# Predicting Real-time Service-level Metrics from Device Statistics

Rerngvit Yanggratoke*, Jawwad Ahmed†, John Ardelius‡, Christofer Flinta†,
Andreas Johnsson†, Daniel Gillblad‡, and Rolf Stadler*‡
*ACCESS Linnaeus Center, KTH Royal Institute of Technology, Sweden Email: {rerngvit,stadler}@kth.se
†Ericsson Research, Sweden Email:{jawwad.ahmed,christofer.flinta,andreas.a.johnsson}@ericsson.com
‡Swedish Institute of Computer Science (SICS), Sweden Email:{john,dgi}@sics.se

*Abstract*—**While real-time service assurance is critical for emerging telecom cloud services, understanding and predicting performance metrics for such services is hard. In this paper, we pursue an approach based upon statistical learning whereby the behavior of the target system is learned from observations. We use methods that learn from device statistics and predict metrics for services running on these devices. Specifically, we collect statistics from a Linux kernel of a server machine and predict client-side metrics for a video-streaming service (VLC). The fact that we collect thousands of kernel variables, while omitting service instrumentation, makes our approach service-independent and unique. While our current lab configuration is simple, our results, gained through extensive experimentation, prove the feasibility of accurately predicting client-side metrics, such as video frame rates and RTP packet rates, often within 10-15% error (NMAE), also under high computational load and across traces from different scenarios.**

*Keywords*—***Quality of service, cloud computing, network analytics, statistical learning, machine learning, video streaming.***

## I. Introduction

Next-generation telecom and internet services will execute on telecom clouds, which combine the flexibility of today's computing clouds with the service quality of telecom systems. Real-time service assurance will be critical for such environments, and real-time prediction of service-level metrics will be a key capability to achieve service assurance.

Understanding and predicting the performance of telecom cloud services is intrinsically hard. Such services involve large and complex software systems that run on general-purpose platforms and operating systems, which do not provide real-time guarantees. One approach to understand the performance of cloud services is to model the various layers of hardware and software using analytical models and to develop an overall model of the system for end-to-end predictions. Such an approach requires thorough understanding of the functionalities of various components and their interactions, and the resulting system model becomes highly complex.

An alternative approach, which we pursue in this work, is based upon statistical learning whereby the behaviour of the target system is learned from observations. In such a case, a large amount of observational data is needed, but no detailed knowledge about the system components and their interactions is required.

The problem of predicting metrics in cloud and network environments has been studied for some time, for instance, for the purpose of predicting TCP throughput rates, the probability of device failures, and the response times of web applications [1]–[4]. Common to all these works is that a small number (usually up to a dozen) of observation variables, also called features, are selected for predicting a specific metric. Our approach, in contrast, considers all available features (which can be thousands). Also, in our case, feature selection is not guided by the specific metric we want to predict, which makes our approach more general.

The paper contains results from our work with using statistics from a Linux kernel of a server machine in order to predict service-level metrics on a client for a video-streaming service (VLC) [5]. The results are based upon extensive experimentation where we run VLC servers under various load patterns on a laboratory testbed and collect traces with server statistics and client-side metrics. We apply statistical learning methods on these traces, compute models that predict service-level metrics, and evaluate the model accuracies. (Prediction here relates to estimating metrics for current times based on current and past measurements.)

We consider this work as a first step towards engineering a generic functionality for real-time prediction of service-level metrics from device-level statistics. In fact, the current experimental setup is simple (a server machine connected to a client machine), and we measure under idealised conditions (no network congestion, light load on the client machine). Even such a system though has a high complexity when studied at the operating-system level, and a mapping from operating-system level to service-level metrics is far from trivial. Our results show though that this can be done using known methods and technologies.

This paper makes the following contributions. First, we propose and evaluate a novel method for predicting service-level metrics. The method is service independent in the sense that it is designed to work with different services without special configuration or adaptation, and it does not require server-side service-level instrumentation. The method not only predicts metrics for a single execution (trace), but also for a range of realistic executions. Second, our method is based on a simple system model. Although it assumes time synchronisation among system components, the learning method we are using does not consider time dependencies. As the evaluation shows, such a simple system model can be sufficiently accurate. Third, our results, gained through extensive experimentation, prove the feasibility of accurately predicting client-side metrics, such as video frame rates and RTP packet rates, often within 10-15% (normalized mean absolute error), also under high computational load (CPU utilisation $> 90\%$) and across traces from different server load scenarios. We evaluate the accuracy of linear methods and show that non-linear, tree-based methods generally improve the results. Further, we show that preprocessing of kernel statistics is needed and can improve prediction accuracies. We have made the traces from this work available at [6].

The remaining part of this paper is organised as follows. Section II describes the problem setting. Section III highlights concepts from statistical learning used in our work. Section IV discusses the specific statistics and metrics for our work. Section V gives details about the testbed and the experiments. Section VI includes the evaluation of the model accuracies. Section VII compares this work with earlier results. Section VIII concludes this paper and outlines future work.

## II. Problem Setting

Figure 1 gives the basic configuration of the system under investigation. It consists of a server that is connected to a client machine via a network. The client accesses a service $S$, which runs on the server. In this work, we consider a video-on-demand (VoD) service. We are interested in the device statistics $X$ on the server during the time that the client accesses the service. In this setting, device statistics refer to metrics on the operating-system level, for instance, the number of running processes, the rate of context switches, the number of active TCP connections, etc. In contrast, service-level metrics $Y$ on the client side refer to statistics on the application level, for example, video frame rates and RTP packet rates.

The metrics $X$ and $Y$ evolve over time, influenced, e.g., by the load on the server, operating system dynamics, etc. Assuming a global clock that can be read on both the client and the server, we can model the evolution of the metrics $X$ and $Y$ as time series $\{X_t\}_t$, $\{Y_t\}_t$, and $\{(X_t, Y_t)\}_t$.

