Performance Prediction in Dynamic Clouds using Transfer Learning

Farnaz Moradi*, Rolf Stadler†‡, and Andreas Johnsson*
* Ericsson Research, Sweden, Email: {*farnaz.moradi, andreas.a.johnsson*}*@ericsson.com*
† Dept of Network and Systems Engineering, KTH Royal Institute of Technology, Sweden
‡ Swedish Institute of Computer Science (RISE SICS), Sweden, Email: *stadler@kth.se*

***Abstract*—Learning a performance model for a cloud service is challenging since its operational environment changes during execution, which requires re-training of the model in order to maintain prediction accuracy. Training a new model from scratch generally involves extensive new measurements and often generates a data-collection overhead that negatively affects the service performance.**

In this paper, we investigate an approach for re-training neural-network models, which is based on transfer learning. Under this approach, a limited number of neural-network layers are re-trained while others remain unchanged. We study the accuracy of the re-trained model and the efficiency of the method with respect to the number of re-trained layers and the number of new measurements. The evaluation is performed using traces collected from a testbed that runs a Video-on-Demand service and a Key-Value Store under various load conditions. We study model re-training after changes in load pattern, infrastructure configuration, service configuration, and target metric. We find that our method significantly reduces the number of new measurements required to compute a new model after a change. The reduction exceeds an order of magnitude in most cases.

***Index Terms*—Service Management, Performance Prediction, Machine Learning, Neural Networks, Transfer Learning.**

I. Introduction

Telecommunication operators deliver services under strict Service-Level Agreements (SLA), and it is well-known that management of such systems is challenging and demanding. One promising concept supporting service management is the use of performance models that can predict the experienced service quality at the client during execution, based on available observations in the infrastructure (including radio base stations, the network, and multiple data centers; see Figure 1). The ability to learn predictive models from observations simplifies service on-boarding and enables anomaly detection, bottleneck detection, as well as root-cause analysis.

In previous work we proposed and evaluated several data-driven approaches for predicting service-level performance metrics, as experienced by the client, from infrastructure measurements, using the framework of statistical learning [1-5]. The results were shown to generalize across a range of different scenarios.

A key challenge in data-driven model creation is the difficulty to maintain the accuracy of a performance model over time, particularly in a dynamic cloud environment. Cloud services generally rely on a virtualization layer, enabled by virtual machines (VMs) or containers, which allows service components to migrate between physical execution environments. An example is illustrated in Figure 1 in which the service is migrated from a *source domain* to a *target domain*. Further, the resources assigned to a VM or a container are dynamically scaled up or down based on operator policies or user requirements. Such changes reduce the accuracy of a performance model, which has been trained for a specific system configuration and environmental condition. As a consequence, management functions that rely on a performance model are negatively affected, unless the model is updated.

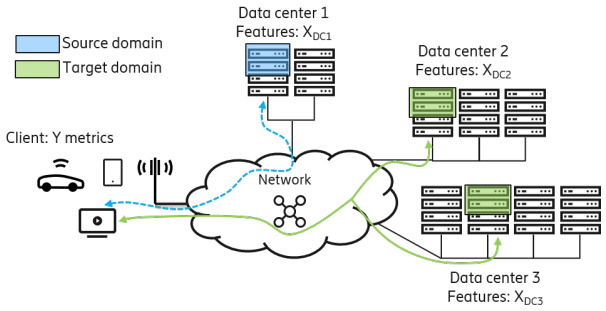

Fig. 1. Services execute in a dynamic and distributed cloud environment, accessed by clients over a network. Measurements X are collected from the infrastructure to predict service-level metrics Y. In the picture, one service is assumed to be migrated from the source to the target domain.

Extensive measurements and data collection (preferably of labeled data) is usually required for training machine-learning models. The data collection process takes time and the overhead associated with measurements and data collection can adversely affect the service itself and potentially co-located services as well. For certain services, such as short-lived Virtual Network Functions (VNFs), there is often not enough time to gather the data required for accurate predictions.

This paper addresses the above challenges and makes the following contributions. First, we show how changes in the execution environment can significantly reduce the accuracy of a predictive model. We adopt and evaluate an approach for re-training neural-network models based on *transfer learning*, whereby a limited number of layers is re-trained. Specifically, the knowledge embedded in neural-network parameters, obtained for one environment (a.k.a. source domain),

978-3-903176-15-7

is transferred to facilitate predictions in a changed execution environment (a.k.a. target domain). We assess the effectiveness of the approach by studying model re-training after changes in load pattern, infrastructure configuration, service configuration, and target metric. Our conclusions are that transfer learning, compared with learning a new predictive model, significantly reduces the number of required measurements in the target domain and reduces the data-collection overhead. In addition, it allows for faster model computation.

This work assumes that deep neural networks are used for performance prediction. Alternative models for prediction, such as random forest, often achieve a similar level of accuracy at a lower cost. For certain tasks however, such as predicting conditional distributions (e.g., [5]), deep architectures are state-of-the-art, which makes studying transfer learning in the context of neural networks an important topic.

The rest of the paper is organized as follows. Section II describes the problem setting. Section III describes our transfer learning approach. Section IV describes the testbed and traces, and Section V provides evaluation results and discussions. Related works are presented in Section VI, and Section VII contains conclusions and future work.

II. Problem setting

Figure 1 outlines the system under consideration, where a set of clients are interacting, over a network, with services executing in one or multiple data centers. For the purpose of this paper we consider experiments and data traces where clients access two networked services executing in one data center; a Video-on-Demand service and a Key-Value Store service. Note however that the concept developed in this paper is not limited to these services.

