Transfer Learning for Cross-Model Regression in Performance Modeling for the Cloud

Francesco Iorio*, Ali B. Hashemi†, Michael Tao‡ and Cristiana Amza§
*‡*Department of Computer Science*
University of Toronto, Toronto, Canada
Email: *francesco.iorio@mail.utoronto.ca, ‡mtao@dgp.toronto.edu
†*Autodesk Research*, Toronto, Canada
Email: ali.hashemi@autodesk.com
§*Department of Electrical and Computer Engineering*
University of Toronto, Toronto, Canada
Email: amza@ece.utoronto.ca

***Abstract*—Performance characteristics of a complex system in different configurations are expensive to obtain due to the cost of sampling the system performance. We introduce *ModelMap*, a novel transfer learning technique for the performance modeling of configurable systems. *ModelMap* captures many explicit and latent types of dynamic system evolution, including configuration changes, scaling and hardware upgrades, by deriving and modeling directly 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, and to interpolate across legacy models to build new models with no new samples at all. We experimentally test our method on a variety of system performance modeling and optimization scenarios. 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 between 0% to 5% of an exhaustive sampling budget. We also 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.**

***Index Terms*—Performance modeling, incremental modeling, transfer learning.**

I. Introduction

Cloud platforms, such as Amazon AWS and Google Cloud, are increasingly used for co-hosting several applications in a highly dynamic, large scale environment. Because they operate at large scales, it is common for cloud providers to optimize the operating costs due to power and cooling by multiplexing the heterogeneous resources of a cloud data center across many applications. At the same time, cloud providers must respect the Quality of Service (QoS) of each application. This strategy is called resource consolidation, and it is usually implemented in virtualized environments.

One of the most useful tools towards resource consolidation is modeling the performance of each application based on how its resource allocation is configured. A performance model is a mathematical function which maps resource configurations to application performance. Specifically, each sample point of the model represents the application performance measured given resource quotas applied to the application, e.g., memory, CPU, disk bandwidth. Performance models are typically costly to build, in the order of days or months.

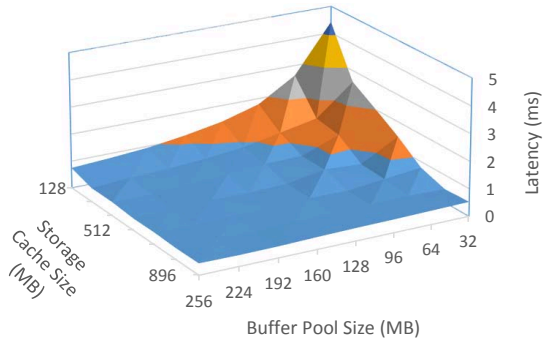


Fig. 1. Graph of database performance *TPC-C*.

The main bottleneck of building a performance model is the sampling time. The sampling time depends on the number of resources modeled, the number of resource quotas sampled for each resource, and other artifacts of ensuring statistical significance per sample taken, such as, the cache warm-up time.

For example, Figure 1 shows a model of the latency of the TPC-C transactional benchmark [1] when running within a virtualized environment where various quotas are allocated for two resources, *storage cache* and *buffer pool*. Building this model by exhaustive sampling of the configuration space required actuating the system to provide the necessary virtual quotas to the application, waiting for cache warm-up and performing repeated performance measurements to ensure statistical significance. Sampling time for building the full model took around 10 days. The greater the number of resources modeled is, the greater the number of dimensions of the model and the higher the cost for sampling for building the model

2330-2186/19/$31.00 ©2019 IEEE
DOI 10.1109/CloudCom.2019.00015

will be. In contrast, the computational cost for building the model is negligible.

Over time a cloud system undergoes software and hardware replacements, extensions and upgrades. Any incremental, but significant, variations of its configuration may render legacy performance models obsolete. Specifically, there are situations where one may wish to incorporate new resource types in a system or application, or situations that would require changing the dimensionality of the system's resource types. For example, one could add a model of the influence of imposing CPU quotas on the existing model of the latency of the storage hierarchy that was shown in Figure 1.

The traditional approach is to build a new model from scratch in any new configuration. It would be a direct model of the application or system in the new configuration based on all of the sampling data available e.g., using black-box [2] or gray-box models [3].

In contrast, we are investigating the reuse and incremental modification of legacy performance models in new resource or application configurations of a cloud system in order to reduce model building costs upon incremental system configuration changes.

In this paper, we introduce a novel incremental modeling technique, called *ModelMap*, 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, 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 variation on 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.

The *ModelMap* technique belongs to the instance-based category of homogeneous transfer learning [4]. Recently multiple attempts to exploit transfer learning techniques [5] have been investigated as a way to reuse prior knowledge to build black-box models of configurable systems.

Our experiments indicate how dynamic variations of systems affect their performance globally across the entire configuration space, rather than in small portions of the configuration space, which imply 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 a direct build of the new model from scratch. Therefore, to get the best of both worlds, we 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.

On the other hand, another important benefit of tracking system and application evolution through a library of historical mapping functions is the additional ability to derive trends and interpolate across existing models, even partial or with different cardinality. In this way, besides obtaining new models with a low sampling budget, we may be able to derive a new model or answer what-if scenarios *with no new samples at all*! We claim that 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 show the usefulness of our approach for the following model mapping scenarios: A) varying the CPU quota from 25% to 50% in an existing 2D model, as part of extending this 2D model to a 3D model, B) modeling the performance variation of a database as it grows in size, C) extrapolating the performance trend for a larger database size based on previously built models of two smaller database sizes and D) resource allocation for optimizing Virtual Machine packing.