Our objective is to predict a service-level metric $Y_{t_i}$ at time $t_i$ on the client side, based on knowing the server metrics $X_{t_i}$. Using the method of statistical learning, the problem is finding a learning model $M : X_t \rightarrow \hat{Y}_t$, such that $\hat{Y}_t$ closely approximates $Y_t$ for a given $X_t$. Our problem formulation assumes that a sample $(X_t, Y_t)$ is drawn uniformly at random from a joint distribution $(X, Y)$, and therefore, $M$ does not change over time.

We further assume that the relationship between $X$ and $Y$ does not depend on the network state and device statistics on the client machine. In practical terms, this means that the network and client machine are lightly loaded. (We are aware that these assumptions do not hold for many real systems, and we plan on relaxing them in future work. Previous research has shown how network measurements can be used to predict client-side metrics [7]; therefore, including this aspect would not necessarily add to the contributions of this work.)

## III. Background: Statistical Learning

In this paper, we are assessing four different regression methods to solve the problem stated in Section II. First, we apply (basic) linear regression, a simple method that serves as a baseline. It models the relationship between $X$ and $Y$ as a linear function $\hat{Y} = \sum_{j=0}^{p} X_j \beta_j$ whereby $X_0 = 1$; $X_j, j = 1..p$ are the features of the feature space $X$; and $\beta_j, j = 0..p$ are the model coefficients. The coefficients are computed such that the sum of the squares of the residuals (RSS) are minimised [8]. The solution can be obtained using QR decomposition or a gradient descent method, for instance [9]. (QR decomposition has a computational complexity of $O(Np^2)$, whereby $N$ is the number of samples.)

Second, we use the Lasso regression method, which is a variant of linear regression and mitigates overfitting in case of a high-dimensional feature space. Lasso regression solves the above linear regression problem with the additional constraint $\sum_j |\beta_j| \leq \lambda$, where $\lambda$ is a regularization parameter. The solution to the Lasso regression problem tends to yield a model with smaller coefficients $\beta_j$ (which can even be zero) than the basic linear regression [10].

In addition to the above linear methods, we apply two tree-based methods: regression tree and random forest. The regression tree method computes region boundaries instead of linear coefficients, with the same objective of minimising RSS. It recursively partitions the feature spaces into regions $R_1, R_2, ..., R_M$. For a given $X$, $Y$ is estimated as $\hat{Y} = \sum_{i \in R_k} \frac{Y_i}{|R_k|}$ where $R_k$ is the region that $X$ falls into, $|R_k|$ is the number of training samples in $R_k$, and $i$ is the index of those samples. The regions are constructed using a greedy algorithm, whereby during each construction step of a selected region, a feature and a threshold are identified that fulfil the optimisation criterium [8]. (The method has a computational complexity of $O(N^2 p.)$)

Finally, random forest is an ensemble method. Each estimated value of $Y$ is an average of predictions from a large number of regression trees [11]. Each of these trees is constructed using a different training set, and each construction step uses a randomised reduced feature set [8].

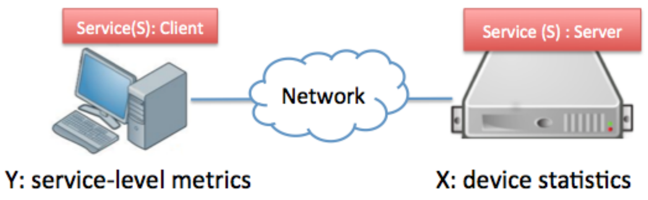

Fig. 1. System configuration for predicting service-level metrics

## IV. Device Statistics and Service-Level Metrics

### A. Device statistics: the feature space X

We obtain device statistics $X$ from the kernel of the Linux operating system that runs on the server (see Figure 1). The Linux kernel is the core of the Linux operating system. It gives applications access to resources, such as CPU, memory, and network, and it schedules requests to those resources. To access the kernel data structures, we use *procfs* [12].

Procfs is based on the Unix file system abstraction. Therefore, kernel data can be accessed as if it was structured in directories and stored in files. For every process, for instance, procfs includes a directory named by the process identifier, and this directory contains invocation parameters, environment variables, status variables, etc., about that particular process.

We rely on two types of feature vectors and feature sets, which are populated using procfs calls. We call these feature sets $X_{proc}$ and $X_{sar}$.

The feature set $X_{proc}$ is based upon features extracted from the leafs of the /proc directory. We include only features with a numerical value and a static structure (which does not change over time). For instance, the amount of free memory has a static structure, while the list of running process identifiers does not, because the length of that list changes over time. Using static structures ensures that the same index value of two feature vectors always corresponds to the same feature. For example, the index 153 may consistently represent the number of active TCP connections. Constructing the feature set in this way, we arrive at a set of some 4000 features. When running experiments on the testbed, a script reads out $X_{proc}$ at the configurable rate. The script takes about 300-400 milliseconds to execute.

The feature set $X_{sar}$ is constructed using System Activity Report (SAR), a popular open source Linux tool [13]. Reading data through procfs, SAR computes various system metrics over a configurable interval. Examples of such metrics are CPU core utilization, memory and swap space utilization, disk I/O statistics, and network statistics. For the feature set $X_{sar}$, we include only numeric features returned by the SAR tool, and we end up with a set size of about 840.

While $X_{proc}$ and $X_{sar}$ share many features, there is a major difference between these feature sets in that $X_{proc}$ contains many operating system counters, but $X_{sar}$ does not. Instead, $X_{sar}$ includes many features that are either derivatives of $X_{proc}$ features (e.g., the current rate of interrupts) or aggregates of such features (e.g., average CPU core utilization). We chose to experiment with both feature sets, using $X_{proc}$ because it contains a large number of features whose values are kernel data and using $X_{sar}$ as an example of a set of preprocessed features that does not include counters.

