics of the data being modeled. As for the Linear Regression model, we used the scikit.learn toolkit implementation.

Support Vector Regression with Linear Kernel

Support Vector Regression with linear kernel (Linear SVR) [13, 124] is a method for classification or regression based on the construction of a set of separating hyper-planes embedded in high dimension Hilbert spaces. Linear SVR has been successfully applied to model a wide variety of phenomena [89, 90]. Scikit.learn was used for its Linear SVR implementation as well.

Gaussian Process Regression

Gaussian Process Regression (GPR) [13, 125] is a method that leverages the statistical properties of Gaussian Processes for the purpose of regression and classification. GPR models have been found to be very effective at modeling high-dimensional phenomena [125]. The selection of the covariance kernel was performed using combinatorial search, and Matern52 [126] emerged as the most effective across all our experiments. We used the implementation of Gaussian Process modeling in the GPy [44] framework.

GPR is a kernel method which leverages the Bayesian treatment of uncertainty. The method iteratively refines a model of a function by sequentially selecting the sampling locations that maximize expected improvement of the model's accuracy. This approach allows for faster convergence with small sampling budgets. However, in our experiments we cannot assume the user has the ability to systematically choose the sampling locations. Therefore, all modeling methods are provided samples selected using a random distribution in the configuration space.

4.3.2 Incremental Modeling Methods

The choice of transfer learning and incremental modeling methods is as follows.

Chorus Incremental Regression

Chen et al. introduced Chorus [24, 25], a state-of-the-art incremental performance modeling framework. Chorus was designed for performance evaluations and resource allocation in data center application deployments, and features model storage, retrieval, and reuse. A detailed description and analysis of Chorus are provided in Section 7.4.1. We use Chorus as a baseline incremental performance model, as it was reported to show substantial savings in training time compared to traditional modeling approaches. In particular, Chorus offers facilities for incremental training of existing models with new samples, similar to the focus of our work. However, Chorus does not have a facility for Model Mapping.

Multi-Task Gaussian Process Prediction

Bonilla et al. introduced Multi-Task Gaussian Process Prediction (MTGP) [14], a transfer learning technique that leverages a presumed correlation among a group of Gaussian Process models being trained simultaneously for the purpose of achieving a higher modeling accuracy compared to training the models individually.

In their work, the authors showed very strong results in two key scenarios:

- Simultaneously training multiple positively or inversely correlated models.
- Training an initial model for a task, and subsequently training a new model for a correlated task with a few samples.

The excellent knowledge transfer characteristics displayed by MTGP in the second scenario motivated our choice to use it as a representative of transfer learning models. Jamshidi et al. [50] also applied this method successfully for performance modeling of software applications.

The covariance kernel we selected was Matern52 [126], using combinatorial search. Similar to Gaussian Process Regression modeling, we used the GPy [44] framework implementation of Multi-Task Gaussian Process modeling.

The goal of our work is not to recommend the most appropriate learning method for specific scenarios, but rather to demonstrate that for many incremental systems applications, a map is likely to be better when applied to model the transformation between two related models. It should not be assumed that using transfer learning is always advantageous, and procedures that are commonly used to perform model selection (e.g. cross-validation) are also valid when using Model Mapping, and should be used to choose the most appropriate modeling approach.

Information Availability and Hyperparameter Tuning

Model hyperparameters, where applicable, have been determined based only on the information presumably available to a system administrator before performing the training of the mod-

els. For direct modeling methods that have hyperparameters, their values were selected under the assumption that the system administrator has available legacy data that can be used to perform model selection. Since the administrator has no information about the effects that variations to the system configuration have on the system performance before it is observed, we could not perform any hyperparameter tuning for the models used to represent the maps. We therefore chose to retain the same hyperparameter values selected for the direct models.

From this perspective, direct modeling and Chorus have an advantage over Model Mapping. In some scenarios, model selection and hyperparameter tuning may be performed on the legacy data and reused effectively for direct modeling, whereas Model Mapping requires performing these two procedures at run-time, as samples from the new configuration become available. In all our experiments, we have not performed any hyperparameter tuning of the map models.

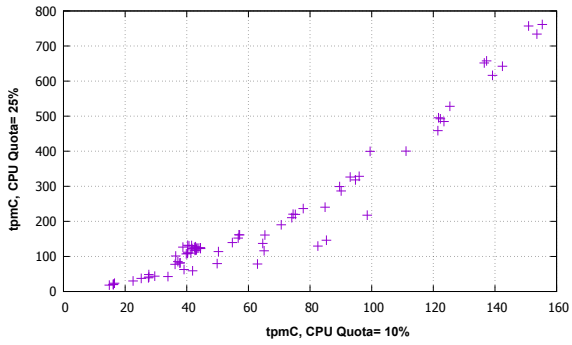
4.4 Scenario 1: Predicting Performance with Increased CPU Resources

In this scenario, a system administrator is interested in improving a virtualized application's performance by increasing its CPU resources. The administrator has already obtained a model of the application's performance under variations of other resource quotas, but has no information on the effects of varying the CPU quota allocation.

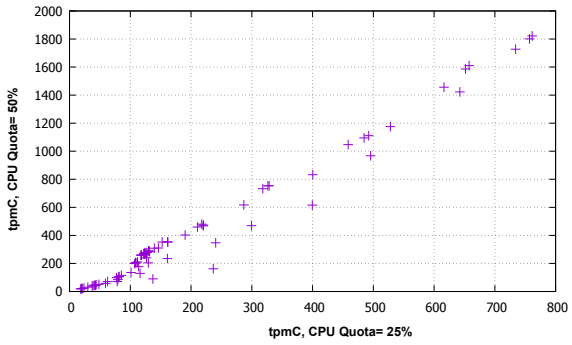
This example shows how, with few new samples, a performance model of the effects that I/O bandwidth and buffer pool have on the application's performance can be extended to a new model that includes CPU resource quota as a new configuration parameter.

Figure 4.2 shows the throughput of the TPC-C benchmark [106] running within a virtualized environment described above, given various quotas of two resources: I/O bandwidth and buffer pool size. In the virtualized environment, these resource quotas correspond to the configuration parameters of the virtual machines (VMs) within which the application runs.

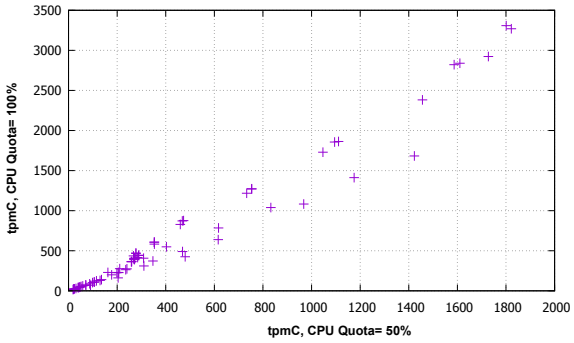
Figure 4.4 represents three maps between pairs of CPU resource quota configurations.



(a) Map of **CPU quota** 10% to 25%



(b) Map of **CPU quota** 25% to 50%



(c) Map of **CPU quota** 50% to 100%

Figure 4.4: **Scenario 1:** Graph of maps of TPC-C throughput (tpmC) with varying **CPU quota** on **Platform 1**.

4.4.1 Formulation

As reported in Section 4.2.4, our TPC-C dataset consists of measurements of the TPC-C benchmark throughput (in queries per minute), by setting the system configuration parameters to all combinations of the values in Table 4.2.

Configuration parameter	**Parameter values**
Disk I/O quota	1, 2, 4, 8, 16, 32, 48 (MBps)
Buffer pool size	1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 (MB)
CPU quota	100000 (10%), 250000 (25%), 500000 (50%), 1000000 (100%)

Table 4.2: TPC-C configuration parameters settings.

The legacy model ρ^0 in this scenario was built using a subset of our TPC-C dataset. The configuration space $\mathcal{C}^0$ is a vector of three-dimensional tuples containing all combinations of the parameters **Disk I/O quota** and **Buffer pool size**, with the fixed third parameter **CPU quota**

= $\{25\%\}$, sampled from Platform 1.

$$\mathcal{C}^0 \in \mathbb{R}^3 = \begin{cases} \textbf{Disk I/O quota } \{1, 2, 4, 8, 16, 32, 48\}\text{(MBps)} \\ \textbf{Buffer pool size } \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024\}\text{(MB)} \\ \textbf{\textit{CPU quota}} \ \{25\%\} \end{cases}$$

We obtain from our dataset a vector $\mathcal{M}^0$ in which each entry contains the result of measuring the TPC-C benchmark throughput using the system configuration parameters in the corresponding entry of $\mathcal{C}^0$.

$$\mathcal{M}^0 \in \mathbb{R} = \text{TPC-C throughput (transactions per minute)}$$

$$Legacy\ model\ \rho^0 : \mathcal{C}^0 \rightarrow \mathcal{M}^0$$

The new model ρ^1 we intend to obtain includes one additional dimension in the configuration space, to cover a new value for the **CPU quota** system parameter. It is assumed that the configuration parameter being evaluated was not subject to variations at the time the legacy model was created.

$$\mathcal{C}^1 \in \mathbb{R}^3 = \begin{cases} \textbf{Disk I/O quota } \{1, 2, 4, 8, 16, 32, 48\}\text{(MBps)} \\ \textbf{Buffer pool size } \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024\}\text{(MB)} \\ \textbf{\textit{CPU quota}} \ \{50\%\} \end{cases}$$

$$Performance\ space\ \mathcal{M}^1 \in \mathbb{R} = \text{TPC-C throughput (transactions per minute)}$$

$$Unknown\ model\ \rho^1 : \mathcal{C}^1 \rightarrow \mathcal{M}^1$$

Recall from Section 3.1 that in order to use our Model Mapping technique, we need to build a model of the map σ such that:

$$\rho^1 = \sigma \circ \rho^0$$

In Section 3.3 we classified maps according to the choice of features selected from the performance function domain and codomain, in the form (ℓ, k). For this scenario, we used the simplest form of map, which includes a single feature from the performance function's codomain:

$$\mathcal{M}^0 = \text{TPC-C transactions per minute (tpmC)}$$

$$\mathcal{K} = \{\mathcal{M}^0\}$$

therefore a map of class $(0, |\mathcal{K}|) = (0, 1)$.

In order to obtain σ, we train a model of the one-dimensional transfer function σ of class $(0, 1)$:

$$\sigma : \mathcal{M}^0 \to \mathcal{M}^1$$

$$\sigma : \rho^0(\mathcal{C}^0) \to \rho^1(\mathcal{C}^1)$$

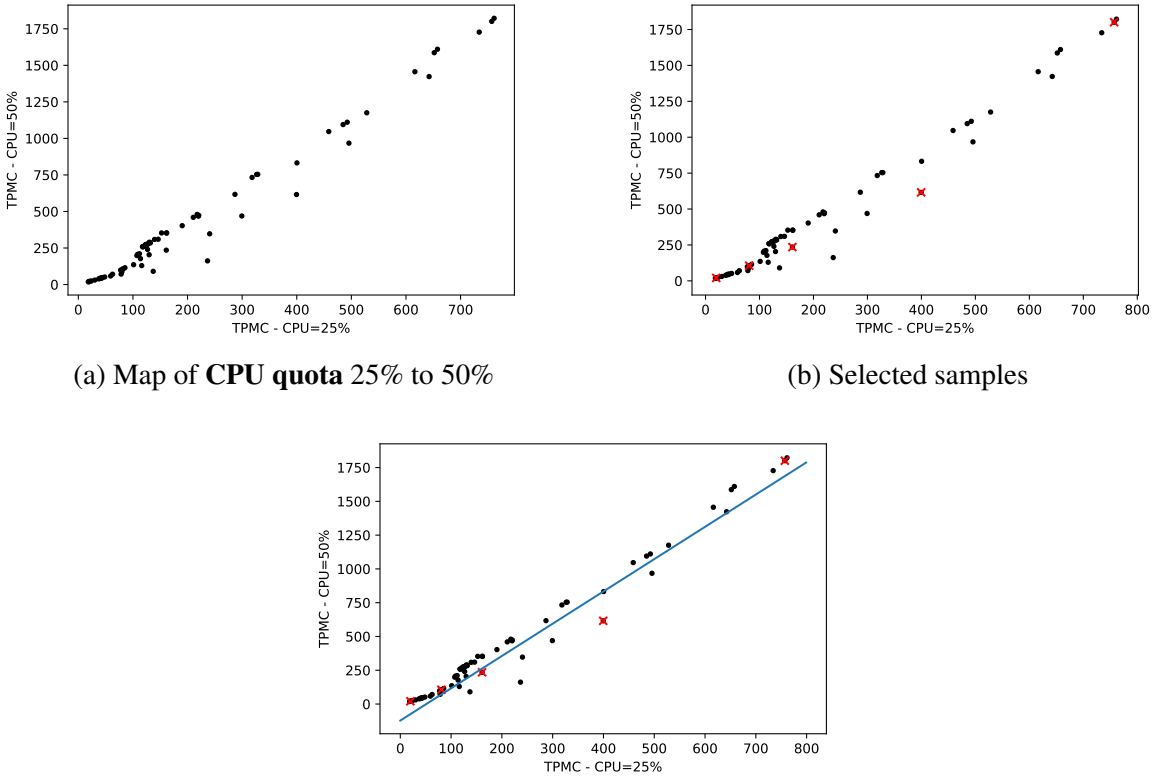
$$s.t.\ \rho^1(\mathbf{x}) = \sigma(\rho^0(\mathit{proj}_{\mathcal{C}^0}\mathbf{x})) \tag{4.1}$$

As an example, consider training a linear model to represent the map. Figure 4.5a shows the plot of ρ^0 against ρ^1. In order to train our linear model, we assume we have a few samples of $\mathcal{M}^1$ that we obtained by measuring the system running with a set of configurations selected from the configuration space $\mathcal{C}^1$, represented by Figure 4.5b. We then sample the legacy system performance by interrogating ρ^0 using the same set of configurations projected into $\mathcal{C}^0$, obtaining a set of tuples,

$$(x, y) \quad x \in \mathcal{M}^0, y \in \mathcal{M}^1,$$

and use it to train a model of the map $\sigma : \mathcal{M}^0 \to \mathcal{M}^1$ using a learning method, such as linear regression, as depicted by Figure 4.5c. We can then interrogate our new model for ρ^1 at any

location $\mathbf{x} \in \mathcal{C}^1$ by using Equation 4.1.



(a) Map of **CPU quota** 25% to 50%

(b) Selected samples

(c) Linear model of the map

Figure 4.5: Fitting a linear model to represent the Scenario 1 map. Selected samples are marked with red x's

4.4.2 Experimental Procedure

Our experiments have the goal of providing a comparison between multiple modeling techniques, when applied to a number of incremental system variations. In particular, we want to ascertain the accuracy of our Model Mapping technique in relation to other direct, incremental and transfer learning approaches. Cross-validation is a commonly used technique to evaluate, compare and contrast the accuracy of selected modeling techniques and their predictive capabilities. To satisfy our incremental modeling scenarios, where only a fixed set of observations (sampling budget) is available to the models, we adopt Monte Carlo cross-validation (MCCV) [78][87][128]. Molinaro [68] and Simon [88] suggest that MCCV is a reasonable

compromise between the bias introduced by simple techniques such as the holdout method [65], the lack of explicit training and test set size selection of v-fold cross-validation [39], and computational expense of exhaustive methods such as leave-one-out cross-validation [56].

4.4.3 Discussion of Error Measurements

In order to measure the accuracy of an individual model, we selected two estimators that are compromises between robustness and popularity in related literature.

We use Mean Absolute Percentage Error (MAPE) and Root Mean Square Error (RMSE):

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{A_i - F_i}{A_i}\right| \tag{4.2}$$

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n}(A_i - F_i)^2}{n}} \tag{4.3}$$

where, A_i is the observed value of sample i and F_i is the predicted value obtained from the model.

We selected MAPE as our primary choice of estimator, as while being biased, it is not overly sensitive to outliers, it provides an absolute estimation, and it is commonly used in literature [61]. For the purpose of comparing performance models, the bias in MAPE towards putting a larger penalty on negative errors leads to conservative estimations, which is preferrable in forecasting performance for systems subject to adhering to Service Level Agreement constraints.

Our second choice of estimators (RMSE) has been shown to weigh outliers heavily [11, 32], which is a sensitive issue in performance modeling. However, RMSE is such a widely used metric for model selection in literature, we elected to use it as it allows us to more directly compare our results to other published work.

Pseudo-code for the full modeling and accuracy comparison procedure implemented by the *ModelMap* toolkit is presented in Algorithm 1, which includes data preparation followed by

multiple iterations of training set and test set creation, model training and evaluation, according to the MCCV cross-validation technique.

Algorithm 1: Pseudo-code for experimental procedure using MCCV

```
Accept ground truth data for the legacy and the unknown functions being modeled;
Perform normalization of all functions' domain and codomain data;
Construct the legacy mathematical model from data;
i ← numIterations;
while i ≥ 0 do
    randomSeed ← i;
    Create training set by performing random selection of samples in the unknown
     function ground truth data set, using the sampling budget;
    Create test set by using all the remaining samples available in the unknown
     function ground truth data set;
    Train an instance of the ModelMap class using the training set;
    Perform model inference on the test set using the ModelMap class and compare the
     accuracy of the different modeling approaches with the ground truth data of the
     unknown function;
    error⃗ ← iterationError;
    i ← i − 1;
error ← mean(error⃗);
```

4.4.4 Results

Table 4.3 and Table 4.4 show the results of our experiments using six different learning approaches. With only five samples (1.5% of total 308 available samples), the best results provided by a direct model that does not exploit transfer learning or incremental modeling (Gaussian Process, Linear SVR, Linear Regression and Polynomial regression) are substantially worse than our Model Mapping technique using Linear Regression. In particular, we observe that the highest relative benefit of Model Mapping is realized when the sampling budget is small. Table 4.3 also shows that our technique outperforms other incremental modeling and transfer learning techniques (Multi-Task Gaussian Process and Chorus).

In particular, we can observe how the other incremental modeling techniques require 10 samples to reach the same level of accuracy afforded by Model Mapping with 5 samples. In

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	453.41±43.9	93.77±6.4
	Polynomial regression	351.60±17.6	153.57±21.3
	Linear SVR	385.55±10.7	87.34±4.5
	Gaussian Process	327.16±18.2	**80.37±2.0**
	Chorus*	86.95±0.7	N/A
	Multi-Task Gaussian Process*	89.69±5.4	N/A
10	Linear regression	336.60±14.6	86.85±4.2
	Polynomial regression	266.87±17.4	151.06±24.4
	Linear SVR	381.99±13.6	82.93±2.2
	Gaussian Process	181.91±19.3	**79.19±1.7**
	Chorus*	84.28±1.4	N/A
	Multi-Task Gaussian Process*	80.96±5.8	N/A

Table 4.3: RMSE mean and standard error of the proposed Model Mapping technique and direct modeling technique with various modeling methods for **CPU quota** configurations (* indicates a direct modeling method that uses incremental or transfer learning). Legacy model **CPU quota**=25% and unknown model **CPU quota**=50%.

real-world scenarios, a seemingly negligible reduction of 5 samples can have substantial implications. Recall how, in this scenario, obtaining each performance measurement requires 5 minutes and in order to retrieve statistically significant samples, 30 runs are performed. Retrieving 5 samples therefore requires 750 minutes (12.5 hours). The difference between 5 and 10 samples is then between letting the method run overnight and acting on the results in the morning, instead of running the system for more than a full day before taking any action.

Examining the dataset, we can see that the reason behind these results is that the relationship between the two configurations' codomains is approximately linear, as highlighted in Figure 4.4b. While we predicted that a linear model would approximate the map most efficiently, we also show the results of learning the map with all of the other types of models we used for comparison.

Recall that when building our TPC-C dataset, we sampled four configurations of the CPU quota allocation: 10%, 25%, 50% and 100%. While in this scenario we reported the results of modeling the effects of changing the **CPU quota** parameter from 25% to 50%, we have also

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	289.55±44.6	40.54±3.1
	Polynomial regression	206.96±20.9	34.55±1.8
	Linear SVR	188.59±22.4	35.36±1.9
	Gaussian Process	133.10±14.0	**33.68±1.2**
	Chorus*	37.41±0.6	N/A
	Multi-Task Gaussian Process*	40.73±5.2	N/A
10	Linear regression	202.51±24.9	34.95±1.9
	Polynomial regression	153.54±19.6	31.36±1.3
	Linear SVR	182.16±24.5	31.69±1.3
	Gaussian Process	50.08±4.5	30.58±1.6
	Chorus*	36.20±0.9	N/A
	Multi-Task Gaussian Process*	**29.87±2.2**	N/A

Table 4.4: MAPE mean and standard error of the proposed Model Mapping technique and direct modeling technique with various modeling methods for **CPU quota** configurations (* indicates a direct modeling method that uses incremental or transfer learning). Legacy model **CPU quota**=25% and unknown model **CPU quota**=50%.

performed additional experiments that represent a subset of all permutations of variations for this parameter: 50% to 25%, 10% to 25%, 25% to 10%, 50% to 100% and 100% to 50%. In all experiments the results showed a similar level of effectiveness for Model Mapping compared to direct modeling. Extending an existing model with a new dimension to represent the CPU resource quota is thus significantly aided by our Model Mapping technique. The results of all these experiments are presented in Appendix B.

4.4.5 Repeated use of Model Mapping

We wanted to further investigate the effects of repeatedly using Model Mapping to obtain multiple consecutive models, starting from a single legacy model/dataset. We therefore performed a variation of the reported scenario in this section. In particular, we are interested in obtaining a model of the TPC-C performance with **CPU quota**=50% by performing two consecutive mapping steps:

1. Use Model Mapping to obtain a model of **CPU quota**=25% from a legacy model of **CPU**

quota=10%, using 5 samples.

2. Use the obtained model as the legacy model, apply Model Mapping again to obtain a model of **CPU quota**=50% using 5 samples.

The goal of this experiment is to observe the effects of error accumulation in applying our technique multiple times against the advantage of having information from intermediate system states. We compare multiple strategies to obtain a model of **CPU quota**=50% with a total budget of 10 samples, as follows.

a) **Map 10→25→50**: the strategy illustrated above consisting of two Model Mapping steps, using 5 samples from **CPU quota**=25% to build the first map, and 5 samples from **CPU quota**=50% to build the second map.

b) **Map 10→50**: Model Mapping with legacy model **CPU quota**=10%, using 10 samples from **CPU quota**=50% to build the map.

c) **Map 25→50**: Model Mapping with legacy model **CPU quota**=25%, using 10 samples from **CPU quota**=50% to build the map.

d) **Direct 50**: direct modeling of **CPU quota**=50%, using 10 samples from **CPU quota**=50% to train the model.

We expect the error resulting from adopting strategy (c) to be lower than the one of (b), given the incremental configuration change is less significant in the former. We also expect strategy (c) to outperform (a), as the approximation errors of **CPU quota**=25% given by our technique compound the fewer samples available from the unknown system configuration. Lastly, strategy (a) may outperform (b) when the compounding of approximation errors does not prevail against the partial information available from the more similar intermediate configuration.

Table 4.5 shows the results of the experiment, with the values in parentheses representing the measured error for the first of the two mapping steps in the *Map 10→25→50* strategy.

The results confirm that mapping between similar system configurations is consistently advantageous, as all modeling methods exhibit a lower error when representing the transformation from **CPU quota**=25% to **CPU quota**=50% (*Map 25→50* in the table), rather than **CPU quota**=10% to **CPU quota**=50% (*Map 10→50* in the table). As expected, we also observe that the additional information available to strategy (c) (*Map 25→50*) and the absence of error accumulation yields better results than the two-step mapping strategy (a) (*Map 10→25→50*).

The results also show that the two-step mapping strategy (a) (*Map 10→25→50*) performs similarly to strategy (b) (*Map 10→50*). While the total sampling budget for all strategies is equal (10), the two-step strategy uses only five samples from the unknown system configuration, while the other five are related to an intermediate configuration. In circumstances when partial information about intermediate system configurations is available, applying Model Mapping multiple times may therefore further reduce the latency of modeling unknown configurations.

Additionally, the results show that the two-step mapping strategy (a) is a better choice than direct modeling (d) in nearly all conditions, and that the error difference between them is often considerable.

These observations lead us to believe that Model Mapping has the ability to compose, by performing multiple mapping steps to build a model through successive transformations of a legacy model. We expect the benefit to be less significant in systems that exhibit behaviors that affect localized areas of the configuration space, rather than global performance variations. Error accumulation from the composition of transformations may also limit the number of mapping steps that can be performed starting from a single legacy dataset, before it outweighs the benefits of capturing partial information. In these circumstances, it is preferable to collapse and/or discard some or all mapping steps.

Modeling Method	Strategy	Error RMSE	Error MAPE
Linear Regression	Map 10→25→50	(66.99±3.15) 207.87±2.56	(40.53±2.41) 90.26±1.45
	Map 10→50	207.44±8.70	103.63±6.71
	Map 25→50	86.85±4.15	34.95±1.91
	Direct 50	336.60±14.57	202.51±24.91
Polynomial Regression	Map 10→25→50	(60.98±10.88) 187.87±2.17	(26.14±1.37) 68.48±0.77
	Map 10→50	196.20±26.72	63.70±3.95
	Map 25→50	151.06±24.39	31.36±1.26
	Direct 50	266.87±17.44	153.54±19.60
Linear SVR	Map 10→25→50	(65.95±1.64) 200.78±1.92	(37.66±2.59) 81.68±1.10
	Map 10→50	205.32±5.15	96.55±5.86
	Map 25→50	82.93±2.18	31.69±1.26
	Direct 50	381.99±13.62	182.16±24.52
Gaussian Process	Map 10→25→50	(44.75±1.78) 169.27±1.86	(29.36±1.43) 72.29±0.85
	Map 10→50	142.00±4.33	80.40±4.80
	Map 25→50	79.19±1.74	30.58±1.61
	Direct 50	181.91±19.27	50.08±4.49

Table 4.5: Comparison of different modeling strategies to obtain the unknown model **CPU quota=**50%. Values in parentheses represent the measured error for the first of the two mapping steps in the *Map 10→25→50* strategy.

4.5 Scenario 2: Modeling a System Downgrade

In this scenario, a system administrator is interested in the feasibility of reducing the cost of operating a database service by downgrading the system specifications to use older, less powerful hardware, while maintaining a determined level of performance. The administrator has already obtained a model of the database's performance under numerous variations of all available resource quotas in the system, but needs to know how to adjust the quotas to compensate for the hardware downgrade, such that the same level of performance can be achieved.

The scenario explains the procedure of incrementally modeling our application's performance when a whole-system architectural modification is performed, moving onto an entirely new hardware platform.

4.5.1 Formulation

The legacy model ρ^0 in this scenario was built from a subset of our TPC-C dataset, using all measurements performed on Platform 1 (Figure 4.2), which represents a recent, high-performance system. The configuration space $\mathcal{C}^0$ is a vector of four-dimensional tuples containing all combinations of the three system configuration parameters **Disk I/O quota**, **Buffer pool size** and **CPU quota**, with the fixed fourth parameter **Platform** = {Platform 1}.

$$\mathcal{C}^0 \in \mathbb{R}^4 = \begin{cases} \textbf{Disk I/O quota } \{1, 2, 4, 8, 16, 32, 48\}(\text{MBps}) \\ \textbf{Buffer pool size } \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024\}(\text{MB}) \\ \textbf{CPU quota } \{10\%, 25\%, 50\%, 100\%\} \\ \textbf{\textit{Platform}} \{\textit{Platform 1}\} \end{cases}$$

We obtain from our dataset a vector $\mathcal{M}^0$ in which each entry contains the result of measuring the TPC-C benchmark throughput using the system configuration parameters in the corresponding entry of $\mathcal{C}^0$.

$$\mathcal{M}^0 \in \mathbb{R} = \text{TPC-C throughput (transactions per minute)}$$

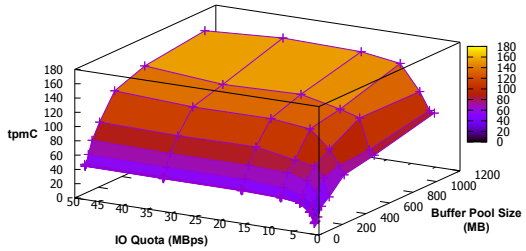
$$Legacy\ model\ \rho^0 : \mathcal{C}^0 \rightarrow \mathcal{M}^0$$

The new model ρ^1 we intend to obtain includes one additional dimension to cover the change between hardware platforms from Platform 1 to Platform 3 (Figure 4.6), which represents an older, lower cost system. Platform 3 is a new configuration for the **Platform** system parameter.