In all of these scenarios, we find that with a sampling budget of only between 0% and 5% of the required samples needed for exhaustive sampling, our method achieves higher accuracy by up to an order of magnitude than the highest performing direct modeling methods and the Chorus incremental modeling technique [6], [7].

II. Related Work

Several modeling techniques exist that explicitly exploit previously acquired knowledge of the systems they model to predict the effects of configuration changes.

A. Semantic Aware Models

Chen et al. [7] extend the SelfTalk approach [8], a query language for encoding data relationships, to the area of performance modeling. They exploit model templates to express performance models and system structure information, and use 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 to proactively trim the configuration space.

While Semantic Aware techniques leverage prior knowledge, they also require expert user guidance to provide specific information, while our technique is entirely black-box and does not rely on this expert information to operate.

B. Transfer Learning

The context for transfer learning is more abstract than that of system models, but rather on *functions*. However, these methods are straightforward to apply to system modeling by just treating each model as a function. There are several disparate categories of approaches:

Instance-based: A subset of the available samples from a legacy function is added to the limited set of samples available for the unknown function. This enriched sample set for the unknown function is used to directly build a model of the unknown function. As an example of instance-based methods, Garcke and Vanck [9], [10] proposed a technique that considers shifts in the domain and co-domain of an unknown function. In their technique, they reduce the bias introduced from legacy samples by assigning each sample a weight. These weights correspond to the estimated similarity of a sample of the legacy function to the existing sample set of the unknown function. The weights are measured by the influence of each sample on the predictive quality of the unknown function. Chen et al. [7] introduced a method that combined transfer learning and active learning [11] using Gaussian Processes to reuse prior information in order to reduce the time required to train a performance model. Similar to other instance-based transfer learning techniques, they assume that there already exists a small sample set of the unknown model. Our proposed technique, in contrast, does not have this assumption and can start from an empty sample set from the unknown function, when multiple legacy models are available.

Feature-representation-based: A shared feature space is created from the limited shared features 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 [12] are among the transfer learning methods that follow this approach. Multi-task learning utilizes multiple tasks with a sparse set of labeled training data for each task in order to jointly learn individual classifiers for different tasks [13]. As an example, the work of Jamshidi et al. [14] used either a linear or a non-linear model (a Gaussian Process) to represent the correlation between different configurations of a configurable software system.

Model-parameter-based: Different from the previous two approaches that try to find shared knowledge in data, the goal of these methods is to find the shared knowledge hidden between the parameters of the models of legacy and unknown functions [15].

The transfer learning techniques reported above do not provide an explanation for the choice of non-linear transfer function modeling and only justify themselves by saying that they had been found to perform well. They also always rely on available samples of the unknown system configuration to build the model.

In contrast, our proposed technique does not depend on this assumption and it can start from as little as an empty sample set and extrapolate across the legacy models to predict the effects of the unknown system configuration. Additionally, in the works employing Gaussian Processes as the basis for transfer learning, the kernel choice is typically not specified, which we found to have a substantial effect on modeling accuracy and to require a form of model selection on its own.

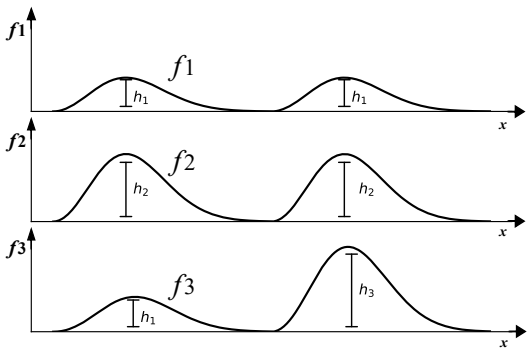


Fig. 2. Graph of the performance function $f1$, with two incremental variations $f2$ and $f3$.

III. Model Mapping

As noted earlier, 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. With this notation, 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 could 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 of a system, which we deem the *legacy model* ρ^0 and take to be trustworthy throughout $\mathcal{C}$. Rather than requiring a costly sampling 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. That is precisely what our proposed technique does, by building a *map* (transformation) σ such that

$$\rho^1 := \sigma \circ \rho^0, \tag{1}$$

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 2 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 2. 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. 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 $f2$:

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

$$f2 := \sigma \circ f1 := 2 \times f1. \tag{3}$$

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 2. 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 $f3$.

Beyond the example we have given, our motivation for computing a mapping transformation applies to other contexts. For instance, multiple performance models would be required to continuously optimize configurations of an evolving system. We claim that, rather than exhaustively sample and build a model for each passing system from scratch, a more effective and efficient strategy would be to build a map to obtain a new model of the evolving system from historically built models.

Even for a single static system, when there are a large number of configuration parameters, the curse of dimensionality makes it infeasible to create a single performance model over the entire configuration space. In such cases, multiple models may be required to capture the performance of different subspaces of configuration space, when a subset of parameters is held still. After a trustworthy model is built with a subset of static parameters, it would be prudent to generalize the model to more parameters to avoid an exhaustive sampling of the configuration space.

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.

IV. Direct and incremental models used for comparison

We performed a variety of experiments against a selection of direct and incremental modeling methods to evaluate the effectiveness of our model mapping technique:

- Linear Regression (LinearRegression) [16]
- Support Vector Regression with Linear kernel (LinearSVR) [16], [17]
- Support Vector Regression with Radial Basis Function kernel (RbfSVR) [16]
- Gaussian Process Regression with Radial Basis Function kernel (GPR) [16], [18]
- Chorus Incremental Regression (Chorus) [6], [7]

In order to measure the accuracy of a model we use Root Mean Squared Error as the error metric:

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n} \left(\hat{y_i} - y_i\right)^2}{n}} \tag{4}$$

where, y_i is the observed value of sample i, $\hat{y_i}$ is the predicted value obtained from the model, and n is the number of samples in the test set.

