Predicting service metrics for cluster-based services using real-time analytics

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

June 8, 2015

***Abstract*—Predicting the performance of cloud services is intrinsically hard. In this work, we pursue an approach based upon statistical learning, whereby the behaviour of a system is learned from observations. Specifically, our testbed implementation collects device statistics from a server cluster and uses a regression method that accurately predicts, in real-time, client-side service metrics for a video streaming service running on the cluster. The method is service-agnostic in the sense that it takes as input operating-systems statistics instead of service-level metrics. We show that feature set reduction significantly improves prediction accuracy in our case, while simultaneously reducing model computation time. We also discuss design and implementation of a real-time analytics engine, which processes streams of device statistics and service metrics from testbed sensors and produces model predictions through online learning.**

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

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 domain experts pre-select a small number (usually up to a dozen) of observation variables, also called features, for predicting a specific metric. Our approach is more general, since it considers a large amount of general statistics (in the thousands) and relies on feature selection through algorithmic reduction.

In this work, features are drawn from the kernel statistics of a server cluster that runs a video-on-demand service (VLC) [5]. We apply statistical learning methods on device statistics and service metrics in real-time, compute models that predict service metrics, and evaluate the model accuracies. (Prediction here relates to estimating metrics for current times based on current and past measurements.) The reported results are based upon extensive experimentation, where we run the video service under various load patterns on a laboratory testbed.

This is the second paper with results from our investigation into real-time prediction of service metrics. In [6], we studied the problem of predicting service metrics from device statistics for a video-on-demand service that runs on a single server. We experimented with various regression methods, performing batch learning on traces, and found that a random forest model allows us to predict various service metrics within an accuracy of 15% for a variety of load patterns. In this work, the system under investigation is more complex, since the video-on-demand service is provided by a server cluster with dedicated components, and the objective is to predict service metrics in real-time during the operation of the system.

We make two main contributions with this paper. First, we identify a learning method that accurately predicts service metrics for a cluster-based video service. The method is service-agnostic in the sense that it takes as input device statistics only, and no service-specific metrics. We also show that feature-set reduction significantly improves prediction accuracy in our case, while simultaneously reducing model computation time. We provide evaluation results in several steps towards real-time prediction: batch learning on traces, online learning on traces, and real-time learning on live statistics. Second, we design and implement a real-time analytics engine, which processes streams of device statistics and service metrics from testbed sensors and produces model predictions through online learning. This engine represents a key building block for an automated-service-assurance system and serves as a powerful tool for exploratory model evaluation and demonstration purposes.

The paper is organized as follows. Section II explains the problem setting. Section III describes 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 design and implementation of the real-time analytics engine. Section VI describes the evaluation of the model accuracies in several steps towards real-time prediction. Section VII discusses related work. Section VIII contains our conclusions and future work.

II. Problem Setting

Figure 1 shows the system under investigation. It consists of a cluster of servers that is connected to a client machine via a network. The client accesses a service S, which runs on the server cluster. In this work, we consider a video-on-demand (VoD) service. We are interested in the device statistics X of the cluster and how these statistics relate to service-level metrics Y on the client. In this setting, device statistics X_i of server $i = 1..n$ refer to metrics on the operating-system level. The device statistics X of the cluster is the concatenation of the server statistics, i.e., $X = [X_1, X_2, ..., X_n]$. The statistics Y on the client side refer to service-level metrics, for example, video frame rate and audio buffer rate.

The metrics X and Y evolve over time, influenced, e.g., by the load on the servers, operating system dynamics, etc. Assuming a global clock that can be read on both the client and the servers, we 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 the service-level metric Y_{t_k} at time t_k on the client, based on knowing the cluster metrics $X_{t_k} = [X_{1t_k}, ..., X_{nt_k}]$. Using the framework 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.

There are two ways to address this problem. First, we consider a set of samples $\{(X_{t_1}, Y_{t_1}), ..., (X_{t_m}, Y_{t_m})\}$. Assuming that each sample (X_{t_i}, Y_{t_i}) in the set is drawn uniformly at random from a joint distribution (X, Y), we apply concepts and methods from statistical learning to find M. The solution M is evaluated using the validation set approach [7]. Note that in this case M does not change over time.