After collecting the samples, we perform a reduction of the feature set $X_{proc}$ or $X_{sar}$. We remove those features that have constant values across all samples, in order to reduce the time required for model computation. Further, some linear regression methods, like those relying on QR decomposition, require that the feature vectors are linearly independent. In such a case, we reduce the feature set in such a way that the feature vectors become linearly independent [9]. In our work, we also applied Principal Component Analysis (PCA) to create a feature space with a small number of dimensions. Since we did not experience a significant reduction in model computation time, and the model accuracy did not improve, we did not include PCA in our prediction method and do not report specific results in this paper.

### B. *Service-level metrics $Y$ for a video-on-demand service*

For this work, we chose the VLC media player software to provide a video-on-demand service on our testbed, for which we predict service-level metrics.

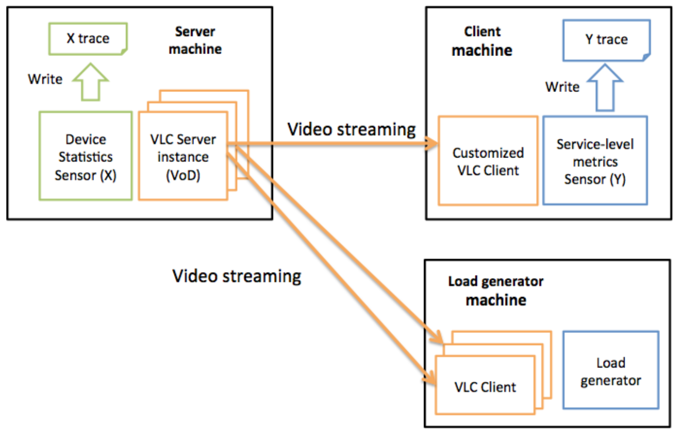

Fig. 2. Testbed setup for creating device statistics $X$ and service-level metrics $Y$ under different load patterns.

The service-level metrics we are considering are measured on the client device (see Figure 1). During an experiment, we capture the following three metrics.

*1) Video frame rate (frames/sec):* the number of displayed video frames per time unit;

*2) Audio buffer rate (buffers/sec):* the number of played audio buffers per time unit;

*3) RTP packet rate (packets/sec):* the number of RTP packets received per time unit.

These metrics are not directly measured, but computed from VLC events like the display of a video frame at the client's display unit, etc. We have instrumented the VLC software to capture these events.

## V. TESTBED AND EXPERIMENTATION

In this section, we describe the hardware and software setup, how we perform the experiments, how we generate load, and how we obtain traces for model computation.

### A. *The testbed*

We run the experiments on our lab testbed at KTH, which includes some 60 rack-based (physical) servers that are interconnected by Ethernet switches. The machines we use in this work are Dell PowerEdge R715 2U rack servers, each with 64 GB RAM, two 12-core AMD Opteron processors, a 500 GB hard disk, and a 1 Gb network controller.

The basic setup for experimentation includes three physical machines, namely, a server machine that provides the video-on-demand service, a client machine that runs a video-on-demand session, and a load generator that creates the aggregate demand of a set of VoD clients. All machines run Ubuntu 12.04 LTS, and their clocks are synchronised through NTP [14]. Figure 2 shows the components that execute on these machines and their interactions during experiments.

The server machine runs one or more VLC servers (version 2.1.3). Each VLC server is configured for video-on-demand service. It transcodes the video and audio streams, and streams the videos over the network to VoD clients. The sensor on the server machine periodically reads out the vector $X$ in form of $X_{proc}$ or $X_{sar}$ (SAR version 10.0.3), as described in Section IV-A. At the start of every second, the sensor reads $X$ and saves it on the local $X$ trace file, together with a timestamp. The server machine is populated with the ten most popular YouTube videos in 2013. The client machine runs a VLC client, whose sensor extracts service-level events. At the start of every second, the sensor collects the events from the last second, computes the $Y$ metrics, and writes them to the local $Y$ trace file, together with a timestamp. The load generator machine dynamically spawns and terminates VLC clients, depending on the specific load pattern that is executed during an experiment.

An experimental run lasts three hours, except the run for creating the periodic-load trace (see below), which lasts 14 hours. At the beginning of a run, the VLC client on the client machine sends a request for playing a specific video to a VLC server, which is selected uniformly at random among the VLC servers on the server machine. Once the video has played, the VLC client sends a new request for the same video to a random VLC server. Also, at the beginning of the run, the load generator starts sending requests according to the selected load pattern, and the sensors on the server and client machines are started. During an experimental run, the server responds to the incoming requests from the load generator.

Depending on the selected load pattern, the load and the utilisation of the server machine can vary significantly.

### B. *Generating load patterns on the testbed*

We have built a load generator that dynamically controls the number of active VoD sessions on the testbed by spawning and terminating VLC clients (Figure 2). When a client is created, it sends a request for a random video to a random VLC server on the server machine. The VLC server then starts streaming the video content to the client. Once the video ends, the client issues a new request, until it is terminated by the load generator. The load generator currently supports the following load patterns.

*1) Constant-load pattern:* the load generator creates 25 clients, which remain active during the experiment.

*2) Linearly-increasing-load pattern:* starting with 0 clients, the load generator creates 5 additional clients every 1500 seconds, until the number of clients reaches 50.

*3) Poisson-load pattern:* the load generator starts clients following a Poisson process with an arrival rate of 15 clients/minute. It terminates a client after an exponentially distributed holding time with an average of one minute.

*4) Periodic-load pattern:* the load generator starts clients following a Poisson process with an arrival rate that starts at 30 clients/minute and changes according to a sinusoid function with a period of 60 minutes and an amplitude of 20 clients/minute. The load generator terminates a client after an exponentially distributed holding time with an average one minute.

