Received August 8, 2019, accepted August 25, 2019, date of publication September 6, 2019, date of current version September 20, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2939805

Prospective: A Data-Driven Technique to Predict Web Service Response Time Percentiles

YASAMAN AMANNEJAD [1], DIWAKAR KRISHNAMURTHY [2], AND BEHROUZ FAR [2]
[1]Mathematics and Computing Department, Mount Royal University, Calgary, AB T3E 6K6, Canada
[2]Department of Electrical and Computer Engineering, University of Calgary, AB T2N 1N4, Canada

Corresponding author: Yasaman Amannejad (yamannejad@mtroyal.ca)

ABSTRACT Delivering fast response times for user transactions is a critical requirement for Web services. Often, a Web service has Service Level Agreements (SLA) with its users that quantify how quickly the service has to respond to a user transaction. Typically, SLAs stipulate requirements for Web service response time percentiles, e.g., a specified target for the 95*th* percentile of response time. Violating SLAs can have adverse consequences for a Web service operator. Consequently, operators require systematic techniques to predict Web service response time percentiles. Existing prediction techniques are very time consuming since they often involve manual construction of queuing or machine learning models. To address this problem, we propose Prospective, a data-driven approach for predicting Web service response time percentiles. Given a specification for workload expected at the Web service over a planning horizon, Prospective uses historical data to offer predictions for response time percentiles of interest. At the core of Prospective is a lightweight simulator that uses collaborative filtering to estimate response time behaviour of the service based on behaviour observed historically. Results show that Prospective significantly outperforms other baseline techniques for a wide variety of workloads. In particular, the technique provides accurate estimates even for workload scenarios not directly observed in the historical data. We also show that Prospective can provide a Web service operator with accurate estimates of the types and numbers of Web service instances needed to avoid SLA violations.

INDEX TERMS Performance engineering, prediction, response time percentiles, system sizing.

I. INTRODUCTION

Web services need to respond quickly to transactions issued by their users. Long response times can frustrate a user and can cause them to discontinue using the service. Often, Web service operators have Service Level Agreements (SLA) with end users that among other things specify requirements, e.g., acceptable thresholds, for service response times. Typically, an SLA will stipulate targets for service response time percentiles. Since the objective is to avoid long response times for users, percentiles that capture the tail of the response time distribution, e.g., the 95*th* percentile, are often used while defining SLAs.

Operators need systematic performance engineering techniques to ensure that their service will meet response time percentile requirements when deployed in production mode. Specifically, given a planning horizon, i.e., a future time period, and an estimate of user workload over this planning horizon, an operator needs predictions of the various response time percentiles of interest. Such predictions can help the operator take remedial actions if percentile thresholds are likely to be exceeded. An example of a remedial action could be scaling out the resources and using more server instances to host the service. Operators need a performance engineering approach to help them select the right types and numbers of server instances to avoid exceeding response time percentile thresholds.

Performance engineering exercises for Web services typically combine load testing and performance modeling [1], [2]. Since load tests are very time consuming to setup and conduct, typically only a very small number of test workloads and system configuration settings are considered. Consequently, a queuing model or a machine learning model, e.g., regression model, may be used to predict the response time behaviour of the service under workloads and system settings that were not explored during the testing phase. Measurements from the load tests are used to parameterize as well as validate the predictive model.

The associate editor coordinating the review of this manuscript and approving it for publication was Fabrizio Messina.

While modeling techniques can be effective, operators can face many challenges in leveraging them. Specifically, queuing analysis techniques can be time consuming since they require an expert to author, parameterize, and validate a model of the Web service. Similarly, applying machine learning models requires an expert to identify the right type of learning technique to employ. Furthermore, expertise is needed to map workload and system characteristics to numerical independent variables that are good at explaining the variation in the dependent variable, i.e., a response time percentile of interest. This process is called featurization. Additionally, some Machine learning and statistical techniques, e.g., least squares regression [3] and quantile regression [4], require the specification of a suitable performance function, e.g., linear or polynomial, that defines how the identified independent and dependent variables are related. Such tasks typically require an expert and can be time consuming.