The second way to address this problem is to consider a time series of samples $\{(X_1, Y_1), (X_2, Y_2), ...\}$. We apply online methods from statistical learning, which process this series sequentially and produce a sequence of models $M_1, M_2, ...$ The solution $M_t, t = 1, 2, ...$ is evaluated using the interleaved test-then-train approach [8]. Note that in this case M changes over time.

The first approach, often called *batch learning*, is the standard approach in statistical learning, where the goal is to explain an existing dataset. The second approach, called *online learning*, is commonly followed, when studying a system that evolves over time. The second approach allows to handle *concept drift*, which requires dynamic adaptation of the model for accurate predictions [9].

In this work, we use both batch learning and online learning methods. We apply batch learning on traces from our testbed to obtain a baseline for model accuracy. We apply online learning as a basis for implementing real-time prediction, since samples become available in a sequential fashion as time evolves.

We do not take into account network statistics and client device statistics in this work, in order to simplify the problem. (Note that previous research has shown the feasibility of using network measurements to predict client-side metrics [10].) In practical terms, this means that in our experiments the network and client machine are lightly loaded. Statistics from network devices and the client machine can be included in our problem setting by extending the X space with the feature sets of those devices. This is part of our future work.

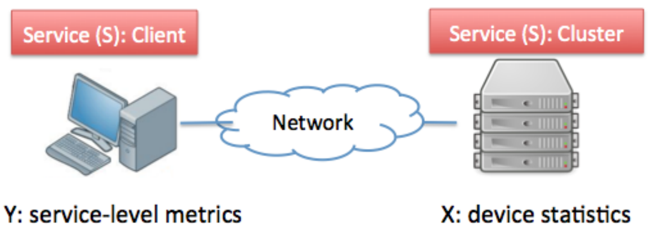

Fig. 1. System under investigation.

III. Statistical Learning Methods Used in This Work

We outline several regression methods that are used in this work. First, we apply 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) is minimised [7]. In our case, the solution is computed using stochastic gradient descent (SGD) [11].

Second, we use the Lasso regression method, which is a variant of linear regression that mitigates overfitting in case of a high-dimensional feature space. Lasso regression solves the linear regression problem with the additional constraint $\sum_j |\beta_j| \leq \lambda$, where λ is a regularization parameter. The solution to the Lasso regression problem tends to yield a model with smaller coefficients β_j (which can even be zero) than those of a linear regression model would [12].

In addition to linear methods, we apply tree-based methods in this work: regression tree, random forest, and fast incremental-model tree. The regression tree method computes region boundaries with the 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 [7]. (The method has a computational complexity of $O(N^2p.)$)

Random forest is an ensemble method. Each estimated value of Y is an average of predictions from several regression trees [13]. Each of these trees is constructed using a different training set, and each construction step uses a randomised reduced feature set [7].

Fast incremental-model tree with concept drift detection (FIMT-DD) is an online regression method [14]. It starts with a leaf and incrementally builds a tree, while reading a stream of samples.

We apply feature selection to reduce the dimensionality of the feature space. We specifically use a method called *forward-stepwise-selection*. Starting from an empty feature set, the method incrementally grows the feature set by including, in each iteration, a new feature that minimises the prediction error [7].

To mitigate concept drift in real-time learning, we use a proactive approach, whereby the model is retrained periodically [9]. An alternative approach would be a reactive one, whereby the model is only retrained when concept drift is detected.

IV. Device statistics and service-level metrics

A. Device statistics: the feature space X

We obtain device statistics X_i from the kernel of the Linux operating system that runs on the server i. 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* [15].

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.

Our earlier study in [6] shows that learning from a preprocessed feature set (X_{sar}) yields better prediction accuracies than from a feature set obtained directly from procfs. Thus, in this work we use the feature set X_{sar} for each server machine. The feature set X_{sar} is constructed using System Activity Report (SAR), a popular open source Linux tool [16]. 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 per server machine.

Later in this work, we use X_i to refer to X_{sar} collected from a server machine i. To represent the device statistics for the cluster of server machines, we concatenate X_i collected from each server to be $(X_1, ..., X_n)$, whereby n is the number of server machines in the cluster. Hence, our feature set for the cluster predictions has about $n * 840$ features. We refer to this feature set as the *full feature set*.

We also apply the feature selection method described in Section III. Then, we end up with a reduced feature set about 10 features. We refer to this feature set as the *minimal feature set*.