For GPR models, after performing model selection, we used the following compound kernel:

```
ConstantKernel(1.0, (1e-3, 1e3))
                *RBF(10, (1e-2, 1e2)).
```

For the LinearRegression and LinearSVR models, we used the default parameters provided by the Scikit.learn 0.19.1 [19]. RbfSVR is svm.SVR with the Radial Basis Function kernel (`kernel='rbf'`), for which, after experimentation with the kernel parameters, we settled on the following parameters: `C=1e3`, `gamma=0.1`.

A. Chorus incremental performance modeling framework

Chen et al. introduced Chorus [6], [7], an incremental performance modeling framework capable of storing, retrieving and reusing models for the purpose of performance evaluations and resource allocation in datacenter application deployments. We use Chorus as a baseline for an incremental performance model in our experiments, as it was reported to show substantial savings in training time compared to traditional modeling approaches, including the modeling of single and multiple concurrent workloads.

The fundamental modeling technique underpinning Chorus consists of subdividing a system's configuration space into a regular grid of regions. These regions are used to fit 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, Gray-Box and Black-box.

Chorus performs incremental modeling by extending a pre-existing legacy model with new samples. These new samples augment the pre-existing system's dataset and re-fit the Chorus ensemble using the new dataset. As reported in the Chorus experiments [6], [7] the more similar the behavior of the two systems, the fewer the number of samples required to obtain an accurate representation of the new system's behavior.

B. Considerations on modeling methods

Some modeling methods, such as Gaussian Processes have the ability to perform a smart selection of the sampling locations based on measuring the model's uncertainty. This smart selection allows for faster convergence with small sampling budgets. However, in order to emulate situations such as real-world cloud platforms, we do not assume the user has the ability to systematically choose the sampling locations in our experiments, and therefore select samples from a random distribution of the configuration space for all modeling methods.

Chorus operates on the concept of domain regions, and builds these regions based on a pre-existing, completely defined domain. This information is not always available for the function's codomain, and therefore Chorus cannot be used to

model our maps, where the range of the codomain features is not known in advance.

For the purpose of incremental modeling, Chorus models are pre-trained using a legacy dataset; in this work we applied Chorus as specified in related work [6], [7], and for all of the experiments this legacy model has been used to perform the baseline training.

A function describing a system's performance and transformation map representing a system's incremental evolution over a previous configuration, operate on different domains and are therefore different categories of functions, often with different dimensionality. It is therefore believable that a modeling method such as Linear Regression, which is easily imagined to be poorly suited for directly modeling a system's performance function, may be fastidious choice of transformation map when trying to accurately model a system's performance function with few samples. Similarly, advanced modeling methods like Gaussian Processes, which can represent complex, multi-dimensional functions effectively, score poorly when the function to be modeled is simple and only few samples are available.

Although we present a variety of modeling techniques, the task of choosing of the most effective modeling method for any given situation is beyond the scope of this project, so we do not perform any automatic model selection, which will be addressed in future work.

V. Experimental Platform

In order to evaluate the effectiveness of our proposed modeling technique, a virtualized platform was configured. The goal was to capture the performance model by monitoring the application's behavior. TPC-C benchmark [1] was used as the scrutinized application. The TPC-C benchmark aims to approximate the running conditions of a database system underpinning a business that operates sales in multiple regions and serving customers from a number of warehouses. Warehouses serve 10 regions each, while every region serves 3000 customers. The performance benchmark uses a variety of transactions operated by a traditional sales operation.

To obtain a sufficiently complex dataset and properly evaluate the applicability of our technique to a real-world scenario, we set up a fully instrumented system to reliably perform a multitude of performance measurements over the course of five months. To this effect we established a complete environment to reproduce the behavior of a standard cloud application leveraging a database system (MariaDB [20], a community-developed fork of MySQL), backed by a separate storage subsystem.

In order to test the database system for our controlled experiments and to be able to reproduce the environment on a variety of hardware, we created an isolated, virtualized environment, which allowed us to simulate the configuration and performance factors of cloud-based solutions.

The Oracle VirtualBox virtualization platform (version 5.1.22) was used as the underlying virtualization platform. All of the virtual machines ran on the same hardware server,

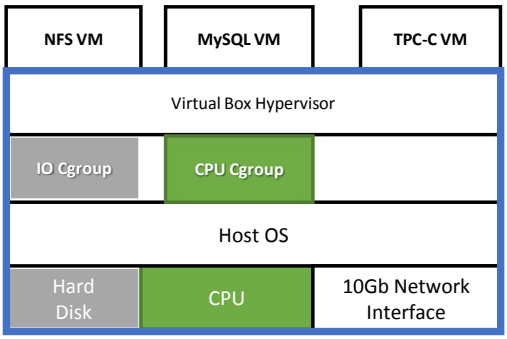


Fig. 3. Data collection platform setup.

with a bridged virtual network connecting them. Three virtual machines were created to host the main system components: the TPC-C application, the Database server, and the Storage system (hosting the database filesystem was on a NFS server). Figure 3 depicts the overall system layout. This setup allowed us to easily isolate and limit resource usage in each service or system component, with similar levels of control to what a cloud environment provides, but with easier management and faster turnaround time for the numerous procedural configuration changes our data collection effort required.