In this paper, we explore a data-driven and automated approach called Prospective to address these limitations. Prospective does not require an operator to author a queuing model. Furthermore, neither does it require the operator to explicitly define a featurization nor does it require the specification of a performance function. Prospective takes as input historical performance data collected either from load testing campaigns or from a live system deployment and a specification for the workload expected at the Web service over a planning horizon. The specification describes the arrival pattern of transactions, i.e., the time series of arrivals, and the transaction mix, i.e., the probabilities of observing each transaction type in the workload. Each transaction type refers to a specific function invoked by the end user, e.g. browse and buy in an e-commerce system. Given such a specification, Prospective uses the historical data to offer predictions for response time percentiles of interest for the specified workload. The key challenge that Prospective needs to address is that the specification of the workload for which a prediction is desired could be quite different from those of the workloads used to generate the historical data.

Prospective employs a novel approach to address this challenge. The central idea behind Prospective is the combined use of load and transaction type for predicting a transaction's response time. We define the load at any given time instant as a vector that records the numbers of concurrent transactions of each type at a service at that time instant. Prospective consists of a lightweight simulator that interacts with a collaborative filtering (CF) [5] based response time prediction module to estimate the service's response time behaviour. The prediction process begins by first generating a synthetic trace of transactions that conforms to the input workload specification. Prospective's simulator traverses this trace to produce initial estimates of the loads experienced by these transactions. Given the initial load estimated for a synthetic transaction and that transaction's type, response time prediction is carried out by analyzing the historical data to obtain past response times detected for the desired transaction type and load combination. For scenarios where the historical data has not encountered this combination, the CF method infers the response time based on response times recorded in the repository for similar combinations. The simulator iteratively refines the load and response time estimates for the synthetic transactions and then finally calculates the desired percentiles.

Results from realistic Web service testbed based on the RUBiS [6] benchmark application show that Prospective achieves high accuracy for a wide variety of workloads. In all of the experimented scenarios, we show that Prospective can offer 95^{th} and 99^{th} percentile predictions with errors less than 14.1%. Furthermore, Prospective outperforms baseline techniques. In particular, it is 16 times more accurate than a similar technique from literature that is agnostic to transaction types. Prospective is also significantly more accurate than existing approaches in literature that are based on building quantile regression models [7], [8]. Finally, we show how Prospective can be used to simulate the impact of horizontal scaling. Specifically, results show that the technique can simulate common load balancing policies used in practice and provide an operator with accurate estimates of the types and numbers of server instances needed to avoid response time percentile violations.

This paper represents a significant extension of the preliminary work described in our previous paper [9]. The extensions are along the following directions:

1) We have further improved the accuracy of Prospective by incorporating two new features. We now explore an additional step where transaction types having similar mean response times are automatically clustered into a small number of transaction groups. We also study a refinement to the earlier approach we used to automatically select Prospective's simulation parameters. On an average, as discussed in Sec. V-C, the two new features combined improve the mean prediction error for the 95^{th} and 99^{th} percentiles from 11.0% and 14.2% to 6.1% and 7.0%, respectively.
2) We have expanded the evaluation by experimentally comparing Prospective with two state-of-the-art quantile regression techniques [7], [8].
3) We present new experiments to characterize the sensitivity of Prospective to the historical data. The results are discussed in Sec. V-E.
4) To support system sizing questions, we have added a new module to Prospective that can simulate common load balancing policies used in practice. We present new experiments in Sec. VI to show how this feature can be used to simulate horizontal scaling scenarios and answer system sizing questions.
5) The related work section is significantly expanded.

The remainder of this paper is organized as follows. Our methodology is discussed in Sec. II. Sec. IV describes the experiments used to validate Prospective. Sec. V provides results that characterize Prospective's prediction accuracy. Sec. VI show how Prospective can be used to support system sizing exercises. Sec. VII discusses related work. Finally, Sec. VIII provides conclusions and future work.

II. PROSPECTIVE

We propose *"Prospective"*, a system for **P**redicting **R**esp**O**n**S**e Time **PE**rcentiles using **C**ollabora**TIVE** Filtering. The high level architecture of Prospective is described in Sec.II-A. Sections II-B to II-D discuss in detail the implementation of Prospective.

