Predicting Distributions of Service Metrics using Neural Networks

Forough Shahab Samani [†‡] and Rolf Stadler[†‡]
[†] Dept of Network and Systems Engineering, KTH Royal Institute of Technology, Sweden
[‡]Swedish Institute of Computer Science (RISE SICS), Sweden
Email: {foro, stadler}@kth.se

***Abstract*—We predict the conditional distributions of service metrics, such as response time or frame rate, from infrastructure measurements in a cloud environment. From such distributions, key statistics of the service metrics, including mean, variance, or percentiles can be computed, which are essential for predicting SLA conformance or enabling service assurance. We model the distributions as Gaussian mixtures, whose parameters we predict using mixture density networks, a class of neural networks. We apply the method to a VoD service and a KV store running on our lab testbed. The results validate the effectiveness of the method when applied to operational data. In the case of predicting the mean of the frame rate or response time, the accuracy matches that of random forest, a baseline model.**

***Index Terms*—Service Engineering, Machine Learning, Generative Models, Network Management**

I. Introduction

Methods from statistical learning are increasingly adopted as tools for network and service engineering. Using measurements collected from infrastructure and services, these methods allow us to learn models that predict service quality, likelihood of component failures, etc.

Such predictions are generally modeled as point values of continuous variables, for instance, the mean response time of a service or the mean time for a system component to fail. Regression methods, based on linear regression, random forest, or neural networks, for instance, predict the mean of the target variable (e.g., the response time), conditioned on the input (i.e., infrastructure measurements).

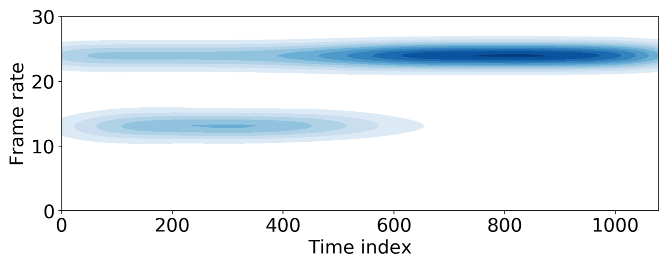


Fig. 1. Evolution of the frame rate, measured on a client, of a VoD service running on the KTH lab testbed under periodic load (see Section IV-C).

Predicting point values like the *conditional mean*, however, provides a limited description of the target variable. This is illustrated in Figure 1, which shows the evolution over time of the frame rate measured on a video-on-demand (VoD) client on our testbed. During the first half of the experiment (time interval 0-600), the distribution of the frame rate is bimodal, with values either close to 12 or 24 frames/sec, and predicting the mean of the frame rate, around 18 frames per second, provides only limited knowledge about the service behavior. Even if the second part of the figure (time interval 600-1200) suggests that the distribution is unimodal, its variance seems to change over time, which may be valuable to know.

In this paper, we present a method for predicting and evaluating the *conditional distributions* of service metrics. From such distributions, key statistics of these metrics, including mean, variance, or percentiles can be derived. Following an approach proposed by C. Bishop, we model the distributions as Gaussian mixtures, whose parameters we predict using *mixture density networks*, a class of neural networks [1]. Keras, a neural-network API, allows us to implement mixture density networks in an elegant and efficient way [2].

We apply the method to a VoD service and a key-value (KV) store running on our lab testbed, where we collect statistics under different load patterns. The results validate the suitability of the method and suggest that accurate prediction of conditional distributions of service metrics from infrastructure statistics can be made. In the case of predicting the mean of the frame rate or response time, the accuracy matches that of random forest, a baseline model. A lesson we learned is that the search for hyper-parameters for mixture density networks (which result in small neural networks and small prediction errors) is hard and time-consuming.

The main contribution of this paper is strong experimental proof that distributions of service metrics can be accurately predicted from operational statistics in a cloud environment. We demonstrate this for two very different, cluster-based services and different load patterns. We believe that the prediction of generative models like Gaussian mixtures for KPIs, configuration parameters, etc., provides a powerful tool for network and service engineering.

The remainder of this paper is organized as follows. Section II formalizes the problem and details the approach to address it. Section III describes the method we are using to obtain the mixture models and the metrics for evaluating the predictions. Section IV details our testbed, the experiments we are conducting, the metrics were are collecting during experiments and the traces we generate from this data. Section V discusses the model predictions and the evaluation results. Section VI surveys related work. Finally, Section VII presents conclusions and future work.

II. Problem Setting and Machine Learning Methods Used in This Work

We investigate how to map infrastructure statistics X to service-level metrics Y using supervised learning. The infrastructure statistics X include measurements from a server cluster and a network. The service-level metrics Y refer to performance indicators on a client, for example, frame rate or response time. Details regarding the composition of X and Y are given in Section IV-B.

The metrics X and Y evolve over time, influenced, e.g., by the load on the servers, operating system dynamics, network traffic, and the number of active clients. Assuming a global clock that can be read on the machines in the server cluster, the network devices, and the clients, we model the combined evolution of the metrics X and Y as a time series $\{(x_t, y_t)\}$.

Our objective is to estimate the distribution (or density) of the service-level metric Y conditioned on knowing the infrastructure metric $x \in X$.

Using the framework of statistical learning, we model X and Y as random variables. We assume that the measurements $\{(x_t, y_t)\}$ are samples drawn from the joint probability distribution of (X, Y). Further, we assume $x_t \in \mathbb{R}^d$ (in the context of this work, d takes values from tens to thousands). Lastly, we assume $y_t \in \mathbb{R}$, i.e., y_t is one-dimensional (or univariate), which simplifies the formal description that follows.