In previous works [1], [2], and [3], we predicted the service-level metrics Y_t at time t on the client, based on knowing the infrastructure metrics X_t. Using the framework of machine learning, we developed and evaluated models $M : X_t \rightarrow \hat{Y}_t$, such that $\hat{Y}_t$ closely approximated Y_t for a given X_t. This was further developed in [5] where the conditional distribution $P(Y_t|X_t)$ was predicted using mixture-density functions. We assume the metrics X and Y evolve over time, influenced, e.g., by the data center and network load, and operating system dynamics. We model the evolution of the metrics X and Y as time series $\{X_t\}_t, \{Y_t\}_t$.

The mapping between X and Y may change in a cloud environment due to scaling of resources for the service execution environment, service migration, changed hardware platform, or other infrastructural dynamics. Hence, the accuracy of a performance model may decrease over time. This challenge is targeted by this paper.

We adopt a transfer learning definition from [6] to formalize the problem. A domain $D = \{X, P(X)\}$ consists of two components: (1) a feature space X, and (2) a marginal probability distribution $P(X)$, where X corresponds to the infrastructure metrics. Further, a task $T = \{Y, M\}$ consists of two components: a target space Y corresponding to the service-level metrics, and an objective predictive model M.

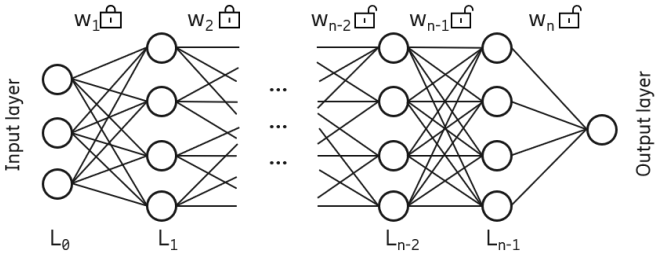

Fig. 2. A neural network with fixed and re-trainable layers. Layers L_{n-2}, L_{n-1}, and L_n are re-trainable. Layer L_0 equals X_t, layer L_n represents Y_t.

Transfer learning is then defined as follows. Given a source domain D_S and learning task T_S, a target domain D_T and learning task T_T, transfer learning aims at reducing the cost of learning the predictive model M in D_T using the knowledge in D_S and T_S, where $D_S \neq D_T$, or $T_S \neq T_T$ [6].

In this paper we specifically study the problem of how the model accuracy of learning task T_T in the target domain varies with respect to transferred insights from the source domain D_S and the source task T_S, and the number of samples obtained at a given time t in the target domain D_T. Recall that the number of samples in the target domain may be limited due to overhead or time constraints, as discussed in Section I.

III. A Transfer Learning Approach for Service Metric Prediction

In this paper, the transfer learning approach builds upon re-training of a machine-learning model M, based on a neural network, inspired by the work in [15]. A neural-network model M generally consists of an input layer, corresponding to samples X_t, $n-1$ hidden layers $L_1, ..., L_{n-1}$, weights $w_1, ..., w_n$, and an output layer (L_n) corresponding to Y_t.

The weights w_i for a neural-network model M_S in the source domain D_S are trained using backpropagation [13]. A transfer model configuration determines which weights of M_S should be kept intact. The remaining weights will be updated during re-training using samples from the target domain D_T.

Note that if the output type of the model in the target domain and the source domain are different, i.e., $Y_S \neq Y_T$, then the output layer of M_S is replaced with a new layer, and the weights w_n are initialized randomly.

Figure 2 shows an example of a neural network and its transfer model configuration where the weights of the first layers, $L_1, ..., L_{n-3}$, of the network are locked and will not change during re-training, while the weights of the last layers, L_{n-2}, L_{n-1}, L_n, will be updated during re-training. The neural network, trained with samples from both D_S and D_T, is denoted M_T and is thus used for service-level metric prediction in D_T.

Note that the neural-network architecture of M_S and M_T are identical in this study.

TABLE I
TESTBED TRACES USED FOR EVALUATION.

Trace ID	Service(s)	Load pattern	# samples
1	KVS	Periodic	28962
2	KVS + VoD	Periodic	26488
3	VoD	Periodic	37036
4	VoD + KVS	Periodic	27699
5	VoD + KVS	Flashcrowd	29151
6	VoD (single server)	Periodic	51043

IV. TESTBED AND TRACES

A. Testbed and services

The work presented in this paper is based on traces obtained from executing experiments in a testbed located at KTH. This section describes the experimental infrastructure, and the structure of the data traces. We also describe the services that run on the infrastructure; a Video-on-Demand (VoD) service and a Key-Value Store (KVS) service. More details are available in [3]. The section ends with an explanation of the load patterns we use and the experiments we run to obtain the traces on which this paper relies.

The testbed consists of a server cluster and a set of clients, similar to the illustration in Figure 1. The server cluster is deployed on a rack with ten high-performance machines interconnected by a Gigabit Ethernet. Nine machines are Dell PowerEdge R715 2U servers, each with 64 GB RAM, two 12-core AMD Opteron processors, a 500 GB hard disk, and four 1 Gb network interfaces. The tenth machine is a Dell PowerEdge R630 2U with 256 GB RAM, two 12-core Intel Xeon E5-2680 processors, two 1.2 TB hard disks, and twelve 1 Gb network interfaces. All machines run Ubuntu Server 14.04 64 bits, and their clocks are NTP [22] synchronized.