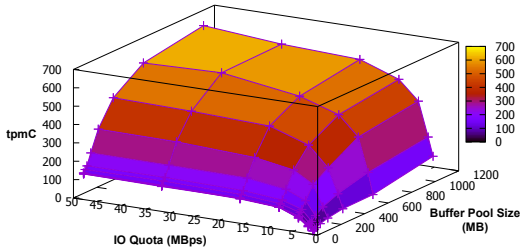
$$\mathcal{C}^1 \in \mathbb{R}^4 = \begin{cases} \textbf{Disk I/O quota } \{1, 2, 4, 8, 16, 32, 48\}(\text{MBps}) \\ \textbf{Buffer pool size } \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024\}(\text{MB}) \\ \textbf{CPU quota } \{10\%, 25\%, 50\%, 100\%\} \\ \textbf{\textit{Platform}} \{\textit{Platform 3}\} \end{cases}$$

$$\mathcal{M}^1 \in \mathbb{R} = \text{TPC-C transactions per minute (tpmC)}$$

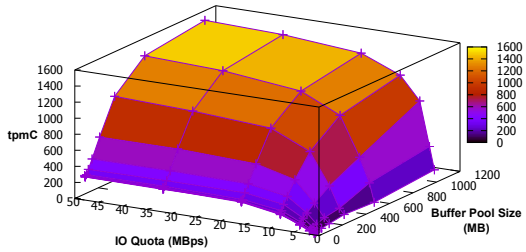
$$Unknown\ model\ \rho^1 : \mathcal{C}^1 \rightarrow \mathcal{M}^1$$



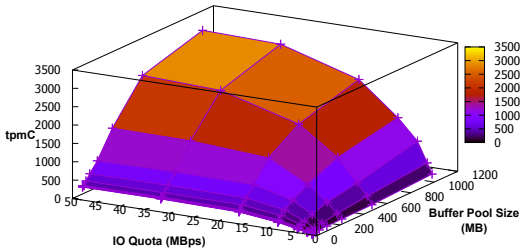
(a) **CPU quota**=10%



(b) **CPU quota**=25%



(c) **CPU quota**=50%



(d) **CPU quota**=100%

Figure 4.6: Performance graph of TPC-C transactions per minute (tpmC) on **Platform 3** when **IO quota** varies between 1 and 48 MBps and **Buffer pool** varies between 1 and 1024 MB.

For this scenario, we choose to use the simplest form of map, using a single feature from the performance function's codomain:

$$\mathcal{M}^0 = \text{TPC-C transactions per minute (tpmC)}$$

$$\mathcal{K} = \{\mathcal{M}^0\}$$

therefore a map of class $(0, |\mathcal{K}|) = (0, 1)$. We follow the formulation described in Section 4.4.1 and the experimental procedure described in Section 4.4.2.

4.5.2 Results

Table 4.6 and Table 4.7 show the results of our experiments using six different learning approaches. With only five samples, we can observe that the best results provided by a direct model that does not exploit transfer learning or incremental modeling (Gaussian Process, Linear SVR, Linear Regression and Polynomial regression) are substantially worse than our Model Mapping technique using any of Gaussian Process, Linear SVR or Linear Regression, to learn the map. In particular, we observe again that the highest benefit of Model Mapping is realized when the sampling budget is small. We observe that the Polynomial regression model applied to the map performs worse with 10 samples than with 5, which we attribute to the 4th order polynomial overfitting the available data.

Table 4.6 shows that our technique outperforms other incremental modeling and transfer learning techniques (Multi-Task Gaussian Process and Chorus) when considering the RMSE error metric. We also observe from Table 4.7 that, for the MAPE metric and using five samples our technique performs marginally worse than Chorus, but still outperforms Multi-Task Gaussian Process.

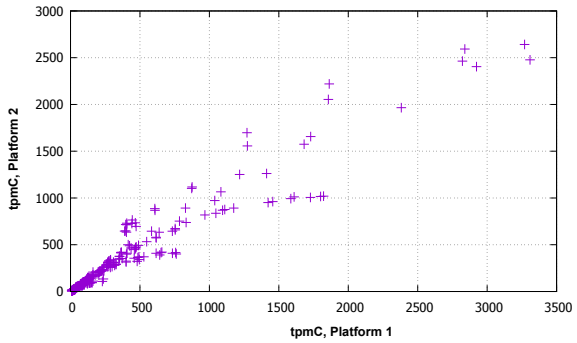
Examining the dataset, we can see that the reason behind these results is that the relationship between the two configurations' codomains (Figure 4.7b) is approximately linear. We expect a linear regression to perform well when modeling the map, although we can see that other modeling methods are just as effective.

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	647.87±67.3	56.63±3.8
	Polynomial regression	380.10±14.4	225.74±73.9
	Linear SVR	417.11±14.1	52.03±2.2
	Gaussian Process	383.90±15.3	**43.60±1.1**
	Chorus*	97.47±0.6	N/A
	Multi-Task Gaussian Process*	48.61±2.3	N/A
10	Linear regression	472.55±40.6	51.02±2.2
	Polynomial regression	296.55±12.5	449.64±115.0
	Linear SVR	392.57±18.2	48.54±1.4
	Gaussian Process	298.91±15.3	**44.22±1.2**
	Chorus*	96.58±1.0	N/A
	Multi-Task Gaussian Process*	50.75±1.9	N/A

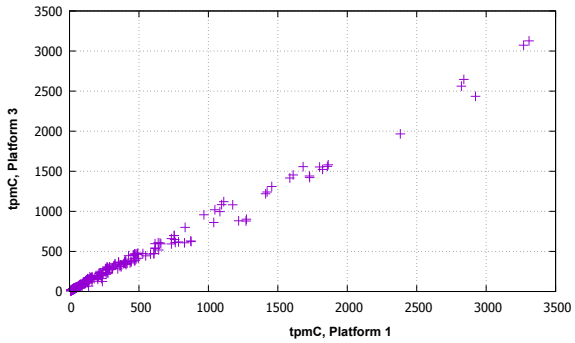
Table 4.6: RMSE mean and standard error of the proposed Model Mapping technique and direct modeling technique with various modeling methods (* indicates a direct modeling method that uses incremental or transfer learning). Legacy model **Platform**=Platform 1 and unknown model **Platform**=Platform 3.

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	921.79±191.7	19.55±2.6
	Polynomial regression	336.17±42.1	19.42±2.3
	Linear SVR	382.90±59.5	15.85±2.3
	Gaussian Process	254.13±23.1	16.41±2.1
	Chorus*	**13.21±0.0**	N/A
	Multi-Task Gaussian Process*	20.97±2.6	N/A
10	Linear regression	702.11±133.9	22.28±2.2
	Polynomial regression	317.78±28.7	19.32±1.4
	Linear SVR	337.74±52.7	15.28±1.7
	Gaussian Process	172.51±12.9	**12.39±1.0**
	Chorus*	13.16±0.0	N/A
	Multi-Task Gaussian Process*	25.01±3.0	N/A

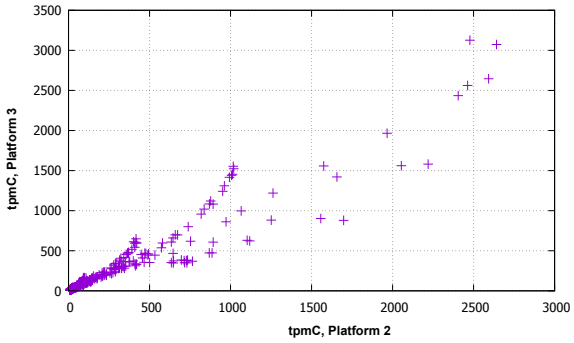
Table 4.7: MAPE mean and standard error of the proposed Model Mapping technique and direct modeling technique with various modeling methods (* indicates a direct modeling method that uses incremental or transfer learning). Legacy model **Platform**=Platform 1 and unknown model **Platform**=Platform 3.



(a) map of **Platform** 1 to **Platform** 2



(b) map of **Platform** 1 to **Platform** 3



(c) map of **Platform** 2 to **Platform** 3

Figure 4.7: Graph of different maps of Transactions per minute (tpmC) between different platforms.

4.6 Scenario 3: Modeling a Substantial System Downgrade

Similar to the previous scenario, a system administrator is interested in the feasibility of reducing the cost of operating a database service by migrating the service from a high-performance system to older, less powerful hardware, while maintaining a determined level of performance. In this case the difference between the specifications is more substantial than in Scenario 2 in terms of the amount of system RAM (512GB to 24GB) and the performance of the storage subsystem (SSD to conventional spinning disks). The administrator has already obtained a model of the database's performance under numerous variations of all available resource quotas in the system, but needs to know how to adjust the quota to compensate for the hardware downgrade.

The scenario represents the procedure of incrementally modeling our application's performance when a larger, whole-system architectural modification is performed, with few samples obtained by running the application on the new platform.

4.6.1 Formulation

The modeling problem is almost identical to the one described in Section 4.5.1. The only difference in this scenario is that the legacy hardware configuration space $\mathcal{C}^0$ represents **Platform 2** (Figure 4.8) instead of **Platform 1**, as follows:

$$\mathcal{C}^0 \in \mathbb{R}^4 = \begin{cases} \textbf{Disk I/O quota } \{1, 2, 4, 8, 16, 32, 48\}(\text{MBps}) \\ \textbf{Buffer pool size } \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024\}(\text{MB}) \\ \textbf{CPU quota } \{10\%, 25\%, 50\%, 100\%\} \\ \textbf{\textit{Platform}} \{\textit{Platform 2}\} \end{cases}$$

4.6.2 Results

Table 4.8 and Table 4.9 show the results of the experimentation. We can observe that the best results provided by a direct model that does not exploit transfer learning or incremental modeling are considerably worse than our Model Mapping technique using any of Gaussian Process, Linear SVR or Linear Regression. In particular, once again the advantage of Model Mapping is largest with the smallest sampling budget. Similar to the previous scenario, Model Mapping outperforms the other incremental modeling techniques as well.

4.7 Scenario 4: Modeling the Effects of Incremental Application Performance Variation

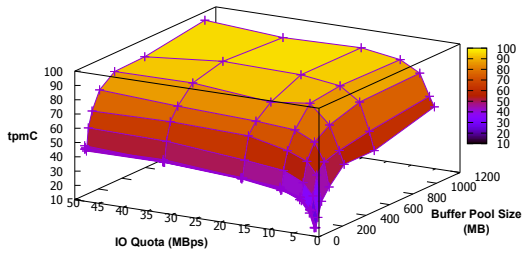
A database system's performance generally decreases as a consequence of data accumulation. In this scenario, a system administrator wants to understand the evolution of the performance characteristics of a database service. The purpose is to alter the system resource allocation quotas to compensate for the performance decreasing over time, while respecting a desired

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	647.87±67.3	168.79±7.3
	Polynomial regression	380.10±14.4	1,131.74±338.3
	Linear SVR	417.11±14.1	161.53±7.3
	Gaussian Process	383.90±15.3	**142.04±3.1**
	Chorus*	159.75±0.3	N/A
	Multi-Task Gaussian Process*	168.97±11.0	N/A
10	Linear regression	472.55±40.6	166.32±8.0
	Polynomial regression	296.55±12.5	1,558.34±442.2
	Linear SVR	392.57±18.2	158.61±8.1
	Gaussian Process	298.91±15.3	**150.77±4.7**
	Chorus*	160.28±1.1	N/A
	Multi-Task Gaussian Process*	171.67±8.6	N/A

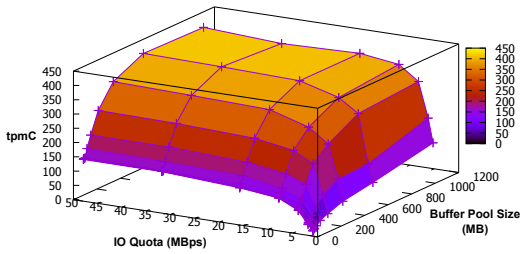
Table 4.8: RMSE mean and standard error of the proposed Model Mapping technique and direct modeling technique with various modeling methods (* indicates a direct modeling method that uses incremental or transfer learning). Legacy model **Platform=** Platform 2 and unknown model **Platform=** Platform 3.

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	921.79±191.7	50.69±8.9
	Polynomial regression	336.17±42.1	34.24±4.1
	Linear SVR	382.90±59.5	32.49±4.8
	Gaussian Process	254.13±23.1	**25.85±3.0**
	Chorus*	31.61±0.2	N/A
	Multi-Task Gaussian Process*	65.26±12.9	N/A
10	Linear regression	702.11±133.9	44.76±5.8
	Polynomial regression	317.78±28.7	34.72±3.2
	Linear SVR	337.74±52.7	27.62±3.5
	Gaussian Process	172.51±12.9	**26.70±3.2**
	Chorus*	32.10±0.4	N/A
	Multi-Task Gaussian Process*	74.20±11.3	N/A

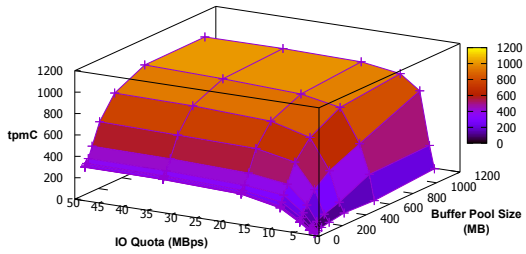
Table 4.9: MAPE mean and standard error of the proposed Model Mapping technique and direct modeling technique with various modeling methods (* indicates a direct modeling method that uses incremental or transfer learning). Legacy model **Platform=** Platform 2 and unknown model **Platform=** Platform 3.



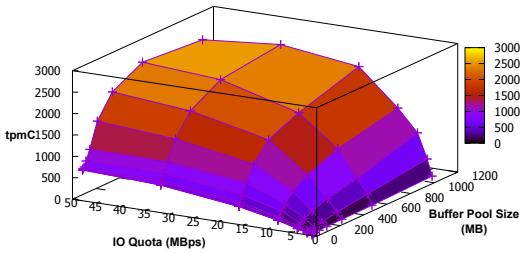
(a) **CPU quota**=10%



(b) **CPU quota**=25%



(c) **CPU quota**=50%



(d) **CPU quota**=100%

Figure 4.8: Performance graph of TPC-C transactions per minute (tpmC) on **Platform 2** when **IO quota** varies between 1 and 48 MBps and **Buffer pool** varies between 1 and 1024 MB.

Service Level Agreement. The administrator has already obtained a model of the system's performance at a specific point in time under numerous variations of all available resource quotas in the system, but needs to know how to adjust the quota over time to maintain the same level of expected performance.

The scenario represents the procedure of incrementally modeling our application's performance as it gets affected by a latent, non-configurable variable, with few samples obtained by running the application in the current state.

4.7.1 Formulation

The legacy model ρ^0 in this scenario was built using a subset of our TPC-C dataset that represents the database engine operating with an on-disk data size of $\sim$1.2GB, by configuring the TPC-C benchmark with 10 warehouses.

The configuration space $\mathcal{C}^0$ is a vector of four-dimensional tuples containing all combina-

tions of the three system configuration parameters **Disk I/O quota**, **Buffer pool size** and **CPU quota**, with the fixed fourth parameter **TPC-C Warehouses**=$\{10\}$.

$$\mathcal{C}^0 \in \mathbb{R}^4 = \begin{cases} \textbf{Disk I/O quota } \{1, 2, 4, 8, 16, 32, 48\}\text{(MBps)} \\ \textbf{Buffer pool size } \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024\}\text{(MB)} \\ \textbf{CPU quota } \{10\%, 25\%, 50\%, 100\%\} \\ \textbf{\textit{TPC-C Warehouses}} \ \{\textit{10}\} \end{cases}$$

We obtain from our dataset a vector in the performance $\mathcal{M}^0$ in which each entry contains the result of measuring the TPC-C benchmark throughput using the system configuration parameters in the corresponding entry of $\mathcal{C}^0$.

$$\mathcal{M}^0 \in \mathbb{R} = \text{TPC-C throughput (transactions per minute)}$$

$$Legacy\ model\ \rho^0 : \mathcal{C}^0 \rightarrow \mathcal{M}^0$$

The new model ρ^1 we intend to obtain represents the database engine operating with an on-disk data size of $\sim$6GB (64 warehouses). This corresponds to an approximately five-fold increase in the size of the database on disk.

$$\mathcal{C}^1 \in \mathbb{R}^4 = \begin{cases} \textbf{Disk I/O quota } \{1, 2, 4, 8, 16, 32, 48\}\text{(MBps)} \\ \textbf{Buffer pool size } \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024\}\text{(MB)} \\ \textbf{CPU quota } \{10\%, 25\%, 50\%, 100\%\} \\ \textbf{\textit{TPC-C Warehouses}} \ \{\textit{64}\} \end{cases}$$

$$\mathcal{M}^1 \in \mathbb{R} = \text{TPC-C throughput (transactions per minute)}$$

$$Unknown\ model\ \rho^1 : \mathcal{C}^1 \rightarrow \mathcal{M}^1$$

For this scenario, we began the experimentation by choosing the simplest form of map, which includes a single feature from the performance function's codomain:

$$\mathcal{M}^0 = \textbf{TPC-C transactions per minute (tpmC)}$$

$$\mathcal{K} = \{\mathcal{M}^0\}$$

therefore a map of class $(0, |\mathcal{K}|) = (0, 1)$. We follow the formulation described in Section 4.4.1 and the experimental procedure described in Section 4.4.2.

4.7.2 Results

Column **Mapping(0,1)** in Table 4.10 and Table 4.11 shows the results of our initial experiment using six different learning approaches and a map of class $(0, 1)$. As we can see, the Model Mapping technique in this situation trails both direct modeling and other transfer learning methods.

Examining the dataset, we can see that a one-dimensional function is insufficient to model the relationship between these configurations and their respective performance characteristics (Figure 4.9). However, this experiment also shows that, even when using a map with the fewest possible dimensions, our technique is still competitive with direct modeling.

We then added features from the legacy function domain, progressively increasing the dimensionality of the map. We experimented with maps of two, three and four dimensions, respectively of class $(1, 1)$, $(2, 1)$ and $(3, 1)$, as defined in Section 3.3.1. Feature selection for the different maps was accomplished by ranking them according to their sensitivity over the legacy model. Given the relatively small size of the configuration space, 2^k factorial sensitivity analysis [15] was used to determine sensitivity.

$$\mathcal{C}^0 = \textbf{Disk I/O quota}, \textbf{Buffer pool size}, \textbf{CPU quota}$$

Sampling Budget	Modeling Method	Modeling Technique				
		Direct	Mapping(0,1)	Mapping(1,1)	Mapping(2,1)	Mapping(3,1)
5	Linear regression	46.32±2.8	50.38±9.0	32.56±3.3	91.27±35.9	313.87±111.6
	Polynomial regression	28.18±0.6	42.78±0.7	23.56±0.5	22.77±0.8	24.04±0.9
	Linear SVR	30.38±0.5	43.58±1.2	23.03±0.8	25.35±1.3	26.83±1.3
	Gaussian Process	**27.68±0.9**	**38.69±2.0**	**22.13±0.8**	**20.89±1.6**	**21.48±1.4**
	Chorus*	43.12±0.0	N/A	N/A	N/A	N/A
	Multi-Task Gaussian Process*	36.32±2.9	N/A	N/A	N/A	N/A
10	Linear regression	27.90±0.5	40.57±4.0	27.21±2.8	26.91±3.0	29.15±3.4
	Polynomial regression	23.22±0.4	39.18±0.7	21.94±0.4	19.28±0.5	20.05±0.5
	Linear SVR	28.57±0.4	38.00±1.2	22.50±0.8	21.42±0.8	21.48±0.9
	Gaussian Process	**20.54±0.6**	**35.57±1.9**	**19.74±1.0**	**12.31±0.9**	**14.60±0.7**
	Chorus*	43.07±0.1	N/A	N/A	N/A	N/A
	Multi-Task Gaussian Process*	23.82±1.4	N/A	N/A	N/A	N/A

Table 4.10: RMSE mean and standard error of the proposed Model Mapping technique and direct modeling technique with various modeling methods (* indicates a direct modeling method that uses incremental or transfer learning). Legacy model: **TPC-C Warehouses**=10, unknown model: **TPC-C Warehouses**=64.

Sampling Budget	Modeling Method	Modeling Technique				
		Direct	Mapping(0,1)	Mapping(1,1)	Mapping(2,1)	Mapping(3,1)
5	Linear regression	196.81±13.6	122.01±10.8	64.47±5.4	249.26±116.0	1,071.83±377.7
	Polynomial regression	110.22±4.4	213.89±13.1	93.32±7.1	86.99±7.7	90.35±8.0
	Linear SVR	108.26±4.0	185.83±18.8	55.00±5.9	74.15±7.5	86.39±7.6
	Gaussian Process	90.62±4.8	**120.62±11.6**	**52.29±5.8**	**57.77±6.7**	66.25±8.5
	Chorus*	**59.52±0.0**	N/A	N/A	N/A	N/A
	Multi-Task Gaussian Process*	60.18±2.4	N/A	N/A	N/A	N/A
10	Linear regression	107.03±3.2	104.29±6.7	49.09±3.3	65.57±3.6	76.95±6.1
	Polynomial regression	71.23±1.9	175.44±9.0	66.23±3.8	56.44±3.3	58.48±3.6
	Linear SVR	87.09±2.7	127.27±13.2	44.77±2.9	50.61±3.6	54.48±4.3
	Gaussian Process	55.36±2.6	**82.21±5.3**	**34.93±1.5**	**27.84±1.9**	**36.84±2.7**
	Chorus*	59.37±0.0	N/A	N/A	N/A	N/A
	Multi-Task Gaussian Process*	**39.82±1.3**	N/A	N/A	N/A	N/A

Table 4.11: MAPE mean and standard error of the proposed Model Mapping technique and direct modeling technique with various modeling methods (* indicates a direct modeling method that uses incremental or transfer learning). Legacy model: **TPC-C Warehouses**=10, unknown model: **TPC-C Warehouses**=64.

$$\mathcal{M}^0 = \textbf{TPC-C transactions per minute (tpmC)}$$

$$\textbf{Mapping(1,1)} = \begin{cases} \mathcal{L} = \textbf{\{Disk I/O quota\}} \\ \mathcal{K} = \{\mathcal{M}^0\} \end{cases}$$

$$\textbf{Mapping(2,1)} = \begin{cases} \mathcal{L} = \textbf{\{Disk I/O quota, CPU quota\}} \\ \mathcal{K} = \{\mathcal{M}^0\} \end{cases}$$

$$\textbf{Mapping(3,1)} = \begin{cases} \mathcal{L} = \textbf{\{Disk I/O quota, Buffer pool size, CPU quota\}} \\ \mathcal{K} = \{\mathcal{M}^0\} \end{cases}$$

Results of these experiments can be seen in Table 4.10 and Table 4.11, in columns **Mapping(1,1)**, **Mapping(2,1)** and **Mapping(3,1)**.

For the second experiment a two-dimensional map was used. The results show the large benefit of adding one feature/map dimension, not only relative to a one-dimensional map, but to the direct modeling and other transfer learning methods. Using 5 samples, the results of adding further dimensions show a more modest benefit, suggesting that the small training set is sufficient to train only a two-dimensional model.

With 10 samples, the benefit of using three features is more substantial, which suggests that the map is best represented by a three-dimensional model, and 10 samples are sufficient to represent its features. Finally, we note that four dimensions delivers worse accuracy, which we attribute to an insufficiently large training set for a four-dimensional model.

We conclude from this experiment that, although maps of higher dimensionality are sometimes necessary to represent complex incremental behavior, increasing the map dimensionality also increases the map complexity, which does not always lead to a more accurate model.

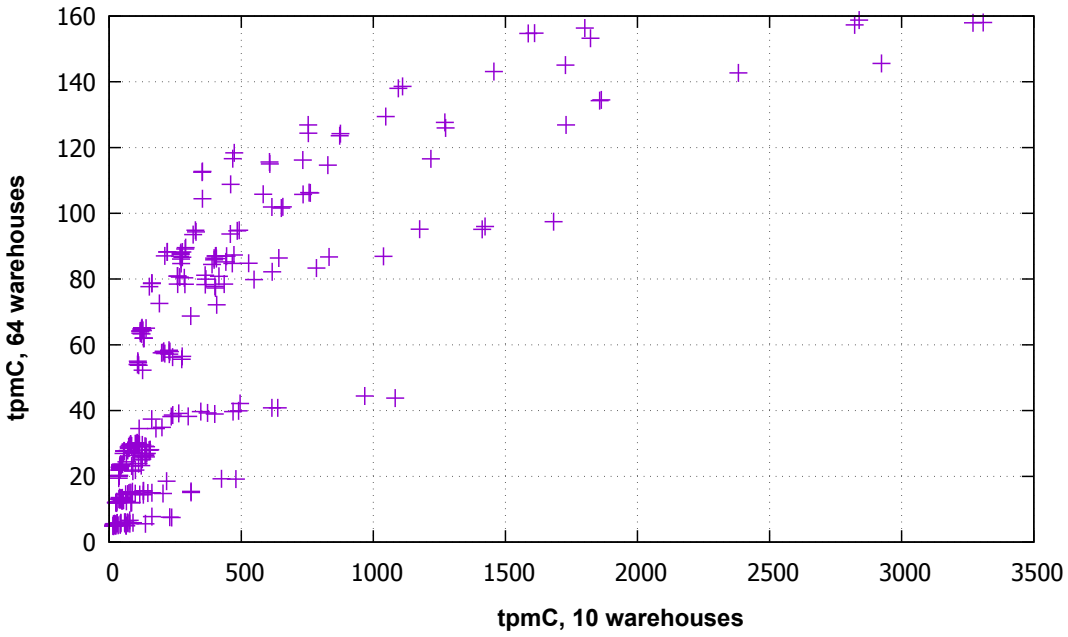


Figure 4.9: Map of Transactions per minute (tpmC) on **Platform** 1, 10 warehouses to 64 warehouses.

4.8 Scenario 5: Resource Optimization for VM Packing

In all our previous scenarios we aimed to model the performance of a single instance of a database Virtual Machine (VM), as a function of the system resources allocated to it. In Scenario 4 we measured the accuracy of incrementally modeling the database VM's performance as its data size varies over time. We now consider a hosted cloud environment, where multiple VMs share a common pool of hardware resources. A system administrator wants to solve a VM packing problem: maximize the aggregate performance of all VMs running database services sharing the same system resources. Achieving this goal requires finding the optimal resource allocation among the VMs.

We define $\vec{P}$ to represent the resource quotas (e.g. CPU, I/O, memory, etc.) reserved to an individual VM running on the system, and ρ to represent the corresponding VM's performance function. The total system performance for a system running n VMs is therefore:

$$T = \sum_{i=1}^{n} \rho_i(\vec{P}_i) \tag{4.4}$$

We also define $\vec{P}_{total}$ as the total available resources in the system. The VM packing problem for n VMs can be defined as finding appropriate resource quotas for all VMs, such that the total system performance is as high as possible:

$$\begin{aligned} \max_{\vec{P_1},\vec{P_2},\ldots,\vec{P_n}} \quad & \sum_{i=1}^{n} \rho_i(\vec{P_i}) \\ \text{s.t.} \quad & \sum_{i=1}^{n} \vec{P_i} \leq \vec{P_{total}} \end{aligned} \tag{4.5}$$

In this scenario, the administrator intends to find a configuration that maximizes the aggregate throughput of four database VMs, running on *Platform 1*, a system equipped with 16 cores (two 8-core Intel Xeon Haswell-series processors). As previously reported, our dataset represents a single VM's throughput as a function of its resource quotas, sampled in the configuration space reported in Table 4.2, for a total of 308 configurations. Finding the optimal resource allocation for a single VM only requires sampling each of the 308 possible resource configurations.

Observing the dataset (Figure 4.3 and Figure 4.2) we notice that, as it is often the case with complex computer systems [93], multiple combinations of resource allocations yield similar performance. For this reason, a resource configuration that evenly subdivides the resource quotas among the four VMs is not likely to be optimal. Enumerating all 308^4 possible configurations is also not feasible. Mathematical optimization is therefore necessary to search for the most appropriate subdivision of the hardware system's resources among the four VMs. While optimal resource allocation in computer systems is outside the scope of this dissertation, several mathematical optimization methods have been employed in the literature for this task [71, 91].

This experiment is designed to evaluate the suitability of the models obtained by Model Mapping to the task of resource allocation optimization. To that effect we need to understand how different models of the performance function, built with a limited training set, affect the solutions found by an optimization process.

4.8.1 Formulation

We assume that the administrator has already obtained an accurate model of the system's performance at a specific point in time, when the size of the databases in each of the four VMs was approximately 1.2GB (the legacy system configuration). The goal is to find the optimal allocation at a later point in time, when the databases inside the VMs have grown to 2.5GB (the unknown system configuration), with a performance model obtained incrementally. While an accurate model does not guarantee convergence on an optimal solution, the ability to obtain an accurate model of the system's performance with very few samples, reusing any available legacy information, is still important.