Formally, the problem is finding a model $M : x \rightarrow P(Y|x)$ for $x \in X$, such that the likelihood function $\mathcal{L}(\{P(y_t|x_t)\})$ is maximized, which is achieved by minimizing the error function $E = -log(\mathcal{L})$ [3]. In other words, the goal is to predict the conditional distribution $P(Y|x)$ for all $x \in X$.

We model this distribution $P(Y|x)$ as a *mixture of Gaussians*, i.e., as a weighted sum of Gaussian distributions.

The resulting density model is a linear combination of kernel functions

$$P(Y|x) = \sum_{i=1}^{K} \alpha_i(x)\phi_i(Y|x) \tag{1}$$

where $\alpha_i(x) \in (0,1)$ are the mixing coefficients, $\phi_i(Y|x)$ are the conditional density functions of the target variable Y for the i^{th} kernel, and $K \geq 1$ is the number of kernels.

The Gaussian kernels have the form

$$\phi_i(Y|x) = \frac{1}{(2\pi)^{\frac{1}{2}}\sigma_i(x)} exp\{\frac{\|Y - \mu_i(x)\|^2}{2\sigma_i(x)^2}\} \tag{2}$$

where $\mu_i(x)$ and $\sigma_i(x)^2$ denote the mean and variance of the i^{th} kernel. It is important to note that the mixture model presented here can model arbitrary distribution functions, which makes this approach general [4], [5].

III. Mixture Density Networks (MDNs)

A mixture density network (MDN) is a specialized neural network that estimates the parameters of a Gaussian mixture model in order to predict a conditional distribution. MDNs have been proposed by C. Bishop in 1994 [1] and have been used regularly by researchers because of good performance on specific data sets [6] [7] [8] [9]. (To clarify a possible confusion: the standard way of finding the parameters of a Gaussian mixture model for a given data set X is applying the expectation-maximization algorithm (EM) to maximize the likelihood of the data [3]. Note that our case is different, as we model a *conditional* distribution $P(Y|x)$.)

An MDN is a fully connected neural network with at least two layers in addition to the input: the input layer, one or more hidden layers, and the output layer. The output layer is specialized to estimate the parameters of the mixture model. It contains three types of nodes which differ in the activation function they are using. The first type of nodes uses softmax as activation function and estimates the mixing coefficients α_i

$$\alpha_i = \frac{exp(a_i^{\alpha})}{\sum_{j=1}^{K} exp(a_j^{\alpha})} \tag{3}$$

for $i = 1,..,K$, where K is the number of mixture components. Softmax ensures that $\alpha_i > 0$ and that they sum up to 1.

The second type of nodes uses identity as activation function and computes the centers of the Gaussian kernels μ_i

$$\mu_i = a_i^{\mu} \tag{4}$$

for $i = 1,..,K$.

The third type of nodes uses the exponential function for activation and computes the variance of the kernels σ^2

$$\sigma_i = exp(a_i^{\sigma}) \tag{5}$$

for $i = 1,..,K$.

The exponential function ensures that the variance is positive. Figure 2 illustrates an MDN for a univariate target.

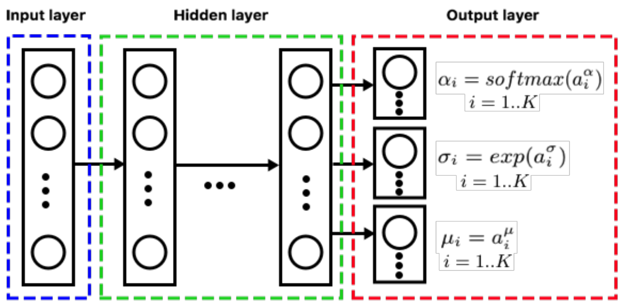


Fig. 2. Structure of the Mixture Density Network for a univariate target. All layers are fully connected.

The error function can be written as

$$E = \sum_q E^q \tag{6}$$

where the contribution to the error from sample q is given by

$$E^q = -ln\{\sum_{i=1}^{K} \alpha_i(x^q)\phi_i(y^q|x^q)\} \tag{7}$$

An MDN is trained trough back-propagation, which computes gradients for iteratively minimizing the error function. Back-propagation requires the derivatives of the error function with respect to the activation variables a_i^α, a_i^μ, and a_i^σ. This is explained in [3].

A. Application of Mixture Models

A mixture density network allows us to estimate the conditional distribution $P(Y|x)$. From this distribution, many statistics of the target variable can be computed. We give two examples of such statistics which will be used later to evaluate the proposed method experimentally. First, we can compute the mean target values for input x from

$$\mathbb{E}[P(Y|x)] = \int_Y \sum_{i=1}^{K} \alpha_i(x)\phi_i(x)p(y)\ \mathrm{d}y = \sum_{i=1}^{K} \alpha_i(x)\mu_i(x). \tag{8}$$

This formula allows us to predict a y value for a given x and thus perform regression. We evaluate the accuracy of such predictions using the metrics $NMAE$ and R^2 (see IV-E).

A second statistic we can compute from the mixture model relates to percentiles of the target variable. Per definition, the $100perc$ percentile of the conditional probability $P(Y|x)$ is the value $a(x)$ in

$$perc = \int_{y=-\infty}^{a(x)} P(Y|x)\ \mathrm{d}y \tag{9}$$

Knowing the parameters of the mixture model $\alpha_i, \mu_i, \sigma_i$, we can numerically compute $a(x)$ for a given $perc$, which means that, knowing $x, perc$, and the model parameters, we can predict $a(x)$.