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. The service-level metrics we are considering are measured on the client device. 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) Network read operation rate (operations/sec): the number of socket read operations issued by VLC 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 prototype

In this section, we describe the hardware and software setup for video-on-demand service and data collection infrastructure, we describe how we perform the experiments, how we generate load, and how we obtain traces for model computation and evaluation. In addition, we discuss design and implementation of the real-time analytics engine.

A. The testbed

We run the experiments on our testbed at KTH, which includes a rack of nine high-performance servers interconnected by Gigabit Ethernet. The servers are Dell PowerEdge R715 2U machines, 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 parts, namely, a server cluster that provides the video-on-demand service over HTTP, a client machine that runs video-on-demand sessions, 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 [17]. Figure 2 shows the components that execute on these machines and the flows of media streams during experiments.

The server cluster consists of six machines: one load-balancer machine, three web server and transcoding machines, and two networked-storage machines. Each machine in the server cluster runs a sensor that periodically reads out the vector X in form of X_{sar} (SAR version 10.0.3), as described in Section IV. At the start of every second, the sensor reads X and saves it on the local X-trace file, together with a timestamp. The load balancer machine runs HAProxy version 1.4.18 [18]. Each web server and transcoding machine runs Apache version 2.2.22 [19] and ffmpeg version 0.8.16 [20]. Each networked-storage machine runs GlusterFS version 3.5.2 [21]. The networked-storage machines are populated with the ten most viewed YouTube videos in 2013.

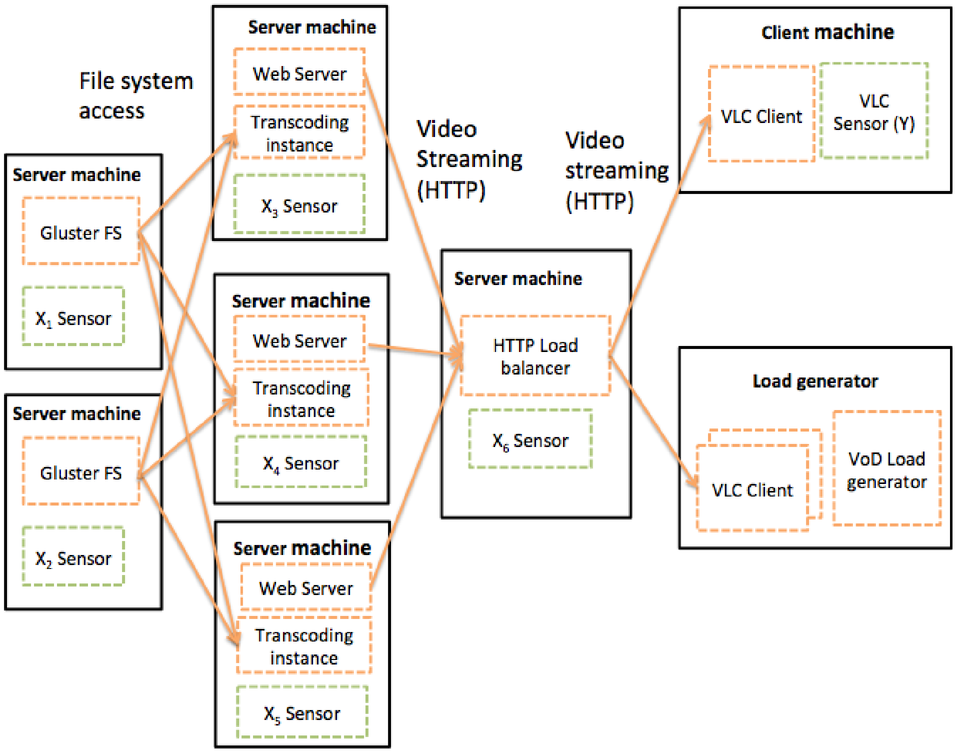

Fig. 2. Testbed configuration for creating X-Y traces and for real-time prediction of service metrics. A solid rectangle represents a physical machine.

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 about 14 hours. At the beginning of a run, the VLC client sends a *VoD session request* for playing a specific video to the load balancer. After the video has played, the VLC client sends a new request for the same video to the load balancer. Also, at the beginning of the run, the load generator starts sending VoD session requests to the load balancer, according to the selected load pattern. Further, the sensors on the server and client machines are started.