The performance function of a single VM in this new, unknown system configuration is represented by ρ^1. Following the example above, let our four Virtual Machine configuration parameter sets be represented as vectors, as follows:

$$\vec{P_1}, \vec{P_2}, \vec{P_3}, \vec{P_4} \in \begin{bmatrix} \{1, 2, 4, 8, 16, 32, 48\} \\ \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024\} \\ \{10\%, 25\%, 50\%, 100\%\} \end{bmatrix}$$

The VMs of all four database instances are hosted on the same hypervisor and are assigned an aggregate quota for IO bandwidth (total=48MBps), memory for database buffer pool (total=1024MB) and CPU (total=100%). We therefore define:

$$\vec{P}_{total} = \begin{bmatrix} \textbf{Disk I/O quota} = 48 \\ \textbf{Buffer pool size} = 1024 \\ \textbf{CPU quota} = 100\% \end{bmatrix}$$

For the purpose of this experiment we assume perfect performance isolation between the concurrently executing applications. Given our single VM performance function ρ^1 we can therefore calculate the total system throughput by establishing individual configurations and sam-

pling ρ^1 once per VM, and adding the contributions. The total system throughput is then defined as:

$$T = \sum_{i=1}^{4} \rho^1(\vec{P}_i) \tag{4.6}$$

We establish a basic reference performance measurement, corresponding to a naively selected system configuration, which evenly divides the available resources among the four VMs. In such configuration, each VM is configured with the following parameters: **CPU quota**=25%, **IO quota**=12Mbps, and **Buffer pool size**=256MB, as follows:

$$\vec{P}_1^e = \vec{P}_2^e = \vec{P}_3^e = \vec{P}_4^e = \begin{bmatrix} 12 \\ 256, \\ 25\% \end{bmatrix}$$

We refer to this resource configuration as *Equal*, to highlight the even partition of the available resources. With our exhaustively sampled dataset, we can obtain an exact model of ρ^1, which we call ρ^e. We can sample ρ^e to retrieve the performance of each VM, as follows:

$$\rho^e(\vec{P}_1^e) = \rho^e(\vec{P}_2^e) = \rho^e(\vec{P}_3^e) = \rho^e(\vec{P}_4^e) = 160.1.$$

The total system performance for the *Equal* resource configuration is therefore:

$$T_{equal} = 160.1 + 160.1 + 160.1 + 160.1 = 640.4. \tag{4.7}$$

We can now proceed to establish the main basis of comparison for our experiment, by retrieving the optimal performance of the system in the unknown configuration. We achieve this goal by

solving a resource optimization problem using ρ^e, our exact model of ρ^1, as follows:

$$\begin{aligned} \max_{\vec{P_1},\vec{P_2},\vec{P_3},\vec{P_4}} \quad & \rho^e(\vec{P_1}) + \rho^e(\vec{P_2}) + \rho^e(\vec{P_3}) + \rho^e(\vec{P_4}) \\ \text{s.t.} \quad & \vec{P_1} + \vec{P_2} + \vec{P_3} + \vec{P_4} \leq \begin{bmatrix} 48 \\ 1024 \\ 100\% \end{bmatrix} \\ & \vec{P_1}, \vec{P_2}, \vec{P_3}, \vec{P_4} \in \begin{bmatrix} \{1, 2, 4, 8, 16, 32, 48\} \\ \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024\} \\ \{10\%, 25\%, 50\%, 100\%\} \end{bmatrix} \end{aligned} \tag{4.8}$$

We then use the following equation to retrieve the expected system performance corresponding to the resulting configuration.

$$T = \rho^e(\vec{P_1}) + \rho^e(\vec{P_2}) + \rho^e(\vec{P_3}) + \rho^e(\vec{P_4}) \tag{4.9}$$

We used Simulated Annealing [119] with 30 random restarts, to increase the probability of retrieving the global optimum. We observed that in all restarts the algorithm converged to a solution with the same cost.

The configuration recovered by the optimization process is represented in Table 4.12. We use this configuration to calculate the aggregated throughput as follows:

$$\begin{aligned} \rho^e(\vec{P_1}) &= 160.5. \\ \rho^e(\vec{P_2}) &= 490.2. \\ \rho^e(\vec{P_3}) &= 41.0. \\ \rho^e(\vec{P_4}) &= 41.0. \end{aligned} \tag{4.10}$$

$$T_{ideal} = 160.5 + 490.2 + 41.0 + 41.0 = 732.7.$$

We refer to this configuration as *Ideal*, as it was obtained using an exact model of ρ^1.

	VM-1	VM-2	VM-3	VM-4
I/O	16	16	8	8
Buffer	256	512	128	128
CPU	25%	50%	10%	10%
Throughput	160.5	490.2	41.0	41.0
Total throughput	732.7			

Table 4.12: *Ideal* resources allocation of four database VMs. This result was obtained by leveraging an exact model of the (ρ^1) performance function in the optimization process in Equation 4.8. TPC-C configured with 20 warehouses.

We now formulate the problem of retrieving a good resource allocation, using models of the system's performance function in the unknown configuration (ρ^1), obtained with six different learning approaches.

The legacy model ρ^0 in this scenario was built using a subset of our TPC-C dataset that represents a single VM of the database system operating on a database size of $\sim$1.2GB, by configuring the TPC-C benchmark with 10 warehouses. The configuration space $\mathcal{C}^0$ is a vector of four-dimensional tuples containing all combinations of the three system configuration parameters **Disk I/O quota**, **Buffer pool size** and **CPU quota**, with the fixed fourth parameter **TPC-C Warehouses** = $\{10\}$.

$$\mathcal{C}^0 \in \mathbb{R}^4 = \begin{cases} \textbf{Disk I/O quota } \{1, 2, 4, 8, 16, 32, 48\}\text{(MBps)} \\ \textbf{Buffer pool size } \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024\}\text{(MB)} \\ \textbf{CPU quota } \{10\%, 25\%, 50\%, 100\%\} \\ \textbf{\textit{TPC-C Warehouses}} \ \{10\} \end{cases}$$

We obtain from our dataset a vector in the performance $\mathcal{M}^0$ in which each entry contains the result of measuring the TPC-C benchmark throughput using the system configuration parameters in the corresponding entry of $\mathcal{C}^0$.

$$\mathcal{M}^0 \in \mathbb{R} = \text{TPC-C throughput (transactions per minute)}$$

$$Legacy\ model\ \rho^0 : \mathcal{C}^0 \rightarrow \mathcal{M}^0$$

The new model of ρ^1 we intend to obtain represents the performance of a single VM when its database size is $\sim$2.4GB (20 warehouses). We call this new model ρ^m. This corresponds to an approximately 100% increase in the size of the database on disk.

$$\mathcal{C}^1 \in \mathbb{R}^4 = \begin{cases} \textbf{Disk I/O quota } \{1, 2, 4, 8, 16, 32, 48\}\text{(MBps)} \\ \textbf{Buffer pool size } \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024\}\text{(MB)} \\ \textbf{CPU quota } \{10\%, 25\%, 50\%, 100\%\} \\ \textbf{\textit{TPC-C Warehouses}} \ \{20\} \end{cases}$$

$$\mathcal{M}^1 \in \mathbb{R} = \text{TPC-C throughput (transactions per minute)}$$

$$Unknown\ model\ \rho^m : \mathcal{C}^1 \rightarrow \mathcal{M}^1$$

In light of the results in Scenario 4, we choose a map which includes two features from the model's domain and a single feature from the performance function's codomain:

$$\mathcal{C}^0 = \textbf{Disk I/O quota}, \textbf{Buffer pool size}, \textbf{CPU quota}$$

$$\mathcal{M}^0 = \textbf{TPC-C transactions per minute (tpmC)}$$

$$\mathcal{L} = \textbf{\{Disk I/O quota, CPU quota\}}$$

$$\mathcal{K} = \{\mathcal{M}^0\}$$

therefore a map of class $(|\mathcal{L}|, |\mathcal{K}|) = (2, 1)$. We follow the same formulation described in Section 4.4.1.

Similar to the procedure we used earlier, we use each model ρ^m we obtain using six different modeling techniques to solve the following resource allocation optimization problem:

$$\begin{aligned} \max_{\vec{P_1},\vec{P_2},\vec{P_3},\vec{P_4}} \quad & \rho^m(\vec{P_1}) + \rho^m(\vec{P_2}) + \rho^m(\vec{P_3}) + \rho^m(\vec{P_4}) \\ \text{s.t.} \quad & \vec{P_1} + \vec{P_2} + \vec{P_3} + \vec{P_4} \leq \begin{bmatrix} 48 \\ 1024 \\ 100\% \end{bmatrix} \\ & \vec{P_1}, \vec{P_2}, \vec{P_3}, \vec{P_4} \in \begin{bmatrix} \{1, 2, 4, 8, 16, 32, 48\} \\ \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024\} \\ \{10\%, 25\%, 50\%, 100\%\} \end{bmatrix} \end{aligned} \tag{4.11}$$

and then measure the real system throughput of the obtained configurations by sampling ρ^e, the exact model of ρ^1, as follows:

$$T_{unknown} = \rho^e(\vec{P_1}) + \rho^e(\vec{P_2}) + \rho^e(\vec{P_3}) + \rho^e(\vec{P_4}) \tag{4.12}$$

The error is measured as a relative percentage difference between the performance of the *Ideal* configuration and the ones retrieved by the optimization processes that use the different models of ρ^1, as follows:

$$Error = \frac{T_{ideal} - T_{unknown}}{T_{ideal}} * 100 \tag{4.13}$$

4.8.2 Experimental Procedure

To compare the effects of different modeling techniques, we train multiple models of ρ^1, and run the resource optimization procedure described above. Each run of the optimization procedure yields a resource configuration, and its associated estimation of total system throughput. We then calculate the error by comparing the performance of each retrieved configuration against the *Ideal* configuration obtained earlier. We repeat the entire procedure 30 times to increase the variation in the selection of the training set. Pseudo-code for the full procedure is

presented in Algorithm 2.

Algorithm 2: Evaluation of Resource Optimization using approximate performance function

```
Accept ground truth data for the legacy and the unknown functions being modeled;
Perform normalization of all functions' domain and codomain data;
Construct the legacy mathematical model from data;
Sample the TPC-C dataset to obtain the performance of the Ideal configuration
  → idealPerformance;
i ← 30;
while i ≥ 0 do
    randomSeed ← s;
    Create training set by performing random selection of samples in the unknown
      function ground truth data set, using the sampling budget;
    Train an instance of the ModelMap class using the training set;
    for model in Models do
        Use Simulated Annealing to find the resource configuration using the model as
          the optimization objective function;
        Sample the ground truth TPC-C dataset to obtain the real performance of the
          resulting configuration → modelPerformance;
        (idealPerformance − modelPerformance)/idealPerformance →
          iterationError;
        e⃗rror ← iterationError;
    i ← i − 1;
error ← mean(e⃗rror);
```

4.8.3 Results

Figure 4.10 shows the performance of individual VMs and total performance obtained as the output of the resource optimization process, using direct modeling and our mapping technique, with a sampling budget of 5.

Table 4.13 shows the configurations retrieved by the resource allocation optimization process, using respectively the best performing direct model, Chorus, Multi-Task Gaussian Process and Model Mapping to model ρ^1, and trained on the TPC-C dataset configured with 20 warehouses, with a sampling budget of 5.

Table 4.14 shows the error of the configurations found using direct models (*Direct*) and

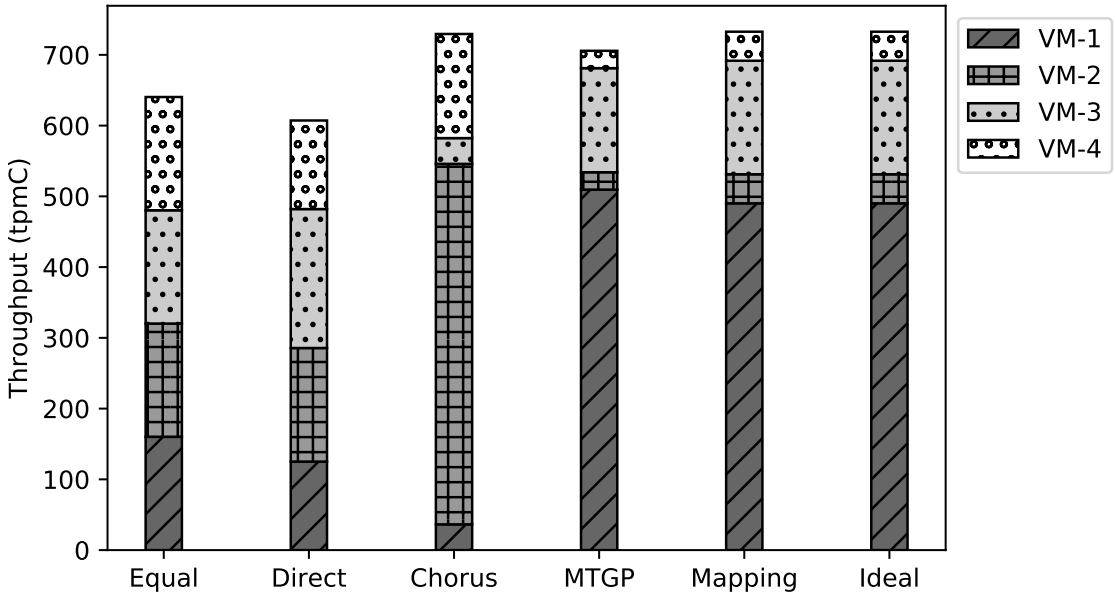


Figure 4.10: Resource optimization for four TPC-C instances. Sampling budget=5. Legacy model **TPC-C Warehouses**=10, unknown model **TPC-C Warehouses**=20. Map class: $(2, 1)$

our technique (*Mapping*), relative to the *Ideal* configuration. The results show how with only 5 samples, our technique accurately models the unknown system configuration, while direct models introduce inaccuracies that induce the optimization process to find suboptimal solutions. The other transfer learning techniques (Chorus and Multi-Task Gaussian Process) also outperform direct modeling, although Model Mapping achieves an error of only 0.1%.

While the configurations retrieved using Chorus and Model Mapping are considerably different, their performance is similar. The explanation for this result is that the two configurations are nearly symmetric.

To verify that the optimization procedure for the two highest-performing techniques (Chorus and Model Mapping) converged to maxima, we calculated the aggregate throughput estimated by the two models of ρ^1, using the best configuration obtained with the other model. Following Equation 4.6, we define T_c and T_m as the aggregate throughput function modeled with Chorus and Model Mapping, respectively. We also define P_a and P_m as the best configurations obtained by Chorus (Table 4.13b) and Model Mapping (Table 4.13d).

	VM-1	VM-2	VM-3	VM-4
CPU	25%	25%	25%	25%
I/O	8	16	16	8
Buffer	128	256	512	128
Thr.	125.1	160.4	196.4	125.1
Total	607.2			

(a)

	VM-1	VM-2	VM-3	VM-4
CPU	10%	50%	10%	25%
I/O	4	32	4	8
Buffer	128	512	128	256
Thr.	36.4	509.5	36.4	147.1
Total	729.4			

(b)

	VM-1	VM-2	VM-3	VM-4
CPU	50%	10%	25%	10%
I/O	32	4	8	4
Buffer	512	1	256	1
Thr.	509.5	24.5	147.1	24.5
Total	705.7			

(c)

	VM-1	VM-2	VM-3	VM-4
CPU	25%	50%	10%	10%
I/O	16	16	8	8
Buffer	256	512	128	128
Thr.	160.5	490.2	41.0	41.0
Total	732.7			

(d)

Table 4.13: Resource allocation optimization results. (a):**Polynomial** direct model. (b):**Chorus**. (c):**Multi-Task Gaussian Process** model. (d):**Model Mapping** model.

We verified that:

$$T_c(P_c) > T_c(P_a)$$

$$T_a(P_a) > T_a(P_c)$$

It is clear from the results that direct modeling methods that do not exploit incremental modeling do not provide accurate models. Using the Model Mapping technique, all learning methods are capable of producing a model that allows the resource allocation optimization procedure to find a resource configuration with performance within 1% of *Ideal*. We conclude that, while direct modeling may in some circumstances show comparable error scores, transfer learning substantially helps in efficiently recovering the general behavior of a system by leveraging correlations between configurations. Our Model Mapping technique in particular is capable of retrieving a near-perfect behavioral model that allows an optimization procedure to discover the most appropriate resource allocation for our scenario.

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	41.81%±5.96%	**0.15%±0.04%**
	Polynomial regression	21.42%±3.97%	0.42%±0.03%
	Linear SVR	29.68%±4.46%	0.18%±0.05%
	Gaussian Process	25.82%±3.96%	1.21%±0.67%
	Chorus*	0.44%±0.00%	N/A
	Multi-Task Gaussian Process*	1.93%±0.61%	N/A
10	Linear regression	21.91%±3.31%	**0.09%±0.04%**
	Polynomial regression	18.08%±2.82%	0.35%±0.04%
	Linear SVR	28.94%±4.18%	0.22%±0.05%
	Gaussian Process	19.18%±3.23%	0.52%±0.15%
	Chorus*	0.44%±0.00%	N/A
	Multi-Task Gaussian Process*	4.07%±1.28%	N/A

Table 4.14: Percentage error for maximum combined throughput in the VM packing scenario, relative to *Ideal* resource configuration, using the proposed Model Mapping technique and direct modeling technique with various modeling methods (* indicates a direct modeling method that uses incremental or transfer learning). Legacy model: **Warehouses**=10, unknown model: **Warehouses**=20.

4.9 Conclusions

Table 4.15 contains a summary of the results of our experiments, as the relative improvement that our technique has over both direct modeling and incremental/transfer learning. The results show that our Model Mapping technique is effective in accurately modeling incremental variations that database systems experience as part of their regular operations and maintenance. The five scenarios represent situations where system administrators intend to quickly and efficiently understand the effects of variations in the system's resource allocation and/or in the system's hardware configuration.

It is clear that in each of these scenarios, our technique outperforms traditionally used direct modeling methods, and presents accuracy advantages over other incremental learning methods, which ultimately translate in system performance improvement.

In order to ascertain the general applicability of Model Mapping, irrespective of the choice of modeling method, we measure the success rate of applying our technique. We define a

	Incremental variation	ΔErr Direct	ΔErr baseline T.Learning
Scenario 1	CPU Quota (25%→50%)	-75.43%	-7.57%
Scenario 2	HW Platform (1→3)	-88.53%	-10.29%
Scenario 3	HW Platform (2→3)	-62.63%	-11.09%
Scenario 4	Database size (1.2GB→6GB)	-24.53%	-42.48%
Scenario 5	Resource optimization	-99.29%	-65.91%

Table 4.15: Summary of database system performance scenarios. %improvement of Model Mapping vs. Direct Modeling and baseline Transfer Learning (Multi-Task Gaussian Process / Chorus). Sampling budget = 5.

successful experiment one where a modeling method applied to represent the map produces a more accurate result than the same method applied to model the unknown function. Table 4.16 and Table 4.17 show how Model Mapping consistently improves over direct modeling, with all modeling methods performing better when used with our technique in most circumstances.

We now continue the evaluation of our technique in the following chapter, by introducing a collection of filesystem performance modeling scenarios, another recurrent situation in cloud environments operations.

	Experiments	Success Rate
Linear regression	9/10	90%
Polynomial regression	5/10	50%
Linear SVR	10/10	100%
Gaussian Process	10/10	100%

Table 4.16: Success rate of Model Mapping vs. direct modeling, with different modeling methods (RMSE).

	Experiments	Success Rate
Linear regression	9/10	90%
Polynomial regression	10/10	100%
Linear SVR	10/10	100%
Gaussian Process	9/10	90%

Table 4.17: Success rate of Model Mapping vs. direct modeling, with different modeling methods (MAPE).

Chapter 5

File Storage System Performance Modeling

5.1 Introduction

The TPC-C dataset used in Chapter 4 was created concurrently with the research and development of the Model Mapping technique and the *ModelMap* toolkit. In order to demonstrate the general applicability of our work, in this chapter we turn to file storage system performance as an additional application scenario for experimentation. To achieve this goal, we use a dataset created independently, by a different research group.

One of the main requirements in operating managed systems, especially large-scale cloud environments, is minimizing the cost of providing users with an acceptable level of service performance [3, 4, 64]. Most hosted applications require accessing an underlying file storage system, and many options are available to manage their price/performance characteristics. From the appropriate choice of filesystem (e.g. ext3, ext4, zfs, etc.), to configuring a filesystem's parameters (e.g. the scheduling policy), to hardware configurations, the parameter space is extensive. Similar to other complex systems, appropriately configuring file storage systems can therefore be time consuming and expensive. To reduce the effort required to establish an

Parameter	Abbrev.	Values
Disk Type	DT	SATA, SAS, SAS500, SSD
File System	FS	Ext3, Ext4
Block Size	BS	1, 2, 4 (KB)
Inode Size, Sector Size	IS	default, 128, 256, 512, 1024, 2048, 4096, 8192 (Bytes)
Block Group	BG	default, 2, 4, 8, 16, 32, 64, 128, 256
I/O Scheduler	I/O	Noop, CFQ, Deadline
Journal Option*	JO	data=ordered
Atime Option*	AO	relatime

Table 5.1: File storage system benchmark configuration parameters settings (* denotes a fixed configuration parameter, for which no exhaustive sampling was available)

appropriate system configuration, it is necessary to accurately model the effect that varying the configuration parameters has on the performance metrics.

We build on the work of Cao et al. [20], who compared different black-box auto-tuning methods for storage systems optimization. As part of their research, the authors instrumented a set of hardware platforms, and measured the systems' performance using four benchmarking workloads. The extensive dataset obtained by profiling these systems under several thousands of file system configurations was publicly released [19].

After examining the dataset, we identified and extracted a portion of it for our experiments: an exhaustive combinatorial sampling of a subset of the available configuration parameters. In particular, we selected the subset obtained profiling the **Filebench-1.4.9.1** [101] benchmark webserver workload, running on the **M2** hardware platform ([20] - Table 3). The resulting dataset is described in Table 5.1.

Our *ModelMap* toolkit does not currently support categorical variables and non-numerical values, therefore we translated them into numerical variables and numbers, respectively, using the following conversion table:

- **Disk Type**: SATA=0, SAS=1, 500SAS=2, SSD=3
- **File System**: Ext3=0, Ext4=1
- **Inode Size**: default=128

- **Block Group**: default=12
- **I/O Scheduler**: CFQ=0, Deadline=1, Noop=2

We selected five scenarios to represent different incremental changes that a storage system may experience. The scenarios exercise our performance modeling technique with increasing the range and significance of change in the file system configuration. The scenarios progress from the incremental change of one configuration parameter of the file system, the disk type, to the incremental change of two configuration parameters, the disk type and the file system scheduling policy, to the change of the entire file system.

The scenarios also exercise variation in the significance of the change in the configuration parameters. For example, in Scenario 1 we change the hard disk type from SATA to SAS, while in Scenario 2 we change the SATA hard disk to an SSD, a storage medium with different principles of operation. In Scenario 3 and 4 we apply two simultaneous variations of increasing effect. In the former we simultaneously change hard disk type from SATA to SAS and the filesystem I/O scheduling policy from CFQ to Deadline, while in Scenario 4, the disk is replaced with an SSD, and the scheduling policy is set to noop. In Scenario 5 we change the filesystem from ext3 to ext4, which includes performance enhancement and different block allocation strategies.

5.2 Scenario 1: Modeling Effects of Changing Disk Type

In this scenario, a system administrator is interested in modeling the effects of upgrading the hard disk in a web server system from a 250GB SATA disk to a more reliable and faster 500GB SAS disk. Such a change is typical in storage systems maintenance and operation and reflects the availability of new products over time and parts obsolescence.

The administrator has obtained a complete model of the web server performance in the original configuration, with the SATA disk. The goal is to obtain an accurate model of the system after the upgrade as quickly as possible, to allow the system administrator to promptly

reconfigure the filesystem and leverage the difference in performance the upgraded disk provides.

This example is intended to demonstrate how, with a few samples, we can obtain a model that covers an incremental change in the system's hardware specifications.

5.2.1 Formulation

As reported above, the file storage system dataset consists of measurements of the Filebench benchmark throughput (in I/O operations per second), by setting the system's configuration parameters to all combinations of values in Table 5.1.

The legacy model ρ^0 in this scenario was built using a subset of the file storage system dataset. The configuration space $\mathcal{C}^0$ is a vector of five-dimensional tuples containing all combinations of the four parameters **I/O Scheduler**, **Block Size**, **Block Group** and **Inode Size**, with the fixed fifth parameter **Disk Type** = SATA, using the ext3 filesystem.

$$\mathcal{C}^0 \in \mathbb{R}^5 = \begin{cases} \textbf{Block Size } \{1, 2, 4\}\text{(KB)} \\ \textbf{Inode Size } \{128, 256, 512, 1024, 2048, 4096, 8192\}\text{(Bytes)} \\ \textbf{Block Group } \{2, 4, 8, 12, 16, 32, 64, 128, 256\} \\ \textbf{I/O Scheduler } \{\text{Noop, CFQ, Deadline}\} \\ \textbf{\textit{Disk Type}} \ \{\textit{SATA}\} \end{cases}$$

We extract from our dataset a vector $\mathcal{M}^0$, where the entries contain the result of measuring the Filebench benchmark throughput using the system configuration parameters of the corresponding entry of $\mathcal{C}^0$.

$$\mathcal{M}^0 \in \mathbb{R} = \text{Filebench throughput (I/O operations per second)}$$

$$Legacy\ model\ \rho^0 : \mathcal{C}^0 \rightarrow \mathcal{M}^0$$

The model of the new configuration we intend to obtain extends the legacy model with one extra dimension in the configuration space, allowing for the representation of the disk drive change from a SATA disk to a 500GB SAS disk, a new value of the **Disk Type** system parameter.

$$\mathcal{C}^1 \in \mathbb{R}^5 = \begin{cases} \textbf{Block Size } \{1, 2, 4\}\text{(KB)} \\ \textbf{Inode Size } \{128, 256, 512, 1024, 2048, 4096, 8192\}\text{(Bytes)} \\ \textbf{Block Group } \{2, 4, 8, 12, 16, 32, 64, 128, 256\} \\ \textbf{I/O Scheduler } \{\text{Noop, CFQ, Deadline}\} \\ \textbf{\textit{Disk Type}} \ \{\textit{500SAS}\} \end{cases}$$

$$\mathcal{M}^1 \in \mathbb{R} = \text{Filebench throughput (I/O operations per second)}$$

$$Unknown\ model\ \rho^1 : \mathcal{C}^1 \rightarrow \mathcal{M}^1$$

We use a three-dimensional map to represent the transformation function between the two models. After conducting a feature sensitivity analysis, as described in Section 4.7.2, we included the **Inode Size** and **Block Group** features from the legacy model's domain, and the only available feature from the model's codomain:

$$\mathcal{C}^0 = \textbf{Block Size}, \textbf{Inode Size}, \textbf{Block Group}, \textbf{I/O Scheduler}$$

$$\mathcal{M}^0 = \textbf{Filebench throughput}$$

$$\mathcal{L} = \textbf{\{Inode Size, Block Group\}}$$

$$\mathcal{K} = \{\mathcal{M}^0\}$$

therefore a map of class $(|\mathcal{L}|, |\mathcal{K}|) = (2, 1)$. Following the formulation described in Section 4.4.1, we train a model of the three-dimensional transfer function $\sigma : \mathcal{M}^0 \rightarrow \mathcal{M}^1$ by using the legacy model and the few samples we obtained from measuring the system in the new configurations selected from $\mathcal{C}^1$. We apply the experimental procedure described in Section 4.4.2 to iteratively select sets of samples, perform cross-validation, and measure the accuracy of all the selected modeling methods.

5.2.2 Results

Results for the experiments are contained in Table 5.2 and Table 5.3. We can observe that the accuracy of all direct models that do not exploit transfer learning or incremental modeling is considerably lower than Model Mapping. With a sampling budget of five samples, our technique shows an RMSE improvement of approximately 40% over the best direct model (Gaussian Process). The accuracy improvement remains consistent with a sampling budget of 10 samples. The results also show that our technique improves over the other incremental modeling methods, which leads us to conclude that Model Mapping is well suited to represent this type of system configuration change.

5.3 Scenario 2: Modeling Effects of Changing Storage Medium

In this scenario, a system administrator faces a more significant shift in storage technology: upgrading the hard disk in a web server system with a solid-state drive. Conventional hard disk drives are electro-mechanical devices, which magnetically store data on rotating platters, and use actuator arms to move magnetic heads to the appropriate position over the platters' surfaces and read/write data. Mechanical friction and inertia set a lower bound on the latency of data access (seek time) for random access patterns. Conversely, solid-state drives store data using persistent semiconductor storage cells and have no moving parts, and thus having extremely low latency for random data access.

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	467.72±54.8	291.36±41.2
	Polynomial regression	236.51±5.5	134.24±3.3
	Linear SVR	239.25±8.3	144.85±4.0
	Gaussian Process	223.19±5.2	**132.55±3.7**
	Chorus*	198.96±0.9	N/A
	Multi-Task Gaussian Process*	164.45±3.6	N/A
10	Linear regression	255.84±16.1	161.80±7.4
	Polynomial regression	224.95±3.7	132.33±2.0
	Linear SVR	226.11±4.5	135.73±2.1
	Gaussian Process	215.38±3.2	**128.70±2.2**
	Chorus*	198.22±1.2	N/A
	Multi-Task Gaussian Process*	151.74±2.2	N/A

Table 5.2: RMSE mean and standard error of the proposed Model Mapping technique and direct modeling technique with various modeling methods (* indicates a direct modeling method that uses incremental or transfer learning). Legacy model: **Disk Type**=SATA. Unknown model: **Disk Type**=500SAS.

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	2.36±0.30	1.40±0.22
	Polynomial regression	1.15±0.04	0.67±0.02
	Linear SVR	1.21±0.04	0.72±0.02
	Gaussian Process	1.07±0.03	**0.65±0.02**
	Chorus*	0.96±0.01	N/A
	Multi-Task Gaussian Process*	0.81±0.02	N/A
10	Linear regression	1.24±0.09	0.78±0.03
	Polynomial regression	1.06±0.02	0.65±0.01
	Linear SVR	1.10±0.03	0.66±0.01
	Gaussian Process	1.01±0.01	**0.63±0.01**
	Chorus*	0.95±0.01	N/A
	Multi-Task Gaussian Process*	0.73±0.01	N/A

Table 5.3: MAPE mean and standard error of the proposed Model Mapping technique and direct modeling technique with various modeling methods (* indicates a direct modeling method that uses incremental or transfer learning). Legacy model: **Disk Type**=SATA. Unknown model: **Disk Type**=500SAS.

Having previously obtained a model of the web server performance using the SATA disk under variations of the filesystem's configuration parameters, the administrator intends to re-configure the filesystem to suit the difference in performance the new storage medium provides. As with the previous scenario, this example aims to extend the model to represent a variation of an additional configuration parameter, which produces a non-linear variation in the performance behavior.