We evaluate the accuracy of this prediction by estimating $perc$ from the predicted values $a(x_t)$ on the test set. An estimator for $perc$ is

$$\frac{1}{n}\sum_{t=1}^{n} \mathbb{1}\{y_t \leq a(x_t)\} \tag{10}$$

where $\mathbb{1}$ is the indicator function. The justification for (10) is based on the Glivenko–Cantelli theorem. Since our data sets contain only one value y_t for a given x_t the prediction is made over the entire X space, namely

$$\int_X \int_{y=-\infty}^{a(x)} P(Y|x)\ \mathrm{d}yP(x)\ \mathrm{d}x = perc\int_X P(x)\ \mathrm{d}x = perc.$$

B. Modeling MDNs in Keras

Keras is a high-level neural network API, which allows users to rapidly define, train, and evaluate neural networks for regression or classification [2]. It is written in Python and runs on top of other platforms, including Tensorflow, CNTK, and Theano.

In Keras, the architecture of a neural network can be defined in a modular way. There are predefined modules that model a dense layer, a convolutional layer, or a recurrent layer, for instance. A range of activation functions is supported, including sigmoid, softmax, relu, to name but a few. Similarly, predefined error functions and regularization schemes can be chosen. Different flavors of forward propagation are supported, such as dropout, as well as various termination criteria for gradient descent, for instance, early stopping. Keras supports auto-differentiation, which means that users do not need to implement back-propagation. Further, input data can be automatically partitioned into training, validation, and test sets.

To implement an MDN in Keras, we create a sequential model with a number of predefined dense layers and build our own output layer. The output layer is written by customizing a base class "Layer". This involves partitioning the layer into three separate layers (with softmax, linear, and exponential activation function, respectively) and constructing the weight matrix. Further, the error function is defined through equation (7).

There are a handful of MDN implementations in the public domain. One of them has been written using Edward, a Tensorflow library which provides an API for probabilistic modeling. We found Edward not suitable for our purposes since it lacks many features of Keras for model training and evaluation. Our implementation is based on [10] which uses an earlier version of Keras.

IV. Testbed and Traces

A. Testbed and services

In this section, we describe the experimental infrastructure and the structure of the data traces that we create. Further, we describe the services that run on this infrastructure, namely, a Video-on-Demand (VoD) service and Key-Value (KV) store. Lastly, we explain the load patterns we use and the experiments we run to obtain the traces.

Figure 3 outlines our laboratory testbed at KTH. It includes a server cluster, an emulated OpenFlow network, and a set of clients. 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 synchronized through NTP [11].

The VoD service uses VLC media player software [13], which provides single-representation streaming with varying

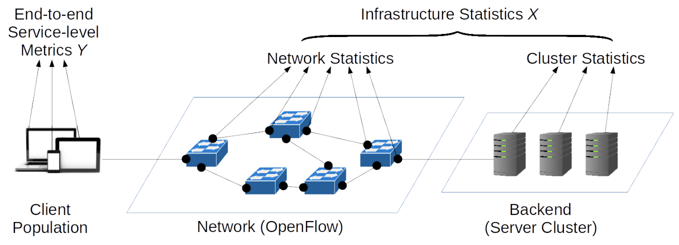


Fig. 3. The testbed at KTH, providing the infrastructure for experiments [12].

frame rate. It is deployed on six PowerEdge R715 machines —one HTTP load balancer, three web server and transcoding machines, and two network file storage machines. The load balancer runs HAProxy version 1.4.24 [14]. Each web server and transcoding machine runs Apache version 2.4.7 [15] and ffmpeg version 0.8.16 [16]. The network file storage machines run GlusterFS version 3.5.2 [17] and are populated with the ten most-viewed YouTube videos in 2013, which have a length of between 33 seconds and 5 minutes. The VoD client is deployed in another PowerEdge R715 machine and runs VLC [13] version 2.1.6 over HTTP.

The KV store service uses the Voldemort software [18]. It executes on the same machines as the VoD service. Six of them act as KV store nodes in a peer-to-peer fashion, running Voldemort version 1.10.22 [18]. The OpenFlow network includes 14 switches, which interconnect the server cluster with clients and load generators. The load generators emulate client populations.

A more detailed description of the testbed setup is given in [12].

B. Collected data and traces

We describe the metrics we collect on the testbed, namely, the input feature sets $X_{cluster}$ and X_{port} —the union of which we refer to as X —as well as the specific service-level metrics Y_{VoD} and Y_{KV}.

The $X_{cluster}$ feature set is extracted from the kernel of the Linux operating system that runs on the servers executing the applications. To access the kernel data structures, we use System Activity Report (SAR), a popular open-source Linux library [19]. SAR in turn uses procfs [20] and computes various system statistics over a configurable interval. Examples of such statistics are CPU core utilization, memory utilization, and disk I/O. $X_{cluster}$ includes only numeric features from SAR, about 1 700 statistics per server.

The X_{port} feature set is extracted from the OpenFlow switches at per-port granularity. It includes statistics from all switches in the network, namely 1) Total number of Bytes Transmitted per port, 2) Total number of Bytes Received per port, 3) Total number of Packets Transmitted per port, and 4) Total number of Packets Received per port.