The *VoD service* uses a modified VLC media player software [7] which provides single-representation streaming with varying frame rate. The VoD service is executed in two configurations; (1) execution on a single PowerEdge R715 machine (referred to as VoD (single server)), and (2) execution on six PowerEdge R715 machines.

The *KVS service* uses the Voldemort software [8]. It is installed on the same machines as the VoD service, and can execute in parallel. Six of them act as KVS nodes in a peer-to-peer fashion and the rest act as load generators emulating client populations.

B. Collected data and traces

This subsection provides a description of the data collected on the testbed, namely the input feature set X as well as the specific service-level metrics Y_{VoD} and Y_{KVS}.

Features X are extracted from the Linux kernels that run on the machines. To access the kernel data, we use System Activity Report (SAR), a popular open-source Linux library [9], which provides approximately 1700 features per server. Examples of such statistics are CPU utilization per core, memory utilization, and disk I/O.

The Y_{VoD} service-level metrics are measured on the client. During an experiment, we capture multiple metrics, but for the purpose of this paper we focus on the number of displayed video frames per second ($FrameRate$).

The Y_{KVS} service-level metrics are also measured on the clients. During an experiment, we capture (1) Read Response Time as the average read latency for obtaining responses over a set of operations performed per second ($ReadsAvg.$), and (2) Write Response Time as the average write latency for obtaining responses over a set of operations performed per second ($WritesAvg.$). These metrics are computed using a customized benchmark tool of Voldemort.

A trace is generated by executing testbed experiments where statistics are collected every second; specifically it includes features X, and service-level metrics Y_{VoD} and Y_{KVS}.

C. Generating load on the testbed

Two load generators are executed in the testbed, one for the VoD application and another for the KVS application. The VoD load generator dynamically controls the number of active VoD sessions, spawning and terminating VLC clients. The KVS load generator controls the rate of KVS operations issued per second. Both generators produce load according to two distinct load patterns.

1) Periodic-load: the load generator produces requests following a Poisson process whose arrival rate is modulated by a sinusoidal function with a starting load level, amplitude, and period of 60 minutes;
2) Flashcrowd-load: the load generator produces requests following a Poisson process whose arrival rate is modulated by a flashcrowd model [17]. The arrival rate starts at a low load level and peaks at flash events. At each flash event, the arrival rate increases within a minute to a peak load, where it stays for one minute, and then decreases to the initial load within four minutes.

All traces we used in the considered scenarios have been created using stochastic models in an attempt to approximate real scenarios. That said, we plan in future studies to complement model-based traces with traces captured from operational systems for evaluation.

D. The experiments chosen for this paper

The transfer-learning approach described in Section III is evaluated using data traces from our experiments. Traces 1-5 are obtained from [4] while trace 6 originates from [1]. Information about the traces is summarized in Table I.

Traces 1, 3, and 6 are based on execution of a single service, while traces 2, 4, and 5 originate from experiments where KVS and VoD services execute in parallel on the infrastructure. Note that for trace 6, the experiment was configured to execute the VoD service on a *single server*.

V. EXPERIMENTAL EVALUATION OF TRANSFER LEARNING

We apply and evaluate the transfer learning approach described in Section III to 6 scenarios, summarized in Table II, and described below. The evaluation specifically targets the

TABLE II
THE SCENARIOS STUDIED IN THIS WORK. D_S CORRESPONDS TO THE SOURCE DOMAIN AND D_T TO THE TARGET DOMAIN.

Scenario #	D_S				D_T			
	Trace ID	Service	T_S	$P(X_S)$	Trace ID	Service	T_T	$P(X_T)$
1	1	KVS	ReadsAvg.	Periodic load	2	KVS + VoD	ReadsAvg.	Periodic load
2	1	KVS	ReadsAvg.	Periodic load	2	KVS + VoD	WritesAvg.	Periodic load
3	3	VoD	FrameRate	Periodic load	4	VoD + KVS	FrameRate	Periodic load
4	3	VoD	FrameRate	Periodic load	5	VoD + KVS	FrameRate	FlashCrowd load
5	6	VoD (single server)	FrameRate	Periodic load	3	VoD	FrameRate	Periodic load
6	6	VoD (single server)	FrameRate	Periodic load	5	VoD + KVS	FrameRate	FlashCrowd load

need for adapting the source model, what to transfer from source to target, and how. Finally, the benefits of using transfer learning to shorten training time is evaluated.

The source domain D_S is based on KVS or VoD executing either on the server cluster or on a single server, whereas the target domain D_T corresponds to a change with respect to the number of services being executed on the platform, the task T as in scenario 2, the distribution of samples $P(X)$ as in scenarios 4 and 6, or the hardware platform as in scenarios 5 and 6. The scenarios, described below, have been defined so that they correspond to changes with different degrees of severity with respect to the service execution environment.