5.3.1 Formulation

The modeling problem and experimental procedure in this scenario are very similar to the one described in Section 5.2.1. The difference in the configuration change is that the SATA disk drive is being replaced by an SSD drive, as follows:

$$\mathcal{C}^1 \in \mathbb{R}^5 = \begin{cases} \textbf{Block Size } \{1, 2, 4\}\text{(KB)} \\ \textbf{Inode Size } \{128, 256, 512, 1024, 2048, 4096, 8192\}\text{(Bytes)} \\ \textbf{Block Group } \{2, 4, 8, 12, 16, 32, 64, 128, 256\} \\ \textbf{I/O Scheduler } \{\text{Noop, CFQ, Deadline}\} \\ \textbf{\textit{Disk Type}} \ \{\textit{SSD}\} \end{cases}$$

5.3.2 Results

Results of our experiments are represented in Table 5.4 and Table 5.5. Relative to Scenario 1, we observe a reduction in the benefit of applying the incremental modeling and transfer learning techniques (Model Mapping, Multi-Task Gaussian Process and Chorus) compared to modeling the new system configuration directly. The difference in behavior between a conventional hard disk drive and a solid-state drive greatly affects the performance profile of the storage system and the applications that leverage it.

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	321.83±74.8	170.81±29.5
	Polynomial regression	123.29±3.7	129.72±4.9
	Linear SVR	143.55±5.1	**122.82±4.5**
	Gaussian Process	126.71±3.9	131.62±6.2
	Chorus*	149.84±1.3	N/A
	Multi-Task Gaussian Process*	138.55±3.6	N/A
10	Linear regression	102.89±5.2	**99.75±6.2**
	Polynomial regression	107.22±2.8	117.40±4.4
	Linear SVR	109.71±4.4	103.96±4.5
	Gaussian Process	110.71±3.5	109.73±5.3
	Chorus*	145.88±1.3	N/A
	Multi-Task Gaussian Process*	116.47±4.5	N/A

Table 5.4: RMSE mean and standard error of the proposed Model Mapping technique and direct modeling technique with various modeling methods (* indicates a direct modeling method that uses incremental or transfer learning). Legacy model: **Disk Type**=500SAS. Unknown model: **Disk Type**=SSD.

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	1.67±0.39	0.85±0.15
	Polynomial regression	0.63±0.02	0.62±0.02
	Linear SVR	0.75±0.03	**0.60±0.02**
	Gaussian Process	0.65±0.02	0.65±0.03
	Chorus*	0.82±0.01	N/A
	Multi-Task Gaussian Process*	0.73±0.02	N/A
10	Linear regression	0.53±0.03	**0.49±0.03**
	Polynomial regression	0.54±0.02	0.57±0.02
	Linear SVR	0.56±0.02	0.51±0.02
	Gaussian Process	0.57±0.02	0.54±0.02
	Chorus*	0.80±0.01	N/A
	Multi-Task Gaussian Process*	0.61±0.02	N/A

Table 5.5: MAPE mean and standard error of the proposed Model Mapping technique and direct modeling technique with various modeling methods (* indicates a direct modeling method that uses incremental or transfer learning). Legacy model: **Disk Type**=500SAS. Unknown model: **Disk Type**=SSD.

This substantial difference translates to a more complex relationship between the two system configurations, which in turn limits the benefit of applying prior knowledge to the modeling problem. At the same time, we observe that using our Model Mapping technique, accuracy is comparable to the best direct modeling technique we tested, which shows the robustness of our technique in this situation.

5.4 Scenario 3: Modeling multiple system configuration changes at once

In this scenario, a system administrator is aware of upcoming variations to a web server system (e.g. a planned hardware upgrade), and intends to introduce related measures generally known to be effective, and quickly verify the effects of the cumulative changes. For this example, the system's 250GB SATA hard disk is being upgraded to a more reliable and faster 500GB SAS disk. The additional modification is to the filesystem I/O scheduler setting, from *CFQ* to *deadline* to exploit the expected increase in data bus I/O performance. The administrator has already obtained a model of the web server performance running on the original system configuration. This example shows how to extend a model to cover multiple concurrent variations of an original performance profile.

5.4.1 Formulation

The legacy model ρ^0 is once again a subset of our file storage system dataset. The configuration space $\mathcal{C}^0$ is a vector of five-dimensional tuples containing all combinations of the three parameters **Block Size**, **Block Group** and **Inode Size**, with the fixed fourth parameter **I/O Scheduler** = CFQ and fifth parameter **Disk Type** = SATA, using the ext3 filesystem.

$$\mathcal{C}^0 \in \mathbb{R}^5 = \begin{cases} \textbf{Block Size } \{1, 2, 4\}\text{(KB)} \\ \textbf{Inode Size } \{128, 256, 512, 1024, 2048, 4096, 8192\}\text{(Bytes)} \\ \textbf{Block Group } \{2, 4, 8, 12, 16, 32, 64, 128, 256\} \\ \textbf{\textit{I/O Scheduler}} \{\textit{CFQ}\} \\ \textbf{\textit{Disk Type}} \{\textit{SATA}\} \end{cases}$$

As with the previous scenarios, we obtain the legacy performance model ρ^0 as follows:

$$\mathcal{M}^0 \in \mathbb{R} = \text{Filebench throughput (I/O operations per second)}$$

$$Legacy\ model\ \rho^0 : \mathcal{C}^0 \rightarrow \mathcal{M}^0$$

The new model adds two extra dimensions to the configuration space to represent the change from a SATA disk drive to a 500GB SAS disk drive, and the filesystem's I/O scheduler setting to *Deadline*, which we consider new configurations for the **I/O Scheduler** and **Disk Type** system parameters.

$$\mathcal{C}^1 \in \mathbb{R}^5 = \begin{cases} \textbf{Block Size } \{1, 2, 4\}\text{(KB)} \\ \textbf{Inode Size } \{128, 256, 512, 1024, 2048, 4096, 8192\}\text{(Bytes)} \\ \textbf{Block Group } \{2, 4, 8, 12, 16, 32, 64, 128, 256\} \\ \textbf{\textit{I/O Scheduler}} \{\textit{Deadline}\} \\ \textbf{\textit{Disk Type}} \{\textit{500SAS}\} \end{cases}$$

$$\mathcal{M}^1 \in \mathbb{R} = \text{Filebench throughput (I/O operations per second)}$$

$$Unknown\ model\ \rho^1 : \mathcal{C}^1 \rightarrow \mathcal{M}^1$$

We use the same map of class $(2, 1)$ used in the previous scenario to represent the transformation across the two models, as follows:

$$\mathcal{C}^0 = \textbf{Block Size}, \textbf{Inode Size}, \textbf{Block Group}, \textbf{I/O Scheduler}$$

$$\mathcal{M}^0 = \textbf{Filebench throughput}$$

$$\mathcal{L} = \textbf{\{Inode Size, Block Group\}}$$

$$\mathcal{K} = \{\mathcal{M}^0\}$$

and we follow the same experimental procedure described in Section 4.4.2 to evaluate the accuracy.

5.4.2 Results

Table 5.6 and Table 5.7 show the results of our experiments using six different learning approaches.

The results show that, even when modeling two different variations at the same time, incremental modeling and transfer learning techniques outperform direct modeling. With only five samples, the best accuracy provided by a direct model that does not exploit incremental modeling is once again substantially worse than our Model Mapping technique.

Differently from Scenario 1, the unknown model is three-dimensional, and with one fewer dimension the direct modeling methods all exhibit a lower error relative to the ones reported in Scenario 1. At the same time, modeling two simultaneous variations increases the complexity of the map. As a consequence, while all transfer learning methods are still advantageous in presence of a very small sampling budget, their advantage over direct models lessens as the sampling budget increases.

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	355.28±51.9	211.97±18.3
	Polynomial regression	223.94±6.0	130.59±3.8
	Linear SVR	227.35±8.9	139.13±5.4
	Gaussian Process	223.15±7.1	**127.22±2.9**
	Chorus*	191.12±3.1	N/A
	Multi-Task Gaussian Process*	183.47±5.0	N/A
10	Linear regression	228.64±7.7	148.65±5.1
	Polynomial regression	207.98±6.9	127.66±3.7
	Linear SVR	214.74±7.2	**125.39±4.2**
	Gaussian Process	200.59±5.4	129.62±4.4
	Chorus*	184.85±5.4	N/A
	Multi-Task Gaussian Process*	164.26±4.5	N/A

Table 5.6: RMSE mean and standard error of Model Mapping technique and direct modeling technique with various modeling methods (* indicates a direct modeling method that uses incremental or transfer learning). Legacy model: **Disk Type**=SATA, **I/O Scheduler**=CFQ. Unknown model: **Disk Type**=500SAS, **I/O Scheduler**=Deadline.

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	1.80±0.28	1.04±0.10
	Polynomial regression	1.09±0.03	0.64±0.02
	Linear SVR	1.12±0.05	0.68±0.03
	Gaussian Process	1.09±0.04	**0.62±0.02**
	Chorus*	0.89±0.01	N/A
	Multi-Task Gaussian Process*	0.90±0.03	N/A
10	Linear regression	1.12±0.04	0.69±0.03
	Polynomial regression	1.00±0.04	0.59±0.02
	Linear SVR	1.05±0.04	**0.58±0.02**
	Gaussian Process	0.96±0.03	0.63±0.02
	Chorus*	0.87±0.02	N/A
	Multi-Task Gaussian Process*	0.80±0.02	N/A

Table 5.7: MAPE mean and standard error of Model Mapping technique and direct modeling technique with various modeling methods (* indicates a direct modeling method that uses incremental or transfer learning). Legacy model: **Disk Type**=SATA, **I/O Scheduler**=CFQ. Unknown model: **Disk Type**=500SAS, **I/O Scheduler**=Deadline.

5.5 Scenario 4: Modeling Multiple Significant System Configuration Changes at Once

Similar to the previous scenario, a system administrator is interested in modeling the effects of modifying multiple system configuration parameters at the same time, to reduce the trial and error in determining the viability of a planned configuration change. For this example, the variations performed are more substantial than in Scenario 3. The existing 250GB SATA hard disk is being upgraded to a Solid-State Disk drive.

The administrator understands that overall throughput is more important than fast access and transfer of large files in a web server. For this reason, a decision is made to implement a second configuration change at the same time: modifying the filesystem I/O scheduler setting from *CFQ* to *Noop* aims to leverage the substantially faster random access properties of the Solid State Disk.

Equipped with a baseline model of the web server performance running in the original configuration, the intention is to quickly model the effects of the configuration change and tune all the remaining filesystem parameters.

5.5.1 Formulation

This scenario uses the same formulation as described in Section 5.4.1, with differences in the legacy and unknown configurations:

$$\mathcal{C}^0 \in \mathbb{R}^5 = \begin{cases} \textbf{Block Size } \{1, 2, 4\}\text{(KB)} \\ \textbf{Inode Size } \{128, 256, 512, 1024, 2048, 4096, 8192\}\text{(Bytes)} \\ \textbf{Block Group } \{2, 4, 8, 12, 16, 32, 64, 128, 256\} \\ \textbf{\textit{I/O Scheduler}} \ \{\mathit{CFQ}\} \\ \textbf{\textit{Disk Type}} \ \{\mathit{SATA}\} \end{cases}$$

$$\mathcal{C}^1 \in \mathbb{R}^5 = \begin{cases} \textbf{Block Size } \{1, 2, 4\}\text{(KB)} \\ \textbf{Inode Size } \{128, 256, 512, 1024, 2048, 4096, 8192\}\text{(Bytes)} \\ \textbf{Block Group } \{2, 4, 8, 12, 16, 32, 64, 128, 256\} \\ \textbf{\textit{I/O Scheduler}} \{\textit{Noop}\} \\ \textbf{\textit{Disk Type}} \{\textit{SSD}\} \end{cases}$$

Once again we use the map of class $(2, 1)$ described in Section 5.2.1, as follows:

$$\mathcal{C}^0 = \textbf{Block Size}, \textbf{Inode Size}, \textbf{Block Group}, \textbf{I/O Scheduler}$$

$$\mathcal{M}^0 = \textbf{Filebench throughput}$$

$$\mathcal{L} = \textbf{\{Inode Size, Block Group\}}$$

$$\mathcal{K} = \{\mathcal{M}^0\}$$

and follow the experimental procedure described in Section 4.4.2.

5.5.2 Results

Table 5.8 and Table 5.9 show the results of our experiments. We observe that the substantial variation in the system's performance between the two configurations greatly diminishes the effectiveness of incremental modeling.

The simultaneous modification of two highly sensitive parameters translates to a change that is more complex to model than the system's behavior itself. In this situation, with five samples, Polynomial Regression appears to model the unknown configuration best, although we can see that Model Mapping still performs reasonably well, offering a lower accuracy degradation relative to the other two incremental modeling and transfer learning techniques. We can also observe that Linear regression applied to the map improves substantially between

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	166.00±23.9	195.18±34.8
	Polynomial regression	**110.23±2.7**	132.57±4.9
	Linear SVR	120.59±4.6	120.03±4.8
	Gaussian Process	119.13±3.5	121.00±6.2
	Chorus*	128.05±1.7	N/A
	Multi-Task Gaussian Process*	122.21±3.5	N/A
10	Linear regression	91.74±1.6	**91.46±2.7**
	Polynomial regression	102.46±1.8	117.97±2.6
	Linear SVR	106.82±3.7	102.07±2.8
	Gaussian Process	102.64±3.2	99.68±3.5
	Chorus*	123.94±2.1	N/A
	Multi-Task Gaussian Process*	112.10±3.8	N/A

Table 5.8: RMSE mean and standard error of the proposed Model Mapping technique and direct modeling technique with various modeling methods (* indicates a direct modeling method that uses incremental or transfer learning). Legacy model: **Disk Type**=SATA, **I/O Scheduler**=CFQ. Unknown model: **Disk Type**=SSD, **I/O Scheduler**=noop.

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	0.88±0.13	0.98±0.17
	Polynomial regression	**0.57±0.02**	0.64±0.02
	Linear SVR	0.63±0.03	0.59±0.02
	Gaussian Process	0.61±0.02	0.58±0.03
	Chorus*	0.67±0.01	N/A
	Multi-Task Gaussian Process*	0.63±0.02	N/A
10	Linear regression	0.48±0.01	**0.47±0.01**
	Polynomial regression	0.54±0.01	0.58±0.01
	Linear SVR	0.56±0.02	0.51±0.01
	Gaussian Process	0.53±0.02	0.50±0.02
	Chorus*	0.65±0.01	N/A
	Multi-Task Gaussian Process*	0.58±0.02	N/A

Table 5.9: MAPE mean and standard error of the proposed Model Mapping technique and direct modeling technique with various modeling methods (* indicates a direct modeling method that uses incremental or transfer learning). Legacy model: **Disk Type**=SATA, **I/O Scheduler**=CFQ. Unknown model: **Disk Type**=SSD, **I/O Scheduler**=noop.

5 and 10 samples. We attribute this effect and the large standard error for the smaller sampling budget to the peculiar shape of this map, which causes a large error when outliers are selected among the 5 samples used to train the model.

5.6 Scenario 5: Modeling Effects of Changing Filesystem

In this scenario, a system administrator is interested in modeling the effects of changing the type of filesystem deployed in a web server platform. As new versions and revisions of filesystem software are released that increase reliability and performance, administrators need to understand the effects of the changes and perform regression tests before deployment.

For this example, the filesystem is being upgraded from ext3 to ext4, and the administrator is interested in understanding the effects of this change quickly, by running only a handful of experiments. The administrator has already obtained a model of the web server performance running on the baseline system under numerous variations of the remaining parameters.

5.6.1 Formulation

In this scenario the configuration space $\mathcal{C}^0$ is a vector of six-dimensional tuples containing all combinations of the five filesystem configuration parameters **I/O Scheduler**, **Block Size**, **Block Group**, **Inode Size** and **Disk Type**, with the fixed sixth parameter **File System** = Ext3.

$$\mathcal{C}^0 \in \mathbb{R}^6 = \begin{cases} \textbf{Disk Type} \{\text{SATA, SAS, SAS500, SSD}\} \\ \textbf{Block Size} \{1, 2, 4\}\text{(KB)} \\ \textbf{Inode Size} \{128, 256, 512, 1024, 2048, 4096, 8192\}\text{(Bytes)} \\ \textbf{Block Group} \{2, 4, 8, 12, 16, 32, 64, 128, 256\} \\ \textbf{I/O Scheduler} \{\text{Noop, CFQ, Deadline}\} \\ \textbf{\textit{File System}} \{\textit{Ext3}\} \end{cases}$$

The unknown model we are looking for covers the configuration change to a different filesystem type, as described below:

$$\mathcal{C}^0 \in \mathbb{R}^6 = \begin{cases} \textbf{Disk Type } \{\text{SATA, SAS, SAS500, SSD}\} \\ \textbf{Block Size } \{1, 2, 4\}\text{(KB)} \\ \textbf{Inode Size } \{128, 256, 512, 1024, 2048, 4096, 8192\}\text{(Bytes)} \\ \textbf{Block Group } \{2, 4, 8, 12, 16, 32, 64, 128, 256\} \\ \textbf{I/O Scheduler } \{\text{Noop, CFQ, Deadline}\} \\ \textbf{\textit{File System}} \{\textit{Ext4}\} \end{cases}$$

After verifying the parameters' sensitivities using the procedure described in Section 4.7.2, we use a five-dimensional map, which includes **Inode Size**, **Block Group**, **Disk Type** and **I/O Scheduler**, the four features with the highest sensitivity in the legacy model's domain, and a single feature from the performance function's codomain:

$$\mathcal{C}^0 = \textbf{Disk Type}, \textbf{Block Size}, \textbf{Inode Size}, \textbf{Block Group}, \textbf{I/O Scheduler}$$

$$\mathcal{M}^0 = \textbf{Filebench throughput}$$

$$\mathcal{L} = \textbf{\{Inode Size, Block Group, Disk Type, I/O Scheduler\}}$$

$$\mathcal{K} = \{\mathcal{M}^0\}$$

therefore a map of class $(|\mathcal{L}|, |\mathcal{K}|) = (4, 1)$. We proceed using the same experimental procedure described in Section 4.4.2.

5.6.2 Results

Results for the experiment are contained in Table 5.10 and Table 5.11. We observe that in this scenario Model Mapping and the other incremental modeling techniques perform effectively

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	166.87±9.2	201.65±21.8
	Polynomial regression	110.51±3.3	131.48±7.2
	Linear SVR	138.31±3.9	141.71±4.0
	Gaussian Process	98.54±2.1	**97.80±1.8**
	Chorus*	102.33±0.1	N/A
	Multi-Task Gaussian Process*	109.17±3.1	N/A
10	Linear regression	135.50±5.8	137.37±5.3
	Polynomial regression	113.72±2.1	162.25±13.3
	Linear SVR	122.00±2.9	126.72±3.6
	Gaussian Process	**92.96±1.0**	93.92±0.9
	Chorus*	102.08±0.2	N/A
	Multi-Task Gaussian Process*	101.96±2.4	N/A

Table 5.10: RMSE mean and standard error of the proposed Model Mapping technique and direct modeling technique with various modeling methods (* indicates a direct modeling method that uses incremental or transfer learning). Legacy model: **File System**=ext3. Unknown model: **File System**=ext4.

on par with direct modeling. The differences between the ext3 and ext4 filesystem behaviors are sufficient to substantially reduce the advantage of incremental modeling. At the same time, adopting incremental modeling does not lead to any loss of accuracy, which makes the approach viable even in these circumstances.

Comparing the values in the ”Polynomial regression” rows of the ”5” and ”10” Sampling Budget segments in Table 5.10 and Table 5.11, we also notice the performance of the Polynomial model worsens as the sampling budget increases. We attribute the effect to the multimodal and discontinuous behavior of both the performance function and the transformation function. With limited sampling budgets, a small increase in the number of samples results in a more than proportional increase in the complexity of the model necessary to fit the data. Further tests seem to confirm our intuition, and show that accuracy begins improving with sampling budgets larger than 30% of the total size of the dataset.

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	0.85±0.05	1.03±0.11
	Polynomial regression	0.55±0.02	0.60±0.02
	Linear SVR	0.70±0.02	0.72±0.02
	Gaussian Process	0.51±0.01	**0.50±0.01**
	Chorus*	0.54±0.00	N/A
	Multi-Task Gaussian Process*	0.56±0.01	N/A
10	Linear regression	0.68±0.03	0.69±0.03
	Polynomial regression	0.55±0.01	0.64±0.02
	Linear SVR	0.61±0.02	0.64±0.02
	Gaussian Process	**0.48±0.00**	**0.48±0.00**
	Chorus*	0.54±0.00	N/A
	Multi-Task Gaussian Process*	0.52±0.01	N/A

Table 5.11: MAPE mean and standard error of the proposed Model Mapping technique and direct modeling technique with various modeling methods (* indicates a direct modeling method that uses incremental or transfer learning). Legacy model: **File System**=ext3. Unknown model: **File System**=ext4.

5.7 Conclusions

A summary of the results is present in Table 5.12, as a percentage improvement in RMSE our technique has over both direct modeling and incremental/transfer learning approaches.

	Incremental variation	ΔErr Direct	ΔErr baseline T.Learning
Scenario 1	Disk Type (SATA→500SAS)	-40.61%	-19.4%
Scenario 2	Disk Type (SATA→SSD)	-0.38%	-11.35%
Scenario 3	Disk Type (SATA→500SAS) I/O Scheduler (CFQ→Deadline)	-42.99%	-30.66%
Scenario 4	Disk Type (SATA→SSD) I/O Scheduler (CFQ→Noop)	**+8.89%**	-1.78%
Scenario 5	Filesystem (ext3→ext4)	-0.75%	-4.43%

Table 5.12: Summary of storage system performance scenarios, %reduction in RMSE of Model Mapping vs. Direct Modeling and baseline Transfer Learning (Multi-Task Gaussian Process / Chorus). Sampling budget = 5.

The results of our experiments show that Model Mapping is generally effective in accurately modeling different incremental variations that emerge in file storage systems as part of

their regular maintenance and performance tuning. The five scenarios represent a system administrator's goal to rapidly experiment with changes in hardware to leverage new technology to quickly and efficiently adapt the system configuration to these variations. In most of these circumstances our technique outperforms both traditional direct modeling methods and other incremental learning methods. In some scenarios, where a configuration variation results in substantial difference of system behavior, Model Mapping is less beneficial, but rarely worse than direct modeling.

As previously discussed in Section 4.9, we report the success rate of applying Model Mapping over direct modeling, using the different modeling methods. Table 5.13 and Table 5.14 show that Model Mapping improves over direct modeling in most scenarios.

We conclude that Model Mapping is a generally applicable technique to the problem of incremental file system storage performance modeling, affording a varying degree of advantage based on the significance of the incremental variation.

	Experiments	**Success Rate**
Linear regression	7/10	70%
Polynomial regression	5/10	50%
Linear SVR	8/10	80%
Gaussian Process	7/10	70%

Table 5.13: Success rate of Model Mapping vs. direct modeling, with different modeling methods (RMSE).

	Experiments	**Success Rate**
Linear regression	7/10	70%
Polynomial regression	5/10	50%
Linear SVR	8/10	80%
Gaussian Process	10/10	100%

Table 5.14: Success rate of Model Mapping vs. direct modeling, with different modeling methods (MAPE).

Chapter 6

Service Performance Modeling

6.1 Introduction

As we discussed in the previous chapters, in cloud systems it is important to accurately model the impact of resource availability on fundamental hosted platform services such as databases and file storage systems. Several large hosted applications are composed of a variety of other services that are invoked as a response to user demands [60, 104, 113]. These services are highly configurable and extremely heterogeneous in nature, and range from micro-databases to audio processing and video transcoding/streaming; their behaviors and resource demands are substantially different. It is therefore very important to model their performance, as the hardware platforms, operating systems, and other platform services change. To further improve our understanding of system performance modeling, and in order to study the broader applicability of our technique to performance problems, we turned our attention to these types of application services.

After our work on the *ModelMap* -toolkit was completed, we discovered the work of Valov et al. [116], who analyzed the accuracy of applying linear models to efficiently transfer application performance models across hardware changes. Their approach is similar to ours, as it is effectively equivalent to using a linear model to represent a map of class $(0, 1)$, and using a

subset of the available legacy data. While we observed in the other chapters that linear models are insufficient to represent several system and application level configurations, we investigate applying our technique to the three scenarios reported by the authors. For our experiments we use the three datasets Valov et al. released publicly as part of their research [117].

As reported in the previous chapter, our *ModelMap* toolkit does not currently support non-numerical values, so we performed a translation of the datasets using the following conversion table:

- **X.bail**: TRUE=1, FALSE=0
- **X.stats**: TRUE=1, FALSE=0
- **synchronous**: ON=1, OFF=0

The three scenarios we report represent realistic hardware changes that a cloud system may incur over its lifetime. The scenarios provide insight in the efficiency of our performance modeling technique relative to other techniques, including the linear model transfer proposed by Valov et al.

Similar to the previous chapters, in each of the three scenarios we verify the accuracy of modeling the system behavior as a consequence of a modification in its configuration. The scenarios involve modeling the performance of three configurable applications/services that exhibit different performance profiles and resource usage patterns, when the underlying hardware platform changes.

For each scenario, the dataset represents measurements of an application's performance profile as a function of its configuration parameters. The three applications are: *x264* [121], *XZ* [102], and SQLite [94]. x264 is a popular tool for compressing and streaming digital video using the H.264/MPEG-4 AVC video compression standard [49]. XZ is a general-purpose data compression tool that relies on the LZMA2 compression method [83]. SQLite is a library for integrating a very compact, file-oriented SQL database into applications.

Differently from the previous chapters, in all the following scenarios we include the linear transfer function method outlined by Valov et al. (which we denote "Linear model transfer" in

the tables) as a baseline for comparison among transfer learning techniques, and we represent it in the results as a form of mapping. The results we report for the ”Linear model transfer” technique are based on our own implementation in the ModelMap toolkit.

6.2 Scenario 1: Modeling Effects of a Hardware Change on a Digital Video Encoding Service

As reported earlier, the scenarios involve a system administrator who intends to understand the effects of upgrading the hardware platform serving a number of different services, a typical scenario in both traditional computing clusters and larger managed/hosted systems. In this scenario, the service under consideration is *x264*, a digital video encoding/decoding/transcoding tool that uses the H.264/MPEG-4 AVC video compression standard.

The administrator starts the investigation with a model of the service performance on the current hardware platform, and intends to obtain an accurate model of the service after the hardware upgrade as efficiently as possible. The purpose of obtaining the new model is to judge whether the new platform is yielding the necessary performance increase, and whether a variation in the service configuration would be advisable as a consequence of the platform upgrade.

This example is intended to demonstrate how, with a few samples, we can obtain a model that covers an incremental change in the system’s hardware specifications.

6.2.1 Formulation

The x264 application has several configuration options, which the dataset authors converted to a set of 11 binary variables to express the configuration space. When translated to a set of binary features, some features are mutually exclusive, therefore the configuration space is not 2^{11}.

The authors did not exhaustively sample the configuration space, but instead selected 165 configurations and measured the application performance in those. The x264 performance dataset therefore consists of measurements of the application's performance (wall clock time to completion in seconds), by setting the system's configuration parameters to 165 possible parameters combinations, and performing the experiments on 11 hardware platforms [116], for a total of 1815 samples.

The legacy model ρ^0 in this scenario was built using a subset of the x264 dataset. The configuration space $\mathcal{C}^0$ is a vector of twelve-dimensional tuples containing 165 combinations of the eleven binary parameters **NO_DEBLOCK, NO_FAST_PSKIP, NO_MBTREE, NO_MIXED_REFS, NO_WEIGHTB, –rc-lookahead-20, –rc-lookahead-40, –rc-lookahead-60, A, B** and **C**, with the fixed twelfth parameter **Worker ID** = 75.

$$\mathcal{C}^0 \in \mathbb{R}^{12} = \begin{cases} \textbf{NO\_DEBLOCK} \ \{0, 1\} \\ \textbf{NO\_FAST\_PSKIP} \ \{0, 1\} \\ \textbf{NO\_MBTREE} \ \{0, 1\} \\ \textbf{NO\_MIXED\_REFS} \ \{0, 1\} \\ \textbf{NO\_WEIGHTB} \ \{0, 1\} \\ \textbf{–rc-lookahead-20} \ \{0, 1\} \\ \textbf{–rc-lookahead-40} \ \{0, 1\} \\ \textbf{–rc-lookahead-60} \ \{0, 1\} \\ \textbf{A} \ \{0, 1\} \\ \textbf{B} \ \{0, 1\} \\ \textbf{C} \ \{0, 1\} \\ \textbf{\textit{Worker ID}} \ \{75\} \end{cases}$$

We extract from the dataset a vector $\mathcal{M}^0$, where the entries contain the result of measuring

the x264 application performance using the system configuration parameters of the corresponding entry of $\mathcal{C}^0$.

$$\mathcal{M}^0 \in \mathbb{R} = \text{x264 wall clock time (seconds)}$$

$$Legacy\ model\ \rho^0 : \mathcal{C}^0 \rightarrow \mathcal{M}^0$$

The model of the new configuration we intend to obtain extends the legacy model with one extra dimension in the configuration space, allowing for the representation of the hardware platform change, a new value of the **Worker ID** parameter.

$$\mathcal{C}^1 \in \mathbb{R}^5 = \begin{cases} \textbf{NO\_DEBLOCK}\ \{0, 1\} \\ \textbf{NO\_FAST\_PSKIP}\ \{0, 1\} \\ \textbf{NO\_MBTREE}\ \{0, 1\} \\ \textbf{NO\_MIXED\_REFS}\ \{0, 1\} \\ \textbf{NO\_WEIGHTB}\ \{0, 1\} \\ \textbf{–rc-lookahead-20}\ \{0, 1\} \\ \textbf{–rc-lookahead-40}\ \{0, 1\} \\ \textbf{–rc-lookahead-60}\ \{0, 1\} \\ \textbf{A}\ \{0, 1\} \\ \textbf{B}\ \{0, 1\} \\ \textbf{C}\ \{0, 1\} \\ \textbf{\textit{Worker ID}}\ \{81\} \end{cases}$$

$$\mathcal{M}^1 \in \mathbb{R} = \text{x264 wall clock time (seconds)}$$

$$Unknown\ model\ \rho^1 : \mathcal{C}^1 \rightarrow \mathcal{M}^1$$

The variation from **Worker ID** 75 to 81 corresponds to a significant modification of the

underlying hardware platform, an increase of CPU cores from 2 to 16, a reduction of CPU core frequency of 800MHz and a 32-fold increase in system memory.