Detailed hardware and software specifications we used to generate the data is provided below:

- **CPU**: 2 × Intel Xeon CPU E5-2640 v3 @ 2.60GHz
- **Chipset**: Supermicro B10DRT-TP, BIOS Ver. 2.0a
- **Memory**: 8 × Hynix HMA84GL7AMR4N-TF 32GB DDR4-2133 ECC LRDIMM (total 256GB)
- **Hard disk**: Seagate Constellation.2 ST9500620NS, 500GB, 7200 RPM, 64MB Cache, SATA 6.0Gb/s
- **Network interface**: MT27520, 10 Gbps
- **Operating System**: Base installation of CentOS Linux release 7.3.1611, Kernel 3.10.0-514.16.1.el7.x86_64
- **Database**: MariaDB[1] [20] 10.2.6-1.el7.centos with InnoDB: 5.7.14.

A. Data collection

The TPC-C benchmark tests the response of a modern database software to a variety of application request types. We measured and modeled the variation in the database's response throughput under varying configuration parameters, both in terms of database configuration parameters and system-level parameters. For data collection we configured the TPC-C benchmark to populate and exercise a database representing, alternatively, 10, 20, 40 and 64 warehouses, for a total database size on disk of approximately 1.2GB to 6.4GB.

For each database configuration change and system configuration change, we allowed the database cache to warm up for 15 seconds. We then ran the benchmark for 3 minutes with 10 simultaneous connections to the database before gracefully

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

shutting down the database, changing the configuration, replacing database files with the initial database, and finally restarting the database server. Each iteration of this process to obtain a single configuration sample took approximately 5 minutes.

The data collection involved in executing the TPC-C benchmark while exhaustively varying three system configuration parameters across the identified domain. We wrote a set of bash scripts to alter the three chosen system configuration parameters as reported below:

- **CPU quota**: maximum allowed percentage of the system's CPU performance allocation for the database, using cgroups [21] CPU quota allocation:
  `cpu.cfs_quota_us`=1000000
  `cpu.cfs_period_us`={100000 (10%), 250000 (25%), 500000 (50%), 1000000 (100%)}
- **Disk I/O quota**: maximum allowed system I/O throughput allowed for the storage system, using cgroups I/O performance allocation:
  `blkio.throttle.write_bps_device` = `blkio.throttle.read_bps_device` = {1, 2, 4, 8, 16, 32, 48} (MBps)
- **Database buffer pool size**: size configuration of the MySQL innodb database buffer pool:
  `innodb_buffer_pool_size` = {1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024} (MB)

The total size of the system configuration space for the performance function we sampled to model its behavior is therefore $4 \times 7 \times 11 = 308$ different configurations and the sampling coordinates in the parameters domain are non-uniform.

For the purpose of the modeling experiments we modified the TPC-C benchmark to capture data directly in the form of overall transactions per minute metric (tpmC[2]) rather than parsing the default output. We ran the benchmark 30 times for each configuration to collect statistically sensitive information, and stored all of the aggregated results to disk. In the all of the modeling experiments, for each configuration, we used the average value of the tpmC throughput as the samples of the performance function we intended to model. Visual examples of the gathered data are depicted in Figure 4(a) and Figure 4(b) (CPU quota slices).

VI. Experimental results

We present a collection of scenarios to evaluate our method. Crucially, our evaluation is focused on the comparison between direct modeling and our model mapping technique, rather than the comparison between the various modeling methods. The effectiveness of our proposed method was ascertained through four scenarios:

A. changing the configuration between two resource configuration scenarios where we vary the CPU quota from 25% to 50%,
B. modeling changes in a database's performance as it grows in size,
C. extrapolating the performance trajectory of a large database increasing in size based on previously built models of smaller databases,
D. resource allocation for optimizing virtual machine packing.

In all the presented scenarios, only a fixed amount of samples (sampling budget) was available to train the models. We therefore adopted Monte Carlo cross validation (MCCV) [23]–[25] to reduce the chance of overfitting or selection bias.

We found that prior work generally asserted statistical significance of the results with a small number of repetitions per experiment, usually around 3. When training the transfer learning model, we observed that different selections of samples cause substantial fluctuations in accuracy, especially when the amount of available samples is minuscule. Given the random selection of samples used to construct the map, the direct models in our experiments and the high sensitivity of the models' accuracy to the set of samples used for training the models, we chose to run our training and accuracy testing procedure for 20 repetitions.

A. Extending a model with a new configuration parameter

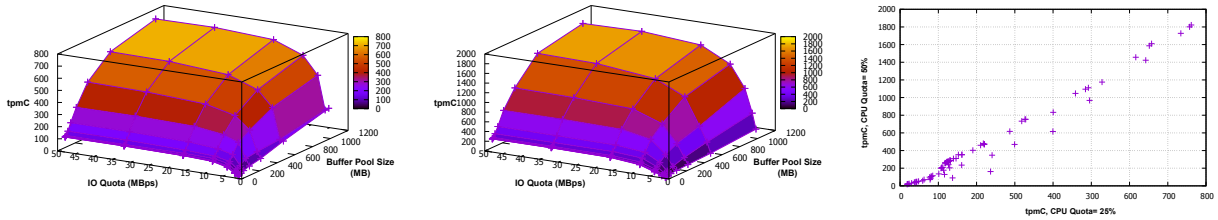
This example shows an efficient way of extending an original 2D performance model of the I/O bandwidth and buffer pool to a 3D model that includes CPU resources as a new configuration parameter with few new samples overall.

Figure 4 shows the throughput of the well-known TPC-C benchmark [1] running within a virtualized environment involving a database and storage server system, allocated with 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.

The results show that our mapping technique performs better than all of the direct modeling methods we have used for the comparison (Table I). This is because the relationship between the two configurations' codomains is approximately linear, as highlighted in Figure 4(c). In particular, for our goal of improving modeling accuracy using a small and fixed sampling budget, it is important to note how the highest benefit of model mapping is realized when the sampling budget is small.