In **Scenario 1**, a model is trained in the source domain D_S to predict the response time for reads, that is ReadsAvg. (T_S), for the KVS service under Periodic load ($P(X_S)$). The model is then transferred to a target domain (D_T) where both KVS and VoD services execute concurrently on the same platform and share the same resources. The transferred model in this scenario is then used to predict ReadsAvg. (T_T) under Periodic load ($P(X_S)$). That is, this scenario investigates the impact of moving a service from a dedicated hardware platform to a shared platform. **Scenario 2** uses the same source model as scenario 1 which is transferred to the target environment. However, the target model is used to predict a different target variable, namely the response time for writes, i.e., WritesAvg. In **Scenario 3**, a source model is trained for prediction of frame rate for the VoD service under periodic load on the testbed. The model is then transferred to a target domain where both VoD and KVS services execute concurrently, that is changing the service execution from a dedicated platform to a shared one. **Scenario 4** shares the same source domain and target domain with scenario 3 except that the load pattern in the target domain has changed to Flashcrowd. In **Scenario 5**, the source model is trained for the VoD service running on a single server under periodic load. The model is transferred to a target domain where the VoD service is scaled up and migrated to a cluster with 6 servers. **Scenario 6** shares the same source domain with scenario 5 and the same target domain as scenario 4. That is, a model is learned for the VoD service running on a single server under Periodic load which is then transferred to an environment where both VoD and KVS services execute in parallel on 6 servers under a Flashcrowd workload. In this case the execution environment is scaled up and migrated, and also changed from dedicated to shared platform.

In this paper, the *Normalized Mean Absolute Error* is used as the primary evaluation metric and is defined as

$$NMAE = \frac{1}{\bar{y}}(\frac{1}{m}\sum_{t=1}^{m}|y_t - \hat{y}_t|)$$

where $\hat{y}_t$ is the model prediction for the measured performance metric y_t, and $\bar{y}$ is the average of the samples y_t of the test set of size m. In this paper, the evaluation results using *NMAE* metric are presented. We chose NMAE over popular metrics like R^2 or *RMSE*, because it gives a better intuitive understanding of the error in the application domain and better mitigates outliers.

We applied a feature selection method based on feature importance obtained from a tree-based model on each source domain. For the traces in Table I, 62 features for KVS and 46 for VoD were selected, respectively. For scenarios 5 and 6, where the VoD service is executing on a single server, a slightly different feature set was obtained compared to scenarios 3 and 4, where it executes on the cluster. As a future work we will look into other approaches for feature selection.

After evaluating several neural-network architectures and hyper parameters, two different architectures are selected for the two services. For the KVS traces used in scenarios 1 and 2, a 4-layer neural network was used which consists of an input layer with 62 nodes, 3 hidden layers with 64 nodes each, and a single-node output layer. For the VoD traces used in scenarios 3 to 6, a 5-layer neural network was used with 46 nodes in the input layer, 256 nodes in each hidden layer, and a single-node output layer.

For the implementation of the neural networks, Keras library [18] running on top of TensorFlow [19] was used. We used the rectified linear unit (ReLu) activation function for all the layers, Adam optimizer [20] with a learning rate of 0.001, L2 regularization of 0.001, and mean absolute error (MAE) as the loss function. Each experiment was run for 300 epochs with a batch size of 256.

To obtain the source model M_S for each scenario, the neural network is initialized with random weights and is trained on samples from the source domain. For training, the samples

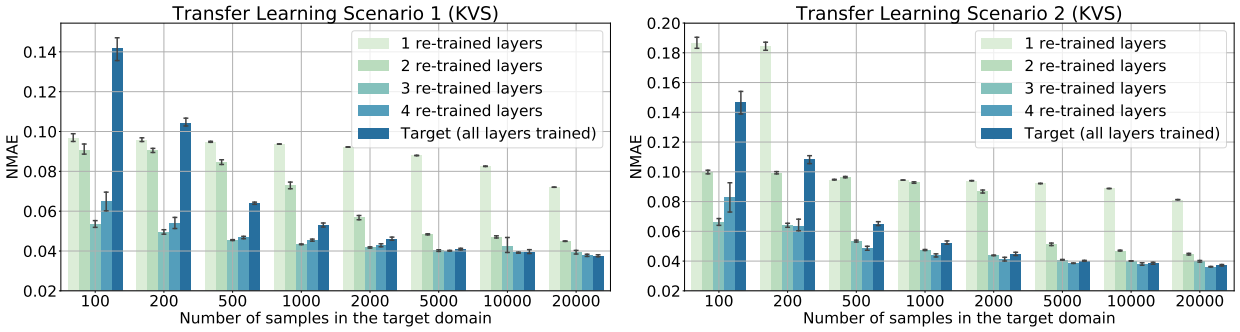

Fig. 3. Prediction error in the target domain, showing the impact of re-training layers of transferred models with samples in target compared to training new target models. The plots show results for the KVS scenarios 1 and 2.

TABLE III
PREDICTION ERROR (*NMAE%*) IN THE TARGET DOMAIN FOR MODELS TRAINED IN THE TARGET DOMAIN (TARGET MODEL) AND IN THE SOURCE DOMAIN (SOURCE MODEL). MODELS ARE CREATED WITH LASSO REGRESSION (LR), RANDOM FOREST (RF), NEURAL NETWORK (NN).

Scenario #	Target model			Source model		
	LR	RF	**NN**	LR	RF	**NN**
1	4.7	3.8	**3.8**	5.5	6.1	**10.5**
2	4.6	3.8	**3.7**	50.4	49.3	**51.5**
3	16.1	11.0	**10.3**	14.7	14.8	**12.5**
4	12.3	6.4	**6.4**	12.6	14.0	**9.5**
5	11.8	8.6	**6.7**	26.7	29.3	**23.1**
6	12.5	6.8	**7.4**	26.9	30.2	**27.5**

TABLE IV
NUMBER OF WEIGHTS OF THE NEURAL NETWORK THAT ARE UPDATED WHEN RE-TRAINING A CERTAIN NUMBER OF LAYERS.