We use a two-dimensional map to represent the transformation function between the two models. After conducting a feature sensitivity analysis, as described in Section 4.7.2, we included the **NO_MBTREE** feature from the legacy model's domain, and the only available feature from the model's codomain:

$$\begin{aligned}\mathcal{C}^0 = \ & \textbf{NO\_DEBLOCK}, \textbf{NO\_FAST\_PSKIP}, \textbf{NO\_MBTREE}, \textbf{NO\_MIXED\_REFS}, \\ & \textbf{NO\_WEIGHTB}, \textbf{–rc-lookahead-20}, \textbf{–rc-lookahead-40}, \textbf{–rc-lookahead-60}, \\ & \textbf{A}, \textbf{B}, \textbf{C}\end{aligned}$$

$$\mathcal{M}^0 = \textbf{x264 wall clock time}$$

$$\mathcal{L} = \{\textbf{NO\_MBTREE}\}$$

$$\mathcal{K} = \{\mathcal{M}^0\}$$

The map is therefore of class $(|\mathcal{L}|, |\mathcal{K}|) = (1, 1)$. Following the formulation described in Section 4.4.1, we train a model of the two-dimensional transfer function $\sigma : \mathcal{M}^0 \rightarrow \mathcal{M}^1$ by using the legacy model and the few samples obtained from measuring the system in the new configurations selected from $\mathcal{C}^1$. We apply the experimental procedure described in Section 4.4.2 to iteratively select sets of samples, perform cross-validation, and measure the accuracy of all the selected modeling methods.

6.2.2 Results

Results for the experiments are contained in Table 6.1 and Table 6.2. We can observe that transfer learning is effective in this scenario, where the models that do not leverage prior knowledge

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	1.98±0.1	0.66±0.1
	Polynomial regression	2.22±0.1	0.76±0.1
	Linear SVR	1.69±0.1	**0.55±0.0**
	Gaussian Process	2.05±0.1	0.55±0.0
	Chorus*	0.63±0.0	N/A
	Multi-Task Gaussian Process*	0.77±0.1	N/A
	Linear model transfer*	N/A	0.81±0.0
10	Linear regression	2.45±0.3	**0.42±0.0**
	Polynomial regression	1.90±0.0	0.68±0.1
	Linear SVR	1.71±0.1	0.45±0.0
	Gaussian Process	1.42±0.1	0.49±0.0
	Chorus*	0.63±0.0	N/A
	Multi-Task Gaussian Process*	0.51±0.0	N/A
	Linear model transfer*	N/A	0.71±0.0

Table 6.1: RMSE mean and standard error of the proposed Model Mapping technique and direct modeling technique with various modeling methods (* indicates a direct modeling method that uses incremental or transfer learning). Application: x264. Legacy model: **Worker ID**=75. Unknown model: **Worker ID**=81.

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	30.03±1.6	9.81±1.8
	Polynomial regression	35.34±1.5	9.38±0.5
	Linear SVR	25.37±1.1	**8.21±0.5**
	Gaussian Process	32.26±1.2	8.43±0.3
	Chorus*	10.15±0.0	N/A
	Multi-Task Gaussian Process*	11.05±0.6	N/A
	Linear model transfer*	N/A	12.42±0.5
10	Linear regression	36.36±5.0	**6.05±0.2**
	Polynomial regression	29.53±1.1	7.69±0.3
	Linear SVR	25.11±1.2	6.31±0.2
	Gaussian Process	20.75±1.1	7.30±0.3
	Chorus*	10.10±0.0	N/A
	Multi-Task Gaussian Process*	7.42±0.4	N/A
	Linear model transfer*	N/A	11.09±0.2

Table 6.2: MAPE mean and standard error of the proposed Model Mapping technique and direct modeling technique with various modeling methods (* indicates a direct modeling method that uses incremental or transfer learning). Application: x264. Legacy model: **Worker ID**=75. Unknown model: **Worker ID**=81.

show considerably lower accuracy. In particular, comparing the values in the ”Direct” Modeling Technique column, ”Linear regression” rows of the ”5” and ”10” Sampling Budget segments in Table 6.2, we can see that the Linear regression model exhibits a degradation of performance when increasing the sampling budget from 5 to 10 ($30.03 \pm 1.6 \rightarrow 36.36 \pm 5.0$). We attribute the error increase to the strongly multi-modal behavior of this system. Increasing the sampling budget also increases the non-linearity of the dataset and exposes the limitations of the linear model.

Among the transfer learning methods, we see how using a simple linear model to transfer information (”Linear model transfer”) shows a substantial improvement over direct modeling using conventional techniques, which denotes how these hardware platform changes produce a mostly linear variation in application performance. More sophisticated transfer learning techniques perform better, and offer a more substantial increase in accuracy. In particular, we observe that Model Mapping is the most effective among them, by using an expressive model to represent the transfer function and by carefully selecting important features from the original information.

6.3 Scenario 2: Modeling Effects of a Hardware Change on a Data Compression Service

Similar to the previous scenario, we are interested in understanding the effects of upgrading the hardware platform hosting an application or a service. In this scenario, we consider *XZ*, a general-purpose data compression library/utility that leverages the LZMA2 lossless compression method.

We investigate a similar change in the underlying hardware with the purpose of understanding the ability of transfer learning methods to efficiently model an incremental change in the system’s hardware specifications.

6.3.1 Formulation

The XZ application is highly configurable, and the dataset authors converted all the available parameters into a set of 16 binary variables to express the configuration space. Some of the binary variables originate from multiple settings of individual numerical or categorical configuration parameters, therefore the configuration space is substantially smaller than 2^{16}.

As with the previous dataset, the authors did not sample the entire configuration space, but instead selected a subset of 154 representative configurations for the purpose of performance measurement. The XZ performance dataset thus consists of measurements of the application's performance (wall clock time to completion in seconds), by setting the system's configuration parameters to 154 parameters combinations, and the measurements of those 154 configurations were performed on 7 different hardware platforms [116], for a total of 1078 samples.

The legacy model ρ^0 in this scenario was built using a subset of the XZ dataset. The configuration space $\mathcal{C}^0$ is a vector of seventeen-dimensional tuples containing 154 combinations of the sixteen binary parameters **extreme**, **sparse**, **crc32**, **crc64**, **none**, **sha256**, **-9**, **-8**, **-7**, **-6**, **-5**, **-4**, **-3**, **-2**, **-1** and **0**, with the fixed seventeenth parameter **Worker ID** = 75, representing one hardware platform.

$$\mathcal{C}^0 \in \mathbb{R}^{17} = \begin{cases} \textbf{extreme}\ \{0,1\} & \textbf{-9}\ \{0,1\} & \textbf{-3}\ \{0,1\} \\ \textbf{sparse}\ \{0,1\} & \textbf{-8}\ \{0,1\} & \textbf{-2}\ \{0,1\} \\ \textbf{crc32}\ \{0,1\} & \textbf{-7}\ \{0,1\} & \textbf{-1}\ \{0,1\} \\ \textbf{crc64}\ \{0,1\} & \textbf{-6}\ \{0,1\} & \textbf{0}\ \{0,1\} \\ \textbf{none}\ \{0,1\} & \textbf{-5}\ \{0,1\} & \textbf{\textit{Worker ID}}\ \{75\} \\ \textbf{sha256}\ \{0,1\} & \textbf{-4}\ \{0,1\} & \end{cases}$$

As with the previous scenario, the vector $\mathcal{M}^0$ represents the application performance measurements, as wall clock time measured in seconds, and the new configuration space $\mathcal{C}^1$ involves a different hardware platform, which we represent as a new value for the **Worker ID** parameter, which is set to the value of 80.

The variation from **Worker ID** 75 to 80 is a substantial change in the underlying hardware platform, an increase of CPU cores from 2 to 8, an increase in CPU core frequency of 200MHz

and a 8-fold increase in system memory.

For this scenario we use a three-dimensional map to represent the transformation function between the two models, as follows:

$$\mathcal{C}^0 = \textbf{extreme}, \textbf{sparse}, \textbf{crc32}, \textbf{crc64}, \textbf{none}, \textbf{sha256}, \textbf{-9}, \textbf{-8}, \textbf{-7}, \textbf{-6}, \textbf{-5}, \textbf{-4}, \textbf{-3}, \textbf{-2}, \textbf{-1}, \textbf{0}$$

$$\mathcal{M}^0 = \textbf{XZ wall clock time}$$

$$\mathcal{L} = \textbf{\{0, -3\}}$$

$$\mathcal{K} = \{\mathcal{M}^0\}$$

therefore a map of class $(2, 1)$. We follow the formulation described in Section 4.4.1 and the experimental procedure described in Section 4.4.2.

6.3.2 Results

Results for the experiments are contained in Table 6.3 and Table 6.4. We can observe that in this scenario the difference between direct modeling and incremental modeling/transfer learning is even more substantial, with a difference in accuracy by up to an order of magnitude in favor of the latter.

Once again, we see how the system configuration changes considered in these scenarios result in mostly linear variation in the performance of the considered application. The results show that for a very limited sampling budget, Chorus and Model Mapping both perform well in this scenario.

While a generic linear model describes well the relationship between these configurations, increasing the features available to the model has a significant impact on accuracy when the available sampling budget is very small.

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	9.74±0.3	0.74±0.1
	Polynomial regression	11.20±0.2	0.66±0.0
	Linear SVR	8.44±0.2	0.72±0.1
	Gaussian Process	10.44±0.2	**0.57±0.0**
	Chorus*	0.58±0.0	N/A
	Multi-Task Gaussian Process*	0.83±0.1	N/A
	Linear model transfer*	N/A	0.79±0.1
10	Linear regression	5.78±0.4	0.55±0.0
	Polynomial regression	10.57±0.1	0.49±0.0
	Linear SVR	5.49±0.3	0.56±0.0
	Gaussian Process	9.24±0.2	**0.45±0.0**
	Chorus*	0.58±0.0	N/A
	Multi-Task Gaussian Process*	0.89±0.1	N/A
	Linear model transfer*	N/A	0.64±0.0

Table 6.3: RMSE mean and standard error of the proposed Model Mapping technique and direct modeling technique with various modeling methods (* indicates a direct modeling method that uses incremental or transfer learning). Application: XZ. Legacy model: **Worker ID**=75. Unknown model: **Worker ID**=80.

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	101.44±6.3	6.27±1.0
	Polynomial regression	119.95±6.3	6.19±0.5
	Linear SVR	88.01±3.1	6.07±0.8
	Gaussian Process	118.49±4.6	**4.46±0.2**
	Chorus*	4.97±0.0	N/A
	Multi-Task Gaussian Process*	7.37±1.4	N/A
	Linear model transfer	N/A	7.30±1.2
10	Linear regression	55.17±4.6	4.24±0.2
	Polynomial regression	111.10±4.9	4.57±0.4
	Linear SVR	52.25±4.0	4.14±0.2
	Gaussian Process	100.70±4.3	**3.42±0.2**
	Chorus*	4.94±0.0	N/A
	Multi-Task Gaussian Process*	7.77±1.3	N/A
	Linear model transfer*	N/A	5.52±0.1

Table 6.4: MAPE mean and standard error of the proposed Model Mapping technique and direct modeling technique with various modeling methods (* indicates a direct modeling method that uses incremental or transfer learning). Application: XZ. Legacy model: **Worker ID**=75. Unknown model: **Worker ID**=80.

6.4 Scenario 3: Modeling Effects of a Hardware Change on a Filesystem-backed Database Service

As in the previous two scenarios, we are interested in modeling the performance of an application as the underlying hardware is changed.

For this experiment we investigate *SQLite*, a library for embedding a SQL database engine on applications backed by simple filesystems.

The goal is still to understand the potential advantages of reusing legacy information for the purpose of improving the accuracy of the performance models after the configuration change has been applied.

6.4.1 Formulation

The SQLite database engine exposes a mix of numerical and boolean configuration parameters, and the dataset authors explored multiple variations of five such parameters for the purpose of gathering performance data.

The SQLite performance dataset thus consists of measurements of the application's performance (wall clock time to completion in seconds), by setting the system's configuration parameters to 32 parameters combinations, and the sampling was performed on 10 different hardware platforms [116], for a total of 320 samples.

The legacy model ρ^0 in this scenario was built using a subset of the SQLite dataset. In particular, the configuration space $\mathcal{C}^0$ is a vector of six-dimensional tuples containing 32 combinations of the configuration parameters **X.bail**, **X.stats**, **mmap_size**, **page_size**, **synchronous**, with the fixed sixth parameter **Worker ID** = 75, representing one hardware platform.

Using the conversion table described in Section 6.1, the configuration space is therefore as follows:

$$\mathcal{C}^0 \in \mathbb{R}^6 = \begin{cases} \textbf{X.bail}\ \{0, 1\} \\ \textbf{X.stats}\ \{0, 1\} \\ \textbf{mmap\_size}\ \{0, 268435456\} \\ \textbf{page\_size}\ \{512, 65536\} \\ \textbf{synchronous}\ \{0, 1\} \\ \textbf{\textit{Worker ID}}\ \{75\} \end{cases}$$

Once again, the vector $\mathcal{M}^0$ represents the application performance measurements, as wall clock time measured in seconds, and the new configuration space $\mathcal{C}^1$ represents a different underlying hardware, represented as a new value for the **Worker ID** parameter, set to 157.

The variation from **Worker ID** 75 to 157 represents a significant change in the underlying hardware platform, an increase of CPU cores from 2 to 36, an decrease in CPU core frequency of 900MHz and a 16-fold increase in system memory.

For this scenario we use a three-dimensional map to represent the transformation function between the two models, as follows:

$$\mathcal{C}^0 = \textbf{X.bail}, \textbf{X.stats}, \textbf{mmap\_size}, \textbf{page\_size}, \textbf{synchronous}$$

$$\mathcal{M}^0 = \textbf{SQLite wall clock time}$$

$$\mathcal{L} = \{\textbf{X.stats}, \textbf{page\_size}\}$$

$$\mathcal{K} = \{\mathcal{M}^0\}$$

therefore a map of class $(2, 1)$. We follow the formulation described in Section 4.4.1 and the experimental procedure described in Section 4.4.2.

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	6.86±0.9	6.69±4.3
	Polynomial regression	13.84±0.4	1.75±0.1
	Linear SVR	5.60±0.5	1.73±0.1
	Gaussian Process	12.22±0.4	1.58±0.0
	Chorus*	**1.45±0.0**	N/A
	Multi-Task Gaussian Process*	3.67±1.2	N/A
	Linear model transfer*	N/A	2.54±0.6
10	Linear regression	1.90±0.1	1.68±0.1
	Polynomial regression	11.28±0.3	1.66±0.0
	Linear SVR	2.12±0.1	1.55±0.0
	Gaussian Process	2.11±0.1	1.55±0.0
	Chorus*	**1.47±0.0**	N/A
	Multi-Task Gaussian Process*	2.07±0.1	N/A
	Linear model transfer*	N/A	1.66±0.0

Table 6.5: RMSE mean and standard error of the proposed Model Mapping technique and direct modeling technique with various modeling methods (* indicates a direct modeling method that uses incremental or transfer learning). Application: SQLite. Legacy model: **Worker ID**=75. Unknown model: **Worker ID**=157.

6.4.2 Results

Results for the experiments are contained in Table 6.5 and Table 6.6. We can observe that in this scenario, Chorus, Linear model transfer and Model Mapping all perform very well, substantially improving on the accuracy of direct modeling.

Observing the dataset in Figure 6.1, we can see that one feature (X.stats) has by far the highest sensitivity, and the sharp discontinuity in the performance function caused by its two states privileges Chorus' ability to segment the configuration space into separate regions.

6.5 Conclusions

A summary of the results is present in Table 6.7, as a percentage improvement in RMSE our technique has over both direct modeling and incremental/transfer learning approaches.

The results of these scenarios show the benefits of using transfer learning in the context of

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	13.86±1.9	10.78±5.8
	Polynomial regression	29.44±1.1	3.57±0.1
	Linear SVR	11.39±1.2	3.56±0.2
	Gaussian Process	27.45±1.1	3.25±0.1
	Chorus*	**3.03±0.0**	N/A
	Multi-Task Gaussian Process*	8.01±3.2	N/A
	Linear model transfer*	N/A	5.37±1.5
10	Linear regression	3.78±0.2	3.39±0.1
	Polynomial regression	24.26±0.6	3.39±0.1
	Linear SVR	4.19±0.2	3.16±0.1
	Gaussian Process	4.25±0.2	3.21±0.1
	Chorus*	**3.04±0.0**	N/A
	Multi-Task Gaussian Process*	4.17±0.2	N/A
	Linear model transfer*	N/A	3.43±0.1

Table 6.6: MAPE mean and standard error of the proposed Model Mapping technique and direct modeling technique with various modeling methods (* indicates a direct modeling method that uses incremental or transfer learning). Application: SQLite. Legacy model: **Worker ID**=75. Unknown model: **Worker ID**=157.

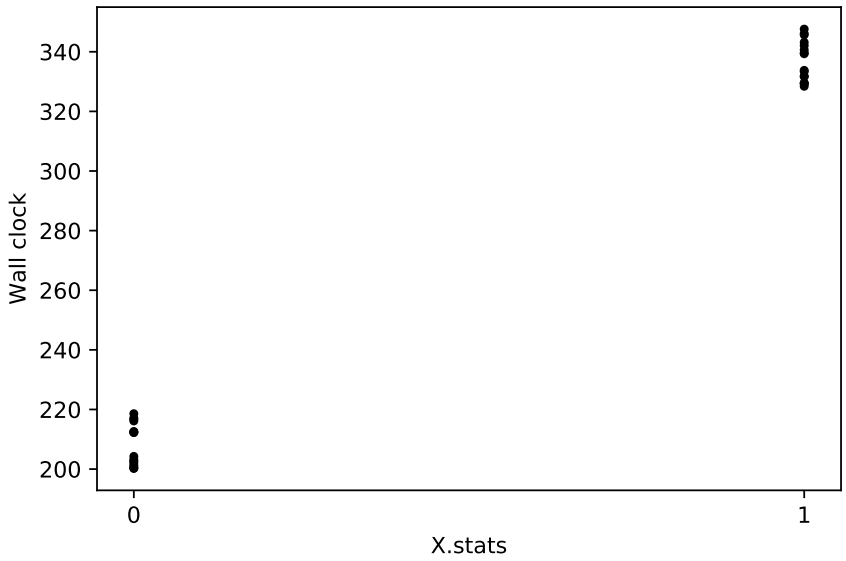


Figure 6.1: Graph of SQLite performance relative to **X.stats** configuration parameter.

	Incremental variation	ΔErr Direct	ΔErr Linear Transf.	ΔErr baseline T.Learning
Scenario 1	Worker ID (75→81)	-67.46%	-32.1%	-12.7%
Scenario 2	Worker ID (75→80)	-93.25%	-27.85%	-1.72%
Scenario 3	Worker ID (75→157)	-71.79%	-37.8%	**+8.97**%

Table 6.7: Summary of application/service performance scenarios. %reduction in RMSE of Model Mapping vs. Direct Modeling, Linear model transfer and baseline Transfer Learning (Multi-Task Gaussian Process / Chorus). Sampling budget = 5.

modeling applications and services performance in the presence of system hardware changes, which is a recurrent circumstance in the maintenance of hosted systems. In all the scenarios, we observed that an incremental upgrade in the hardware specifications has a predominantly linear effect on the applications under scrutiny. While this relationship may be modeled explicitly by applying a linear or polynomial model as a transfer function, we noted that Model Mapping, by explicitly considering significant features in the legacy data, and using more expressive modeling methods such as SVR to represent the transfer function, yields considerably better results, capturing subtle nuances of the performance variations.

Again, as previously reported in Section 4.9, we report the success rate of applying Model Mapping over direct modeling, using the different modeling methods. Table 6.8 and Table 6.9 show that Model Mapping improves over direct modeling in all the experiments in this scenario, with a 100% success rate.

We conclude that Model Mapping is a generally applicable technique to the problem of incremental hosted applications and services performance modeling, with a substantial accuracy advantage over traditional modeling approaches, and generally outperforming all other incremental and transfer learning methods we studied.

	Experiments	Success Rate
Linear regression	6/6	100%
Polynomial regression	6/6	100%
Linear SVR	6/6	100%
Gaussian Process	6/6	100%

Table 6.8: Success rate of Model Mapping vs. direct modeling, with different modeling methods (RMSE).

	Experiments	Success Rate
Linear regression	6/6	100%
Polynomial regression	6/6	100%
Linear SVR	6/6	100%
Gaussian Process	6/6	100%

Table 6.9: Success rate of Model Mapping vs. direct modeling, with different modeling methods (MAPE).

Chapter 7

Related Work

System performance modeling is a vast field and many techniques have been applied to this task in literature, each offering a compromise between ease of deployment, usage complexity, data availability requirements, level of operator involvement, etc.

In order to appropriately compare and relate our contributions to the extensive work present in this field, we classify existing approaches according to their characteristics. While generally the distinction between modeling approaches is made from the perspective of model interpretability vs. model accuracy [59], we adopt a classification that uses availability of information as the main discriminant. In particular, in this chapter we first introduce prior work according to how each approach leverages a-priori information about a system, and then we provide context for our method in relation to the state-of-the-art in its category. From this perspective, the main distinction between approaches stems from the amount of information available when a performance model is built.

At one end of the spectrum, in circumstances where very detailed information of a system's design and implementation are available, and interactions between parameters are known to be clearly observable and readily explainable, analytical or simple regression models are used, also referred to as *white-box* models.

At the opposite end of the spectrum, in situations where little to no information is available

about a system, and more complex, non-linear interactions exist between parameters, data-driven models that rely purely on observations, otherwise defined as *black-box* models, are used.

A third category of hybrid approaches also exists, based on a combination of partial a-priori information, such as parameter correlation and smoothness characteristics of the modeled function's codomain, and data. Models with these characteristics are referred to as *grey-box*.

In this framework, we consider Transfer Learning as a type of black-box modeling approach, as modeling techniques in this category may leverage existing information in the form of data or parametric models, but they generally rely on further observations to formulate the relation between existing and new information.

In the following sections, we provide an overview of the different categories of modeling techniques according to our classification, providing for each category examples of its application to the field of system performance modeling.

7.1 White-box Models

A white-box model is generally built from first principles, and ideally it either describes a system by providing a closed form mathematical representation of its behavior, or it explains a system's behavior using one or more simple, observable relationships, such as linear models or decision trees [82]. For example, the designers of a system, using intimate knowledge of its characteristics, may build a performance model that satisfies the theoretical system behavior according to its specifications. If an accurate analytical model of a system can be derived, it is preferred to any other models of the system, as it is generally extremely efficient to evaluate and requires no direct observations of a system in action.

As an example of analytical models, Baghsorkhi et al. [7] propose an analytical model of GPU performance, to provide performance information to an optimizing compiler as a means to producing efficient code.

Several examples of advanced white-box models applied to system performance modeling exist in literature. Uysal et al. formulate an analytical model to predict the throughput of disk arrays [115], reaching a level of accuracy that was deemed to be sufficient for the purpose of resource provisioning. Elnikety et al. [30] build an analytical model to predict application workload scalability on a replicated database system, and tune the parameters of their model by sampling performance metrics of a workload running on a standalone system. The limitation of this approach is in its strong reliance on the data access distribution, which is only satisfied in rare circumstances in real-world applications.

Multi-class analytic queuing models [10, 114] have also been explored for resource control on CPU-bound multi-tier web servers and general hosted applications. These techniques use analytical models to accurately perform demand profiling at each service tier, and in turn appropriately balance resource usage. This approach is effective only in circumstances where demand follows specific patterns, and does not capture workloads that deviate from the established patterns, such as in presence of I/O intensive operations.

Building white-box models requires substantial, in-depth understanding of the underlying system, which may not always be available with complex heterogeneous environments. Additionally, a system's behavior may be governed by a very large set of parameters with substantial dependencies, which would require compromising modeling accuracy in order to achieve the level of interpretability that makes using a white-box model a reasonable choice.

In all these circumstances, machine learning approaches based on modeling a system's behavior solely based on observations of the system in action constitute a powerful alternative to white-box models.

7.2 Black-box Models

Obtaining a white-box model of a system is a challenging task for many practical applications, and especially for complex systems with many parameters, due to the explicit need to capture

analytical information or to represent complex interactions among the parameters. To build a performance model of such systems, purely sampling-based black-box methods are more practical.

Real-world cloud systems and applications may exhibit a considerable number of configuration parameters and parameters may have broad ranges, therefore it would require a prohibitively long time to exhaustively sample them [93]. In black-box methods, the performance model of a system is thus created by tuning the parameters of a mathematical model to fit a set of observation samples, which are tuples of input parameters and observable characteristics of the system's behavior. Different machine learning techniques have been applied to provide effective approximations for a variety of system performance modeling tasks.

Ganapathi et al. [37] use Kernel Canonical Correlation Analysis (KCCA) [57] to model database query performance characteristics, such as elapsed time, record used, and disk IO. In their method, instead of building a cost model for each metric, they extract high level query features and use them as inputs to KCCA to model all required performance characteristics directly from the queries themselves. The authors show that in this application, KCCA can predict the overall execution time of TPC-DS [110] test database queries with considerable accuracy. The authors also report that their method can be applied to other tasks, such as MapReduce [26], by customizing the feature vectors, and in subsequent work they evaluate the performance of their technique in this scenario [36].

Soror et al. [91] build a cost model of a workload for a given resource allocation configuration by sampling various configuration parameters and use this model to estimate an initial allocation of CPU resources to virtual machines. Gambi et al. [34] introduce a model-driven approach to automatically engineer an application-specific cloud controller. Leveraging multi-dimensional Kriging models [97], the authors approximate the performance profile of applications through a utility function, which is then used to support the controller's decision-making process.

Wang et al. [123] use a different machine learning model, CART [16], to predict the per-

formance of a storage device. The authors improve on prior work by Anderson [2], which relies on tables that contain pre-sampled performance data, defined as the response time as a function of input workloads. Building the performance tables is very time consuming, due to the amount of sampling required, and the models are closely tied to the definitions of the input workloads themselves. CART is used to interpolate across tables, resulting in a model that better adapts to variations in the workloads.

Soundararajan et al. [92] propose a method for dynamic partitioning of resource utilization, based on latency models of multiple applications concurrently accessing a shared database system. The authors train a set of support vector machine regression (SVR) models that approximate the latency of each application running in isolation under a variety of database cache configurations, which they refer to as *application surfaces*. The performance models are then used to find an optimal partitioning of the database cache among a group of applications, using a greedy search procedure. While the approach shows good accuracy, when the runtime system detects significant discrepancies between the expected application behavior and the actual performance measurements, the application surfaces are rebuilt from scratch, which may incur in a considerable delay.

Duan et al. [29] propose iTuned, a system designed to tune a database configuration and optimize its performance. iTuned uses Gaussian Process Regression (GPR) to represent the response surface of the database, defined as the average running time of a given workload as a function of the database configuration parameters. The authors initialize the GPR by employing latin hypercube sampling, and then proceed to use an adaptive sampling method, which automatically selects sample points based on the Gaussian Process ability to efficiently calculate the points that maximize expected improvements in the model uncertainty. The results are favorable when compared with other methods, although any variation of additional system parameters or any latent change in the system's behavior may invalidate the model's accuracy and the ability of the optimization method to tune the system's configuration.

7.3 Grey-box Models

In specific circumstances, when individual components of a system exhibit a predictable and clearly explainable behavior, but the components exhibit a complex interaction to produce an aggregated system performance, it is possible to use hybrid, grey-box models.

Grey-box models leverage both partial, a-priori information about a system's behavior and additional, experimentally measured data to refine the accuracy of the overall models.

For example, Thereska and Ganger [103] analyze why and how performance models get out-of-sync with the system over time. Through case studies with models for common resources found in distributed systems, they find that traditional models are brittle because they assume idealized system-workload interactions. The authors show that, in most real-world scenarios, both the system itself and its models are often erratic or misconfigured, and unanticipated behavior is the norm rather than the exception. To address this problem, they propose IRONModel, a performance modeling framework which uses a series of expectation-based analytical models, and decision tree-based machine learning model (Z-CART) to tune the parameters of the analytical models at runtime.

The authors apply the framework to model a storage system, by constructing a set of analytical models for the fundamental subsystems: a CPU model, a network model, a buffer pool model and a disk model. While these individual models on their own are not accurate due to various factors such as the complexity of caching algorithms, IRONModel uses Z-CART to continuously tune the parameters for the analytical models designed for their storage system and to refine their predictions over time. The results show that IRONModel is very effective for detecting system anomalies against high-level specifications. While the Z-CART decision tree reacts to performance discrepancies between models and observations, the fundamental underlying reliance on a set of analytical models does not allow modeling of emergent, complex parameters interactions that are due to latent variations.