The Y_{VoD} service-level metrics are measured on the client device. During an experiment, we capture the following metrics: *1) Display Frame Rate (frames/sec)*, i.e., the number of displayed video frames per second; *2) Audio Buffer Rate (buffers/sec)*, i.e., the number of played audio buffers per second. These metrics are not directly measured, but computed from VLC events like the display of a video frame at the client's display unit. We have instrumented the VLC software to capture these events and log the metrics every second.

The Y_{KV} service-level metrics are measured on the client device. During an experiment, we capture the following metrics: *1) Read Response Time* as the average read latency for obtaining responses over a set of operations performed per second; *2) Write Response Time* as the average write latency for obtaining responses over a set of operations performed per second. These metrics are computed using a benchmark tool of Voldemort, which we modified for our purposes. The read and write operations follow the request–reply paradigm, which allows for tracking the latency of individual operations. We instrumented the benchmark tool to log the metrics every second.

Generating the traces: During experiments, X and Y statistics are collected every second on the testbed. For each application running on the testbed, the data collection framework produces a trace in form of a time series $\{(x_t, y_t)\}$. We interpret this time series as a set of samples $\{(x_1, y_1), ..., (x_m, y_m)\}$. Assuming that each sample in the set is drawn uniformly at random from a joint distribution of (X, Y), we obtain predictions models using methods from statistical learning.

C. Generating load on the testbed

We have built two load generators, one for the VoD application and another for the KV application. The VoD load generator dynamically controls the number of active VoD sessions, spawning and terminating VLC clients. The KV load generator controls the rate of KV operations issued per second. Both generators produce load according to two distinct load patterns.

1) Periodic-load pattern: the load generator produces requests following a Poisson process whose arrival rate is modulated by a sinusoidal function with starting load level P_S, amplitude P_A, and period of 60 minutes;

2) Flashcrowd-load pattern: the load generator produces requests following a Poisson process whose arrival rate is modulated by the flashcrowd model described in [21]. The arrival rate starts at load level F_S and peaks at flash events, which are randomly generated at rate F_E events/hour. At each flash event, the arrival rate increases within a minute to a peak load F_R. It stays at this level for one minute and then decreases to the initial load within four minutes.

Table I shows the configurations of the load generators during the experiments reported in Section IV-A. We used a single load generator for the VoD experiments (see [22]) and three for the KV experiments (see [12]).

All traces we used have been created by stochastic models in an attempt to approximate real scenarios. The flash-crowd load pattern, for instance, is based on a model from the research literature and produces arrival patterns that are quite hard to predict. We plan to use in future work traces from operational environments in addition to synthetic traces.

TABLE I
CONFIGURATION PARAMETERS OF VOD AND KV LOAD GENERATORS.

Application	Load Generator	Periodic-load		Flashcrowd-load		
		P_S	P_A	F_S	F_E	F_R
VoD	1	70	50	10	10	120
KV	1	1 000	800	200	10	1 800
	2, 3	350	150	200	3	500

D. The experiments chosen for this paper

The prediction method proposed in this paper has been evaluated using data from four experiments. Two of them involve running the VoD service and two the KV service.

1) *VoD periodic*: In this experiment, we run the VoD service and generate a periodic load pattern on the testbed. Load generator and client are directly connected to the server cluster, and the testbed does not include the network (see Figure 3). Data is collected every second over a period of $50\,000$ seconds. The X feature set contains $4\,984$ features. After cleaning the dataset, $1\,409$ features remain for processing. Using tree-based feature selection [23], the input space is further reduced to the top 30 features. From the service metrics Y only the video frame rate is used in this investigation. For model training, we use the first $21\,600$ samples of the trace. More details about the experiment are given in [22], and the trace is available at [24].
2) *VoD flashcrowd*: This experiment relies on the same setup as VoD periodic, except that the testbed is loaded using the flashcrowd pattern. We use the first $3\,600$ samples of the trace to train the model. We process the trace the same way as described above and the given references contain more information.
3) *KV periodic*: In this experiment, we run the KV service to generate a periodic load pattern on the testbed. Unlike the VoD periodic experiment, we connect load generator and clients to the server cluster via an OpenFlow network (seeIV-A). Measurements are collected every second over a period of $28\,962$ seconds. The $X\_cluster$ feature set contains $10\,374$ features. After cleaning the data set, $1\,649$ features remain for the processing. We use univariate feature selection [23] to decrease number of features to the 200 top features. (The fact that we use two different methods for feature selection in different scenarios has historical, not methodological reasons. We could have used a single method throughout.) The X_port feature set contains 176 features, and after cleaning 134 features remain. From the service metrics Y only the response time for read operations is used in this work. For model training, we use the first $3\,600$ samples of the trace. More details about the experiment are given in [12], and the trace is available at [25].
4) *KV flashcrowd*: This experiment relies on the same setup as KV periodic, except that the testbed is loaded using the flashcrowd pattern. We process the trace the same way as described above and the given references contain more information.

E. Evaluation metrics for mean estimates

To evaluate the model accuracy in predicting mean values we use two metrics. The first is Normalized Mean Absolute Error (NMAE) which is defined as $\frac{1}{\bar{y}}(\frac{1}{m}\sum_{i=1}^{m}|y_i - \hat{y_i}|)$ where $\hat{y_i}$ is the mean value of the target variable, $\bar{y}$ is the average value of the target variable, and m is the number of samples. NMAE is an intuitive and useful measure in the application domain.