# Re-trained layers	# Trainable weights	
	KVS	VoD
1	65	257
2	4225	66049
3	8385	131841
4	12417	197633
5	N/A	209665

in the source domain is split into training (80%) and testing (20%) sets. The model and its weights are then transferred to be used for predictions in the target domain. Similarly, in the target domain the samples are split into training and testing sets where the training set is used for either re-training the transferred source model or to train a new target model, while the testing set is used for evaluation.

A. The need for adapting the source model

First, we investigate whether a source model can be used without modifications for service-metric prediction in the corresponding target domain. In addition to neural networks (NN), we also evaluate lasso regression (LR) and random forest (RF) models.

Table III shows the *NMAE* (%) in target domain by using the source models, trained on samples from the source domains, compared to using target models trained on the data from the corresponding target domains as the baseline. It can be seen that regardless of the machine-learning algorithm used, a source model will have lower accuracy if reused without modification when the domain changes, particularly in scenario 2 where the source and target tasks are different ($T_S \neq T_T$), and for scenarios 5 and 6 where the hardware platform (i.e., execution environment) has changed. For these three scenarios a source performance-prediction model *must* be re-trained, due to the major change in the domain, whereas it is highly beneficial in the other scenarios.

B. How and what to transfer from source to target domain

From the above results it is obvious that a source model has to be adapted before it can be used for making predictions in the target domain. For each scenario in Table II, we determine the transfer configuration, i.e., the neural-network layers where the weights can remain intact after transfer and the layers where the weights need to get updated during re-training. Table IV shows the number of weights that are updated during re-training the different number of layers for KVS and VoD neural-network architectures.

It is known that the weights of the last layers of a neural network are more specific to a particular dataset/task, while the weights of the first layers are more general [15]. We investigate this by keeping the weights on the first layers unchanged while re-training additional layers. For example, for the VoD traces, re-training 3 layers means that the weights on the first 2 layers of the network are kept unchanged while the weights of the last 3 layers (i.e. 131841 weights) are updated during re-training in the target domain. Note that in scenario 2, since the outputs Y_S and Y_T are different, before transfer learning can be used, the output layer of the source neural-network model is replaced with a new output layer initialized with random weights.

We evaluate if a source model can be transferred to the target domain and which layers need to be re-trained. As discussed in Section I, the number of samples from the target

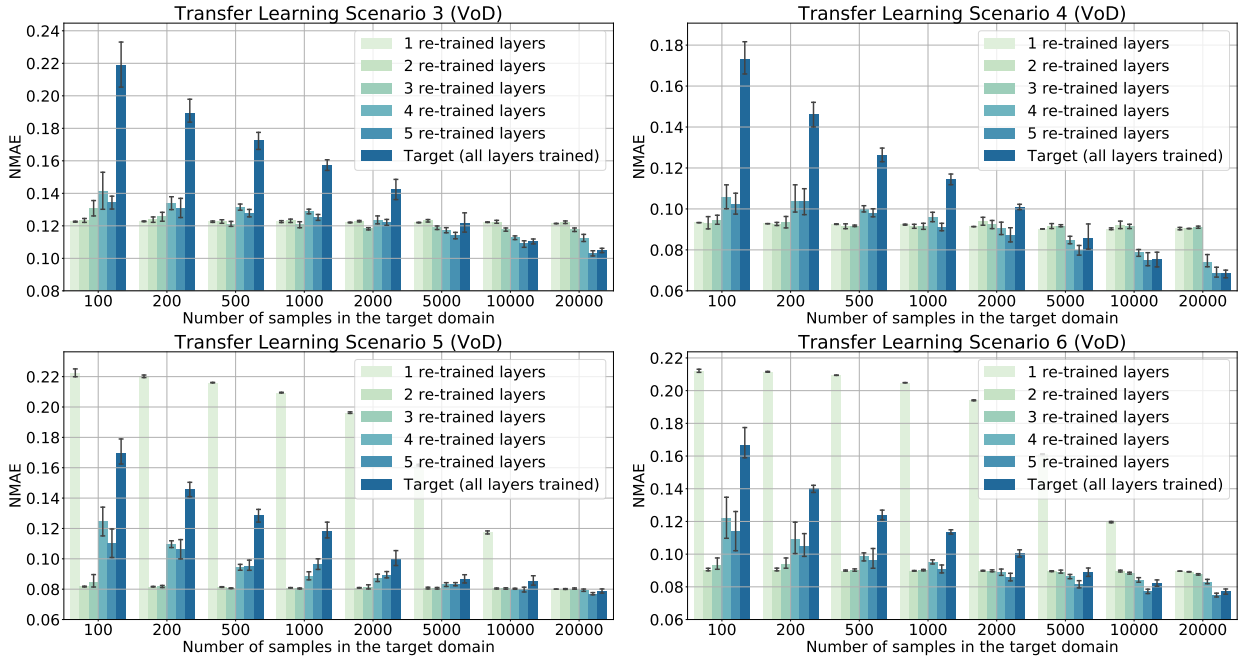

Fig. 4. Prediction error in the target domain, showing the impact of re-training layers of transferred models with samples in target compared to training new target models. The plots show results for VoD scenarios 3 to 6.

domain can be limited due to overhead and also the time it takes to run new measurements during service execution. Therefore, in addition to evaluating the transferability of the model weights, we evaluate the impact of the number of samples in the target on the service metric predictions tasks.