Herodotou and Babu [48] introduce a What-if Engine as part of Duke University's Starfish project [96] for the optimization of cluster resource selection for MapReduce task execution.

The authors classify MapReduce task profiles as sets of direct fields, such as execution times of specific portions of a task, and statistical information fields, such as average time to execute a portion of the dataflow per input record. A mix of analytical models and black-box models are used to represent the fields. Given a representative task profile, the system predicts the total execution time of a related task, with different input data, cluster resources and configuration parameters. Results show that the accuracy of the predictions was sufficient to perform configuration tuning and resource allocation for multiple classes of MapReduce tasks. The What-If Engine in Starfish is very effective, although it requires a considerable amount of a-priori knowledge to select the most appropriate models to represent each of the system's features.

When partial information on a system's behavior is available, and it can be offered by a user or an administrator to the modeling framework in an appropriate form, it can be leveraged to reduce the modeling effort, improve accuracy and detect anomalies. This approach leverage information about a system's characteristics, expressed by domain-specific languages, to impart specific hints and/or expected behavior.

To assist the inspection of performance behavior of the system, Ghanbari et al. [40] introduce *SelfTalk*, a declarative, domain-specific language that allows users to encode their high-level hypotheses about system invariants, expected correlations between system metrics, or other a-priori derived performance models. While SelfTalk was designed to allow system administrators to query and understand the status of a large-scale storage system, and detect anomalies, Chen et al. [24] in their Ensemble system extend the concept to the area of performance modeling. The authors leverage model templates to express explicit information on system behavior and system structure, and their system has the ability to answer to queries to validate these models. They further propose leveraging Direct Sampling Guidelines or *clues* in SelfTalk to indicate areas of the configuration space that can be safely ignored due to being already known, known to be noisy, or known to be invalid configurations. The purpose of the semantic information is to guide the Ensemble run-time modeling engine to proactively trim

the configuration space.

All the aforementioned techniques do not explicitly leverage any previously acquired observations of the systems they are required to model, other than the information that was initially used to establish the performance modeling approach itself. When legacy data is actively stored and re-used for the purpose of improving accuracy and to model discrete or continuous variations in a system's behavior, the modeling approach becomes incremental rather than static.

7.4 Incremental Modeling and the Chorus Performance Modeling Framework

Incremental models capitalize on all or part of the legacy data available from past observations of a system, and rely on the realization that variations commonly occur in real-world applications. These models represent a system's behavior by incorporating new observation and adapting existing models instead of re-training them, such that the training time is reduced and accuracy is improved.

7.4.1 Chorus

Chorus [25] is a comprehensive modeling framework that was created by our group at the University of Toronto, which explicitly supports incremental modeling. The aim is to use a combination of a-priori information about a system being modeled, capturing an expert system administrator's intuition on the overall system behavior, and actual observations in the form of stored reusable models.

The fundamental modeling technique underpinning Chorus consists of subdividing a system's configuration space into a regular grid of regions, then fitting an ensemble model in which multiple template models are trained concurrently and then individually ranked on a

per-region basis. The template models are of three different categories: Analytical, Grey-Box and Black-box. The models used by Chorus to create ensembles are the following: Analytical (white-box), Linear (black-box), Support Vector Machine (black-box), Inverse Exponential (grey-box), and Curve fitting (grey-box). When training, Chorus uses the training set to fit all the models in the ensemble, and ranks them according to their accuracy on a per-region basis using k-fold cross-validation, according to the following error metric:

$$\begin{aligned} n &= \text{Number of samples} \\ p &= \text{Model prediction} \\ m &= \text{Observed value} \\ \text{RERR} &= \frac{\sum_{i=1}^{n} \left(\left| \frac{p_i - m_i}{m_i} \right| \right)}{n} \end{aligned} \tag{7.1}$$

One of the key aspects in Chorus is the ability for the user to specify prior knowledge of a system's behavior, by selecting the parameters of the grid of regions, choosing a suitable ensemble of template models to represent the expected behavior, and appropriately configuring the template models' configuration parameters. When performing regression, for each desired regression location, Chorus detects the appropriate region, then selects the template model with highest local accuracy and performs the final regression using the selected model. The ensemble covers the entire configuration space, although some of the underlying models use a single set of parameters, while others support per-region fitting and regression.

Chorus Template Models

In Chorus, template models used to represent individual regions are of three different categories: Analytical, Grey-Box and Black-box. The system allows the creation of model ensembles that mix different categories.

Static Analytical Models (A-STOR) represent information that the Chorus user has obtained by detailed inspection of the system under consideration. Rather than being represented

directly as analytical formulae, Chorus represents these models as databases of measurements that have been obtained in advance by sampling user-supplied analytical models at a representative set of locations. A limitation of these models is that they do not interpolate, and therefore their evaluation outside of the locations that were sampled to construct the database is unavailable.

Linear Models (B-LIN) represent black-box approximations of a system's performance behavior obtained by fitting a set of linear models, one per region, each with its own set of coefficients. This model supports full multi-dimensional regression.

Support Vector Machine Models (B-SVM) represent black-box approximations of a system's performance behavior obtained by fitting a single Support Vector Machine model with Radial Basis Functions kernel. This model supports full multi-dimensional regression.

Curve Fitting Models (G-RGN) represent grey-box approximations of a system's performance behavior, where the toolkit user has pre-determined the behavior to be representable by a set of simple functions such as polynomial, exponential, logarithmic. The system performs fitting of 1-dimensional curve models, each with its own set of coefficients. The user selects one dimension of the configuration space, which the system uses to fit the models. A regular grid of locations is then selected by Chorus in the remaining configuration space dimensions, and for each location, a 1-dimensional curve fitting model is fitted to the available training data at those locations. Interpolation is therefore only available along the selected dimension.

Inverse Exponential Models (G-INV) represent the system behavior as a set of an 1-dimensional inverse exponential curves, but are otherwise identical to G-RGN models.

Since not all the template models allow for full, multi-dimensional regression over the entire configuration space, when queried for the value at a location where the highest-ranked model cannot be evaluated, Chorus progressively interrogates lower-ranked models as a fallback mechanism until one is found that covers the desired location. For example, the highest ranked model for a region may be an A-STAT model, which does not interpolate between the available training data. If Chorus is requested to evaluate a location that the A-STAT model

cannot resolve, it automatically interrogates a lower-ranked model until one is found that has interpolation capabilities, such as B-LIN.

Model Reuse and Incremental Modeling with Chorus

Using the Chorus toolkit, incremental modeling starts from a pre-existing legacy model, then adds new samples to an existing dataset, and finally re-fits the Chorus ensemble using the new dataset. The new samples replace the original samples in the locations where they overlap. The ensemble model then adapts to the new dataset by fitting its template models and ranking them. The authors report that ([25], Section 5.6.4) the more similar the behavior of the two systems, the fewer samples are necessary for Chorus to obtain an accurate representation of the new system's behavior.

The incremental modeling technique implemented in Chorus has definite advantages over previous methods that did not leverage existing information, and by using a strategy of replacing old samples with new observations, it performs best in scenarios where variations produce effects localized to limited portions of the configuration space. In order to capture the more complex and nuanced relationships between observations of a system undergoing configuration changes, recently multiple attempts to exploit transfer learning techniques are being investigated as a way to reuse prior knowledge to build black-box models of configurable systems. As previously introduced in Section 2.2, transfer learning leverages similarity between tasks to improve modeling accuracy and to reduce the need for a large number of samples in a model's training set.

7.5 Transfer Learning for Systems Performance Modeling

In the field of systems performance modeling, homogeneous transfer learning [72] approaches have so far been the most prominent, as in all the work we found in literature, the domains of the legacy and unknown models are identical, or a valid bijection of the unknown model's

domain onto the legacy model's domain always exists.

Additionally, for the purpose of performance optimization, models have to be obtained by inductive learning, which allows the generalization of the transfer of information between the source and target domains to different codomains. This characteristic is in contrast with Domain Adaptation [52] and other transductive learning methods [35] that are limited to transferring information between functions that share a common support.

In particular, inductive homogeneous transfer learning is further classified into three main categories:

- Instance-based approaches, in which a set of available samples from the legacy function (the source domain) are added to the limited samples of the unknown function (the target domain). The enriched sample set of the unknown function is used to directly build a full model of the unknown function. As an example of instance-based methods, Garcke and Vanck [38, 120] proposed a technique that considers shifts in domain and co-domain of the unknown function. In their technique, each sample of the legacy function is individually assigned a weight corresponding to the estimated similarity of a sample of the legacy function to an existing sample set of the unknown function. The weight is measured by influence of the sample on the prediction quality of the unknown function. Our group at University of Toronto introduced a form of combined transfer learning and active learning [86] using Gaussian Processes to reuse prior information for the purpose of reducing training time of a performance model, as part of the Ensemble performance modeling tool [24].

  As part of Ensemble [24], an early method was proposed by our group at the University of Toronto for extending trained models, to represent the effects of using heterogeneous hardware environments on application performance. Specifically, a Gaussian Process was used to model the effects that changes to the hardware configuration have on the performance of the NPAIRS [98] neuroscience application. The approach (*GP Mapping*) leveraged the ability of Gaussian Processes to find the locations that maximize the

expected improvement in modeling accuracy in closed form, therefore using a form of active learning [86] to select the samples used as training set. This method represents an early use of the concept of leveraging transfer functions, or maps, to correlate performance models, and exhibited significant advantages over employing traditional modeling methods. Our work builds upon that intuition and extends it to use a more general form of transformation maps. At the same time, our approach adds the ability to represent the map using multiple modeling methods, and removes the limitation of using active learning to obtain the model training set.

- Feature-representation-based approaches, in which a shared feature space is created from the limited features shared between legacy and unknown functions. The shared features are used to encode and transfer the knowledge from the legacy to the unknown function. Multi-task learning methods [21] are among the transfer learning methods that follow this approach. In multi-task learning, given multiple tasks with a few labeled training data for each task, the goal is to jointly learn individual classifiers for different tasks by exploring those tasks [5].

  Chen et al. [23] propose *Experience Transfer*, a technique that leverages a Bayesian network to model the correlation between a system's configuration parameters, and then use it to optimize the system's performance. The authors show how a Bayesian Network trained with samples of a system in a given configuration can be effectively used as the starting point to tune the configuration of a similar system. While Experience Transfer is very effective for transferring knowledge between systems that exhibit strong similarities in the covariance of their configuration parameters and for parameter tuning, the technique does not build a full model of a system and therefore cannot be used for performance reasoning. It relies on having complete freedom in the choice of sampling positions, which may not be possible in all scenarios.

  Jamshidi et al. [50] use either a linear or a non-linear model (a Gaussian Process) to

represent the correlation between different configurations of a configurable software system. Their approach leverages the similarity between a legacy model and an unknown new model in the form of a covariance matrix that is formed as samples from the legacy dataset and samples from the new system configuration are added to a Multi-Task Gaussian Process model. While this method has been found to be very effective in modeling system performance, it does not directly exploit the intuition that the sparsity of the covariance matrix, when only a few samples are available of the second task (the unknown model), may yield only a local effect based on the choice of Gaussian Process kernel. These effects may prevent the model from detecting opportunities for simpler, lower-dimensional constructs that explain the covariance globally. Additionally, the research does not provide a correlation between the error metric and the ultimate purpose of the model, e.g. for performance reasoning/forecasting and/or for the purpose of acting as a surrogate model for resource allocation optimization.

Similar to our work, Valov et al. [116] study the effects of using linear models to represent a transfer function that transforms a legacy model to cover a new, unknown system or application configuration. The choice of a linear model for the transfer function was justified experimentally, by noticing that a class of system modifications results in largely linear changes in a system's behavior. While the work shows substantial accuracy improvements with limited samples, the technique is inherently limited to representing simple incremental variations. Additionally, no significant features are identified and selected in the modeling process, or used in the representation of the transfer function, which results in suboptimal accuracy, relative to our work (Sections 6.4.2) and other incremental and transfer learning techniques.

- Model-parameter-based approaches, in which the information to be shared is not directly in the data, but rather in the models' hyperparameters. In these approaches the goal is therefore not to reuse data from the legacy model and perform different forms of resampling to obtain a dataset representative of the unknown function, but instead to

focus on modeling the relation between the hyperparameters of the legacy model and the model representing the unknown function [58]. In our search, we could not find an example of model-parameter-based approach applied to system performance modeling, and an interesting future research direction could be applying Model Mapping to model the relation between models' hyperparameters.

Most transfer learning techniques [33, 38, 75, 99, 120] rely on available samples of the unknown system configuration to build the model. As discussed in Section 1.1, in principle our technique may extrapolate across an ordered set of legacy models to predict the effects of the unknown system configuration without any additional observations. Additionally in the works employing Gaussian Processes [50, 86] as the basis for transfer learning, the choice of kernel is not specified. We found that the kernel choice has a considerable contribution to modeling accuracy, and requires a form of model selection by itself.

Model Mapping is completely independent of the approach used to create the legacy models, whereas several other transfer learning techniques apply are created with the same formalism (e.g. neural networks [100]). Similar to Multi-Task Learning we express the relationship between models, although with Model Mapping we do so in a compact mathematical form that is more interpretable than other constructs (e.g. a full covariance matrix for Multi-Task Gaussian Process [14]). We therefore argue that our technique has some advantages over other transfer learning approaches that adapt existing models to generalize them.

Chapter 8

Conclusion and Future Work

In this chapter, we analyze our contributions in light of all the experimental work, provide a summary of the findings, and highlight future opportunities for research that emerged as a direct consequence of this investigation.

8.1 Conclusion

In this dissertation we proposed *Model Mapping*, a novel inductive transfer learning technique for system performance modeling and optimization. The motivation for this work is rooted in the extensive work performed in our group on the topic of capturing, analyzing and understanding the behavior of scalable, multi-tiered virtual environments, the computer systems that underpin them, and the services and applications they host.

The constant effort of operating and maintaining datacenters requires both a deep level of understanding of the applications' characteristics and a reactive, dynamic approach to monitoring and balancing the variables that determine a datacenter's performance. Conventional techniques for modeling systems performance use static models and are capable of adapting to variations of system configurations only within the boundaries of their pre-defined heuristics.

We propose to systematically leverage experience in the form of monitoring and observations, and use this information to trace a virtual performance trajectory a system follows

throughout its lifetime. To this extent, our contribution is the development, implementation, and detailed experimentation for a targeted form of transfer learning that is particularly effective at this task.

Model Mapping lends itself to representing the behavior of a system as it varies due to explicit and latent changes. Our approach transforms the problem of directly modeling system performance for the changed system into a problem of modeling the changes themselves, where the latter is likely to be a problem of lower dimensionality and/or lower complexity than the former. The technique exploits structural similarity between two system configurations to construct a compact map between their performance behaviors. In particular, we observed that the technique is effective in the three use cases we originally set out to explore: extending existing performance models, modeling incremental variations, and resource optimization/consolidation.

This dissertation documents the work behind Model Mapping, from the initial intuition behind the technique to the experimentation phases. We discussed the reasoning that led to applying transfer learning to the domain of system performance modeling and outlined the motivations behind the choice of introducing a dedicated approach over other forms of knowledge transfer (Chapter 2). We then formally described the technique, its generalization, and overall limitations, and discussed the *ModelMap* toolkit design considerations and implementation details (Chapter 3).

We proceeded to evaluate and compare the effectiveness of our technique against different traditional modeling methods and other forms of incremental modeling (e.g. Chorus) and transfer learning in a number of scenarios. These scenarios were carefully selected to represent a variety of realistic circumstances that cloud systems experience at design stage and/or while in operation. The data used for the experiments was collected from direct instrumentation of real systems, and used a number of applications and benchmarks that are realistic representations of production hosted workloads and services.

As part of researching and developing the Model Mapping technique, we designed, setup

and instrumented a performance measurement testbed, for the purpose of creating a new dataset using the industry-standard TPC-C benchmark. The goal was to document how the performance of a database system is affected by different configurations of a number of system-level and application-level parameters (Chapter 4).

Our technique proved to be very effective in most scenarios, dealing with changes in hardware specifications, application configurations and performance optimization (Chapters 4, 5, 6). We found Model Mapping to be especially efficient in improving accuracy over other techniques where the sampling budget is very low, making it a strong guidance tool for resource consolidation and optimization, which is very responsive to system changes.

In order to ensure the generality of our technique, we also studied how often the usage of Model Mapping results in an accuracy improvement, while using the same modeling method. We reported that in the large majority of the experiments we conducted, using a modeling method with Model Mapping results in higher accuracy than when the same method used directly to model the unknown system configuration. On a total of 104 experiments, applying Model Mapping resulted in a RMSE error reduction in 84 cases (81%), and in a MAPE error reduction in 92 cases (88%).

We also observed that Model Mapping may be used multiple times consecutively, by performing successive mapping steps to build a model through consecutive transformations of a legacy model. However, error accumulation introduced by the composition of transformation may limit the number of steps that can be done before the error becomes prohibitive.

Overall, we found our technique to be successful when the modeling problem presents the following characteristics:

- Explicit or latent system variations produce incremental, smooth, linear or nonlinear changes in the performance metrics being modeled. Examples are hardware upgrades such as CPU and disk improvements, or software configuration alterations such as variations in resource allocation to Virtual Machines.

- Variations produce global performance changes across the parameter space, and result

in a nonlinear scaling of the system performance behavior. Examples are changes in processor clock speed, gradual variations in application demand and latent effects of data accumulation.

We also observed that, similar to other transfer learning techniques, Model Mapping is not effective in the following circumstances:

- System configuration changes that result in introducing new discontinuities or removing sharp features from the performance function. This behavior can be observed when examining the performance characteristics of an SSD drive compared to a regular Hard Disk (Sections 5.3 and 5.5).

- Configuration changes that cause shifting of performance features as a result of triggering different performance bottlenecks. A sudden system failure, or a drastic increase in available resources may have effects on performance to render its performance characteristic too dissimilar from a legacy configuration, negating the advantage of using prior knowledge.

We conclude that our technique affords increased levels of accuracy for quickly and accurately modeling the performance characteristics of systems as they experience implicit and explicit incremental variations.

8.2 Future Work

A promising research direction is to verify the effectiveness of Model Mapping in scenarios that require model extrapolation, for example to capture and analyze the performance trend of a system as a transformation function, and then use it to predict when a system upgrade may be necessary. To this effect, we conducted a preliminary study on model extrapolation using Model Mapping, using a variation of the scenario described in Section 4.7, which analyzes how a database performance changes as data accumulates over time.

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	46.74±5.7	143.6±47.7
	Polynomial regression	28.39±1.2	**22.76±0.7**
	Linear SVR	30.92±1.1	30.34±0.7
	Gaussian Process	33.87±1.6	32.73±1.1
10	Linear regression	28.20±1.1	27.03±1.1
	Polynomial regression	23.61±0.9	**20.18±0.6**
	Linear SVR	28.84±0.9	30.73±0.6
	Gaussian Process	23.06±1.6	27.30±1.5

Table 8.1: RMSE mean and standard error of the proposed Model Mapping technique and direct modeling technique with various modeling methods for **TPC-C number of warehouses** configurations. Legacy models **warehouses**=10, 20 and 40, unknown model **warehouses**=64.

In particular, we used two complete legacy models, representing authoritative snapshots of the database's performance at specific moments in time, using our dataset for **TPC-C number of warehouses**=10 and 20, respectively. We also used a third legacy model, built with a limited sampling budget, representing the latest information we have on the database's performance behavior (**TPC-C number of warehouses**=40). The unknown model in this what-if scenario is represented by the database performance when configured with 64 warehouses.

In Table 8.1, the "Direct" column represents the error of using direct modeling, using samples from the system operating in the unknown configuration (**TPC-C number of warehouses**=64). The "Mapping" column represents the error of using Model Mapping, using samples from the system operating in the third intermediate configuration (**TPC-C number of warehouses**=40), and *no samples* from the unknown configuration.

We can see that Model Mapping effectively extrapolates the information contained in the full and partial legacy models and provides a model of reasonable accuracy even without using any samples from the system in the unknown configuration. Substantial additional verification will be necessary to ascertain the extrapolation capabilities of our transformation functions, but we believe that this characteristic may be leveraged most effectively in other scenarios where a system is subject to constant or steady variations.

Furthermore, we believe that our approach of expressing system variations explicitly has the additional advantage of improved interpretability over direct models. Due to their lower-dimensional nature relative to the models they span, maps can be used to more easily and visually understand how systems vary over time. Databases of maps may be created to represent templates of expected behaviors for multiple classes of transient systems, and comparisons between expected and measured maps may be used to detect and diagnose failures in dynamic systems.

Our work can also be expanded by improving the modeling and validation capabilities of our modeling toolkit. We believe that these extensions will give additional benefit to employing Model Mapping in the field of systems performance reasoning and optimization, by making it more accessible and readily available for integration into production systems. Additionally, we believe that our technique is suitable for use in a large number of computer systems application scenarios. We therefore intend to further explore the use of Model Mapping and the *ModelMap* toolkit in the following directions:

- **Consecutive mapping error mitigation**

  Future work can address the issue of error accumulation by introducing techniques such as systematically limiting the number of consecutive mapping steps used to transform a legacy model, by collapsing transformations into a smaller set, and by maintaining a sliding window of sample sets to use only the last N sets to build the maps.

- **Automated map class and model selection**

  In this work, we classified transfer functions according to their cardinality and usage of features, but in each modeling scenario, we left it to the user/system administrator to choose the most appropriate form of map (pure composition, difference, etc.) and its class. We can improve our *ModelMap* toolkit to automatically perform sensitivity analysis and feature extraction from the legacy model, and to recommend to the user the most appropriate class of map to be used.

Additionally, the toolkit can be extended to automatically train numerous map classes at the same time, and perform cross-validation to select the most accurate combination of modeling method and map class for a given sampling budget. With this improvement, as more samples are available of the unknown configuration being modeled, the toolkit can internally perform model evaluation and transparently select the most appropriate model and map class to perform regression with.

- **Database of models and automated selection of appropriate legacy models**

  Our work focused on reusing information of a system's behavior, encoded as raw data or a mathematical model, to assist in reducing the effort of modeling the behavior of the same system, operating in a different configuration.

  We can expand our toolkit to perform the classification of partial and full models, according to their overall behavioral similarity, measured as covariance. With such a classification capability we can build a database of system performance models, which we can use to serve as legacy models for other systems and applications that exhibit a similar behavior. Additionally, we can also build a database of map models, which may be used to directly compare and contrast how different systems evolve as a consequence of the configuration changes they experience over time.

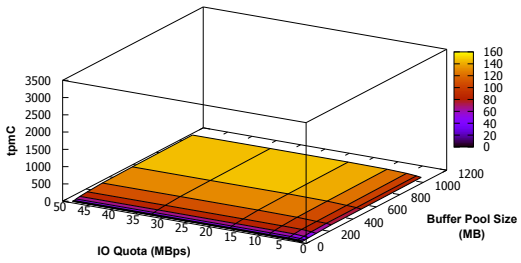
- **Applying Model Mapping to other types of performance modeling applications**

  The scenarios we presented, while being representative of cloud systems, are only a small fraction of the use cases where our technique can be applied.

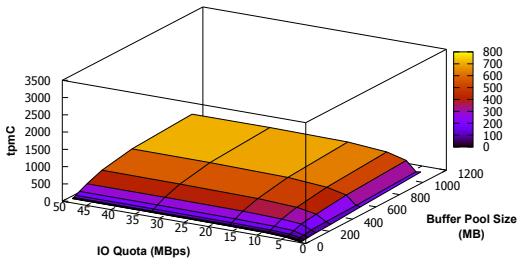
  We can apply Model Mapping to other scenarios, to analyze other kinds of transient systems present in datacenters as their performance metrics change according to the variation in their configurations. For example, Krzywda et al. [53] show the importance of CPU power throttling to reduce the chances of exceeding power consumption limits in datacenters. We can apply Model Mapping to extend a service or a system's performance model to account for the performance variations that result from CPU power throttling.

Appendix A

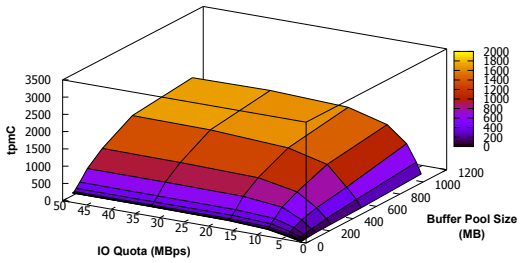
TPC-C Performance Data



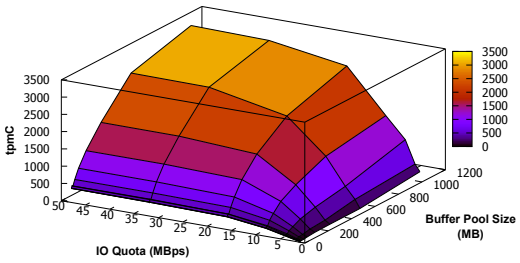
(a) **CPU quota**=10%



(b) **CPU quota**=25%

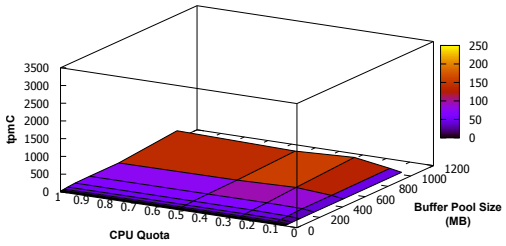


(c) **CPU quota**=50%

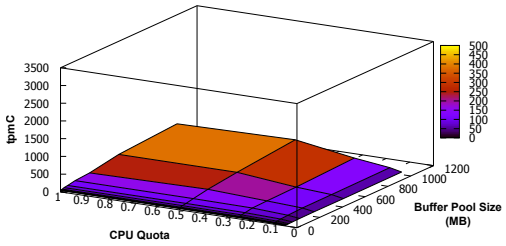


(d) **CPU quota**=100%

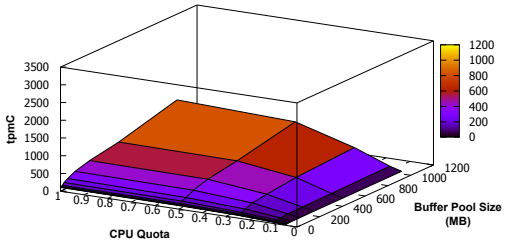
Figure A.1: Performance graph of TPC-C transactions per minute (tpmC) on **Platform 1** when the **IO quota** varies between 1 and 48 MBps and the **Buffer pool** varies between 1 and 1024 MB.



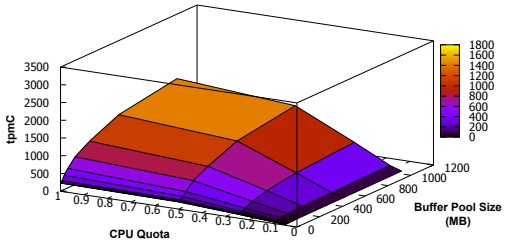
(a) **IO quota** = 1MBps



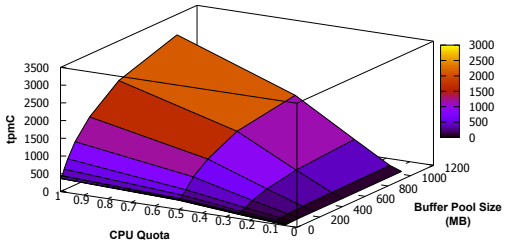
(b) **IO quota** = 2MBps



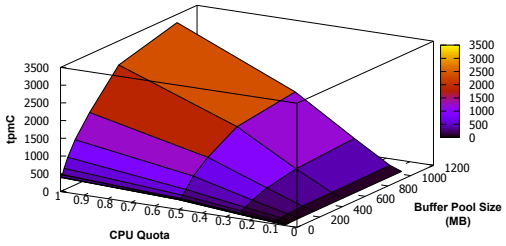
(c) **IO quota** = 4MBps



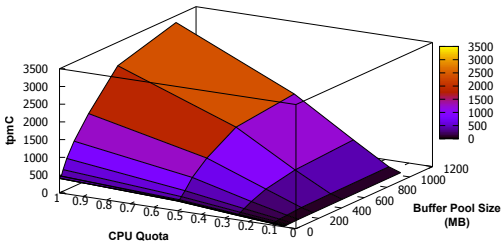
(d) **IO quota** = 8MBps



(e) **IO quota** = 16MBps

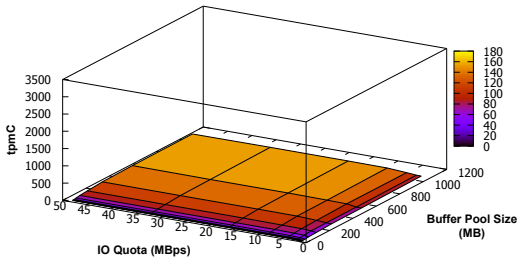


(f) **IO quota** = 32MBps

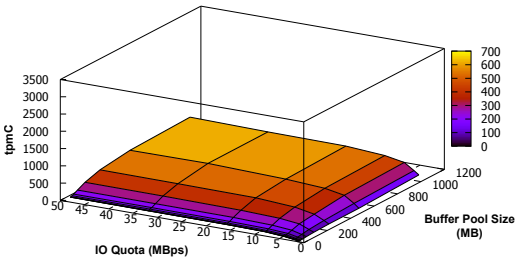


(g) **IO quota** = 48MBps

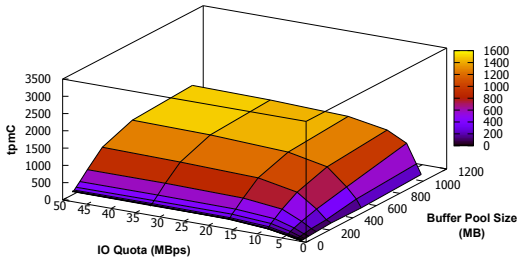
Figure A.2: Performance graph of Transactions per minute (tpmC) on **Platform 1**, when the **CPU quota** varies between 10% and 100% and the **Buffer pool** varies between 1 and 1024 MB.