*5) Flashcrowd-load pattern:* the load generator starts and terminates clients according to a flash-crowd model described in [15]. The creation of clients follows a Poisson process with

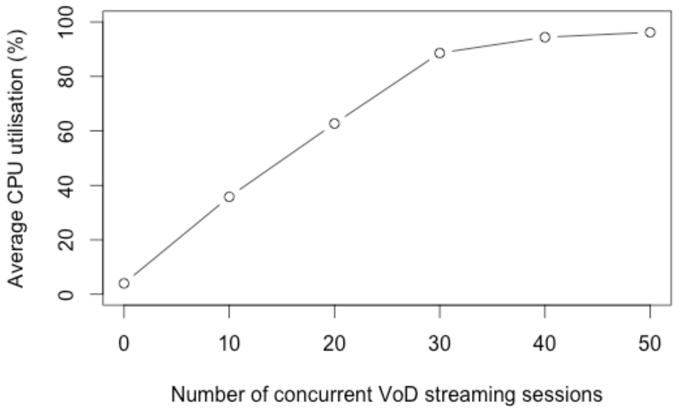

Fig. 3. Average CPU utilisation across cores versus the number of concurrent VoD streaming sessions

an arrival rate that starts at 5 clients/minute and peaks at flash events, which are randomly created at a rate of 10 events/hour. At each flash event, the arrival rate linearly increases 10-fold to 50 clients/minute within one minute, it sustains this level for one minute, and then linearly decreases to 5 clients/minute within 4 minutes. The load generator terminates a client after an exponentially distributed holding time with an average of one minute.

The above load patterns allow us to perform experiments that cover the full range of CPU utilisation of the server machine. The server on our testbed is saturated when it concurrently serves 50 VoD streams. (At 50 concurrent VoD streams, the required bandwidth between the server machine and the load generator machine is about 117 Mbps, which is much lower than the capacity of the switched connection of 1Gbps.) Recall that each audio and video stream is transcoded on the server, which is CPU intensive. Figure 3 shows the average CPU utilisation (i.e., the utilisation across the 24 cores of the machine) as function of the number of concurrent VoD streams from the server machine. The measurements have been generated based on the constant-load patterns. Each point on the graph is an average of three measurements during a ten-second interval.

## VI. Evaluation of the prediction models

In this section, we collect traces produced on our testbed under different load patterns, we apply well-known statistical learning methods on the traces to produce models for predicting service-level metrics, and we evaluate these models against test data from the traces. Then, we draw conclusions on the accuracies of the models for different traces, device statistics, service-level metrics, and learning methods. We have made the traces from this work available at [6].

### A. *Evaluation method*

To evaluate a model, we apply the *validation set* approach, whereby we (1) randomly assign each sample $(X_t, Y_t)$ of a trace to either a training set or a test set, (2) compute the models from the training set, and (3) evaluate the models using the test set [8]. The training set contains 70% of the samples, and the test set 30%. The third evaluation is an exception of