The second metric is the Coefficient of Determination (R^2), which is defined as $1 - (\frac{\sum_{i=1}^{m}(y_i - \hat{y_i})^2}{\sum_{i=1}^{m}(y_i - \bar{y})^2})$. R^2 provides theoretical insight: $R^2 = 1$ means perfect prediction, $R^2 = 0$ is the accuracy of a naïve estimator that predicts the sample mean. R^2 takes values in $(-\infty, 1]$.

V. MODEL COMPUTATION AND RESULTS

A. Regression using neural networks

Finding the parameters of a Gaussian mixture model using an MDN means solving a regression problem with a neural network. To gain experience with neural network regression on our data, we first performed a range of experiments with the goal of predicting the mean of the target variable (i.e., frame rate or response time). For comparison, we performed the same predictions using random forest [26], a method we found gave us the best predictions on our traces in the past [12] [22]. For each trace, we performed an extensive search of the space of hyper-parameters to find the most accurate neural network regressor. The process included finding the number of layers and nodes, the regularization parameter, the batch size, the number of epochs, and the parameters of the optimizer like the learning rate [27]. In particular, we applied parallel search and “baby-sitting” as search strategies.

We eventually found hyper-parameters that predict the frame rate or response time with an accuracy that matches that of random forest, for all four traces. The neural network architecture that performs best on both VoD traces has the structure [30,50,50,50,1] (which means that the input layer has 30 features, the three hidden layers have 50 nodes each, and the output layer has a single node). The regularization parameter λ used here is 0.01. Similarly, the architecture that performs best on both KV traces has the structure [200,20,20,20,1], and we used $\lambda = 0.06$. (The figures suggest that learning the frame rate from the VoD data set is harder than learning the response time from the KV data sets, which is consistent with Figure 5, where the VoD trace shows large fluctuations.)

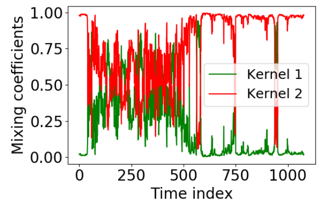
Overall, we experienced the search for hyper-parameters hard and time-consuming, which confirms that this topic is still an active research area.

B. *Predicting distributions $P(Y|x)$ using MDNs*

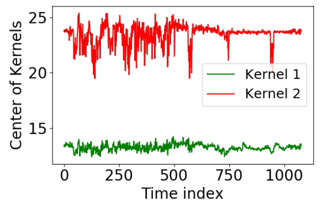
For each trace, we determine the parameters of a Gaussian mixture model. This process includes searching for the best hyper-parameters of the MDN and the number of kernels K that gives the best prediction for the particular trace. We find that $K = 6$ gives the smallest error for VoD periodic; for all other experiments, $K = 4$ gives the smallest error.

Figure 4 illustrates the obtained mixture model for the VoD periodic load pattern. For better readability, we show here the suboptimal case with two kernels, i.e., $K = 2$. This means, that for each input of X, the model gives 6 parameters. These 6 parameters describe the behavior of the system during the experiment. The graphs (a) to (c) show the time series of the mixing coefficients α_i, the kernel centers μ_i and the kernel variance σ_i^2. Graph (b) suggests a bimodal distribution of the frame rate with center at 12 and 24 frames/sec, respectively, which is consistent with Figure 1. Graph (a) suggests a fluctuation of the frame rate during the time interval [0,500] and a more steady frame rate during [501,1 080]. This observation is consistent with the measurements shown in Figure 5 (a). Graph (c) shows that the variance is larger during the time interval [0,500] than during the following interval.

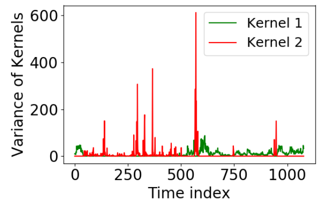
Figure 4 (d) shows the predicted distribution of the frame rate for the specific time index 150. It indicates the measured value at that time as a vertical bar. Also, the figure suggests that the frame rate is either close to 12 or 24 frames/sec.



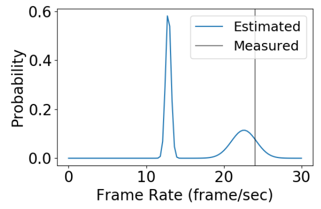
(a) $\alpha_i, i = 1, 2$



(b) $\mu_i, i = 1, 2$



(c) $\sigma_i^2, i = 1, 2$



(d) Distribution at time index 150

Fig. 4. (a) - (c): Time series of predicted parameters for a Gaussian mixture model. The parameters are learned from the trace (VoD periodic load pattern). (d) The predicted distribution and the measured value at time index 150.

C. *Predicting means $\mathbb{E}[P(Y|x)]$*

Using the parameters of the mixture model, we predict the mean of the target variable by applying equation (8).

Figure 5 shows the time series of the predicted target values for the test set. For comparison, the measured values from the test set and the average of the target, which is the output of the naïve estimator, are also shown. The time axis on the graphs covers an interval of 3 600 seconds. This interval captures a single period of the periodic load pattern. The part of the test set covered by this interval includes 1 080 samples. Each sample corresponds to exactly one index value. The graphs show that, for the most parts, the predicted values are close to the measured values.

Table II shows the estimation error (see Section IV-E) for expected target values for two different load pattern of the VoD service. K is the number of kernels of the mixture model that gives the best estimation. It turns out that the two load patterns produce two different Gaussian mixtures, resulting in predictions with slightly different accuracies.