Receiving a VoD session request from a client, the load balancer forwards the request to the backend web server that has the least number of pending connections with the load balancer. To respond to a request forwarded from the load balancer, the web server spawns a transcoding instance for a selected video, whereby the raw video content is retrieved over the network from one of the networked storage machines.

B. Generating load 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. When a client is created, it sends a request for a random video to the load balancer. The server cluster then starts streaming the video content to the client. Whenever the video ends, the client keep issuing a new request, until the client is terminated by the load generator. We use the following load patterns in the reported experiments.

1) Periodic-load pattern: the load generator starts clients following a Poisson process with an arrival rate that starts at 70 clients/minute and changes according to a sinusoid function with a period of 60 minutes and an amplitude of 50 clients/minute. The load generator terminates a client after an average holding time of one minute, which is exponentially distributed.

2) Flashcrowd-load pattern: the load generator starts and terminates clients according to the flash-crowd model described in [22]. The creation of clients follows a Poisson process with an arrival rate that starts at ten clients/minute and peaks at flash events, which are randomly created at a rate of ten events/hour. At each flash event, the arrival rate increases within a minute to 120 clients/minute. The rate stays at this level for one minute, and then decreases to ten clients/minute within four minutes. The load generator terminates a client after an average holding time of one minute, which is exponentially distributed.

C. Prototype of the real-time analytics engine

We designed and implemented a real-time analytics engine that runs on a separate machine on our testbed. It receives streams of device statistics and service metrics from the sensors and produces model predictions through online learning. In this work, we use this engine for predicting service metrics in real-time and evaluating the prediction accuracies. We have also built an application that visualises the outputs of this analytics engine.

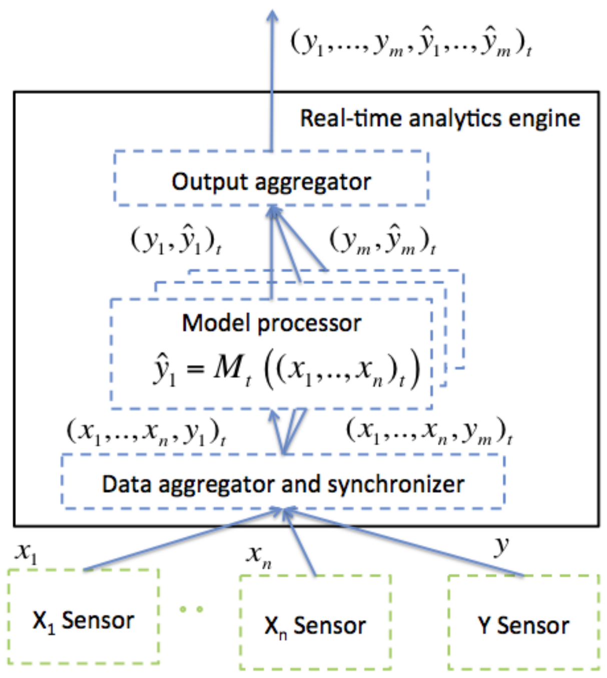

Fig. 3. Design of the real-time analytics engine for predicting service metrics. The analytics engine executes on a separate machine not shown in Figure 2.

Figure 3 shows the software design of the analytics engine and the data flow through the engine's pipeline architecture. It includes three main components: (1) a data aggregator and synchroniser, (2) model processors, and (3) an output aggregator.

The data aggregator and synchroniser component collects the data items $x_i, i = 1..n$ and $y_j, j = 1..m$ from the sensors, aggregates and time-synchronises them, and produces a stream of samples $(x_1, x_2, ..., x_n, y_j)_t$ that is fed to the model processors.

For each service metric y_j there is one model processor. The processor implements the learning method. It reads a stream of samples, computes the learning models M_t and the model prediction $\hat{y}_j$ at time t, and streams the prediction to the output aggregator.

The output aggregator time-synchronises the predictions and sends the aggregate $(\hat{y}_1, ..., \hat{y}_m)_t$ to the management application.

All components of the real-time analytics engine are written in R (version 3.1.2) [23]. The communication between components is realised through TCP network sockets, using the connection package [24]. The communication between sensors and the data aggregator and synchroniser is realised using Netcat [25] version 1.89. All communication channels are setup at initialisation times and kept alive during operation. The output aggregator, the data aggregator and synchroniser, and each model processor run in a separate Linux process.