We can note in Table I that with only five samples (1.5% of total 308 available samples), the best results provided by a direct model (the Polynomial Regressor) had a RMSE of 234% higher than the best results for model mapping (using Linear Regression). While we could have predicted that a linear model would approximate the map most efficiently, we show results with all of the other types of models we used for comparison. Although we do not report mapping across all possible pairs of CPU quota configurations, the results were similar, with the same impact in effectiveness of *ModelMap* compared to direct modeling. Adding a new dimension to represent the CPU resource to the performance model is thus significantly aided by our *ModelMap* technique.

[2]In TPC-C, throughput is defined as number of New-Order transactions per minute a system generates while the system is executing four other transactions types (Payment, Order-Status, Delivery, Stock-Level) [22].



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

Fig. 4. Performance graph of TPC-C in transactions per minute (tpmC) when **Disk I/O quota** varies between 1 and 48 MBps and **Buffer pool size** varies between 1 and 1024 MB for two **CPU quotas** 25% (a) and 50%(b). Graph of maps of TPC-C throughput (tpmC) on **CPU quota** configurations (c)

Sampling Budget	Modeling Technique Modeling Method	Direct	Mapping
5	GaussianProcessRegressor	392.28±27.3	420.41±38.1
	RbfSVR	437.64±25.1	192.75±27.3
	LinearRegression	443.97±42.8	**109.62±9.2**
	LinearSVR	380.37±11.5	212.97±20.9
	PolynomialRegressor	299.59±19.5	351.34±21.0
10	GaussianProcessRegressor	195.4±20.4	41,068.8±40,550.8
	RbfSVR	293.83±21.3	901.42±720.1
	LinearRegression	327.17±11.0	**95.43±4.7**
	LinearSVR	374.98±12.3	135.60±12.1
	PolynomialRegressor	241.53±16.8	274.72±21.6

TABLE I
RMSE OF THE PROPOSED MAPPING TECHNIQUE AND DIRECT MODELING TECHNIQUE WITH VARIOUS MODELING METHODS FOR **CPU QUOTA** CONFIGURATIONS, (LEGACY MODEL **CPU QUOTA**=25% AND UNKNOWN MODEL **CPU QUOTA**=50%).

B. Modeling effects of incremental application performance variation

We experimented with modeling the performance of a database as it accumulated data over time for the sake of exploring how different methods model an incrementally evolving application's performance characteristics.

The legacy model for this experiment is represented by the full performance data (Figure 4(a) and Figure 4(b)), using the TPC-C benchmark with 10 warehouses and configured as reported in section V-A. The unknown model we wanted to obtain represents the performance characteristics of the same benchmark running on the same platform, but with having changed the TPC-C benchmark configuration parameter controlling the number of warehouses from 10 to 64. This increased the size of the database on disk approximately six times, from ∼1.2GB to ∼6GB.

The graph of the resulting map in Figure 5 highlights how it may be impossible to represent the overall relationship between different configurations and their respective performance characteristics with a 1-dimensional function between performances. Applying a 1D map to model the performances of the new configurations would, therefore, only yield an inaccurate approximation. The results of using a 1D map confirms that our technique performs worse than the direct modeling methods we have used for the comparison in this task (Table II), as the 1D map is incapable of capturing the relationship without an appropriate set of additional of features.

It is clear that, in the presence of complex, multi-dimensional, non-linear relationships, such as the one in this experiment, the hope of discerning a model of lower dimensionality may not always be realizable. However, this experiment also shows how in this scenario, even when using a map with the fewest possible dimensions, our technique is still competitive with direct modeling.

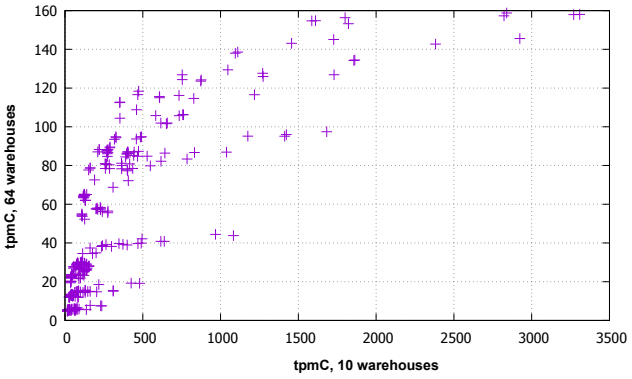


Fig. 5. Map of Transactions per minute (tpmC), 10 warehouses to 64 warehouses.

We obtain our extra dimensions by adding different features from our configuration space to the map, which increased the dimensionality of our mapping functions. The first experiment was to apply our mapping technique to two dimensions. The results in Table II, column *2D map*, show the benefit of adding one dimension yields better modeling accuracy of not only *1D map*, but better modeling accuracy than the direct modeling methods as well.

We then experimented with adding further dimensions to determine its effect on the model's accuracy. The results in Table II, column *3D map*, show a more modest benefit of adding yet another dimension on modeling accuracy, suggesting that a 2D model is sufficient to represent the map accurately enough to outperform direct modeling. Finally, as depicted in Table II column *4D map*, using a map with four dimensions is actually counter-productive, as having a restricted sampling is problematic for training such a complex map model.