Figure 3 shows the evaluation results for KVS scenarios 1 and 2, and Figure 4 shows the evaluation results for VoD scenarios 3 to 6. In each figure, the horizontal axis show the number of samples available in the target domain. The bars show the model prediction error NMAE (%) for varying number of re-trained layers using target samples. The right-most bar in each group (colored in dark blue) shows the prediction error when transfer learning is not used and rather a new target model, initialized with random weights, is created by using only the available target samples. Each experiment is performed 5 times where the samples are selected randomly from the target domain. The variance is indicated for each bar.

For **scenario 1**, as shown in Figure 3, re-training three layers of the transferred model leads to low *NMAE* values, particularly when the number of samples are low. This means that the weights of the first layer of the source model are more general and therefore can remain intact, while the weights of the other layers are more specific to the source domain and have to be re-trained after the transfer. The figure also shows that when there are more samples available in the target, re-training all four layers of the transferred model can further reduce the prediction error.

In **scenario 2**, the task in the source and target differ (i.e., $T_S \neq T_T$) and therefore, the last layer of the transferred model is replaced with a new layer which is initiated with random weights. It can be seen in Figure 3 that re-training only the newly initialized layer while keeping the rest of the weights intact leads to even higher errors compared to training a new target model. Similar to scenario 1, re-training at least three layers of the model reduces the *NMAE* values.

For **scenarios 3 and 4**, shown in Figure 4, a low number of samples suggests that the weights of most of the layers can remain intact, that is re-training only 1 layer, for example. This means that the weights of the first layers of the network are quite general and re-training them with small number of samples leads to over-fitting. But as the number of samples in the target domain grows, more layers can and should be re-trained to further improve the model performance.

In **scenarios 5 and 6**, the hardware platform in the source and target has changed, which means that the input features $X_S \neq X_T$. In order to be able to use transfer learning, we first aggregated the features in the target domain. In this study the aggregation is done by calculating the mean value for the features category collected from each server. We had similar observations using median or maximum values for aggregation, however the mean value is chosen for aggregation as it leads to slightly lower *NMAE* values. In other words, for each feature $x_S \in X_S$ which is collected from a single server, there are 6 features in X_T which are collected from the 6 servers in the target domain. These features are then aggregated into one feature by calculating $x_T = \frac{1}{J}\sum_{j=1}^{J} x_{t_i}$, where J in this case equals 6. As a future work we will study other methods for feature aggregation and selection.

The results for scenarios 5 and 6 are shown in the bottom graphs of Figure 4. Re-training two or three layers of the transferred model in both scenarios leads to low *NMAE* values, whereas re-training only one layer leads to high errors. For the

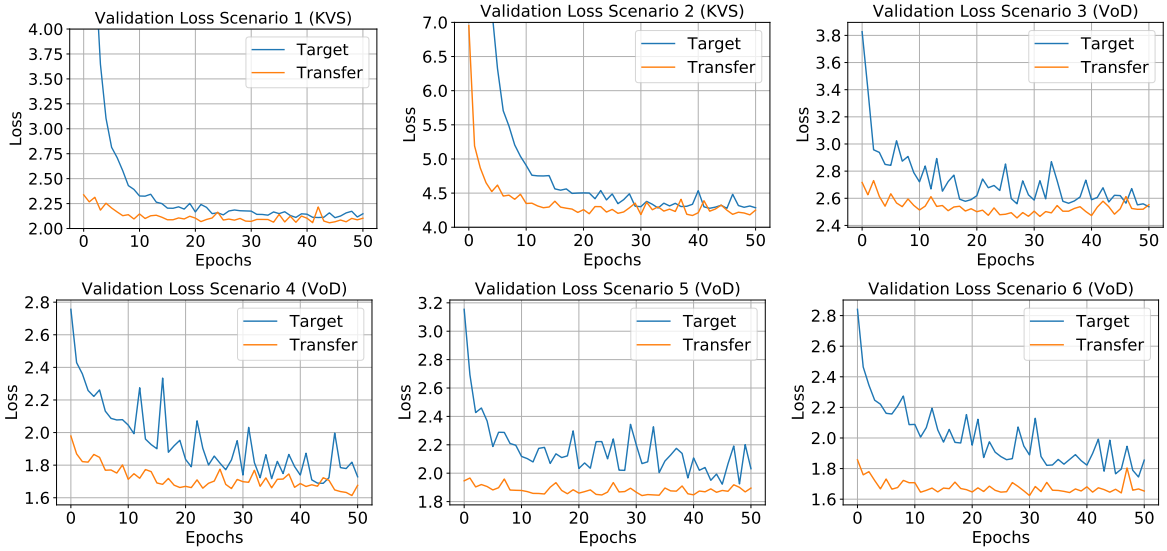

Fig. 5. Decrease in loss function during neural-network model training in D_T, using all samples, comparing a case with transferred weights from D_S and randomly initialized weights and training all layers in D_T.

cases with many samples in the target even lower *NMAE* is achieved when all layers of the model are re-trained.

In **scenarios 4 and 6**, the transferability of two different source models, trained in different source domains using different features, towards the same target domain is evaluated. It can be seen that in scenario 6, in contrast to scenario 4, re-training only the last layer of the transferred model leads to very high errors. In scenario 4, the change from source to target is minor, while in scenario 6 the change to target domain is more sever, and has impacted the transferability of the source model. Our results show that in scenarios 4 only the last layer of the source model is specific to the source domain, while in scenario 6 the last three layers are specific to source domain and need re-training. Based on our evaluations, we did not observe any significant impact in model performance depending on the similarity of the source and target domains if correct number of layers are re-trained. Future work will study selection strategies for the source domain.