Figure 4 shows a screenshot of an application we built that visualises the output of the analytics engine. The screen shows time series of the predicted service metrics and the measured

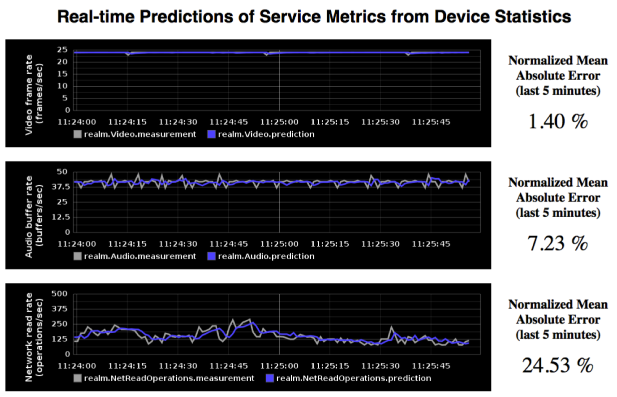
Real-time Predictions of Service Metrics from Device Statistics
Video frame rate (frames/sec)
realm.Video.measurement
realm.Video.prediction
Normalized Mean Absolute Error (last 5 minutes)
1.40 %
Audio buffer rate (buffers/sec)
realm.Audio.measurement
realm.Audio.prediction
Normalized Mean Absolute Error (last 5 minutes)
7.23 %
Network read rate (operations/sec)
realm.NetReadOperations.measurement
realm.NetReadOperations.prediction
Normalized Mean Absolute Error (last 5 minutes)
24.53 %
11:24:00
11:24:15
11:24:30
11:24:45
11:25:00
11:25:15
11:25:30
11:25:45

Fig. 4. Visualising the output of the real-time analytics engine.

service metrics. The application was built using Graphite [26] and JQuery [27]. We currently use this application for exploratory model evaluation and for demonstration purposes, as we did at IM2015 [28].

VI. Model computation and evaluation

Our goal is to provide accurate predictions of service metrics in real-time. Towards that goal, we perform model computation and evaluation in several steps, increasing the required effort and the obtained realism in each step. We start with batch learning using traces from the testbed to obtain a baseline for the accuracies of different learning models. As a second step, we use online learning on the same traces, since online learning, while more complex than batch learning, is required for real-time adaptation of prediction models. Finally, we perform model prediction and evaluation in real-time on the testbed. In this case, samples are collected from live statistics instead of from traces, and the learning models are computed by our testbed prototype, which includes the real-time analytics engine described in Section V-C.

We expect higher prediction accuracy with batch learning than with online learning. This is because batch learning methods perform model computation and evaluation using all samples from an experiment, while online learning methods perform model computation and evaluation on subsets of all samples. (We assume here that the effects of concept drift is limited during experimental runs.) Second, we expect that online learning on a trace can be achieved with similar accuracy than real-time learning on the testbed under the same load pattern.

In order to obtain baseline predictions, we collect traces that were obtained under different load patterns from our testbed. We apply a range of statistical learning methods on these traces, compute models for predicting service metrics, and evaluate these models against ground-truth data from the traces.

A. *Platform and packages for learning on traces*

We compute and evaluate models on a server with specification given in Section V-A. We run R version 3.1.2 [23], specifically the following packages: glmnet version 1.9-8 [29] for Lasso regression using coordinate descent (with regularlization parameter 0.02), rpart version 4.1-8 [30] for regression tree, RMOA [31] for Hoeffding-based regression tree with concept drift detection (FIMT-DD), and randomForest version 4.6-7 [32] for random forest (with 120 trees). We run Matlab version R2014b for least-square linear regression, using stochastic gradient ascent with learning rate of 0.01 for video frame rate and audio buffer rate and 0.001 for network read rate. The model computation parameters have been determined through experimentation.

B. *Batch learning on traces*

Using traces collected from the testbed, we compute models for a variety of regression methods, including Lasso regression, regression tree, and random forest. (For an overview of these methods, see Section III.) We apply the *validation set* approach for model evaluation, which is generally used when large datasets are available. According to this approach, 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 them using the test set [7]. Following standard practise, the training set contains 70% of the samples, and the test set 30%.