Sampling Budget	Modeling Technique	Direct	1D map	2D map	3D map	4D map
5	GaussianProcessRegressor	33.87±1.6	40.09±1.5	25.17±1.1	23.85±1.4	24.63±1.2
	RbfSVR	45.75±3.0	72.66±15.4	37.06±7.1	34.98±2.8	30.15±1.9
	LinearRegression	46.74±5.7	42.66±3.5	29.35±1.9	54.64±13.1	291.11±176.9
	LinearSVR	30.92±1.1	43.85±1.2	**22.76±0.9**	25.82±1.4	26.97±1.3
	PolynomialRegressor	28.39±1.2	43.00±0.7	23.56±0.5	**22.97±0.8**	**24.23±0.9**
	ChorusIncrementalRegressor	40.23±0.1	—	—	—	—
10	GaussianProcessRegressor	23.06±1.6	33.80±1.2	**21.54±0.8**	**16.67±1.2**	**17.07±1.0**
	RbfSVR	34.03±2.9	50.95±9.7	46.43±16.3	17.45±1.6	22.17±1.4
	LinearRegression	28.20±1.1	40.36±4.0	26.96±2.5	25.39±2.2	28.52±3.4
	LinearSVR	28.84±0.9	38.43±1.1	22.61±0.8	21.63±0.9	21.77±0.9
	PolynomialRegressor	23.61±0.9	39.26±0.7	21.90±0.4	19.46±0.5	20.41±0.6
	ChorusIncrementalRegressor	40.08±0.1	—	—	—	—
30	GaussianProcessRegressor	9.37±0.5	26.49±0.5	16.24±0.3	11.61±0.5	**8.31±0.2**
	RbfSVR	11.23±0.6	37.09±5.8	19.29±1.8	**8.65±0.6**	8.75±0.3
	LinearRegression	22.95±0.3	30.58±1.3	21.07±0.8	18.69±0.5	19.55±0.8
	LinearSVR	24.43±0.5	30.09±0.3	20.46±0.3	19.04±0.3	18.93±0.3
	PolynomialRegressor	15.70±0.3	33.44±0.4	19.65±0.2	14.66±0.4	14.28±0.4
	ChorusIncrementalRegressor	38.92±0.1	—	—	—	—

TABLE II

RMSE OF THE PROPOSED MAPPING TECHNIQUE AND DIRECT MODELING TECHNIQUE WITH VARIOUS MODELING METHODS FOR **TPC-C NUMBER OF WAREHOUSES** CONFIGURATIONS. LEGACY MODEL **WAREHOUSES**=10, UNKNOWN MODEL **WAREHOUSES**=64.

We therefore 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.

C. Extrapolating application performance variation for what-if scenarios

An active database's size will vary over time, which we can consider as a latent parameter that we can extrapolate against. In this experiment, we examine our technique's ability beyond modeling incremental variations in performance characteristics of a reference application (TPC-C benchmark), but rather to measure its efficacy at performing extrapolation, i.e its ability to generate models for what-if scenarios, such as a future state, for which it is impossible to gather data. In our case, the variations are induced by accumulating data in a database over time, and the extrapolation is created by analyzing across multiple models representing the application performance at different points in time.

For this experiment, we used two legacy models, each representing a snapshot of the application's performance at a given point in time in the past as its underlying database grows in size. We also used a third model, which we had built with a limited sampling budget, which represents the latest information we have on the application's performance behavior.

The progressive increase in the database's size is represented by three data collections, using the TPC-C benchmark running on the database configured as reported in section V-A, using 10, 20 and 40 warehouses. The unknown model in this what-if scenario is represented by the application's performance with the database configured with 64 warehouses.

As explained in section IV-B, we could not use the Chorus toolkit for this experiment, as it does not allow for building an incremental model without using any samples from the unknown system configuration.

The results show that our technique performs better than the direct modeling methods we have used for the comparison (Table III) for small sampling budgets (5 and 10), but as the sampling budget increased, direct modeling converged more quickly towards an accurate representation of the model. We had to consider that the direct models had access to data coming from measurements of the system in the unknown configuration, whereas our mapping technique was limited to inferring the unknown model in a what-if scenario, hence, without any access to that data. We conclude that for what-if scenarios, when no sampling data is available, modeling the trend across prior models using our proposed model mapping offers an advantage. The prediction obtained with model mapping is more accurate than that of direct modeling with small sample sizes in the unknown configuration.

D. Resource allocation optimization for VM packing

This experiment was designed to find the overall accuracy of the models we could build using our technique, and to ensure that such models are capable of reproducing the system's behavior closely enough, without introducing differences such that would force an optimization process that relied on such models to converge to a significantly suboptimal solution.

In this experiment, we intended to identify the ideal resource allocation for four concurrently running instances of the TPC-C benchmark, each with a database configured with 20 warehouses, so that the aggregated throughput of the four TPC-C instances is the highest possible.

VMs of all four instances were hosted on the same hypervisor and were assigned a quota for CPU (total=100%), I/O bandwidth (total=48MBps), and memory for database buffer pool (total=1024MB). For the purpose of this experiment we

Sampling Budget	Modeling Technique Modeling Method	Direct	Mapping
5	GaussianProcessRegressor	33.87±1.6	32.73±1.1
	RbfSVR	45.75±3.0	40.57±3.8
	LinearRegression	46.74±5.7	143.6±47.7
	LinearSVR	30.92±1.1	30.34±0.7
	PolynomialRegressor	28.39±1.2	**22.76±0.7**
10	GaussianProcessRegressor	23.06±1.6	27.30±1.5
	RbfSVR	34.03±2.9	52.41±9.5
	LinearRegression	28.20±1.1	27.03±1.1
	LinearSVR	28.84±0.9	30.73±0.6
	PolynomialRegressor	23.61±0.9	**20.18±0.6**
30	GaussianProcessRegressor	**9.37±0.5**	26.14±2.7
	RbfSVR	11.23±0.6	78.9±31.5
	LinearRegression	22.95±0.3	24.45±0.3
	LinearSVR	24.43±0.5	28.68±0.4
	PolynomialRegressor	15.70±0.3	17.59±0.5

TABLE III
RMSE OF THE PROPOSED MAPPING TECHNIQUE AND DIRECT MODELING TECHNIQUE WITH VARIOUS MODELING METHODS FOR **TPC-C NUMBER OF WAREHOUSES** CONFIGURATIONS. LEGACY CONFIGURATIONS **WAREHOUSES**=10, 20 AND 40, UNKNOWN CONFIGURATION **WAREHOUSES**=64.