Overall, in all scenarios when the number of available samples in the target domain is low ($\leqslant 1000$), transfer learning improves the *NMAE* value with at least 20% compared to training a new target model.

Further, it is clear that when the number of samples in the target is low, the prediction error for a model trained only with samples in target (i.e., the right-most bar) is high. This observation is expected and visible for both KVS and VoD scenarios, since training a neural network with many parameters requires large amount of training data, otherwise the model tends to over-fit to the relatively small data set. In general, additional measurements reduce the *NMAE*.

Moreover, our results show that to reach a given prediction error, transfer learning requires fewer training samples from the target domain compared to training a new accurate target model, in some cases less than one to two orders of magnitude, which means fewer additional measurements, and lower overhead, after a change in the cloud environment during service execution. For example, in scenario 3, an *NMAE* of 12% is achieved by re-training the transferred source model on 500 samples, while to reach the same *NMAE* without transfer learning around 5000 samples are needed to train a new target model. As another example, in scenario 5 an *NMAE* of around 8% is achieved by re-training two layers of the transferred model with as little as 100 samples from the target, but to reach the same *NMAE* a randomly initialized target model has to be trained with more than 10000 samples.

C. Transfer learning can shorten training time

It is evident from the results presented above that transfer learning lowers the prediction error for the target model when few samples are available in the target domain. Hence, the approach is suggested to be used as part of a management system to shorten the time until an accurate performance model is available in the target. This is critical specifically for short-lived services such as VNFs, as discussed in Section I, but also for VMs and containers being migrated or scaled.

Transfer learning can also be used to train a source model in a testbed environment, and then transfer the performance model to an operational environment where it is known that data collection overhead is costly or time consuming. This is exemplified in scenarios 5 and 6 where the VoD service first runs on a small infrastructure and then scales to an operational environment with a larger number of servers.

When a large number of samples is available during service execution ($\geqslant$ 10000), the difference in prediction error between re-training a neural network model initialized with transferred weights and training a neural-network model initialized with random weights becomes negligible. This is visible in all graphs in Figures 3 and 4, corresponding to both KVS and VoD scenarios. *However, using transfer learning is*

still beneficial since it can speed up the model training. We elaborate on this below.

Figure 5 shows a comparison between the validation loss (*mean absolute error*) over the number of epochs of a transferred model, where all the layers are re-trained in target, and a target model initialized with random weights and trained on samples from the target domain. Both the transferred model and the target model are trained on all samples in the target domain. For all 6 scenarios, when transfer learning is used, the loss value is lower and decreases faster (fewer number of epochs required) compared to when a new target model is trained. In scenario 2, since the last layer of the transferred model is replaces and also initiated with random weights, the initial loss value is high but it is decreased after a few epochs.

VI. Related Work

Transfer learning has received considerable attention in areas such as image processing and natural language processing (NLP). This section provides a review of relevant literature.

In [15], the authors investigated the transferability of features in a neural network for image processing and show that the transferability decreases when the distance between source and target tasks increases. It was also shown that transfer learning leads to better results compared to a new model with randomly initialized weights and that re-training all layers of the network improves its generalizability. Further, in [16], the authors investigate how transferable the layers of a neural-network model in the field of NLP are and show that the semantic similarity of the source and target tasks impacts the transferability of the neural-network models.

Recently, transfer learning has also been used in other areas including performance predictions in the network and data center domains. A number of studies have looked into using transfer learning for identifying the best application configurations. For example, in [11], the authors present a transfer learning approach for performance prediction of configurable software across different hardware platforms. The source model is built using a regression tree from a random sample of configurations on the source hardware, then a linear regression model transfers the results into the target domain. Further, in [10], an empirical study was performed on four software systems, with varying software configurations and environmental conditions, to identify the key knowledge pieces that can be exploited for transfer learning. Insights from the paper include that for non-severe hardware changes, a linear transfer model can be deployed across environments. However, virtualization may hinder transfer learning. Further, even for some severe environmental changes when the performance distributions are similar there is a potential for learning a non-linear transfer function. In comparison, this paper adopts a more advanced transfer-learning approach, and also studies a different use case, compared to the above mentioned works.

In [21] the authors propose a deep-learning based approach for identifying software configurations for a high-performing application, when limited resources are available for data collection in the target domain by combining information from exhaustive observations collected at a smaller scale with limited observations collected at a larger target scale. The work has similarities with this paper, but the neural networks are trained given feature sets, outputs, and assumptions that do not generalize to the domain of this paper. Rather, the work may serve as inspiration for how to select a source domain for the use case discussed in this paper.

In [14] the authors study the challenge of predicting server behavior and proposed a random-forest-based transfer learning approach. The challenge is that small data centers exhibit too few labeled training examples to build a proper model, since the distribution of problematic and normal server behavior is highly skewed. The approach is to combine training examples from several small data centers into one pool of training samples. A model for the target domain is then built based on samples from all small data centers that resembles the target domain good enough.

VII. Conclusions and Future Work

In this paper, we investigated an approach for model re-training based on transfer learning whereby only a limited number of layers of a neural network are re-trained. We evaluated our approach for multiple scenarios with realistic datasets obtained from a testbed for two different services under varying load.