We compute two metrics to evaluate the learning models. The first metric 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 metric y_i, and $\bar{y}$ is the average of the samples y_i of the test set of size m. The second metric is the *training time*, which measures the times it takes to train a model on the training set. We report the average time of three model computations.

Table I shows the evaluation results for different load patterns, feature sets, regression methods, and service metrics. Based on these results, we make the following observations. First, in terms of prediction accuracies, the *random forest method performs best* across all traces, feature sets, and service metrics. For example, the NMAE value of the audio buffer rate is 21 for the combination of the flashcrowd-load trace, the full feature set, and the random forest method. When Lasso regression or regression tree are used instead, the NMAE values of the audio buffer rate increase to 42 or 31, respectively.

Second, Lasso regression allows for the fastest model computation. It takes below 22 seconds for each service metric, trace, and feature set, while random forest takes a thousand times longer on average. In this case, fast computation comes at the price of high prediction errors for audio buffer rate and network read rate.

Third, and most importantly, random forest on the minimal feature set significantly reduces training time and improves prediction accuracy than random forest on the full feature set. (This is expected, since the training time increases linearly with the number of features, and improving prediction accuracy has been the objective of feature selection.) This is true for all tested service metrics and traces. For example, considering the video frame rate of the periodic trace, the NMAE value is 6 and the training time is less than 10 minutes for the minimal feature set, while the NMAE value is 12 and the training time is more than 10 hours in case of the full feature set.

We conclude from the batch learning experiments that random forest on the minimal feature set is a good candidate method for real-time prediction.

C. *Online learning on traces*

Using testbed traces, we compute models for several online regression methods, including linear regression using stochastic gradient descent, Hoeffding-based regression tree with concept drift detection, and random forest on a sliding-window of samples. (For an overview of these methods, see Section III.) We evaluate a model as follows. During a warm up phase (the first 20K samples), we compute an initial model. After that, during the evaluation phase (the subsequent 30K samples), we follow the interleaved-test-then-train approach, whereby we repeatedly read in a new sample, test the model on the sample, and then adapt the model [33]. For model evaluation, we use $NMAE$ (defined above) computed from all samples used in the evaluation phase.

Table II presents the evaluation results using the minimal feature set for different load patterns, online regression methods, and service metrics. It allows the following observations.

First, we observe that the prediction error differs for each service metric. For instance, video frame rate has consistently the lowest error. Interestingly, the service metric with the lowest error has also the lowest coefficient of variation, and the one with the highest error shows the highest coefficient of variation.

Second, random forest on a sliding-window of samples achieves the best prediction accuracy across traces and service metrics.

Third, while it is difficult to directly compare the evaluation results from batch learning methods with those of online learning methods (since the algorithms are different), Table I and Table II confirm our intuition that batch learning generally achieves higher prediction accuracy than online learning. For example, the minimum prediction error of audio buffer rate for the online method on flashcrowd trace is 19% (Table II), while the minimum error for the batch learning method is 15% (Table I).

These experiments further confirm that random forest with the minimal feature set is a promising candidate method for real-time prediction.

D. *Real-time learning*

Collecting live statistics from the testbed, our prototype computes learning models for the random forest method on a sliding window of samples using the minimal feature set. The service predictions from the prototype are evaluated in the same ways as described above for the case of online learning. The prototype computes an initial model during the warm up phase and computes a series of models during the evaluation phase. Specifically, the random forest model uses the window size of 600 samples, and the model is retrained for every new sample.

Table III presents the evaluation results. Comparing these results with Table II, our key observation is that there is no significant difference in prediction errors for the tested

Trace	Feature set	Regression method	Video frame rate		Audio buffer rate		Network read rate	
			$NMAE$ (%)	Training (secs)	$NMAE$ (%)	Training (secs)	$NMAE$ (%)	Training (secs)
Periodic-load	Full	Lasso regression	22	9	53	9	51	11
		Regression tree	17	871	42	875	50	788
		Random forest	12	5.97E4	32	7.83E4	33	9.18E4
	Minimal	Lasso regression	22	0.05	53	0.05	52	0.05
		Regression tree	22	1.7	53	0.28	51	1.6
		Random forest	**6**	862	**22**	1.6E3	**24**	1.56E3
Flashcrowd-load	Full	Lasso regression	18	9	42	11	50	11
		Regression tree	13	664	31	704	48	840
		Random forest	8	5.56E4	21	7.98E4	33	9.4E4
	Minimal	Lasso regression	18	0.05	42	0.05	50	0.05
		Regression tree	19	0.31	42	0.37	50	1.9
		Random forest	**4**	778	**15**	1.75E3	**22**	1.95E3