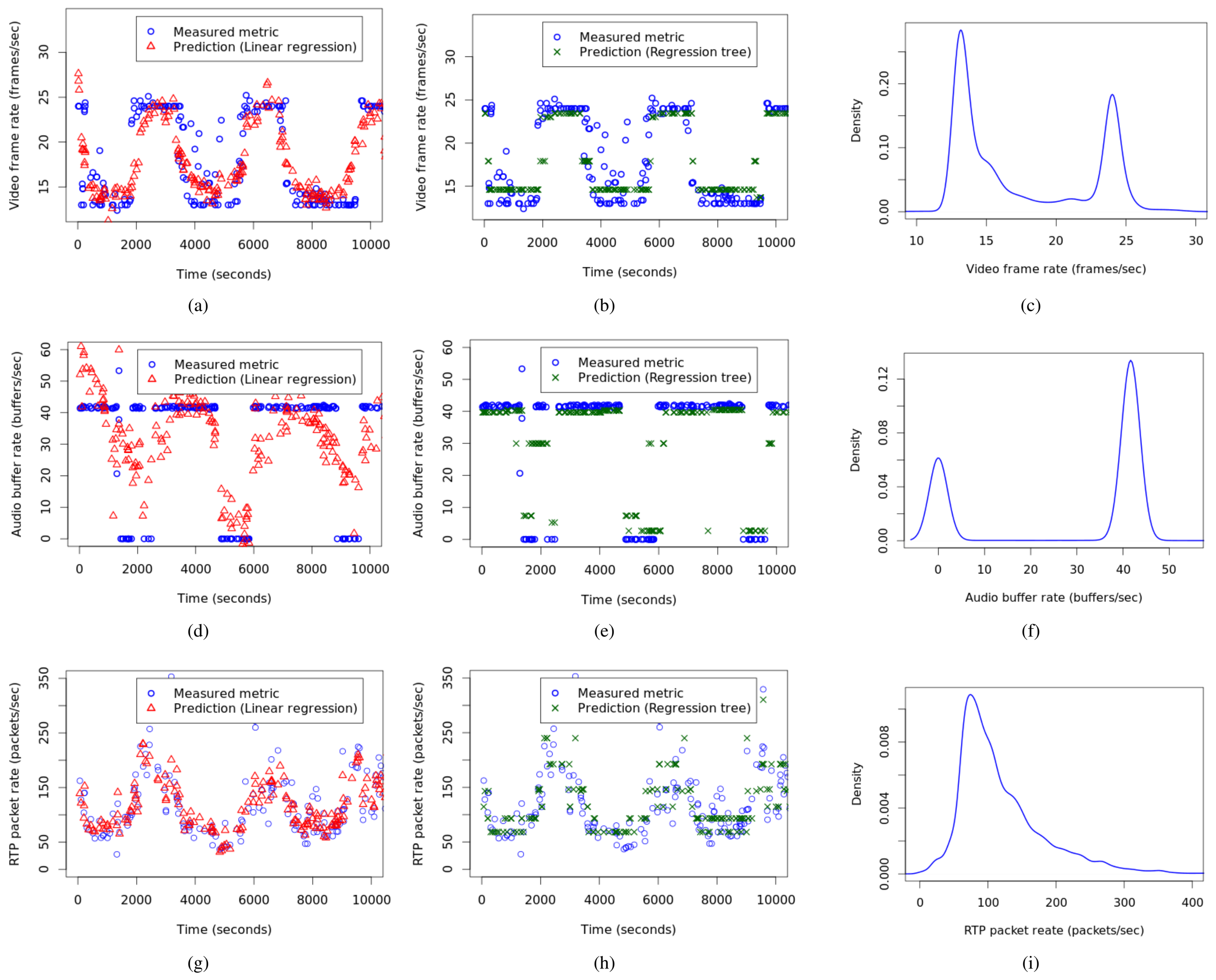

Fig. 4. The left side shows time series of measurements versus model predictions for video frame rate, audio buffer rate, and RTP packet rate, while the right side shows the sample distribution. The measurements are taken from the trace with the periodic load pattern.

this procedure in that we obtain the training set from one trace and the test set from another trace.

We use two accuracy measures to evaluate the learning models. The first is the *Normalized Mean Absolute Error* ($NMAE$), computed as $\frac{1}{\bar{y}}(\frac{1}{m}\sum_{i=1}^{m}|y_i - \hat{y}_i|)$, whereby $\hat{y}_i$ is the model prediction for the measured service-level metric $y_i$, and $\bar{y}$ is the average of the samples $y_i$ of the test set, which is of size $m$. The second measure is the *Normalized* $90^{th}$ *percentile of prediction errors* ($N90^{th}E$), computed as the $90^{th}$ percentile of the prediction errors $|y_i - \hat{y}_i|$, normalized by $\bar{y}$. We use normalized accuracy measures to better compare model accuracies across service-level metrics.

We use R version 3.1.0 for producing and evaluating the learning models [16]. The tool includes implementations of many regression methods in forms of R packages. In this work, we make use of the following R packages: [17] for linear regression using QR decomposition, [18] version 1.9-8 for Lasso regression using coordinate descent (we use the regularlization parameter of 0.02), [19] version 4.1-8 for regression tree, and [20] version 4.6-7 for random forest (we use 200 trees).

## B. Evaluation results

Figure 4 gives a first impression of our predictions of the three service-level metrics, namely, video frame rate, audio buffer rate, and RTP packet rate. The data is taken from the test set of the periodic-load trace that has been produced with the device statistics $X_{sar}$. The right side shows the sample distribution of the service-level metrics, the left side shows the time series of the metrics, and the model predictions for two learning methods, namely, linear regression and regression tree. (Note that the time series show only 1000 random samples from the test set, to make reading the plots easier.)

We observe that, for the video frame rate and the RTP packet rate (4(a), 4(b), 4(g) and 4(h)), the evolution of the metrics is periodic, with the same period as the load pattern.

Also, the model predictions from both linear regression and regression tree seem to follow the measurements well.

For the audio buffer rate (4(d) and 4(e)), a periodic pattern of measurements or predictions is not visually apparent (although a frequency analysis using Fourier transform shows its presence). In addition, the model predictions from linear regression have large errors, while the predictions from regression tree are more accurate. (See further experiments and discussion below.)

The video and audio buffer rates (4(c) and 4(f)) show clearly a bi-modal distribution, which is not the case for the RTP packet rate (4(i)). Inspecting the VLC source code on the client side explains the bi-modal distribution: in the case of a missing video frame at play out time, VLC displays the last frame at 12.5 frames/sec; in case of a missing audio buffer, no buffer is played out.

We have performed four rounds of evaluation, during which we determine model accuracies for predicting the three above mentioned service-level metrics. Each evaluation allows us to answer a specific set of questions.

The first evaluation addresses the questions: how accurate are model predictions for various regression methods, and how does the choice of device statistics, $X_{sar}$ or $X_{proc}$, influence the accuracy? The evaluation is performed on the periodic-load trace.

Table I shows the results. First, $X_{sar}$ consistently offers better predictions than $X_{proc}$ across service-level metrics and regression methods. We believe this is due to the fact that $X_{proc}$ contains many counters, which is not the case for $X_{sar}$ (see Section IV-A).

Second, for $X_{sar}$, the linear methods produce prediction errors ($NMAE$) below 20% for the video frame rate and RTP packet rate, while the corresponding figures for the audio buffer rate are more than 40%. The non-linear methods produce $NMAE$ values below 20% and $N90^{th}E$ values below 42% across all service-level metrics. Further, random forest consistently outperforms regression tree for both $X_{proc}$ and $X_{sar}$ (while the differences are less prominent in case of $X_{proc}$). This indicates that the prediction errors from regression tree are the result of overfitting, which random forest mitigates.

We conclude that (1) random forest with $X_{sar}$ gives the best results; we can predict all service-level metrics with $NMAE$ at most 15% and $N90^{th}E$ at most 27%; (2) as expected, non-linear methods perform significantly better than linear methods; (3) if a 20% error margin is acceptable, a linear method for $X_{sar}$ can be used for predicting video frame rate and RTP packet rate.

In the remaining of the paper, we give results only for $X_{sar}$, the linear regression method, and the random forest method. Linear regression is a representative of a linear method, random forest of a nonlinear method.

The second evaluation addresses the questions: how accurate are model predictions for various load patterns (i.e., traces), and how does the choice of a particular regression method (linear regression and random forest) influence the accuracy?