	VM-1	VM-2	VM-3	VM-4
CPU	25%	50%	10%	10%
I/O	16	16	8	8
Buffer	256	512	128	128
Throughput	160.5±1.4	490.2±11.1	41.0±0.6	41.0±0.6

TABLE IV
OPTIMAL PACKING OF FOUR TPC-C INSTANCES

assumed perfect performance isolation between the concurrently executing applications.

In order to set a baseline for the experiment, we measured the cumulative throughput of the four TPC-C instances while equally distributing the available resources among them. We will further refer to this configuration as *Equal*. In such configuration, each instance is configured with the following parameters: **CPU quota**=25%, **Disk I/O quota**=12Mbps, and **Buffer pool size**=256MB. We used our exhaustive sampling dataset to find the throughput each TPC-C instance produced with this configuration, which was 160.1±1.4; the total aggregated throughput was therefore 640.6±2.8. We then proceeded to search for the optimal set of configuration parameters for all four instances of TPC-C.

Finding the optimal solution for our resource allocation problem can be an extremely difficult task since the space is discrete and exhaustive search is not a prudent solution. Among the available algorithms, Chen [6] used Hill Climbing for this purpose. Although Hill Climbing is fast and simple, the solution is likely to be one of the local optima. Thus, we used Simulated Annealing [26], a heuristic global optimization algorithm, to find the optimal solution.

The configuration recovered by the optimization process is represented in Table IV, and it produced an aggregated throughput of 732.7±11.2. We will further refer to this configuration as *Ideal*, since it was produced using the exhaustively sampled dataset.

We then proceeded to perform the same optimization us-

Sampling Budget	Modeling Technique Modeling Method	Direct	Mapping
5	GaussianProcessRegressor	56.0%±6.4%	1.7%±0.8%
	RbfSVR	58.0%±5.5%	1.3%±0.7%
	LinearRegression	55.4%±5.4%	21.9%±6.4%
	LinearSVR	62.8%±5.2%	1.8%±0.4%
	PolynomialRegressor	36.0%±8.0%	**0.1%±0.1%**
	ChorusIncrementalRegressor	12.3%±2.5%	—
10	GaussianProcessRegressor	39.3%±6.2%	1.5%±0.9%
	RbfSVR	54.4%±5.9%	0.8%±0.3%
	LinearRegression	44.6%±4.8%	1.4%±0.5%
	LinearSVR	42.2%±4.8%	0.9%±0.3%
	PolynomialRegressor	21.5%±7.0%	**0.0%±0.0%**
	ChorusIncrementalRegressor	18.3%±2.1%	—
30	GaussianProcessRegressor	21.0%±3.3%	2.2%±1.2%
	RbfSVR	8.1%±2.2%	2.1%±1.1%
	LinearRegression	34.3%±1.2%	**0.1%±0.1%**
	LinearSVR	40.2%±3.5%	0.3%±0.2%
	PolynomialRegressor	2.6%±1.7%	0.1%±0.1%
	ChorusIncrementalRegressor	14.0%±2.5%	—

TABLE V
RELATIVE ERROR FOR MAXIMUM COMBINED THROUGHPUT IN THE VM PACKING EXPERIMENT USING THE PROPOSED MAPPING TECHNIQUE AND DIRECT MODELING TECHNIQUE WITH VARIOUS MODELING METHODS. LEGACY CONFIGURATION: **WAREHOUSES**=10, UNKNOWN CONFIGURATION: **WAREHOUSES**=20.

ing performance models as approximation of the exhaustive dataset, to understand the impact of approximating the actual performance function with limited sampling budgets. To obtain the performance model of TPC-C with 20 warehouses, the legacy model used for our technique was the full performance model for TPC-C using a database configured with 10 warehouses.

Figure 6 shows the best individual and aggregated performance using direct modeling and our mapping technique, with sampling budgets of 5, 10 and 30, respectively. Table V shows the relative error in aggregated throughput of the configurations found using direct models (*Direct*) and our technique (*Mapping*), relative to the *Ideal* configuration. The results show that with only 5 samples, our technique already produced a model that exactly predicted the correlation between resource constraints and throughput, while direct modeling produced models that induced the optimizer to find suboptimal solutions.

With 30 samples, direct modeling (Polynomial model) provided a model that is of similar accuracy to our technique. This particular type of applications generally produces a polynomial response in all its configuration parameters, so this was not entirely unexpected that the Polynomial model performed this well. All other direct modeling methods failed to provide accurate models, whereas using the model mapping technique, all of the underlying modeling methods for the map were capable of producing a model that provided a resource configuration within 1 percent of the optimal one, which indicates the generality of our approach.

We conclude that, while direct modeling may, in some circumstances, show comparable RMSE, our technique reproduces the overall behavior of a system better by leveraging correlations between configurations in a configuration space, and therefore it substantially outperforms direct modeling when the performance models are used for the purpose of

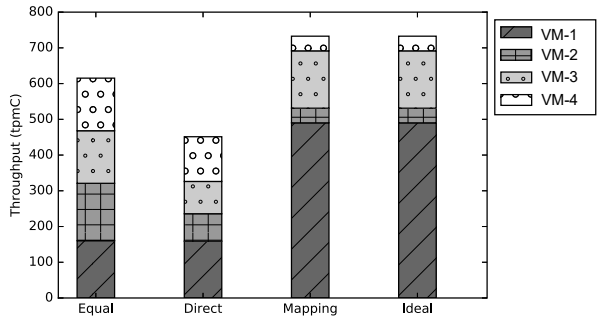


Fig. 6. Running four TPC-C instances. Sampling budget=5. Legacy configuration **warehouses**=10, unknown configuration **warehouses**=20.

resource allocation optimization with small sample counts.