TABLE I. OFFLINE-MODEL ACCURACIES AND TRAINING TIMES FOR DIFFERENT SERVICE METRICS, FOR THE FULL AND THE MINIMAL FEATURE SET, DIFFERENT BATCH REGRESSION METHODS, AND DIFFERENT TRACES.

Trace	Regression method	$NMAE(\%)$		
		Video frame rate	Audio buffer rate	Network read rate
Periodic-load	Linear regression using SGD	18	37	51
	Hoeffding-based regression tree (FIMT-DD)	21	51	49
	Sliding-window random forest	**6**	**25**	**26**
Flashcrowd-load	Linear regression using SGD	13	32	49
	Hoeffding-based regression tree (FIMT-DD)	18	42	49
	Sliding-window random forest	**5**	**19**	**25**

TABLE II. ONLINE-MODEL ACCURACIES FOR DIFFERENT SERVICE METRICS, FOR THE MINIMAL FEATURE SET, FOR DIFFERENT ONLINE REGRESSION METHODS, AND DIFFERENT TRACES.

Real-time load pattern	NMAE(%)		
	Video frame rate	Audio buffer rate	Network read rate
Periodic-load pattern	3.6	14	28.5
Flashcrowd-load pattern	5.6	11	28

TABLE III. REAL-TIME-MODEL ACCURACIES FOR DIFFERENT SERVICE METRICS, FOR THE MINIMAL FEATURE SET, FOR THE ONLINE RANDOM-FOREST METHOD AND DIFFERENT LOAD PATTERNS.

random forest method. For instance, the NMAE values of video frame rate for both online and real-time learning are below 6%, the values of audio buffer rate are below 20%. This result is not obvious, since (1) the online learning experiments use different implementation for data collection, as well as, model computation and evaluation, than the real-time learning experiment, and (2) the load patterns are probabilistic and produce different executions.

Our main conclusion from all the experiments is that we were able to identify a method and build the system that accurately predicts service metrics (NMAE values below 14% for video frame rate and audio buffer rate, and NMAE values below 28.5% for network read rate) in real-time for a cluster-based service.

VII. Related work

This work relates to recent research on predicting service metrics and real-time prediction in networking and cloud environments, using analytics methods.

While our approach to prediction is service independent, many other works propose methods that are targeted towards a specific service and metric. Important examples of such research in the context of cloud and statistical learning are [4], [34]–[36]. In contrast to our method, these works consider a small set (less than 10) of selected features to learn from, which is fixed at the design stage of the method. In our case, the set of features for model computation is automatically derived from a very large, general feature set (see Section IV-A).

The authors in [37] predict quality-of-service metrics for IPTV using decision trees. The features are selected by a domain expert. In [10], 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.

Other works like [38]–[41] use statistical learning models to estimate quality-of-experience metrics of a multimedia service. A review of such metrics can be found in [42].

Recent work by [43] proposes a system for service-level prediction using statistical models. The authors use Bayes classifiers with a set of statistical metrics as features to determine the probability of SLA violations. They solve a classification problem, while our work is based on methods for regression.

The authors in [44] describes a method that dynamically allocates run-time resources for MapReduce tasks under unbalanced data distribution. In particular, the work applies linear regression to predict partition sizes for reduce tasks and use the predictions to guide run-time resource allocation.

VIII. Conclusion

We have shown the feasibility of predicting service metrics for cluster-based video streaming service in real-time. Through testbed evaluation and measurements, we found that a random forest model gives a prediction accuracy of 14% or better for video frame rate and audio buffer rate, and an accuracy of 28.5% or better for network read rate. We found that feature-set reduction not only reduces model computation time, with is critical for real-time prediction, but even improves prediction accuracy (which we think is likely due to reduction of overfitting). Our investigation progresses in a methodical fashion: from batch learning on traces, to online learning on traces, and finally to real-time learning on live statistics. Each step serves as a baseline for the next, more complex step.