Table II shows the results of the second evaluation. First, we observe that the prediction error is positively correlated with the variation in load, across all service-level metrics and for both linear regression and random forest. For instance, the constant-load trace has the lowest variation in load and the lowest prediction error, whereas the flashcrowd-load trace has the highest variation in load and the highest prediction error. Second, for all service-level metrics, random forest consistently outperforms linear regression, with $NMAE$ at most 11% and $N90^{th}E$ at most 26%.

Our conclusions from the second evaluation are consistent with those of the first evaluation: a tree-based method gives better results than a linear method; for predicting the video frame rate and the RTP packet rate, both a tree-based method or a linear method provide good prediction accuracy, while, for predicting the audio buffer rate, only a tree-based method is suitable. This means that the suitability of the investigated models is consistent across load patterns (or traces).

The third evaluation addresses the questions: can a model be trained on one load pattern and then used to predict service-level metrics on a different load pattern? If true, which regression method should be used? This issue is important, because it relates to the deeper question of whether one can learn the system behaviour for load patterns that the system has not seen before.

Table III shows the prediction accuracies of linear regression and random forest for the three service-level metrics. The models are learned from the periodic-load trace and are evaluated using the test sets from the four other traces.

We observe that the model learned from the periodic-load trace, with either linear regression or random forest, can predict the video frame rate and RTP packet rate from all other traces with $NMAE$ at most 16% and $N90^{th}E$ at most 35%. Additionally, we observe that random forest consistently outperforms linear regression, which confirms the findings from the first and second evaluations. From this, we conclude that the relationship between the device statistics and service-level metrics is fundamentally nonlinear.

Comparing Table III with Table II, we see that a model learned from the periodic-load trace and evaluated against a different trace has a higher prediction error than a model learned from one trace and evaluated against the same trace. For instance, the model learned with linear regression from the flashcrowd-load trace in Table II has an $NMAE$ value of 9% for the video frame rate, while the corresponding $NMAE$ value in Table III is 14%.

The key lesson from this evaluation is that it is possible to train a model on one load pattern and then use the same model to predict service-level metrics for a different pattern. The evaluation shows this is the case for learning from the periodic-load trace and predicting the video frame rate and the RTP packet rate for other traces. Unfortunately, this is not the case for the audio buffer rate. We currently cannot explain the high prediction error of the audio buffer rate for the flashcrowd-load trace in Table III.

The fourth evaluation addresses the question: can our method be used to accurately predict service-level metrics under high computational load? It is well known that performance

| Device statistics | Regression method | Video frame rate | | Audio buffer rate | | RTP packet rate | |
|---|---|---|---|---|---|---|---|
| | | $NMAE(\%)$ | $N90^{th}E(\%)$ | $NMAE(\%)$ | $N90^{th}E(\%)$ | $NMAE(\%)$ | $N90^{th}E(\%)$ |
| $X_{sar}$ | Linear regression | 12 | 28 | 41 | 85 | 15 | 30 |
| | Lasso regression | 16 | 29 | 51 | 85 | 17 | 31 |
| | Regression tree | 11 | 28 | 19 | 42 | 19 | 41 |
| | Random forest | **6** | **17** | **0.94** | **5.8** | **15** | **27** |
| $X_{proc}$ | Linear regression | 26 | 71 | 59 | 118 | 39 | 69 |
| | Lasso regression | 23 | 44 | 63 | 102 | 35 | 66 |
| | Regression tree | 23 | 44 | 61 | 103 | 36 | 68 |
| | Random forest | 22 | 44 | 60 | 103 | 34 | 68 |

TABLE I. MODEL ACCURACIES FOR $X_{sar}$ AND $X_{proc}$. FOR THE EVALUATION OF VARIOUS MODELS, THE TRAINING AND TEST SETS ARE TAKEN FROM THE PERIODIC-LOAD TRACE.

| Regression method | Trace | Video frame rate | | Audio buffer rate | | RTP packet rate | |
|---|---|---|---|---|---|---|---|
| | | $NMAE(\%)$ | $N90^{th}E$ (%) | $NMAE(\%)$ | $N90^{th}E$ (%) | $NMAE(\%)$ | $N90^{th}E$ (%) |
| Linear regression | Constant-load trace | 0.47 | 1.0 | 0.62 | 1.1 | 12 | 24 |
| | Poisson-load trace | 3 | 6.1 | 3.6 | 6 | 12 | 26 |
| | Linearly-increasing-load trace | 6.1 | 18 | 7.0 | 15 | 13 | 24 |
| | Flashcrowd-load trace | **9** | **22** | **28** | **63** | **14** | **30** |
| Random forest | Constant-load trace | 0.34 | 0.6 | 0.57 | 0.81 | 10 | 23 |
| | Poisson-load trace | 2.0 | 4.1 | 1.3 | 1 | 11 | 24 |
| | Linearly-increasing-load trace | 3.4 | 12 | 0.69 | 0.89 | 11 | 24 |
| | Flashcrowd-load trace | **6.0** | **17** | **4.4** | **10** | **11** | **26** |

TABLE II. MODEL ACCURACIES FOR DIFFERENT TRACES, FOR $X_{sar}$. FOR THE EVALUATION OF THE LINEAR REGRESSION OR RANDOM FOREST MODELS, THE TRAINING AND TEST SETS ARE TAKEN FROM THE SAME TRACE.