Table III compares three baseline models with the above mixture model from periodic load. The errors on the test set for the neural network and random forest are very close to those of the mixture model—0.118 and 0.119 compared with 0.12. The naïve estimator performs much worse. As expected, random forest produces a high-variance model (i.e., the difference between training error and test error is large), while the variance of other predictors is low.

Table IV gives the estimation error for the KV service. Similar to the results in Table II, the mixture models produced by the two load patterns give slightly different errors.

Our first conclusion from the above evaluation is that the predictions produced by the Gaussian mixture models match the accuracy of the best baseline model, namely, random forest. (We gave here only the results for VoD under periodic load. The remaining three evaluations gave the same results.)

Second, we find that our approach for predicting mean values is not only accurate for the VoD service, but also for the KV service. Both services are very different in nature and are built on fundamentally different designs. Our results thus suggest that the approach generalizes to different services and load patterns.

Third, We found that K is an important parameter of the prediction model and is costly to identify, since it increases the complexity of the search space of the MDN.

TABLE II
ESTIMATION ERROR OF MEAN FRAME RATE FOR VOD SERVICE UNDER DIFFERENT LOAD PATTERNS. K INDICATES THE NUMBER OF KERNELS THAT PRODUCE THE BEST PREDICTION.

Trace	Train		Test		K
	NMAE	R^2	NMAE	R^2	
VoD periodic Load	0.10	0.53	0.12	0.43	6
VoD Flash-crowd Load	0.08	0.44	0.08	0.43	4

TABLE III
ESTIMATION ERROR FOR BASELINE MODELS FOR VOD PERIODIC LOAD.

VoD periodic Load	Train		Test	
	NMAE	R^2	NMAE	R^2
Neural Network Regressor	0.113	0.28	0.118	0.24
Random Forest	0.04	0.76	0.119	0.46
Naïve estimator	0.22	0.0	0.218	0.0

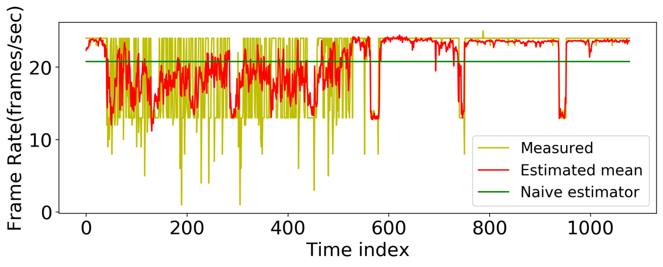
D. *Predicting Percentiles*

Our objective is to predict the 25, 50, and 95 percentiles of the service metrics in all four experiments. Using the

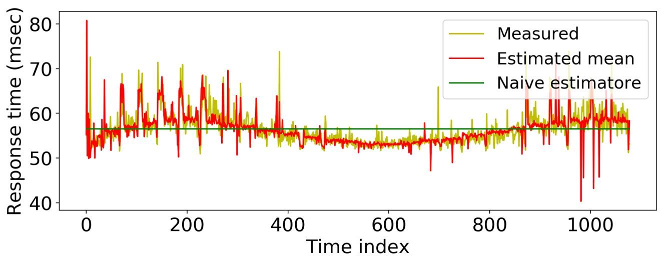
TABLE IV

ESTIMATION ERROR OF MEAN VALUES FOR KV SERVICE UNDER DIFFERENT LOAD PATTERNS. THE VALUE OF KS INDICATES THE NUMBER OF KERNELS THAT PRODUCE THE BEST PREDICTION.

Trace	Train		Test		K
	NMAE	R^2	NMAE	R^2	
KV periodic Load	0.024	0.52	0.027	0.26	4
KV Flash-crowd Load	0.0188	0.71	0.020	0.59	4



(a) VoD periodic



(b) KV periodic

Fig. 5. Time-series of measured and predicted values of the test set for VoD service and KV service. (a) shows the time-series of measured and predicted values of VoD service periodic load pattern. (b) shows the time-series of measured and predicted values of KV service periodic load pattern

parameters of the mixture density model, we compute the percentile values $a(x)$ as discussed in Section III-A. Figure 6 illustrates this by giving the predictions for 25 and 95 percentiles as a time series over the test set, both for VoD and KV under the periodic load.

In order to evaluate the accuracy of the predictions, we perform the evaluation over the test set, use the estimator in equation (10) to predict $perc$, and compare the result with the ground truth of 25, 50, and 95, respectively (see Section III-A).

Tables V and VI show the outcome of the evaluation. We observe that the predicted values are close to the true values, and that the predictions for KV are slightly better than those for VoD.

In contrast to the prediction of mean values above, we do not have a baseline model with whom to compare the accuracy of our predictions. This is part of future work.

E. Predicting the aggregate distribution

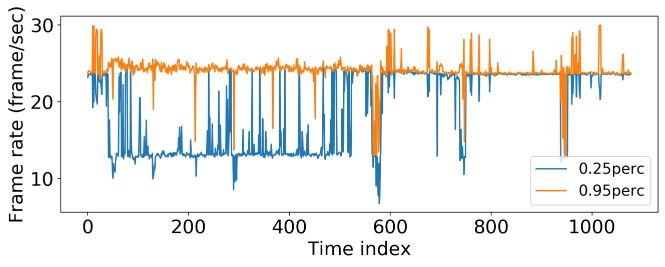
Our objective is to predict the aggregated distribution of the service metrics over the test set in all four experiments. As in the evaluation above, we show the results for VoD and KV under the periodic load. (The results for flash-crowd load are very similar.)