We have designed and implemented a real-time analytics engine that performs model learning and metrics prediction in a centralised framework. This engine will serve as a building box for analytics-based management functions, including a service assurance system and anomaly detection system. At the same time, we currently use it for real-time experimentation and demonstration purposes. We can observe, in real-time, the effects of system perturbation on service quality. For instance, we can study the effects of load spikes, background executions of maintenance jobs, changes of system configuration, or different types of system failures.

Our plans include the following research directions. First, we plan to extend our prediction method towards clusters that run multiple services and towards virtualised environments. Second, we plan to extend the results from this paper towards end-to-end predictions over network infrastructure. Third, we plan to make the prediction method scale, by distributing the analytics engine and performing model computation and metrics prediction inside network nodes. Finally, we plan to engineer novel management applications that are directly driven by the analytics engine.

Acknowledgment

Zuoying Jiang, KTH, contributed to the development of the real-time analytics engine. Ruairi de Frein, KTH, participated in the discussion of the learning methods used in this work. 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, “Predicting real-time service-level metrics from device statistics,” in *Integrated Network Management (IM 2015), 2015 IFIP/IEEE International Symposium on*, April 2015.

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

[8] J. Gama, R. Sebastião, and P. P. Rodrigues, “Issues in evaluation of stream learning algorithms,” in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009, pp. 329–338.

[9] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, “A survey on concept drift adaptation,” *ACM Computing Surveys (CSUR)*, vol. 46, no. 4, p. 44, 2014.

[10] 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.

[11] L. Bottou, “Large-scale machine learning with stochastic gradient descent,” in *Proceedings of COMPSTAT'2010*, Y. Lechevallier and G. Saporta, Eds. Physica-Verlag HD, 2010, pp. 177–186. [Online]. Available: http://dx.doi.org/10.1007/978-3-7908-2604-3_16

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

[13] 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

[14] E. Ikonomovska, J. a. Gama, and S. Džeroski, “Learning model trees from evolving data streams,” *Data Min. Knowl. Discov.*, vol. 23, no. 1, pp. 128–168, Jul. 2011. [Online]. Available: http://dx.doi.org/10.1007/s10618-010-0201-y

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

[16] S. Godard. SAR. http://linux.die.net/man/1/sar.

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

[18] HAProxy. http://www.haproxy.org/.

[19] Apache http server. http://httpd.apache.org/.

[20] Ffmpeg. https://www.ffmpeg.org/.

[21] Gluster FS. http://www.gluster.org/.

[22] 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.

[23] [Online]. Available: http://www.r-project.org/

[24] “R functions to manipulate connections,” https://stat.ethz.ch/R-manual/R-devel/library/base/html/connections.html.

[25] “Netcat: the swiss army knife of networking,” http://nc110.sourceforge.net/.

[26] “Graphite - scalable realtime graphing,” https://graphite.readthedocs.org/en/latest/.

[27] jQuery.com. jquery: write less, do more. https://jquery.com/.

[28] R. Yanggratoke, J. Ahmed, J. Ardelius, C. Flinta, A. Johnsson, and R. Stadler, "A platform for predicting real-time service-level metrics from device statistics," in *Integrated Network Management (IM 2015), 2015 IFIP/IEEE International Symposium on*, May 2015, demonstration session.

[29] 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/

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

[31] J. Wijffels, "RMOA: Connect r with moa to perform streaming classifications," 2014, r package version 1.0. [Online]. Available: https://github.com/jwijffels/RMOA

[32] 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/

[33] J. Gama, R. Sebastio, and P. Rodrigues, "On evaluating stream learning algorithms," *Machine Learning*, vol. 90, no. 3, pp. 317–346, 2013. [Online]. Available: http://dx.doi.org/10.1007/s10994-012-5320-9

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] P. Leitner, J. Ferner, W. Hummer, and S. Dustdar, "Data-driven and automated prediction of service level agreement violations in service compositions," *Distributed and Parallel Databases*, vol. 31, no. 3, pp. 447–470, 2013.

[44] Z. Liu, Q. Zhang, M. F. Zhani, R. Boutaba, Y. Liu, and Z. Gong, "Dreams: Dynamic resource allocation for mapreduce with data skew," in *Integrated Network Management (IM 2015), 2015 IFIP/IEEE International Symposium on*, May 2015 2015.