A. OVERVIEW

Fig. 1 shows the architecture of Prospective. The target application is a Web service with n types of transactions with each transaction type assigned a unique id in the range 1 to n. Similar to other performance prediction technique, Prospective requires that the Web service operator specifies as input a workload specification $\mathbf{W}$ over a planning horizon. $\mathbf{W}$ represents the workload for which a prediction is desired. It consists of the tuple of transaction mix $\mathbf{M}$, transaction arrival model $\mathbf{A}$, and the total number of transactions N. $\mathbf{M}$ defines the probabilities of observing each transaction type in the workload. $\mathbf{A}$ provides a statistical model that governs the time instants at which transactions in the workload arrive at the Web service, i.e., the arrival pattern. $\mathbf{W}$ can be obtained by exploiting workload prediction techniques [10], [11] or using expert knowledge. It can also be obtained by perturbing the current workload of the service for the purpose of a sensitivity analysis. As shown in Fig. 1, a workload generator is used to transform $\mathbf{W}$ to a trace $\mathbf{T}$ of N workload records. Each record pertains to a transaction and contains an id indicating the transaction's type and also the time at which the transaction arrives at the Web service.

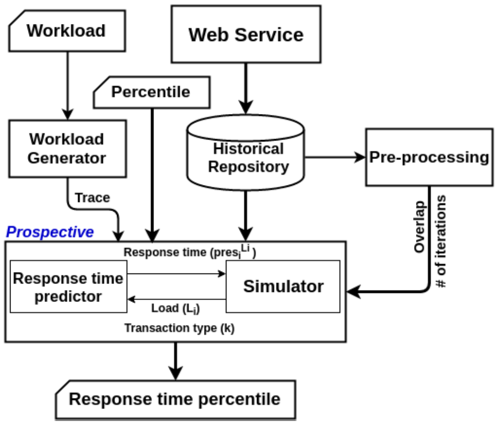


FIGURE 1. High level architecture of prospective.

From Fig. 1, Prospective predicts the response time percentile for an operator-defined percentile value P. Prospective also takes as input a historical data repository $\mathbf{D}$. Each record in the repository pertains to a transaction i. It includes the transaction's type k, load $\mathbf{L_i}$, and its load-dependant response time $res_i^{\mathbf{L_i}}$. $\mathbf{L_i}$ is an n-dimensional vector where the j^{th} element in $\mathbf{L_i}$, $L_i[j]$, shows the number of transactions of type j executing concurrently on the service during the execution of transaction i. The response time of the transaction is indicated by $res_i^{\mathbf{L_i}}$. $\mathbf{D}$ can be constructed by processing trace data $\mathbf{T_{rep}}$ from a load testing campaign or the actual Web service deployment. Prospective only requires that each line of $\mathbf{T_{rep}}$ contain the arrival instant of a transaction, its transaction type, and its response time.

As shown in Fig. 1, the final inputs to Prospective are the overlap parameter O and the number of iterations parameter I. O defines the minimum overlap between the execution of two transactions for them to be considered as concurrent transactions during load calculations. I is the number of times Prospective attempts to refine the response time estimates for a transaction. O and I together provide a mechanism to simultaneously improve prediction accuracy and limit simulation time. The parameters are automatically calculated in a pre-processing phase described in Sec. II-C.

Prospective uses a lightweight simulator in combination with a CF based response time prediction module to estimate response times of transactions in the input trace $\mathbf{T}$. These response times are then used to calculate the P^{th} percentile of response times. Multiple independent simulation runs can be carried out to calculate a confidence interval (CI) for the mean. The Web service operator can use the predictions to take system sizing and application deployment decisions.

Fig. 1 also shows the high-level interaction between the simulator and the response time prediction modules. The simulator module takes as input the workload trace $\mathbf{T}$. It then calculates the load $\mathbf{L_i}$ experienced by a transaction i in the trace and queries the response time prediction module to get a response time estimate $pres_i^{\mathbf{L_i}}$ at this load for the transaction. This process is repeated for all N transactions to obtain the response time percentile of interest. We next discuss in detail the implementation of these two modules.

B. SIMULATOR

Algorithm 1 describes Prospective's simulator which takes as input the trace $\mathbf{T}$, the desired percentile P, and the input parameters derived from the pre-processing phase namely, the overlap parameter O and the number of iterations parameter I. We defer the description of the pre-processing phase to Sec. II-C. The output of Prospective is $res^P_{predicted}$, a prediction for the P^{th} percentile of response time.