In order to estimate the aggregated distribution, we draw a sample from the mixture-model distribution for each x in the test set. We then use Kernel Density Estimation (KDE) to fit a density distribution to the sample set. Figure 7 shows the predicted aggregated distributions for VoD periodic and KV periodic. For comparison, we add the distributions from the measured values of the test set. Visual inspection suggests that the mixture models predict the measured density distributions very closely. An alternative approach to MDN, which will allow us to compare the accuracy of prediction to a baseline model is planned for future work.

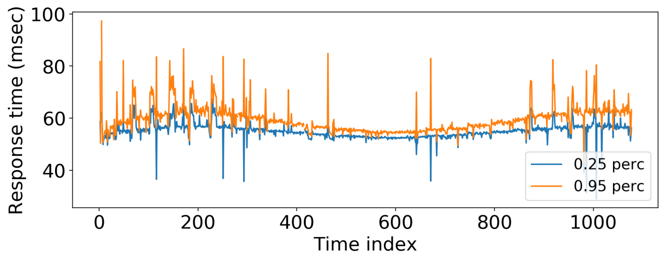
TABLE V

PREDICTED PERCENTILES OF FRAME RATE OVER TEST SET.

Trace	0.25 perc	0.50 perc	0.95 perc	K
VoD periodic Load	0.31	0.52	0.94	6
VoD Flash-crowd Load	0.26	0.54	0.90	4



(a) VoD periodic



(b) KV periodic

Fig. 6. Time-series of predicted percentiles of the test set for VoD and KV services.

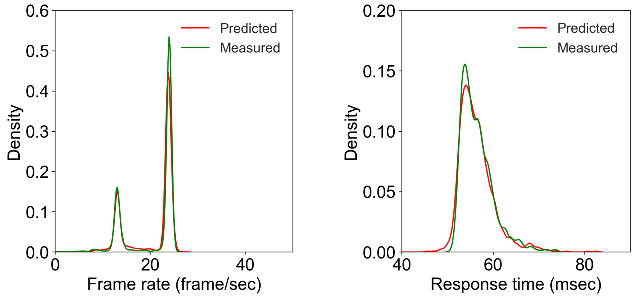
TABLE VI

PREDICTED PERCENTILES OF RESPONSE TIME OVER TEST SET.

Trace	0.25 perc	0.50 perc	0.95 perc	K
KV periodic Load	0.25	0.49	0.93	4
KV Flash-crowd Load	0.24	0.49	0.90	4

VI. RELATED WORK

Several studies have applied statistical learning to predicting KPI metrics from infrastructure measurements. To our knowledge, all of these works predict point estimates of the target variables [28] [29] [30]. This includes our own past research [22] [12] [31].



(a) VoD periodic (b) KV periodic

Fig. 7. Aggregated density of VoD and KV periodic load over test set.

The Authors in [29] present an approach to predict percentiles of the response time for a web service that supports several types of transactions. They build a model that predicts the response time from the current carried load for each transaction type, based on a collaborative filtering technique and table lookup of stored measurements. For a time series of transactions, they then compute the sequence of expected response times, from which they obtain the sample percentile. The work in [30] aims at estimating the tail end of the response-time distribution of a web service from workload measurements using nonlinear quantile regression. The work is conducted in the context of controlling power strategies to achieve SLA conformance at minimum cost.

The above research aims at producing point estimates, while this work aims at estimating conditional distributions, which is much more general. From such distributions, the above point estimates and other metrics can be derived, as we demonstrate experimentally in Section V.

While the research discussed so far is solely based on observations, i.e., measurements from the infrastructure and services, a different direction of investigation uses explicit knowledge about the structure and functionality of the system to construct a system model that can be analyzed and used for prediction. Examples of such model-based approaches that are applied to predict percentiles of system metrics include [32][33].

In several works, the conditional distribution of the target variable is estimated by discretizing the target domain and thus solving a classification instead of a regression problem [34] [8]. For instance, the authors in [8] argue that using general multidimensional MDNs for modeling the conditional distribution of a multidimensional target variable requires complex computations, which can be avoided by discretizing the target domain. The authors in [35] state that it is easier to model skewed, peaked, or long tail distributions with a discretized target variable than with a Gaussian mixture model. A drawback of the discretization approach is that changing the discretization (e.g., the bin size of a histogram) requires computing a new model, which is not needed in our work.

Finally, an alternative to estimating Gaussian mixture models is to estimate generative models of the target using deep learning architectures [36].

VII. Conclusions and Future Work

We have shown that mixture density networks form an effective method to predict conditional distributions of service metrics from infrastructure measurements. Conditional distributions capture the behavior of a system much better than point estimates, which are currently considered in the domain of network and service engineering. Using traces from our lab testbed, we assessed the accuracy of prediction for three derived statistics—the conditional mean of the target (e.g., expected frame rate or response time), percentile predictions, and the distribution of the target variable over a given time interval.

While we found the predictions we produced surprisingly accurate, it took us considerable time to find effective models. Further work is thus required to efficiently identify suitable models, which will be important in real-time settings.

As an alternative to MDNs for conditional density estimation, several groups have proposed to discretize the target space, especially if the space is high-dimensional. We find a different direction equally promising. It involves deep architectures and directly produces generative models, without relying on mixture models. We plan to pursue this direction for our application domain.