| Regression method | Trace | Video frame rate | | Audio buffer rate | | RTP packet rate | |
|---|---|---|---|---|---|---|---|
| | | $NMAE(\%)$ | $N90^{th}E$ (%) | $NMAE(\%)$ | $N90^{th}E$ (%) | $NMAE(\%)$ | $N90^{th}E$ (%) |
| Linear regression | Constant-load trace | 16 | 24 | 19 | 37 | 14 | 27 |
| | Poisson-load trace | 13 | 20 | 15 | 28 | 13 | 27 |
| | Linearly-increasing-load trace | 14 | 34 | 28 | 62 | 14 | 29 |
| | Flashcrowd-load trace | **14** | **30** | **45** | **85** | **15** | **32** |
| Random forest | Constant-load trace | 9.0 | 10 | 4 | 7 | 13 | 28 |
| | Poisson-load trace | 10 | 16 | 12 | 23 | 14 | 30 |
| | Linearly-increasing-load trace | 13 | 27 | 10 | 38 | 14 | 31 |
| | Flashcrowd-load trace | **15** | **34** | **45** | **103** | **16** | **35** |

TABLE III. MODEL ACCURACIES FOR DIFFERENT TRACES, FOR $X_{sar}$. FOR THE EVALUATION OF THE LINEAR REGRESSION OR RANDOM FOREST MODELS, THE TRAINING SET IS TAKEN FROM THE PERIODIC-LOAD TRACE, WHILE THE TEST SET IS TAKEN FROM ANOTHER TRACE.

predictions for high load scenarios is a hard problem. We consider the system to be under high load when the average CPU core utilisation exceeds 90%.

Three of our traces contain samples collected under high load: the linearly-increasing-load trace, the periodic-load trace, and the flash-crowd load trace. From each of these traces, we create a test set with those samples that have been collected under high load. We evaluate the random forest model that we have trained using the (complete) training set from the periodic-load trace against the three test sets. The results (for video frame rate and RTP packet rate) show NMAE values of at most 14.7%, 19.8%, and 12% for the the periodic-load trace, linearly-increasing trace, and the flashcrowd-load trace, respectively.

We conclude that our approach allows for predicting video frame rate and RTP packet rate under high-load conditions. In fact, there is no significant difference in prediction accuracy between samples collected under high load and samples collected under any load (see Tables I and III).

## VII. Related work

While our prediction method aims to be service independent, other works on predicting service-level metrics that we are aware of develop methods that are targeted towards a specific service and metric. Important examples of such works in the context of cloud and statistical learning are [4], [21]–[23]. In contrast to our method, these works consider a small set (less than 10) of carefully selected features to learn from.

Other works like [24]–[27] use statistical learning models to estimate quality-of-experience metrics of a multimedia service. A review of such metrics can be found in [28].

The authors in [29] predict quality-of-service metrics for IPTV using decision trees. The features are selected by a domain expert. In [7], the authors present an approach for learning from a set of network-level metrics, e.g., delay, loss, and jitter measurements, to estimate the quality-of-service metrics for IPTV streaming clients. The authors conclude that their prediction method is accurate, as long as the packet loss ratio is low.

Predicting service-level metrics using statistical learning has been investigated in other contexts. For instance, the authors in [3] predict the effects of ageing software components using various learning techniques.

## VIII. Discussion

In this paper, we proposed a novel method for predicting service-level metrics. The method is service independent and does not requires server-side service-level instrumentation. The cost of service independence is a very large feature set, which requires a significantly larger number of samples for learning and significantly more computational resources than a small feature set would.

We evaluated two feature sets obtained from known interfaces to kernel data: the proc file system, from which $X_{proc}$ is obtained, and the SAR interface, from which $X_{sar}$ is gained. SAR itself is implemented using procfs. It contains no counters but aggregate statistics instead. We believe that this difference is the primary reason why the models computed with $X_{sar}$ are significantly more accurate than those computed with $X_{proc}$.

Our evaluation, which includes linear and nonlinear statistical learning methods applied to various traces, shows that accurate prediction of service-level metrics is possible in the following sense. For linear methods, the prediction error is below 20% (NMAE values, see Section VI); for random forest, which is a nonlinear ensemble method, the error reduces to below 15%. This reduction in error comes at the cost of computational overhead, which increases by a factor of 60 when changing from linear regression to random forest. Note that the error figures are obtained from executions with significant periods of high load (CPU utilisation $> 90\%$).

Our results further show that learning from statistics of one execution and predicting metrics for another execution is possible, although generally difficult. The difficulties stem from the fact that, under identical load conditions, there can be a large number of possible kernel states and thus a large number of feature vectors $X$ associated with these states. For example, the utilization of a CPU core that runs a specific video server can appear in one feature during a first execution and in a different feature during a second execution, due to operating system scheduling. The same applies, for instance, to the interrupt vector of a video server. Consequently, in order to learn the relationship between $X$ and $Y$ for all possible executions of the system, learning must consider all permutations of such features. This however is likely to result in a state space that is infeasible to sample due to its size. One approach to mitigate this issue is to aggregate (e.g., summing up) features that belong to the same group, e.g., the utilisation of various CPU cores or the rate of various interrupts. Domain knowledge can be applied to manually identify these groups, or an automatic procedure can be attempted.

Figures 4(c) and 4(f) suggest that the video frame rate and the audio buffer rate have bimodal distributions. We could therefore formulate the task of predicting these two metrics as a *classification problem* rather than as a regression problem (as done in this paper). Since our objective with this work has been to develop a generic method, rather than a method targeting one or more specific metrics, we decided to apply regression for predicting all metrics.

The models developed in this work are applicable beyond metrics prediction, for instance for model inference. Well-known methods are available for linear models and regression tree methods to identify the features in $X$ that significantly impact a specific metric $Y$ [8]. This can be particularly beneficial in our case, since our method relies on a very large feature set.

Recall that the methods used in this work do not exploit the fact that the samples $(X_t, Y_t)$, which are obtained from measurements, are in fact points in a time series, rather than drawn from a general joint distribution $(X, Y)$. Our evaluation shows that such a simplification still gives accurate predictions in many cases. It will be interesting to investigate, to which extent model accuracy can be improved when time dependencies are explicitly taken into account. One approach would be to expand the feature set $X$ with derivatives of its features. Another option is to include for a feature not only its current value, but an array of values from a given time window.