As shown in Fig. 2 (a), the simulator first generates a time line consisting of the arrival instants of all transactions in $\mathbf{T}$ arranged in chronological order (lines 2-7, Algorithm 1). It then starts processing these arrival events as described in the second loop of Algorithm 1. Specifically, processing each arrival instant involves calculating the response time of that transaction and hence the time instant at which the transaction completes, i.e., the departure time. The simulator uses the response time prediction module to compute the response time of that transaction using the historical repository $\mathbf{D}$. As mentioned previously, to calculate the response time of any given transaction i the simulator requires the

Algorithm 1 Main Steps of Prospective

```
1  Input: T, $P$, $O$, $I$,  Output: $res^P_{predicted}$
2  foreach (transaction in T) do
3      $arrival \leftarrow transaction$.arrivalTime
4      $k \leftarrow transaction$.type
5      event ← createEvent($k$, $arrival$, null)
6      AddToTimeline(event)
7  end
8  foreach event in Timeline do
9      i ← i + 1
10     k ← $event$.transactionType
11     $arrival_i \leftarrow event$.arrivalTime
12     $\mathbf{L_i} \leftarrow$ calcStartLoad(i, $arrival_i$)
13     $pres_i^{\mathbf{L_i}} \leftarrow$ PredictResponseTime($i$, $k$, $\mathbf{L_i}$)
14     $departure_i \leftarrow arrival_i + pres_i^{\mathbf{L_i}}$
15     foreach iteration in (1, I) do
16         $\mathbf{L_i} \leftarrow$ calcLoad($arrival_i$, $departure_i$, $O$)
17         $pres_i^{\mathbf{L_i}} \leftarrow$ PredictResponseTime($i$, $k$, $\mathbf{L_i}$)
18         $departure_i \leftarrow arrival_i + pres_i^{\mathbf{L_i}}$
19     end
20     RESPs.add($pres_i^{\mathbf{L_i}}$)
21 end
22 $res^P_{predicted} \leftarrow$ calculatePercentile(RESPs, $P$)
```

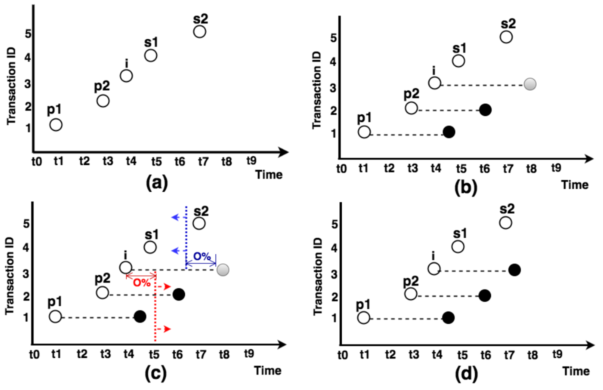


FIGURE 2. Example of load and response time estimation.

load $\mathbf{L_i}$ over the execution of that transaction. However, $\mathbf{L_i}$ is unknown when the simulator processes the arrival instant. As a result, the simulator has to guess $\mathbf{L_i}$ and iteratively refine that estimate.

The simulator uses the known instantaneous load at the arrival instant of i as an initial estimate of $\mathbf{L_i}$. The instantaneous load is a known quantity since the simulator knows the arrival instants of all previous transactions and has also calculated the departure instants of those transactions. The simulator provides this $\mathbf{L_i}$ estimate as well as the transaction's type k as inputs to the response time prediction module and obtains $pres_i^{L_i}$ as an initial response time estimate. This response time is then used to compute an initial departure time of transaction i. As shown in Fig. 2 (b), calculation of the departure time (denoted by a solid grey circle) allows the simulator to obtain a different estimate of $\mathbf{L_i}$ that captures other concurrently executing transactions. This triggers an iterative process where the initial estimate of $\mathbf{L_i}$ is refined taking into account the departures and arrivals that happen during the newly estimated execution time line of i (lines 16-20, Algorithm 1).

We first refine $\mathbf{L_i}$ using O. The parameter O allows the simulator to only include those transactions that have a significant overlap with the execution of transaction i. For any given transaction i and any other concurrent transaction j, the overlap $O_{i,j}$ between them is described as the difference between the departure time of the predecessor transaction and the arrival time of the successor transaction expressed as a percentage of the response time of transaction i. The simulator only considers those transaction pairs whose $O_{i,j}$ values are greater than or equal to O for computing the load $\mathbf{L_i}$.