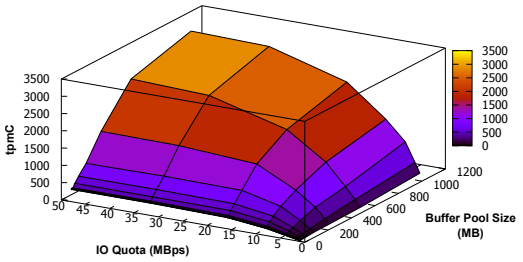
(a) **CPU quota**=10%



(b) **CPU quota**=25%

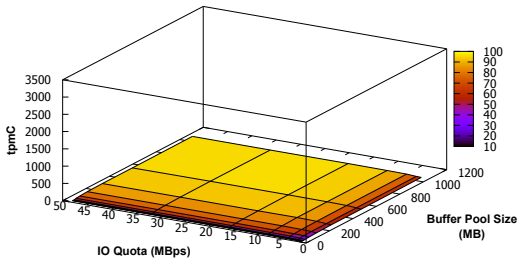


(c) **CPU quota**=50%

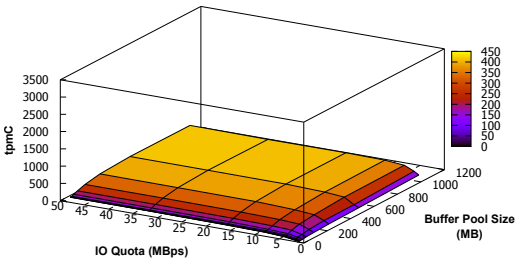


(d) **CPU quota**=100%

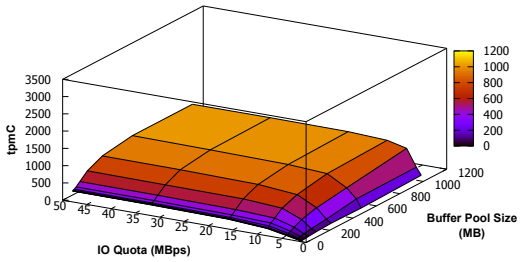
Figure A.3: Performance graph of TPC-C transactions per minute (tpmC) on **Platform 3** when **IO quota** varies between 1 and 48 MBps and **Buffer pool** varies between 1 and 1024 MB.



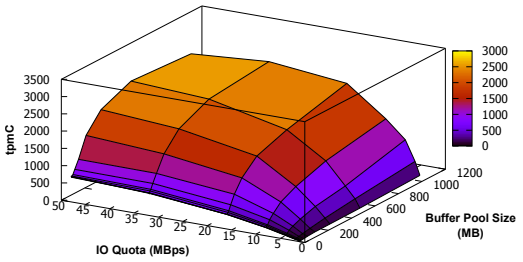
(a) **CPU quota**=10%



(b) **CPU quota**=25%



(c) **CPU quota**=50%



(d) **CPU quota**=100%

Figure A.4: Performance graph of TPC-C transactions per minute (tpmC) on **Platform 2** when **IO quota** varies between 1 and 48 MBps and **Buffer pool** varies between 1 and 1024 MB.

Table A.1: Average TPC-C Throughput (tpmC) with its 95% confidence interval - Platform 1

CPU Quota	Buffer pool (MB)	Disk IO Quota (MBps) 1	2	4	8	16	32	48
5%	**1**	8.7±0.3	12.5±0.4	14.4±0.4	15.2±0.4	15.8±0.5	15.7±0.4	16.3±0.5
	2	9.2±0.3	12.8±0.4	14.6±0.4	15.4±0.4	16.2±0.5	16.1±0.6	16.1±0.5
	4	9.9±0.3	13.3±0.3	15.2±0.4	16.5±0.6	15.9±0.6	16.8±0.5	16.6±0.6
	8	9.0±0.2	12.6±0.3	14.1±0.4	15.0±0.5	15.3±0.5	15.7±0.5	15.4±0.5
	16	8.9±0.3	12.1±0.4	13.3±0.3	14.7±0.5	15.1±0.4	15.0±0.5	15.2±0.5
	32	13.2±0.5	17.0±0.5	19.7±0.8	21.1±1.0	21.1±0.8	21.4±1.0	21.6±1.0
	64	13.9±0.5	20.3±0.7	22.4±1.0	24.8±1.3	23.2±1.2	25.1±1.2	24.1±1.3
	128	15.6±0.8	23.1±0.9	24.6±1.2	27.5±1.6	29.3±1.6	28.0±1.6	28.0±1.5
	256	16.6±0.9	26.3±1.6	27.1±1.6	28.9±2.0	29.0±1.8	29.7±1.7	30.1±1.9
	512	15.7±1.1	22.7±1.2	25.6±2.3	29.4±1.7	28.1±1.6	30.6±2.4	31.0±2.3
	1,024	18.3±1.1	25.2±1.5	29.0±1.6	33.6±1.8	32.9±2.0	32.1±2.0	33.4±2.1
10%	**1**	14.9±0.7	25.2±1.1	36.2±0.8	39.8±0.8	42.9±1.1	43.7±1.1	44.4±0.9
	2	16.1±0.3	27.6±0.8	37.7±0.6	41.3±0.8	43.1±0.8	43.1±1.0	44.5±1.1
	4	16.2±0.3	29.6±0.5	37.9±0.8	40.3±1.1	42.6±1.1	44.4±0.9	43.3±0.9
	8	16.2±0.3	27.7±0.6	37.1±0.6	40.3±0.7	41.7±1.1	42.5±1.0	42.8±1.0
	16	16.5±0.6	27.7±0.6	36.4±0.7	38.7±0.7	40.3±0.9	40.7±1.1	41.6±0.8
	32	22.6±0.3	39.1±0.7	50.3±1.0	54.8±1.2	56.7±1.4	57.2±1.5	56.9±1.4
	64	33.9±0.6	49.9±1.1	65.4±1.0	70.6±1.5	74.1±1.2	75.2±1.8	74.5±1.7
	128	41.9±0.9	65.1±1.5	84.9±1.8	90.1±2.4	94.7±2.6	95.9±2.7	93.0±2.9
	256	62.9±2.3	82.5±2.6	89.6±3.3	111.2±3.8	121.5±4.1	123.5±4.6	122.3±3.9
	512	64.7±2.1	85.3±3.6	99.5±4.2	125.4±7.2	139.2±5.1	136.5±5.0	137.3±5.4
	1,024	77.8±4.0	98.6±5.1	121.7±7.2	142.4±7.8	153.6±9.1	155.4±8.0	150.9±8.8
25%	**1**	18.3±1.0	37.3±1.5	77.5±1.7	107.2±1.4	118.1±1.1	123.5±1.6	124.6±1.5
	2	19.9±0.4	39.0±1.4	81.0±1.7	109.5±1.2	118.6±1.4	123.4±1.6	122.4±1.1
	4	19.9±0.5	43.9±0.9	81.8±1.0	108.6±1.4	117.5±1.4	124.5±1.9	123.2±1.8
	8	21.0±0.8	41.3±1.5	85.4±1.3	111.9±1.3	122.5±1.4	126.0±1.5	127.8±1.6
	16	23.7±0.5	48.3±1.2	101.2±1.6	126.7±2.2	131.8±2.7	130.0±2.0	131.1±2.0
	32	30.1±0.5	62.7±1.1	113.8±1.3	139.5±1.3	152.4±1.6	161.6±1.6	161.7±1.7
	64	42.8±0.7	79.7±1.9	161.0±1.9	190.2±1.8	210.6±1.9	219.9±2.1	220.5±2.7
	128	59.2±1.1	116.1±3.8	240.5±3.6	286.8±2.3	318.2±3.8	328.5±3.1	326.5±3.4
	256	78.4±6.6	129.7±9.9	299.4±8.1	400.4±8.8	458.6±5.4	484.8±6.8	492.4±5.8
	512	137.2±10.7	146.2±17.4	399.6±12.7	528.2±13.0	616.4±9.6	651.8±11.3	657.7±8.0
	1,024	236.6±13.1	217.6±19.7	495.5±19.5	642.4±12.1	734.1±17.8	761.4±15.2	757.1±17.3
50%	**1**	18.5±0.8	38.7±2.3	98.6±3.1	198.8±2.6	258.3±3.5	271.4±2.4	270.2±2.3
	2	20.6±0.6	41.2±1.7	105.7±2.1	204.1±2.4	259.7±2.3	272.5±3.1	271.7±2.5
	4	20.6±0.5	47.8±1.2	106.0±2.9	206.7±2.3	259.2±2.2	270.9±2.5	268.7±3.2
	8	21.6±0.6	45.8±1.9	115.0±1.3	210.6±1.9	267.2±2.0	277.3±1.8	277.3±2.6
	16	24.7±0.8	51.5±1.5	135.7±1.4	240.3±2.7	284.5±2.8	289.3±3.6	287.7±3.3
	32	30.5±0.6	70.7±2.0	177.3±2.2	308.7±3.3	353.2±2.8	353.4±3.1	352.2±3.1
	64	43.5±0.9	86.8±2.7	235.2±3.0	402.8±3.9	459.8±3.2	467.9±4.5	473.4±3.2
	128	58.4±1.8	129.6±3.4	347.2±3.8	617.2±4.1	733.1±6.0	753.9±4.9	753.6±6.3
	256	71.5±9.6	203.9±10.8	469.2±13.0	832.6±13.1	1046.8±14.2	1094.8±14.4	1110.8±12.9
	512	90.4±14.7	310.0±13.5	615.9±8.1	1175.2±13.9	1456.3±21.5	1586.4±20.3	1610.2±16.7
	1,024	162.0±36.5	480.0±36.6	967.7±27.8	1423.1±28.4	1727.4±23.3	1822.0±28.5	1800.7±29.0
100%	**1**	16.1±0.9	41.3±1.3	103.0±2.3	226.8±3.0	363.5±3.5	397.6±6.4	406.1±4.7
	2	19.8±0.5	41.3±1.8	107.4±2.0	229.8±4.4	366.2±3.6	402.9±5.7	404.9±4.8
	4	20.2±0.8	46.8±1.2	110.7±2.8	227.6±4.0	363.7±4.4	389.6±10.2	397.2±5.8
	8	22.2±0.6	47.1±1.0	125.2±2.7	276.7±3.2	435.8±3.4	472.3±4.2	466.3±5.8
	16	25.4±0.5	54.4±1.1	139.5±1.3	274.7±2.4	415.6±4.1	441.6±10.6	444.5±6.5
	32	30.3±0.7	75.2±1.8	199.6±2.8	407.1±2.7	583.3±3.8	608.1±4.9	607.2±5.4
	64	44.4±0.9	88.1±2.2	263.6±5.4	548.6±3.5	828.0±5.5	872.7±6.7	875.8±8.8
	128	61.3±1.4	130.1±4.5	372.3±6.7	784.7±4.0	1217.9±10.8	1273.4±10.9	1270.6±12.5
	256	68.0±9.5	162.1±14.3	489.8±14.5	1038.6±8.9	1729.3±20.3	1856.5±37.0	1863.1±35.4
	512	70.2±11.1	310.0±18.4	638.4±17.4	1412.0±13.2	2382.0±28.2	2821.8±71.3	2839.2±56.5
	1,024	229.2±29.9	425.3±60.4	1082.7±28.3	1682.3±22.8	2923.5±106.7	3269.8±195.6	3308.7±179.9

Table A.2: Average TPC-C Throughput (tpmC) with its 95% confidence interval - Platform 2

CPU Quota	Buffer pool (MB)	Disk IO Quota (MBps) 1	2	4	8	16	32	48
5%	**1**	8.0±0.6	10.8±1.0	12.6±0.9	13.7±0.5	14.6±1.1	14.6±1.2	14.2±0.9
	2	8.6±0.3	10.5±0.7	12.2±0.9	13.8±0.7	14.1±0.9	14.7±1.1	14.9±1.0
	4	8.3±0.5	10.2±0.5	12.7±0.6	14.0±1.0	14.4±1.0	14.4±1.0	14.9±1.0
	8	8.0±0.5	10.4±0.5	12.5±0.9	13.6±0.9	14.0±1.1	14.6±1.0	14.7±0.7
	16	7.9±0.7	10.1±0.5	12.1±0.7	12.7±0.6	14.0±1.0	14.4±0.7	14.1±1.2
	32	10.5±0.8	13.0±0.5	15.3±0.7	16.7±1.0	16.9±1.0	16.9±1.3	17.9±1.3
	64	10.7±0.8	15.7±1.1	17.2±1.2	18.4±1.2	19.0±2.1	19.7±1.0	21.0±2.2
	128	12.0±0.7	17.0±0.9	18.9±2.6	20.5±2.4	21.2±2.2	23.1±2.6	21.4±2.8
	256	13.2±1.1	14.3±0.8	19.1±1.1	21.6±1.5	21.7±2.1	23.5±2.2	21.5±2.5
	512	12.2±1.0	16.2±1.4	18.5±0.6	20.8±1.4	22.7±1.9	23.8±1.5	23.2±2.4
	1,024	16.7±1.2	18.3±1.3	22.0±1.6	24.2±2.3	24.7±2.4	26.9±3.5	24.5±2.9
10%	**1**	15.6±0.8	25.6±1.7	34.2±1.2	40.7±1.4	43.0±1.6	46.1±2.2	48.8±2.0
	2	15.9±0.8	28.4±1.1	33.9±0.9	39.8±1.3	42.6±1.4	45.7±2.2	45.9±1.6
	4	15.9±0.6	28.0±0.8	35.6±0.9	39.5±2.1	42.9±2.0	47.0±1.6	46.1±1.8
	8	15.7±0.6	27.3±1.0	35.1±1.1	37.9±1.2	42.0±2.3	45.4±2.7	47.3±1.5
	16	15.1±0.7	26.3±1.3	32.2±1.3	37.9±1.4	42.7±1.1	44.9±1.3	44.4±2.1
	32	21.7±0.7	36.4±1.3	45.0±1.5	50.9±2.3	54.2±3.1	59.0±3.7	59.7±2.1
	64	32.0±1.2	50.0±1.1	58.2±1.7	64.1±2.9	69.8±2.7	71.5±3.0	70.3±3.5
	128	39.3±1.5	55.5±4.2	69.4±2.9	76.0±5.7	77.9±3.8	80.2±5.0	82.9±2.8
	256	50.7±3.1	60.2±4.5	78.5±5.5	90.9±5.9	86.2±5.2	91.1±4.6	91.7±5.6
	512	49.7±5.1	63.0±4.8	83.7±2.8	92.7±7.2	96.7±5.8	97.6±4.9	92.2±3.8
	1,024	61.4±6.4	70.5±6.3	85.4±8.7	92.0±7.9	93.7±8.1	93.0±6.0	98.4±6.7
25%	**1**	17.5±1.6	36.7±1.7	81.6±3.0	115.6±2.8	129.4±1.3	145.4±2.9	145.1±3.6
	2	19.3±1.3	42.3±0.9	83.8±2.7	114.9±2.5	132.9±2.9	146.6±2.4	146.9±4.0
	4	19.0±1.1	44.7±2.2	84.9±2.6	115.4±3.1	133.1±2.7	144.3±3.1	148.3±2.5
	8	21.5±1.0	49.1±1.4	90.0±2.4	118.3±2.1	134.0±1.9	145.0±4.5	147.9±2.9
	16	23.7±1.3	53.8±2.6	103.6±4.0	130.3±5.6	141.6±4.4	144.3±3.6	149.4±6.1
	32	30.3±1.4	63.6±1.7	117.1±4.4	142.4±2.3	165.2±3.4	174.1±4.7	178.4±2.6
	64	43.1±1.5	86.9±2.1	160.8±4.9	189.0±2.4	207.0±5.5	219.2±4.7	219.0±4.6
	128	54.0±5.5	116.0±13.0	220.2±5.2	257.8±4.4	278.5±7.4	289.0±7.5	293.0±4.7
	256	76.9±16.5	147.3±24.6	265.1±14.8	314.2±6.6	355.8±8.0	364.2±6.1	371.2±7.1
	512	90.8±18.0	155.5±28.1	326.3±8.0	374.0±11.8	410.5±13.8	413.9±13.8	421.3±11.3
	1,024	117.5±46.2	208.6±26.7	334.9±32.8	392.9±13.8	407.2±16.5	399.9±13.6	417.2±10.3
50%	**1**	17.1±2.0	40.0±4.2	101.0±4.9	207.1±5.5	283.6±5.0	308.9±5.8	313.5±4.8
	2	19.3±1.5	46.9±3.5	103.8±5.3	213.5±3.8	282.7±4.7	308.6±4.2	314.1±3.5
	4	19.2±1.4	46.2±2.4	106.4±6.3	210.2±2.9	284.7±3.5	309.7±4.1	308.3±5.7
	8	22.9±1.5	51.5±2.0	111.9±3.8	221.5±4.2	293.6±3.2	326.4±4.8	328.0±3.9
	16	24.4±1.7	56.3±2.4	138.0±5.6	253.9±5.2	309.3±7.0	334.2±5.9	339.1±5.8
	32	30.1±1.6	72.9±1.7	177.2±3.8	313.2±7.7	350.2±5.7	372.1±7.1	376.7±6.0
	64	44.0±2.0	93.7±2.1	232.0±5.5	399.2±3.5	448.7±6.9	471.2±5.9	477.6±5.7
	128	59.4±6.3	128.4±8.9	344.0±6.5	573.5±8.1	645.1±9.2	660.8±7.2	671.6±7.4
	256	78.6±25.4	183.1±29.8	468.3±13.3	741.8±6.4	833.5±9.2	869.9±12.6	878.4±11.0
	512	74.0±28.6	304.8±18.9	579.2±21.1	894.0±14.3	962.4±15.3	996.1±14.1	1009.8±10.8
	1,024	116.7±44.8	330.0±111.6	826.5±36.5	950.6±14.9	1000.2±19.1	1017.7±21.5	1017.0±16.7
100%	**1**	18.4±1.9	40.5±1.4	107.1±5.8	235.6±8.0	418.6±7.6	648.4±17.9	716.1±22.1
	2	20.3±1.3	48.5±3.4	110.3±6.5	233.1±9.4	421.0±11.4	636.2±15.0	737.3±12.0
	4	20.2±1.0	46.4±4.8	114.3±4.0	233.9±6.3	409.9±11.2	647.4±17.9	712.2±18.6
	8	22.4±0.8	52.2±1.8	123.8±3.3	275.9±8.9	466.6±9.5	700.3±23.8	739.8±24.4
	16	24.9±1.0	58.4±2.0	140.0±3.3	290.4±3.5	499.5±4.3	723.0±18.4	770.4±23.7
	32	30.6±1.3	73.8±2.8	201.1±3.6	409.4±6.6	644.3±14.9	862.2±12.3	885.2±45.5
	64	43.3±1.6	94.9±2.9	264.4±12.1	532.0±13.1	897.7±18.2	1112.8±55.0	1119.5±59.4
	128	57.4±5.2	134.2±8.3	352.1±18.3	754.2±8.3	1242.3±37.7	1566.8±48.6	1702.5±59.9
	256	60.2±29.0	203.0±36.3	494.2±12.8	968.7±12.1	1659.9±39.3	2040.2±72.8	2240.7±89.9
	512	64.4±22.1	307.8±16.6	636.7±13.8	1259.7±18.3	1970.8±50.5	2447.7±89.0	2574.0±112.6
	1,024	94.3±68.7	465.0±134.7	1069.5±45.8	1568.4±61.7	2391.0±90.6	2631.0±113.1	2481.2±173.9

Table A.3: Average TPC-C Throughput (tpmC) with its 95% confidence interval - Platform 3

CPU Quota	Buffer pool (MB)	Disk IO Quota (MBps) 1	2	4	8	16	32	48
5%	**1**	10.0±0.6	14.4±0.6	16.3±0.7	17.7±0.8	17.9±1.2	17.5±0.8	17.8±1.0
	2	10.9±0.6	14.3±0.5	16.0±0.6	16.9±0.9	17.7±1.2	17.3±1.1	17.2±1.2
	4	10.6±0.8	14.6±0.8	16.3±1.1	16.3±0.9	16.9±0.9	17.2±1.2	16.3±1.6
	8	10.0±0.5	14.0±1.0	15.2±1.5	16.0±1.3	16.8±1.0	17.7±1.5	17.4±1.2
	16	10.5±0.8	14.0±0.8	15.5±1.6	15.8±0.6	17.2±1.2	16.4±1.6	16.8±1.3
	32	14.4±0.9	17.8±1.1	20.2±1.6	21.3±1.6	22.8±1.2	22.8±2.6	23.4±2.7
	64	17.3±2.1	23.8±2.4	25.6±3.7	26.8±3.8	28.9±4.3	27.6±4.5	28.1±3.4
	128	20.8±3.0	26.6±3.5	29.0±2.4	29.5±3.7	32.5±4.6	31.2±3.8	32.4±3.7
	256	21.5±4.4	25.4±4.0	30.2±3.8	34.3±5.1	35.6±2.2	37.2±1.9	33.8±3.7
	512	21.9±3.5	25.2±3.7	31.1±4.5	36.8±4.4	34.0±2.0	36.9±1.9	36.2±3.4
	1,024	27.0±4.0	26.0±5.8	31.6±4.5	39.0±7.1	43.5±5.3	40.2±3.7	36.2±3.1
10%	**1**	15.0±1.0	29.6±1.4	40.3±1.4	45.5±0.8	47.3±1.3	48.1±1.3	49.1±1.4
	2	16.1±0.8	30.4±1.2	41.4±1.5	44.9±1.6	45.6±1.5	46.1±4.6	46.7±2.8
	4	16.7±0.7	29.4±2.3	40.5±3.3	44.4±2.5	47.3±4.0	46.8±3.9	46.3±3.4
	8	16.8±1.1	30.0±2.9	39.6±3.9	43.2±3.7	44.6±3.4	45.1±3.4	46.2±3.8
	16	17.6±1.7	31.4±3.2	40.0±3.4	43.3±3.3	45.7±3.2	47.0±4.3	47.4±4.1
	32	22.2±2.1	39.4±5.2	49.1±5.3	55.1±4.8	59.0±6.6	59.5±7.5	59.7±7.8
	64	34.0±4.0	53.9±7.5	66.0±6.7	74.3±10.1	76.2±8.0	75.9±10.5	77.8±8.4
	128	42.7±5.1	68.1±12.0	83.7±11.4	87.1±10.6	97.2±13.3	95.3±11.9	96.9±11.8
	256	68.2±8.4	74.2±14.0	95.8±10.0	121.8±7.9	128.3±8.9	127.9±6.6	133.9±4.9
	512	72.0±4.5	78.0±8.0	123.1±10.4	129.2±15.4	140.9±18.9	140.4±7.0	148.8±8.5
	1,024	92.7±5.6	93.0±7.4	123.7±11.2	155.3±10.6	163.3±11.0	166.0±10.1	159.5±12.8
25%	**1**	19.2±1.9	37.3±2.3	77.1±2.6	120.1±2.5	136.1±4.8	141.6±3.9	143.8±3.1
	2	17.7±3.3	42.9±2.2	82.2±3.1	121.1±3.8	136.0±4.7	134.2±20.7	138.0±15.2
	4	19.1±1.5	41.9±3.8	76.2±8.4	116.7±11.9	130.6±12.3	138.7±12.5	136.2±15.9
	8	19.8±2.2	45.4±5.4	84.8±11.5	116.6±12.8	132.0±11.4	137.3±15.6	139.1±15.5
	16	23.1±2.4	50.0±5.3	101.9±14.9	131.2±17.4	150.8±14.5	159.6±19.3	158.1±19.1
	32	27.5±3.4	56.5±9.5	112.5±18.1	148.5±22.8	167.3±16.9	172.8±27.2	172.1±28.9
	64	40.0±4.1	79.8±13.6	151.0±29.3	197.5±34.4	226.4±37.1	230.0±36.4	232.4±41.6
	128	50.5±10.0	112.1±22.0	219.6±47.1	281.9±47.7	326.3±57.0	339.5±56.4	344.4±61.3
	256	65.3±24.3	141.4±31.8	273.0±58.6	379.1±29.1	456.6±28.5	468.2±25.1	476.8±19.8
	512	66.3±20.7	178.4±75.2	375.0±29.4	469.5±27.4	548.1±18.7	609.9±32.6	591.2±23.8
	1,024	131.3±30.8	244.6±52.7	404.4±54.8	517.5±17.6	593.6±30.6	612.2±24.3	648.3±14.4
50%	**1**	16.4±1.5	39.7±2.5	93.5±3.4	201.7±5.8	279.1±4.1	303.1±4.6	305.5±4.7
	2	20.2±1.6	44.3±3.0	102.1±3.7	195.6±7.6	285.6±6.7	275.4±55.4	284.3±40.8
	4	19.4±1.4	42.6±4.7	98.5±12.5	183.4±23.8	263.1±36.0	280.1±45.4	288.0±45.6
	8	21.8±2.3	48.0±6.1	102.4±14.2	196.5±32.6	269.5±42.0	281.2±44.8	278.8±56.6
	16	22.4±3.5	52.4±6.1	111.2±23.7	222.9±33.6	290.9±43.2	304.6±46.2	302.4±49.5
	32	28.1±3.2	64.8±9.6	158.2±27.4	289.5±47.9	362.0±53.4	360.2±81.5	356.0±77.5
	64	37.9±6.4	81.1±14.4	198.4±42.5	366.4±90.5	447.5±96.3	462.6±94.2	464.0±89.0
	128	53.0±8.6	118.0±29.3	280.1±62.6	533.2±123.5	650.1±150.4	693.2±146.1	687.5±144.0
	256	78.4±31.3	158.0±36.6	408.2±100.3	808.7±23.9	1011.8±62.4	1101.3±36.4	1119.0±23.8
	512	62.4±19.5	297.3±15.1	595.7±42.7	1085.5±31.3	1306.4±38.0	1422.6±28.9	1456.5±34.3
	1,024	124.3±81.9	367.9±122.5	964.7±99.9	1241.0±22.7	1441.2±21.0	1523.4±36.5	1554.2±38.6
100%	**1**	17.2±1.3	41.9±3.2	98.4±6.8	208.9±5.7	331.6±6.9	373.7±6.7	378.3±4.8
	2	18.8±1.8	44.7±1.3	98.3±4.4	207.0±6.0	330.5±6.2	351.7±53.9	346.9±54.3
	4	18.7±2.0	42.4±4.1	96.7±13.5	195.3±27.1	310.4±45.9	345.5±61.7	348.3±62.7
	8	20.4±2.5	49.0±6.6	112.3±16.7	225.8±38.1	353.3±64.3	380.4±70.5	377.2±67.5
	16	23.4±2.5	53.1±5.9	119.9±18.1	235.1±36.1	350.2±59.2	374.3±61.9	365.7±64.0
	32	29.3±3.1	67.5±9.3	172.0±28.6	334.5±55.6	461.1±79.8	465.9±119.7	465.5±113.5
	64	40.0±5.1	84.4±13.4	210.3±54.5	436.6±110.7	600.7±150.3	622.1±145.7	611.5±142.8
	128	53.5±9.6	118.3±24.0	300.0±66.4	604.4±142.4	863.9±225.9	884.7±215.2	870.7±215.3
	256	65.7±30.6	178.3±40.6	462.5±24.3	850.2±32.6	1430.1±25.2	1558.2±40.3	1580.1±32.5
	512	77.9±32.8	280.2±46.0	615.5±38.7	1207.3±46.2	1964.4±76.2	2570.7±75.7	2630.9±98.1
	1,024	145.9±89.5	460.5±135.0	983.9±135.8	1557.4±27.5	2434.8±98.2	3072.8±48.3	3126.4±97.1

Appendix B

Additional Results

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	176.52±13.8	38.07±2.1
	Polynomial regression	144.74±7.6	69.42±14.0
	Linear SVR	155.09±4.9	36.56±1.9
	Gaussian Process	132.16±7.1	34.48±1.3
	Chorus*	**33.23±0.3**	N/A
	Multi-Task Gaussian Process*	34.94±2.1	N/A
10	Linear regression	133.80±5.9	35.72±1.3
	Polynomial regression	113.10±7.1	71.61±14.6
	Linear SVR	153.18±5.4	33.60±0.7
	Gaussian Process	73.34±7.6	**31.50±0.6**
	Chorus*	32.52±0.4	N/A
	Multi-Task Gaussian Process*	32.00±1.9	N/A

Table B.1: RMSE of the proposed Model Mapping technique and direct modeling technique with various modeling methods for **CPU quota** configurations (* indicates a direct modeling method that uses incremental or transfer learning). Legacy model **CPU quota**=50% and unknown model **CPU quota**=25%.