We have begun extending our testbed and our prediction method towards a more realistic cloud networking environment. Our plans include introducing a virtualisation layer between operating system and service, distributing the service components across several machines, and including statistics from network devices for metric prediction. Obviously, such extensions to the system make the learning problem more complex, and mastering this complexity is key to our research agenda.

## ACKNOWLEDGMENT

Marcello Lagana, KTH, contributed to the testbed setup and developed several tools for experimentation. This research has been supported by the Swedish Governmental Agency for Innovation Systems, VINNOVA, under grant 2013-03895.

## REFERENCES

[1] J. Bogojeska, D. Lanyi, I. Giurgiu, G. Stark, and D. Wiesmann, “Classifying server behavior and predicting impact of modernization actions,” in *Network and Service Management (CNSM), 2013 9th International Conference on*, Oct 2013, pp. 59–66.

[2] M. Mirza, J. Sommers, P. Barford, and X. Zhu, “A machine learning approach to tcp throughput prediction,” *Networking, IEEE/ACM Transactions on*, vol. 18, no. 4, pp. 1026–1039, Aug 2010.

[3] A. Andrzejak and L. Silva, “Using machine learning for non-intrusive modeling and prediction of software aging,” in *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*. IEEE, 2008, pp. 25–32.

[4] H. Hlavacs and T. Treutner, “Predicting web service levels during vm live migrations,” in *Systems and Virtualization Management (SVM), 2011 5th International DMTF Academic Alliance Workshop on*. IEEE, 2011, pp. 1–10.

[5] VLC. http://www.videolan.org/vlc.

[6] R. Yanggratoke, J. Ahmed, J. Ardelius, C. Flinta, A. Johnsson, D. Gillblad, and R. Stadler, “Linux kernel statistics from a video server and service metrics from a video client.” 2014, distributed by Machine learning data set repository [MLData.org]. http://mldata.org/repository/data/viewslug/realm-im2015-vod-traces.

[7] 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.

[8] T. H. Gareth James, Daniela Witten and R. Tibshirani, *An Introduction to Statistical Learning with Applications in R*. Springer, 2014.

[9] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[10] R. Tibshirani, “Regression shrinkage and selection via the lasso,” *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.

[11] L. Breiman, “Random forests,” *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, Oct. 2001. [Online]. Available: http://dx.doi.org/10.1023/A:1010933404324

[12] T. Bowden, B. Bauer, J. Nerin, and S. Feng, “The /proc filesystem,” https://www.kernel.org/doc/Documentation/filesystems/proc.txt.

[13] S. Godard, “SAR,” http://linux.die.net/man/1/sar.

[14] NTP. http://www.ntp.org/.

[15] I. Ari, B. Hong, E. Miller, S. Brandt, and D. D. E. 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*, Oct 2003, pp. 246–249.

[16] R Development Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2008, ISBN 3-900051-07-0. [Online]. Available: http://www.R-project.org

[17] R stats package. http://stat.ethz.ch/R-manual/R-devel/library/stats/html/lm.html.

[18] J. Friedman, T. Hastie, and R. Tibshirani, “Regularization paths for generalized linear models via coordinate descent,” *Journal of Statistical Software*, vol. 33, no. 1, pp. 1–22, 2010. [Online]. Available: http://www.jstatsoft.org/v33/i01/

[19] T. Therneau, B. Atkinson, and B. Ripley, “rpart,” http://cran.r-project.org/web/packages/rpart/rpart.pdf.

[20] A. Liaw and M. Wiener, “Classification and regression by randomforest,” *R News*, vol. 2, no. 3, pp. 18–22, 2002. [Online]. Available: http://CRAN.R-project.org/doc/Rnews/

[21] P. Bodık, R. Griffith, C. Sutton, A. Fox, M. Jordan, and D. Patterson, “Statistical machine learning makes automatic control practical for internet datacenters,” in *Proceedings of the 2009 conference on Hot topics in cloud computing*, 2009, pp. 12–12.

[22] A. Matsunaga and J. A. Fortes, “On the use of machine learning to predict the time and resources consumed by applications,” in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE Computer Society, 2010, pp. 495–504.

[23] S. Kundu, R. Rangaswami, A. Gulati, M. Zhao, and K. Dutta, “Modeling virtualized applications using machine learning techniques,” in *ACM SIGPLAN Notices*, vol. 47, no. 7. ACM, 2012, pp. 3–14.

[24] V. Menkovski, A. Oredope, A. Liotta, and A. C. Sánchez, “Predicting quality of experience in multimedia streaming,” in *Proceedings of the 7th International Conference on Advances in Mobile Computing and Multimedia*. ACM, 2009, pp. 52–59.

[25] V. Menkovski, G. Exarchakos, and A. Liotta, “Online qoe prediction,” in *Quality of Multimedia Experience (QoMEX), 2010 Second International Workshop on*. IEEE, 2010, pp. 118–123.

[26] H. H. Song, Z. Ge, A. Mahimkar, J. Wang, J. Yates, Y. Zhang, A. Basso, and M. Chen, “Q-score: Proactive service quality assessment in a large iptv system,” in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. ACM, 2011, pp. 195–208.

[27] A. Khan, L. Sun, and E. Ifeachor, “Learning models for video quality prediction over wireless local area network and universal mobile telecommunication system networks,” *Communications, IET*, vol. 4, no. 12, pp. 1389–1403, 2010.

[28] D. Hands, O. V. Barriac, and F. Telecom, “Standardization activities in the itu for a qoe assessment of iptv,” *IEEE Communications Magazine*, p. 79, 2008.

[29] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and H. Zhang, “Developing a predictive model of quality of experience for internet video,” in *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*. ACM, 2013, pp. 339–350.