Acknowledgements

The authors are grateful to Erik Ylipää with RISE SICS, as well as to Andreas Johnsson, Farnaz Moradi, Christofer Flinta, and Jawaad Ahmed with Ericsson Research for fruitful discussion around this work. This research has been partially supported by the Swedish Governmental Agency for Innovation Systems, VINNOVA, through project SENDATE-EXTEND.

References

[1] C. M. Bishop, "Mixture density networks," 1994.
[2] Keras, 2018. [Online]. Available: https://keras.io/
[3] C. Bishop, *Pattern Recognition and Machine Learning*. Springer-Verlag New York Inc., 2006.
[4] G. J. McLachlan and K. E. Basford, *Mixture models: Inference and applications to clustering*. Marcel Dekker, 1988, vol. 84.
[5] V. R. da Silva and A. Yongacoglu, "Em algorithm on the approximation of arbitrary pdfs by gaussian, gamma and lognormal mixture distributions," in *Communications (LATINCOM), 2015 7th IEEE Latin-American Conference on*. IEEE, 2015, pp. 1–6.
[6] M. Sugiyama, I. Takeuchi, T. Suzuki, T. Kanamori, H. Hachiya, and D. Okanohara, "Conditional density estimation via least-squares density ratio estimation," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, pp. 781–788.
[7] H. Zen and A. Senior, "Deep mixture density networks for acoustic modeling in statistical parametric speech synthesis," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 3844–3848.
[8] W. Tansey, K. Pichotta, and J. G. Scott, "Better conditional density estimation for neural networks," *arXiv preprint arXiv:1606.02321*, 2016.
[9] M. Walton, M. Ayache, L. Straatemeier, D. Gebhardt, and B. Migliori, "Unsupervised anomaly detection for digital radio frequency transmissions," in *Machine Learning and Applications (ICMLA), 2017 16th IEEE International Conference on*. IEEE, 2017, pp. 826–832.
[10] O. Alemi, "Mdn implementation in keras," 2018. [Online]. Available: https://github.com/omimo/Keras-MDN
[11] NTP, 2016. [Online]. Available: http://www.ntp.org/

[12] R. Stadler, R. Pasquini, and V. Fodor, "Learning from network device statistics," *Journal of Network and Systems Management*, vol. 25, no. 4, pp. 672–698, 2017.

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

[14] HAProxy, 2016. [Online]. Available: http://www.haproxy.org/

[15] Apache HTTP Server, 2016. [Online]. Available: http://httpd.apache.org/

[16] FFmpeg, 2016. [Online]. Available: https://www.ffmpeg.org/

[17] Gluster FS, 2016. [Online]. Available: http://www.gluster.org/

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

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

[20] T. Bowden, B. Bauer, J. Nerin, S. Feng, and S. Seibold, "The /proc Filesystem," *Linux Kernel Documentation*, 2000.

[21] I. Ari, B. Hong, E. L. Miller, S. A. Brandt, and D. D. Long, "Managing Flash Crowds on the Internet," in *Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003. 11th IEEE/ACM International Symposium on*. IEEE, 2003, pp. 246–249.

[22] 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 *Network and Service Management (CNSM), 2015 11th International Conference on*. IEEE, 2015, pp. 135–143.

[23] D. Cournapeau, "Scikit learn," 2018. [Online]. Available: http://scikit-learn.org/stable/modules/feature_selection.html

[24] Mldata, 2018. [Online]. Available: http://mldata.org/repository/data/viewslug/realm-cnsm2015-vod-traces/

[25] R. Pasquini, "traces-jnsm-2017," 2018. [Online]. Available: https://github.com/rafaelpasquini/traces-jnsm-2017

[26] L. Breiman, "Random Forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[27] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016.

[28] S. Handurukande, S. Fedor, S. Wallin, and M. Zach, "Magneto Approach to QoS Monitoring," in *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*. IEEE, 2011, pp. 209–216.

[29] Y. Amannejad, D. Krishnamurthy, and B. Far, "Predicting web service response time percentiles," in *Network and Service Management (CNSM), 2016 12th International Conference on*. IEEE, 2016, pp. 73–81.

[30] P. Bodík, R. Griffith, C. A. Sutton, A. Fox, M. I. Jordan, and D. A. Patterson, "Statistical Machine Learning Makes Automatic Control Practical for Internet Datacenters," *HotCloud*, vol. 9, pp. 12–12, 2009.

[31] J. Alonso, L. A. Belanche Muñoz, and D. Avresky, "Predicting software anomalies using machine learning techniques," in *2011 IEEE International symposium on network computing and applications, NCA 2011: 25-27 August 2011, Cambridge, Massachusetts, US: proceedings*. IEEE Computer Society Publications, 2011, pp. 163–170.

[32] J. F. Pérez and G. Casale, "Assessing sla compliance from palladio component models," in *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2013 15th International Symposium on*. IEEE, 2013, pp. 409–416.

[33] Y. Su, D. Feng, Y. Hua, and Z. Shi, "Predicting response latency percentiles for cloud object storage systems," in *Parallel Processing (ICPP), 2017 46th International Conference on*. IEEE, 2017, pp. 241–250.

[34] E. Frank and R. R. Bouckaert, "Conditional density estimation with class probability estimators," in *Asian Conference on Machine Learning*. Springer, 2009, pp. 65–81.

[35] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," *arXiv preprint arXiv:1601.06759*, 2016.

[36] Y. Li, K. Swersky, and R. Zemel, "Generative moment matching networks," in *International Conference on Machine Learning*, 2015, pp. 1718–1727.