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	138.95±17.5	22.93±2.2
	Polynomial regression	103.51±10.4	22.51±2.0
	Linear SVR	101.20±10.4	**18.50±1.0**
	Gaussian Process	72.33±7.4	18.70±1.2
	Chorus*	19.08±0.2	N/A
	Multi-Task Gaussian Process*	20.36±1.7	N/A
10	Linear regression	99.58±10.2	20.06±1.3
	Polynomial regression	74.80±8.6	21.05±1.3
	Linear SVR	101.05±11.0	16.11±0.6
	Gaussian Process	30.28±2.2	**15.98±0.7**
	Chorus*	18.69±0.3	N/A
	Multi-Task Gaussian Process*	16.77±1.1	N/A

Table B.2: MAPE of the proposed Model Mapping technique and direct modeling technique with various modeling methods for **CPU quota** configurations (* indicates a direct modeling method that uses incremental or transfer learning). Legacy model **CPU quota**=50% and unknown model **CPU quota**=25%.

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	176.52±13.8	69.24±2.9
	Polynomial regression	144.74±7.6	57.59±5.4
	Linear SVR	155.09±4.9	66.99±2.1
	Gaussian Process	132.16±7.1	**49.10±1.1**
	Chorus*	53.65±0.4	N/A
	Multi-Task Gaussian Process*	63.60±3.0	N/A
10	Linear regression	133.80±5.9	66.99±3.2
	Polynomial regression	113.10±7.1	60.98±10.9
	Linear SVR	153.18±5.4	65.95±1.6
	Gaussian Process	73.34±7.6	**44.75±1.8**
	Chorus*	52.43±0.6	N/A
	Multi-Task Gaussian Process*	57.19±3.7	N/A

Table B.3: RMSE of the proposed Model Mapping technique and direct modeling technique with various modeling methods for **CPU quota** configurations (* indicates a direct modeling method that uses incremental or transfer learning). Legacy model **CPU quota**=10% and unknown model **CPU quota**=25%.

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	138.95±17.5	41.65±3.7
	Polynomial regression	103.51±10.4	**29.19±1.5**
	Linear SVR	101.20±10.4	39.37±2.9
	Gaussian Process	72.33±7.4	34.93±1.9
	Chorus*	45.09±0.7	N/A
	Multi-Task Gaussian Process*	42.66±4.6	N/A
10	Linear regression	99.58±10.2	40.53±2.4
	Polynomial regression	74.80±8.6	**26.14±1.4**
	Linear SVR	101.05±11.0	37.66±2.6
	Gaussian Process	30.28±2.2	29.36±1.4
	Chorus*	43.57±0.9	N/A
	Multi-Task Gaussian Process*	36.62±3.0	N/A

Table B.4: MAPE of the proposed Model Mapping technique and direct modeling technique with various modeling methods for **CPU quota** configurations (* indicates a direct modeling method that uses incremental or transfer learning). Legacy model **CPU quota**=10% and unknown model **CPU quota**=25%.

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	28.61±2.6	15.18±1.1
	Polynomial regression	29.67±1.6	17.84±2.1
	Linear SVR	25.33±1.4	14.82±0.7
	Gaussian Process	22.21±1.7	10.70±0.1
	Chorus*	**9.24±0.1**	N/A
	Multi-Task Gaussian Process*	13.88±0.9	N/A
10	Linear regression	20.50±0.9	14.27±1.1
	Polynomial regression	23.51±1.3	14.97±1.5
	Linear SVR	23.14±1.0	14.08±0.6
	Gaussian Process	11.26±1.5	10.44±0.1
	Chorus*	**9.07±0.1**	N/A
	Multi-Task Gaussian Process*	11.82±1.7	N/A

Table B.5: RMSE of the proposed Model Mapping technique and direct modeling technique with various modeling methods for **CPU quota** configurations (* indicates a direct modeling method that uses incremental or transfer learning). Legacy model **CPU quota**=25% and unknown model **CPU quota**=10%.

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	48.35±4.5	18.96±1.1
	Polynomial regression	46.63±4.0	17.48±1.0
	Linear SVR	40.76±2.2	18.16±1.6
	Gaussian Process	32.51±2.5	**15.01±0.5**
	Chorus*	16.23±0.2	N/A
	Multi-Task Gaussian Process*	17.23±0.8	N/A
10	Linear regression	34.97±1.9	18.34±0.8
	Polynomial regression	31.22±2.8	14.94±0.6
	Linear SVR	37.85±2.2	16.16±0.7
	Gaussian Process	16.20±2.2	**14.68±0.6**
	Chorus*	15.97±0.2	N/A
	Multi-Task Gaussian Process*	14.90±1.2	N/A

Table B.6: MAPE of the proposed Model Mapping technique and direct modeling technique with various modeling methods for **CPU quota** configurations (* indicates a direct modeling method that uses incremental or transfer learning). Legacy model **CPU quota**=25% and unknown model **CPU quota**=10%.

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	784.71±61.9	210.28±11.1
	Polynomial regression	612.73±30.6	527.84±113.9
	Linear SVR	652.57±18.4	209.48±11.8
	Gaussian Process	577.76±25.7	**171.14±5.6**
	Chorus*	200.36±3.2	N/A
	Multi-Task Gaussian Process*	197.08±11.6	N/A
10	Linear regression	601.57±28.5	201.84±11.5
	Polynomial regression	438.97±30.8	493.99±105.5
	Linear SVR	627.42±21.8	199.75±10.6
	Gaussian Process	348.77±36.3	176.68±5.0
	Chorus*	194.50±4.6	N/A
	Multi-Task Gaussian Process*	**168.13±12.0**	N/A

Table B.7: RMSE of the proposed Model Mapping technique and direct modeling technique with various modeling methods for **CPU quota** configurations (* indicates a direct modeling method that uses incremental or transfer learning). Legacy model **CPU quota**=50% and unknown model **CPU quota**=100%.

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	477.83±86.7	56.25±6.5
	Polynomial regression	321.31±33.7	43.09±3.2
	Linear SVR	281.00±39.3	42.28±6.4
	Gaussian Process	196.84±21.3	**37.70±4.9**
	Chorus*	53.64±1.0	N/A
	Multi-Task Gaussian Process*	60.68±13.6	N/A
10	Linear regression	344.68±54.9	47.90±3.4
	Polynomial regression	211.14±28.8	32.02±2.4
	Linear SVR	246.05±40.6	36.19±3.1
	Gaussian Process	76.65±14.8	**27.01±2.5**
	Chorus*	52.46±1.4	N/A
	Multi-Task Gaussian Process*	38.08±4.9	N/A

Table B.8: MAPE of the proposed Model Mapping technique and direct modeling technique with various modeling methods for **CPU quota** configurations (* indicates a direct modeling method that uses incremental or transfer learning). Legacy model **CPU quota**=50% and unknown model **CPU quota**=100%.

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	453.41±43.9	132.75±10.5
	Polynomial regression	351.60±17.6	406.84±105.8
	Linear SVR	385.55±10.7	119.09±8.7
	Gaussian Process	327.16±18.2	97.72±3.5
	Chorus*	**93.68±1.0**	N/A
	Multi-Task Gaussian Process*	110.38±7.5	N/A
10	Linear regression	336.60±14.6	120.57±8.4
	Polynomial regression	266.87±17.4	356.46±78.0
	Linear SVR	381.99±13.6	117.05±8.5
	Gaussian Process	181.91±19.3	96.23±2.0
	Chorus*	**91.89±1.5**	N/A
	Multi-Task Gaussian Process*	94.25±7.0	N/A

Table B.9: RMSE of the proposed Model Mapping technique and direct modeling technique with various modeling methods for **CPU quota** configurations (* indicates a direct modeling method that uses incremental or transfer learning). Legacy model **CPU quota**=100% and unknown model **CPU quota**=50%.

Sampling Budget	Modeling Method	Modeling Technique	
		Direct	Mapping
5	Linear regression	289.55±44.6	38.42±4.3
	Polynomial regression	206.96±20.9	34.84±3.2
	Linear SVR	188.59±22.4	**25.53±2.9**
	Gaussian Process	133.10±14.0	27.14±1.9
	Chorus*	31.16±0.5	N/A
	Multi-Task Gaussian Process*	35.80±5.1	N/A
10	Linear regression	202.51±24.9	32.04±2.4
	Polynomial regression	153.54±19.6	28.63±1.9
	Linear SVR	182.16±24.5	24.76±1.6
	Gaussian Process	50.08±4.5	**22.24±1.7**
	Chorus*	30.29±0.7	N/A
	Multi-Task Gaussian Process*	25.26±2.8	N/A

Table B.10: MAPE of the proposed Model Mapping technique and direct modeling technique with various modeling methods for **CPU quota** configurations (* indicates a direct modeling method that uses incremental or transfer learning). Legacy model **CPU quota**=100% and unknown model **CPU quota**=50%.

Bibliography

[1] Amazon Web Services (AWS). https://aws.amazon.com.

[2] Eric Anderson. Simple table-based modeling of storage devices. Technical report, Technical Report HPL-SSP-2001-04, HP Laboratories, 2001.

[3] Danilo Ardagna, Giuliano Casale, Michele Ciavotta, Juan F Pérez, and Weikun Wang. Quality-of-service in cloud computing: modeling techniques and their applications. *Journal of Internet Services and Applications*, 5(1):11, 2014.

[4] Danilo Ardagna, Barbara Panicucci, Marco Trubian, and Li Zhang. Energy-aware autonomic resource allocation in multitier virtualized environments. *IEEE transactions on services computing*, 5(1):2–19, 2010.

[5] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Multi-task feature learning. In *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*, volume 19, page 41. The MIT Press, 2007.

[6] Martin Arlitt, Diwakar Krishnamurthy, and Jerry Rolia. Characterizing the scalability of a large web-based shopping system. *ACM Transactions on Internet Technology*, 1(1):44–69, 2001.

[7] Sara S. Baghsorkhi, Matthieu Delahaye, Sanjay J. Patel, William D. Gropp, and Wen-mei W. Hwu. An adaptive performance modeling tool for gpu architectures. *SIGPLAN Not.*, 45(5):105–114, January 2010.

[8] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *ACM SIGOPS operating systems review*, 37(5):164–177, 2003.

[9] Luiz André Barroso and Urs Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis lectures on computer architecture*, 4(1):1–108, 2009.

[10] Mohamed N. Bennani and Daniel A. Menascé. Resource Allocation for Autonomic Data Centers using Analytic Performance Models. In *ICAC'05*, pages 229–240, 2005.

[11] Sergio Bermejo and Joan Cabestany. Oriented principal component analysis for large margin classifiers. *Neural Networks*, 14(10):1447–1461, 2001.

[12] Dimitri P Bertsekas, Robert G Gallager, and Pierre Humblet. *Data networks*, volume 2. Prentice-Hall International New Jersey, 1992.

[13] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[14] Edwin V Bonilla, Kian M Chai, and Christopher Williams. Multi-task gaussian process prediction. In *Advances in neural information processing systems*, pages 153–160, 2008.

[15] George EP Box, William Gordon Hunter, J Stuart Hunter, et al. *Statistics for experimenters*, volume 664. John Wiley and sons New York, 1978.

[16] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

[17] Randal E Bryant. *Computer Systems: A Programmer's Perspective*. Pearson Education Limited, second edition, 2010.

[18] Sean Campbell and Michael Jeronimo. An introduction to virtualization. *Published in "Applied Virtualization", Intel*, pages 1–15, 2006.

[19] Zhen Cao, Vasily Tarasov, Sachin Tiwari, and Erez Zadok. Towards better understanding of black-box auto-tuning: A comparative analysis for storage systems. dataset. http://download.filesystems.org/auto-tune/ATC-2018-auto-tune-data.sql.gz, 2008. [Online; accessed 2020-04-30].

[20] Zhen Cao, Vasily Tarasov, Sachin Tiwari, and Erez Zadok. Towards better understanding of black-box auto-tuning: A comparative analysis for storage systems. In *2018 {USENIX} Annual Technical Conference ({USENIX} {ATC} 18)*, pages 893–907, 2018.

[21] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[22] Linux Control Groups. http://man7.org/linux/man-pages/man7/cgroups.7.html. Online; accessed 2017-06-01.

[23] Haifeng Chen, Wenxuan Zhang, and Guofei Jiang. Experience transfer for the configuration tuning in large-scale computing systems. *IEEE Transactions on Knowledge and Data Engineering*, 23(3):388–401, 2010.

[24] J. Chen, S. Ghanbari, G. Soundararajan, F. Iorio, A. B. Hashemi, and C. Amza. Ensemble: A tool for performance modeling of applications in cloud data centers. *Cloud Computing, IEEE Transactions on*, 2015.

[25] Jin Chen. *Chorus: Model Knowledge Base for Performance Modeling in Datacenters*. PhD thesis, University of Toronto, 2011.

[26] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[27] Kemal A Delic, Jeff A Riley, Claudio Bartolini, and Adnan Salihbegovic. Knowledge-based self-management of apache web servers. *Resource*, 2:3, 2007.

[28] Chuong B Do and Andrew Y Ng. Transfer learning for text classification. In *Advances in Neural Information Processing Systems*, pages 299–306, 2006.

[29] Songyun Duan, Vamsidhar Thummala, and Shivnath Babu. Tuning database configuration parameters with ituned. *PVLDB*, 2(1):1246–1257, 2009.

[30] Sameh Elnikety, Steven Dropsho, Emmanuel Cecchet, and Willy Zwaenepoel. Predicting Replicated Database Scalability from Standalone Database Profiling. In *Proceedings of the 4th ACM European Conference on Computer Systems*, EuroSys '09, pages 303–316, New York, NY, USA, 2009. ACM.

[31] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Transfer learning for time series classification. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 1367–1376. IEEE, 2018.

[32] Robert Fildes. The evaluation of extrapolative forecasting methods. *International Journal of Forecasting*, 8(1):81–98, 1992.

[33] Marcus Frean and Phillip Boyle. Using Gaussian Processes to Optimize Expensive Functions. In Wayne Wobcke and Mengjie Zhang, editors, *AI 2008: Advances in Artificial Intelligence SE - 25*, volume 5360 of *Lecture Notes in Computer Science*, pages 258–267. Springer Berlin Heidelberg, 2008.

[34] A. Gambi, M. Pezze, and G. Toffetti. Kriging-based self-adaptive cloud controllers. *Services Computing, IEEE Transactions on*, PP(99):1–1, 2015.

[35] Alex Gammerman, Volodya Vovk, and Vladimir Vapnik. Learning by transduction. *arXiv preprint arXiv:1301.7375*, 2013.

[36] A. Ganapathi, Yanpei Chen, A. Fox, R. Katz, and D. Patterson. Statistics-driven workload modeling for the cloud. In *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, pages 87–92, March 2010.

[37] Archana Ganapathi, Harumi A. Kuno, Umeshwar Dayal, Janet L. Wiener, Armando Fox,

Michael I. Jordan, and David A. Patterson. Predicting Multiple Metrics for Queries: Better Decisions Enabled by Machine Learning. In *ICDE'09*, pages 592–603, 2009.

[38] Jochen Garcke and Thomas Vanck. Importance Weighted Inductive Transfer Learning for Regression. In Toon Calders, Floriana Esposito, Eyke Hüllermeier, and Rosa Meo, editors, *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2014, Nancy, France, September 15-19, 2014. Proceedings, Part I*, pages 466–481. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[39] Seymour Geisser. The predictive sample reuse method with applications. *Journal of the American statistical Association*, 70(350):320–328, 1975.

[40] Saeed Ghanbari, Gokul Soundararajan, and Cristiana Amza. A Query Language and Runtime Tool for Evaluating Behavior of Multi-tier Servers. In *SIGMETRICS'10*, pages 131–142, 2010.

[41] Github. https://github.com/.

[42] Google Cloud. https://cloud.google.com.

[43] Kasthurirangan Gopalakrishnan, Siddhartha K Khaitan, Alok Choudhary, and Ankit Agrawal. Deep convolutional neural networks with transfer learning for computer vision-based data-driven pavement distress detection. *Construction and Building Materials*, 157:322–330, 2017.

[44] GPy GPy. A gaussian process framework in python. https://sheffieldml.github.io/GPy/, 2012.

[45] Ajay Gulati, Chethan Kumar, Irfan Ahmad, and Karan Kumar. Basil: Automated io load balancing across storage devices. In *Fast*, volume 10, pages 13–13, 2010.

[46] Irfan Habib. Virtualization with kvm. *Linux Journal*, 2008(166):8, 2008.

[47] Jens Happe, Hui Li, and Wolfgang Theilmann. Black-box performance models: prediction based on observation. In *Proceedings of the 1st international workshop on Quality of service-oriented software systems*, pages 19–24. ACM, 2009.

[48] Herodotos Herodotou and Shivnath Babu. A what-if engine for cost-based mapreduce optimization. *IEEE Data Eng. Bull.*, 36(1):5–14, 2013.

[49] Itu h.264 : Advanced video coding for generic audiovisual services. https://www.itu.int/rec/T-REC-H.264. [Online; accessed 2020-04-30].

[50] Pooyan Jamshidi, Miguel Velez, Christian Kästner, Norbert Siegmund, and Prasad Kawthekar. Transfer learning for improving model predictions in highly configurable software. In *Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 31–41. IEEE Press, 2017.

[51] Nobuaki Kimura, Ikuo Yoshinaga, Kenji Sekijima, Issaku Azechi, and Daichi Baba. Convolutional neural network coupled with a transfer-learning approach for time-series flood predictions. *Water*, 12(1):96, 2020.

[52] Wouter M Kouw and Marco Loog. An introduction to domain adaptation and transfer learning. *arXiv preprint arXiv:1812.11806*, 2018.

[53] Jakub Krzywda, Ahmed Ali-Eldin, Eddie Wadbro, Per-Olov Östberg, and Erik Elmroth. Power shepherd: Application performance aware power shifting. In *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 45–53, 2019.

[54] Paul J Kuehn. Energy effciency and performance of cloud data centers: which role can modeling play? In *Proceedings of the 5th International Workshop on Energy Efficient Data Centres*, pages 1–6, 2016.

[55] Brian Kulis, Kate Saenko, and Trevor Darrell. What you saw is not what you get: Domain adaptation using asymmetric kernel transforms. In *CVPR 2011*, pages 1785–1792. IEEE, 2011.

[56] Peter A Lachenbruch and M Ray Mickey. Estimation of error rates in discriminant analysis. *Technometrics*, 10(1):1–11, 1968.

[57] Pei Ling Lai and Colin Fyfe. Kernel and nonlinear canonical correlation analysis. *International Journal of Neural Systems*, 10(05):365–377, 2000.

[58] Neil D Lawrence and John C Platt. Learning to learn with the informative vector machine. In *Proceedings of the twenty-first international conference on Machine learning*, page 65. ACM, 2004.

[59] Octavio Loyola-Gonzalez. Black-box vs. white-box: Understanding their advantages and weaknesses from a practical point of view. *IEEE Access*, 7:154096–154113, 2019.

[60] Zaigham Mahmood and Saqib Saeed. *Software engineering frameworks for the cloud computing paradigm*. Springer, 2013.

[61] Spyros Makridakis. Accuracy measures: theoretical and practical concerns. *International journal of forecasting*, 9(4):527–529, 1993.

[62] MariaDB. https://mariadb.org/. Online; accessed 2017-07-31.

[63] MariaDB versus MySQL. https://mariadb.com/kb/en/mariadb/mariadb-vs-mysql-compatibility. Online; accessed 2017-06-17.

[64] N Ani Brown Mary and K Jayapriya. An extensive survey on qos in cloud computing. *International Journal of Computer Science and Information Technologies*, 5(1):1–5, 2014.

[65] Geoffrey McLachlan. *Discriminant analysis and statistical pattern recognition*, volume 544. John Wiley & Sons, 2004.

[66] Microsoft Azure. https://azure.microsoft.com.

[67] Modelmap model mapping toolkit. https://github.com/frioglobal/ModelMap, 2020. [Online; accessed 2020-06-24].

[68] Annette M Molinaro, Richard Simon, and Ruth M Pfeiffer. Prediction error estimation: a comparison of resampling methods. *Bioinformatics*, 21(15):3301–3307, 2005.

[69] Jaechang Nam, Wei Fu, Sunghun Kim, Tim Menzies, and Lin Tan. Heterogeneous defect prediction. *IEEE Transactions on Software Engineering*, 44(9):874–896, 2017.

[70] David Oppenheimer, Archana Ganapathi, and David A Patterson. Why do internet services fail, and what can be done about it? In *USENIX Symposium on Internet Technologies and Systems*, volume 67. Seattle, WA, 2003.

[71] Pradeep Padala, Kang G Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, Arif Merchant, and Kenneth Salem. Adaptive control of virtualized resources in utility computing environments. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, pages 289–302, 2007.

[72] Sinno Jialin Pan. Transfer Learning. In Charu C Aggarwal, editor, *Data Classification: Algorithms and Applications*, pages 537–570. {CRC} Press, 2014.

[73] Sinno Jialin Pan, Qiang Yang, et al. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[74] pandas - python data analysis library. https://pandas.pydata.org.

[75] David Pardoe and Peter Stone. Boosting for regression transfer. In *Proceedings of the 27th international conference on Machine learning (ICML-10)*, pages 863–870, 2010.

[76] M. Pedram. Energy-efficient datacenters. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(10):1465–1484, 2012.

[77] Pep 8 - style guide for python. https://www.python.org/dev/peps/pep-0008.

[78] Richard R Picard and R Dennis Cook. Cross-validation of regression models. *Journal of the American Statistical Association*, 79(387):575–583, 1984.

[79] Python 3. https://www.python.org.

[80] Asfandyar Qureshi, Rick Weber, Hari Balakrishnan, John Guttag, and Bruce Maggs. Cutting the electric bill for internet-scale systems. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 123–134, 2009.

[81] Peter J Rousseeuw and Mia Hubert. Robust statistics for outlier detection. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(1):73–79, 2011.

[82] S Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674, 1991.

[83] David Salomon. *Data compression: the complete reference*. Springer Science & Business Media, 2004.

[84] Andrea Saltelli, Paola Annoni, Ivano Azzini, Francesca Campolongo, Marco Ratto, and Stefano Tarantola. Variance based sensitivity analysis of model output. design and estimator for the total sensitivity index. *Computer physics communications*, 181(2):259–270, 2010.

[85] scikit-learn - Machine Learning in Python. http://scikit-learn.org/stable.

[86] Burr Settles. Active learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–114, 2012.

[87] Jun Shao. Linear model selection by cross-validation. *Journal of the American statistical Association*, 88(422):486–494, 1993.

[88] Richard Simon. Resampling strategies for model assessment and selection. In *Fundamentals of data mining in genomics and proteomics*, pages 173–186. Springer, 2007.

[89] Alex Smola and Vladimir Vapnik. Support vector regression machines. *Advances in neural information processing systems*, 9:155–161, 1997.

[90] Alex J. Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, 2004.

[91] Ahmed A Soror, Umar Farooq Minhas, Ashraf Aboulnaga, Kenneth Salem, Peter Kokosielis, and Sunil Kamath. Automatic virtual machine configuration for database workloads. *ACM Transactions on Database Systems (TODS)*, 35(1):1–47, 2008.

[92] Gokul Soundararajan, Jin Chen, Mohamed A Sharaf, and Cristiana Amza. Dynamic partitioning of the cache hierarchy in shared data centers. *Proceedings of the VLDB Endowment*, 1(1):635–646, 2008.

[93] Gokul Soundararajan, Daniel Lupei, Saeed Ghanbari, Adrian Daniel Popescu, Jin Chen, and Cristiana Amza. Dynamic resource allocation for database servers running on virtual storage. In *FAST*, pages 71–84, 2009.

[94] Sqlite. https://www.sqlite.org/. [Online; accessed 2020-04-30].

[95] Shekhar Srikantaiah, Aman Kansal, and Feng Zhao. Energy aware consolidation for cloud computing. In *Proceedings of USENIX workshop on power aware computing and systems in conjunction with OSDI, ACM*, pages 1–5, 2008.

[96] Starfish: Self-tuning analytics system. http://www2.cs.duke.edu/starfish/.

[97] Michael L Stein. *Interpolation of spatial data: some theory for Kriging*. Springer Science & Business Media, 2012.

[98] Stephen C. Strother, Jon Anderson, Lars Kai Hansen, Ulrik Kjems, Rafal Kustra, John Sidtis, Sally Frutiger, Suraj Muley, Stephen LaConte, and David Rottenberg. The quantitative evaluation of functional neuroimaging experiments: The NPAIRS data analysis framework. *NeuroImage*, 15(4):747–771, 2002.

[99] Masashi Sugiyama, Shinichi Nakajima, Hisashi Kashima, Paul V Buenau, and Motoaki Kawanabe. Direct importance estimation with model selection and its application to covariate shift adaptation. In *Advances in neural information processing systems*, pages 1433–1440, 2008.

[100] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. In *International conference on artificial neural networks*, pages 270–279. Springer, 2018.

[101] Vasily Tarasov, Erez Zadok, and Spencer Shepler. Filebench: A flexible framework for file system benchmarking. *USENIX; login*, 41(1):6–12, 2016.

[102] The tukaani project. xz utils. http://tukaani.org/xz/. [Online; accessed 2020-04-30].

[103] Eno Thereska and Gregory R. Ganger. Ironmodel: robust performance models in the wild. In *SIGMETRICS*, pages 253–264, 2008.

[104] Huaglory Tianfield. Cloud computing architectures. In *2011 IEEE International Conference on Systems, Man, and Cybernetics*, pages 1394–1399. IEEE, 2011.

[105] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI Global, 2010.

[106] TPC-C. http://www.tpc.org/tpcc/. Online; accessed 2017-06-01.

[107] TPC-C. http://www.tpc.org/tpcc/faq.asp. Online; accessed 2017-06-01.

[108] TPC-C Benchmark implementation for MySQL. https://github.com/Percona-Lab/tpcc-mysql (Commit: 1ec1c5eb5b11b55ecf26f81a74db86b659c4e7b9 [1ec1c5e], Date: Friday, January 20, 2017 2:41:50 PM). Online; accessed 2017-06-01.

[109] Tpc-c performance dataset. https://github.com/frioglobal/ModelMap/tree/master/Datasets, 2020. [Online; accessed 2020-04-30].

[110] TPC-DS. http://www.tpc.org/tpcds/. Online; accessed 2017-06-01.

[111] TPC-W. http://www.tpc.org/tpcw/. Online; accessed 2017-06-01.

[112] Gerard V Trunk. A problem of dimensionality: A simple example. *IEEE Transactions on pattern analysis and machine intelligence*, 3(3):306–307, 1979.

[113] Wei-Tek Tsai, Xin Sun, and Janaka Balasooriya. Service-oriented cloud computing architecture. In *2010 seventh international conference on information technology: new generations*, pages 684–689. IEEE, 2010.

[114] Bhuvan Urgaonkar, Giovanni Pacifici, Prashant J. Shenoy, Mike Spreitzer, and Asser N. Tantawi. An analytical model for multi-tier internet services and its applications. In *SIGMETRICS*, pages 291–302, 2005.

[115] Mustafa Uysal, Guillermo A. Alvarez, and Arif Merchant. A modular, analytical throughput model for modern disk arrays. In *MASCOTS*, pages 183–192, 2001.

[116] Pavel Valov, Jean-Christophe Petkovich, Jianmei Guo, Sebastian Fischmeister, and Krzysztof Czarnecki. Transferring performance prediction models across different hardware platforms. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*, pages 39–50, 2017.

[117] Pavel Valov, Jean-Christophe Petkovich, Jianmei Guo, Sebastian Fischmeister, and Krzysztof Czarnecki. Transferring performance prediction models across differ-

ent hardware platforms. https://bitbucket.org/valovp/icpe2017/src/master/, 2017. [Online; accessed 2020-04-30].

[118] Dana Van Aken, Andrew Pavlo, Geoffrey J Gordon, and Bohan Zhang. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1009–1024, 2017.

[119] Peter JM Van Laarhoven and Emile HL Aarts. Simulated annealing. In *Simulated annealing: Theory and applications*, pages 7–15. Springer, 1987.

[120] Thomas Vanck. *New importance sampling based algorithms for compensating dataset shifts*. Doctoral thesis, Technische Universität Berlin, 2016.

[121] Videolan organization. x264, the best h.264/avc encoder. http://www.videolan.org/developers/x264.html. [Online; accessed 2020-04-30].

[122] Carl A Waldspurger. Memory resource management in vmware esx server. *ACM SIGOPS Operating Systems Review*, 36(SI):181–194, 2002.

[123] Mengzhi Wang, Kinman Au, Anastassia Ailamaki, Anthony Brockwell, Christos Faloutsos, and Gregory R. Ganger. Storage device performance prediction with cart models. *SIGMETRICS Perform. Eval. Rev.*, 32(1):412–413, June 2004.

[124] Chih wei Hsu, Chih chung Chang, and Chih jen Lin. A practical guide to support vector classification. http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf, 2010.

[125] Christopher KI Williams. Prediction with gaussian processes: From linear regression to linear prediction and beyond. In *Learning in graphical models*, pages 599–621. Springer, 1998.

[126] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

[127] Catherine Wong, Neil Houlsby, Yifeng Lu, and Andrea Gesmundo. Transfer learning with neural automl. In *Advances in Neural Information Processing Systems*, pages 8356–8365, 2018.

[128] Qing-Song Xu and Yi-Zeng Liang. Monte carlo cross validation. *Chemometrics and Intelligent Laboratory Systems*, 56(1):1–11, 2001.

[129] Zuoning Yin, Xiao Ma, Jing Zheng, Yuanyuan Zhou, Lakshmi N Bairavasundaram, and Shankar Pasupathy. An empirical study on configuration errors in commercial and open source systems. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 159–172, 2011.

[130] Wei Zheng, Ricardo Bianchini, and Thu D Nguyen. Automatic configuration of internet services. *ACM SIGOPS Operating Systems Review*, 41(3):219–229, 2007.