VII. Conclusion and Future Work

We have proposed *ModelMap*, a technique for leveraging prior knowledge of a system's performance for system and/or application changes using transfer learning. The proposed technique obtains models for previously unknown parts of a configuration space, or even new data sets and application instances utilizing only a sparse set of new samples. This is performed by exploiting presumed structural similarity between two system configurations or application instances to construct a compact map between their performance behaviors. Furthermore, we have performed experiments on four real world application scenarios in cloud systems to show how our proposed technique can be applied to a variety of incremental performance modeling problems.

In the presence of a minuscule sampling budget our technique improves the accuracy by up to an order of magnitude compared to other modeling methods in the described class of problems. This characteristic is particularly useful when modeling and tracking dynamic variations of system's performance over time, even with a limited budget for sampling performance, which enables rapid decision making on how to optimally reconfigure a system.

Of particular note is that the experiments we have conducted, though representative of actual system performance models, are among the worst-case scenarios for our method, due to the overall simplicity of their characteristics. Our method works best in circumstances where the complexity of the performance models is significantly higher than the transformation functions that represent their evolution.

In future work we intend to analyze performance modeling problems with substantially larger numbers of system configuration parameters and where complex cloud workload mixes are present, in which a system's evolving performance behavior that demands sophisticated feature selection to construct optimal transformation functions. Additionally, we plan to study the relation between the dimensionality of the map and its accuracy, and leverage sensitivity analysis of the system's performance function to select both the most appropriate map dimensionality and its parameters. We also intend to experiment with automatic, incremental model selection between our technique and direct modeling, based on incremental cross-validation using multiple error metrics for accuracy estimation, based on the desired model application.

References

[1] "TPC-C," http://www.tpc.org/tpcc/, accessed: 2018-10-06.
[2] A. Ganapathi, H. A. Kuno, U. Dayal, J. L. Wiener, A. Fox, M. I. Jordan, and D. A. Patterson, "Predicting Multiple Metrics for Queries: Better Decisions Enabled by Machine Learning," in *ICDE'09*, 2009, pp. 592–603.
[3] E. Thereska and G. R. Ganger, "Ironmodel: robust performance models in the wild," in *SIGMETRICS*, 2008, pp. 253–264.
[4] S. J. Pan, "Transfer Learning," in *Data Classification: Algorithms and Applications*, C. C. Aggarwal, Ed. {CRC} Press, 2014, pp. 537–570.
[5] S. J. Pan, Q. Yang *et al.*, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.
[6] J. Chen, "Chorus: Model knowledge base for performance modeling in datacenters," Ph.D. dissertation, University of Toronto, 2011.
[7] 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.
[8] S. Ghanbari, G. Soundararajan, and C. Amza, "A Query Language and Runtime Tool for Evaluating Behavior of Multi-tier Servers," in *SIGMETRICS'10*, 2010, pp. 131–142.
[9] J. Garcke and T. Vanck, "Importance Weighted Inductive Transfer Learning for Regression," in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2014 Proceedings, Part I.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 466–481.
[10] T. Vanck, "New importance sampling based algorithms for compensating dataset shifts," Doctoral Thesis, Technische Universität Berlin, 2016.
[11] B. Settles, "Active learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 6, no. 1, pp. 1–114, 2012.
[12] R. Caruana, "Multitask learning," *Machine learning*, vol. 28, no. 1, pp. 41–75, 1997.
[13] A. Argyriou, T. Evgeniou, and M. Pontil, "Multi-task feature learning," in *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*, vol. 19. The MIT Press, 2007, p. 41.
[14] P. Jamshidi, M. Velez, C. Kästner, N. Siegmund, and P. 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*. IEEE Press, 2017, pp. 31–41.
[15] N. D. Lawrence and J. C. Platt, "Learning to learn with the informative vector machine," in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 65.
[16] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.
[17] C. wei Hsu, C. chung Chang, and C. jen Lin, "A practical guide to support vector classification," 2010.
[18] C. K. Williams, "Prediction with gaussian processes: From linear regression to linear prediction and beyond," in *Learning in graphical models*. Springer, 1998, pp. 599–621.
[19] "scikit-learn - Machine Learning in Python," http://scikit-learn.org/stable.
[20] "MariaDB," https://mariadb.org/, accessed: 2018-10-06.
[21] "Linux Control Groups," http://man7.org/linux/man-pages/man7/cgroups.7.html, accessed: 2018-10-06.
[22] "TPC-C," http://www.tpc.org/tpcc/faq.asp, accessed: 2018-10-06.
[23] R. R. Picard and R. D. Cook, "Cross-validation of regression models," *Journal of the American Statistical Association*, vol. 79, no. 387, pp. 575–583, 1984.
[24] J. Shao, "Linear model selection by cross-validation," *Journal of the American statistical Association*, vol. 88, no. 422, pp. 486–494, 1993.
[25] Q.-S. Xu and Y.-Z. Liang, "Monte carlo cross validation," *Chemometrics and Intelligent Laboratory Systems*, vol. 56, no. 1, pp. 1–11, 2001.
[26] P. J. Van Laarhoven and E. H. Aarts, "Simulated annealing," in *Simulated annealing: Theory and applications*. Springer, 1987, pp. 7–15.