Fig. 2 (c) illustrates the use of O. In this example, transactions $p1$ and $p2$ arrive prior to transaction i. The simulator has already computed their departure times (solid black circles) and they are both predicted to end after the arrival of i. Transactions $s1$ and $s2$ arrive after i but their departure times have not been estimated yet. For this example, only $p2$ and $s1$ satisfy the acceptable overlap criterion specified by O. Consequently, the load vector $\mathbf{L_i}$ is updated such that the number of transactions of the type to which $p1$ belongs is decremented by 1. This ensures that $p1$ does not influence the load. Similarly, the number of transactions of the type to which $s1$ belongs is incremented by 1. Transaction $s2$ is not involved in the load calculation since it did not meet the overlap criterion. The updated $\mathbf{L_i}$ is used to calculate a revised departure time for i, as shown in Fig. 2 (d). This refinement process is repeated I times. A pre-processing step, outlined next, is used to select O and I values that can yield accurate predictions.

C. PRE-PROCESSING FOR AUTOMATED PARAMETER SELECTION

The basic idea in this step is to use the simulator to predict the measured response times of transactions in the historical trace data $\mathbf{T_{rep}}$ used to construct the repository $\mathbf{D}$. The simulator employs various O and I values and selects the values that most accurately predict the historically measured response time percentiles embodied by $\mathbf{T_{rep}}$. We now describe two parameter selection approaches namely, arrival independent and arrival dependent.

The basic operation mode of Prospective, as depicted in Fig. 1, uses arrival independent parameter selection. With this approach, the simulator takes as input a workload trace $\mathbf{T_{tune}}$. Each line of $\mathbf{T_{tune}}$ corresponds to a transaction recorded in $\mathbf{T_{rep}}$. It contains the arrival instant of the transaction and the transaction's type. For the sake of simplicity, Prospective independently searches for the best O and I values. During the process for selecting the value of O, the value of I is set to 1. The simulator first conducts experiments using progressively increasing O values. For each of these experiments, we calculate over all percentiles of interest the mean of the prediction errors (defined in Sec. IV-D). We then select the O value that results in the least error. The simulator then uses this value of O and performs experiments with progressively increasing values of I. For any given percentile of interest, we select the I value that yields the least prediction error.

We also study Prospective with arrival dependent parameter selection where the choice of O and I depends on the mean transaction arrival rate λ. During parameter selection, we first split $\mathbf{T_{rep}}$ into a set of sub-traces $< \mathbf{T^1_{rep}}, \mathbf{T^2_{rep}}, \ldots, \mathbf{T^u_{rep}} >$, each containing the same number of transactions. For each sub-trace $\mathbf{T^i_{rep}}$, we calculate the mean transaction arrival rate λ_i and select O and I using a full factorial parameter sweep over a pre-defined range of values for these parameters. The final result after considering all the sub-traces is a look up table τ that records the O and I values discovered for λ_i $\forall i \in \{1, \ldots u\}$.

During the simulation, when arrival dependent parameter selection is enabled, the simulator considers equal duration transaction windows in the input trace $\mathbf{T}$. For each window w, it calculates the arrival rate λ_w. For any transaction i in this window, it provides λ_w to the response time prediction module in addition to the usual inputs, i.e., the load $\mathbf{L_i}$ and the transaction's type k. The response time prediction module uses the look up table τ to locate the entry whose arrival rate is closest to λ_w. It then uses the O and I values pertaining to that entry while predicting response times in that window w. Due to its arrival dependency and the use of full factorial parameter search, parameter selection can take a longer time using this approach. However, it could potentially improve prediction accuracy for workloads with highly variable transaction arrival patterns. We evaluate this in detail in Sec. V.

D. RESPONSE TIME PREDICTION MODULE

As shown in Algorithm 2, the response time prediction module takes as input the transaction type k of transaction i and the load $\mathbf{L_i}$. The module first looks up the repository $\mathbf{D}$ to check if any transactions of type k experienced the load $\mathbf{L_i}$ when the historical data was collected. If there is a hit for the k and $\mathbf{L_i}$ combination, then the module obtains a list $\mathbf{H^{L_i}_k}$ containing past transaction response times observed for this combination. Finally, a response time value is selected at random from this list and returned as $pres^{\mathbf{L_i}}_i$, the response time prediction for transaction i (lines 2-4, Algorithm 2). If there is no hit for the desired transaction type and load combination, then a CF based method is used to predict response time. This involves finding similar transaction types that have experienced the load $\mathbf{L_i}$ in $\mathbf{D}$.