Our results show that transferring the weights of a neural network, trained in a source domain, and re-training them according to a transfer configuration, using samples in a target domain that constitutes a changed operational environment, improves the model performance with at least 20% when the number of available samples is low ($\leqslant 1000$). Further, the paper shows that transfer learning reduces the need for data collection in the target domain, by an order of magnitude in several cases. In the case where a larger number of samples ($\geqslant 10000$) is readily available in the target domain the transfer-learning approach will not further improve the prediction error, however it greatly speeds up the model training in the target domain, for all considered scenarios.

Moreover, the results show that the number of neural-network layers to re-train in the target domain varies with the service type, the severity of execution-environment change, and also the number of available samples in the target domain. Hence, additional work is needed to provide general guidelines for determining a transfer configuration.

In future work, we will study additional scenarios and datasets in more complex infrastructures, investigate the importance of source-domain selection on the performance of the transferred model, and determine how transfer learning can be deployed in operational management systems.

Acknowledgment

The authors are grateful to Jawwad Ahmed and Christofer Flinta, both at Ericsson Research, and Forough Shahab Samani at KTH for fruitful discussions around this work. This research has been partially supported by the Swedish Governmental Agency for Innovation Systems, VINNOVA, through projects Celtic SENDATE EXTEND and ITEA3 AutoDC.

REFERENCES

[1] R. Yanggratoke, J. Ahmed, J Ardelius, C. Flinta, A. Johnsson, D. Gillblad, R. Stadler, "Predicting real-time service-level metrics from device statistics". in IFIP/IEEE International Symposium on Integrated Network Management (IM), 2015.

[2] R. Yanggratoke, J. Ahmed, J. Ardelius, C. Flinta, A. Johnsson, D. Gillblad, and R. Stadler, "Predicting Service Metrics for Cluster-based Services using Real-time Analytics", in Proceedings of 11th IEEE International Conference on Network and Service Management (CNSM), 2015.

[3] R. Yanggratoke, J. Ahmed, J. Ardelius, C. Flinta, A. Johnsson, D. Gillblad, R. Stadler, "A service-agnostic method for predicting service metrics in real-time", in International Journal of Network Management, 2017.

[4] R. Stadler, R. Pasquini, V. Fodor, "Learning from network device statistics", in Journal of Network and Systems Management (JNSM) 25 (4), 672-698, 2017.

[5] F. S. Samani, R. Stadler, "Predicting Distributions of Service Metrics using Neural Networks", in 14th International Conference on Network and Service Management (CNSM) 2018.

[6] S. J. Pan and Q. Yang, "A Survey on Transfer Learning", in IEEE Transactions on Knowledge & Data Engineering, vol. 22, no. , pp. 1345-1359, 2009.

[7] VLC, 2016. [Online]. Available: http://www.videolan.org/vlc/

[8] Voldemort, 2016. [Online]. Available: http://www.project-voldemort.com/voldemort/

[9] SAR, 2016. [Online]. Available: http://linux.die.net/man/1/sar

[10] P. Jamshidi, N. Siegmund, M. Velez, C. Kastner, A. Patel, Y. Agarwal, "Transfer Learning for Performance Modeling of Configurable Systems: An Exploratory Analysis", in Proceedings of IEEE/ACM International Conference on Automated Software Engineering, 2017.

[11] P. Valov, J. Petkovich, J. Guo, S. Fischmeister, K. Czarnecki, "Transferring Performance Prediction Models Across Different Hardware Platforms", in Proceedings of ACM/SPEC International Conference on Performance Engineering, 2017.

[12] V. Nair, R. Krishna, T. Menzies, P. Jamshidi, "Transfer Learning with Bellwethers to find Good Configurations", arXiv:1803.03900 [cs.SE], 2018.

[13] T. Hastie, R. Tibshirani, J. Friedman, "The Elements of Statistical Learning", in Springer Series in Statistics. New York, NY, USA: Springer New York Inc.; 2001.

[14] J. Bogojeska and D. Wiesmann, "Transfer learning for server behavior classification in small IT environments", in Proceedings to IFIP/IEEE Network Operations and Management Symposium (NOMS), 2018.

[15] J. Yosinski, J. Clune, Y. Bengio, H. Lipson, "How transferable are features in deep neural networks?", in Advances in Neural Information Processing Systems (NIPS), 2014.

[16] L. Mou, Z. Meng, R. Yan, G. Li, Y. Xu, L. Zhang, Z. Jin, "How Transferable are Neural Networks in NLP Applications?", in Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, 2016.

[17] I. Ari, B. Hong, E. L. Miller, S. A. Brandt, and D. D. Long, "Managing Flash Crowds on the Internet", in 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems, (MASCOTS) 2003.

[18] Keras, 2018. [Online]. Available: https://keras.io/

[19] TensorFlow, 2018. [Online]. Available: https://github.com/tensorflow/tensorflow

[20] D. Kingma, J. Ba, "Adam: A Method for Stochastic Optimization", in International Conference on Learning Representations, 2014.

[21] A. Marathe, R. Anirudh, N. Jain, A. Bhatele, J. Thiagarajan, B. Kailkhura, J. Yeom, B. Rountree, T. Gamblin. "Performance Modeling Under Resource Constraints Using Deep Transfer Learning", in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC 17). 2017.

[22] Network Time Protocol (NTP), 2018. [Online]. Available http://www.ntp.org/