The CF method relies on the computation of similarity measures between pairs of transaction types in the system. Two transaction types j and k are considered to be similar if their mean response times are similar to each other for all observed loads in the repository. Their similarity measure $S[j, k]$ characterizes the extent of their similarity. We use the commonly used Pearson correlation coefficient (PCC) as our similarity metric. Using PCC, the similarity between two transaction types k and j is defined by Eq. 1.

$$S[k,j] = \frac{\sum\limits_{e\in\mathbf{E_{kj}}} (res^e_k - res^*_k) \times (res^e_j - res^*_j)}{\sqrt{\sum\limits_{e\in\mathbf{E_{kj}}} (res^e_k - res^*_k)^2 \times \sum\limits_{e\in\mathbf{E_{kj}}} (res^e_j - res^*_j)^2}} \qquad (1)$$

In Eq. 1, the similarity $S[k, j]$ between transaction types k and j is calculated based on the mean response times of these two transaction types for all common loads, $\mathbf{E_{kj}}$, they have experienced before. Common loads are loads at the Web service that both k and j have experienced in the past. res^e_k shows the measured response time of transaction type k under one such load $e \in \mathbf{E_{kj}}$. In Equation 1, res^*_k and res^*_j are the mean response times of transaction types k and j under all system loads recorded in $\mathbf{D}$, respectively. As described shortly, the similarity value between each pair of transaction types is calculated and used within Eq. 2 to estimate the response time. We note that the similarity measures calculation need to be computed only once by processing the repository $\mathbf{D}$.

To predict the transaction's response time using CF the module first generates a random number r using a uniform distribution between 0 and 1. It then identifies $\mathbf{N^{L_i}_k}$ as the set of *other* transaction types similar to k that have encountered the load $\mathbf{L_i}$ in $\mathbf{D}$. For each similar transaction type j in $\mathbf{N^{L_i}_k}$, the module then constructs a list $\mathbf{H^{L_i}_j}$ that contains response times recorded for j under $\mathbf{L_i}$. Next, the module selects the r^{th} quantile of response time from this list as $pres^{\mathbf{L_i}}_j$. This process is depicted in lines 6-12 of Algorithm 2.

The final step of the CF method is to aggregate all the $pres^{\mathbf{L_i}}_j$ values pertaining to the similar transaction types into a single response time prediction $pres^{\mathbf{L_i}}_i$ for the transaction i. This aggregation uses the similarity measures between k and the similar transaction types recorded in $\mathbf{N^{L_i}_k}$. Specifically, we use the aggregation approach proposed by Breese *et al.* [12] implemented as Eq. 2 in our context.

$$res^{\mathbf{L_i}}_k = res^*_k + \frac{\sum\limits_{j\in\mathbf{N^{L_i}_k}} (res^{\mathbf{L_i}}_j - res^*_j) \times S[k,j]}{\sum\limits_{j\in\mathbf{N^{L_i}_k}} |S[k,j]|} \qquad (2)$$

When no transactions in $\mathbf{D}$ have experienced the load $\mathbf{L_i}$ the module simply returns the predicted response time $pres^{\mathbf{L_i}}_k$ as the mean response time of all transactions of type k in the repository $\mathbf{D}$ (Lines 14-16, Algorithm 2). We refer to the percentage likelihood of this scenario as the miss rate which provides an estimate of the error in Prospective's prediction.

While the basic operation mode of Prospective distinguishes transactions based on transaction types, we also explore another approach where similar transaction types are further clustered into a smaller number of *transaction groups*. We refer to this as the *clustered approach*. With this approach, we use the well-known K-means clustering algorithm [13] during the pre-processing step. We use a simple method where the centroid of a cluster is the mean response time

Algorithm 2 Response Time Prediction

```
1  Input: D, $i$, $k$, $\mathbf{L_i}$, Output: $pres_i^{\mathbf{L_i}}$
2  $\mathbf{H_k^{L_i}} \leftarrow$ query repository ($k$, $\mathbf{L_i}$)
3  if ($\mathbf{H_k^{L_i}}$ is not empty) then
4      $pres_i^{\mathbf{L_i}} \leftarrow$ randomValue($\mathbf{H_k^{L_i}}$)
5  else
6      $\mathbf{N_k^{L_i}} \leftarrow$ query similar transactions($k$, $\mathbf{L_i}$)
7      if ($\mathbf{N_k^{L_i}}$ is not empty) then
8          $r \leftarrow$ randomNumber()
9          foreach $j \in \mathbf{N_k^{L_i}}$ do
10             $\mathbf{H_j^{L_i}} \leftarrow$ query repository ($j$, $\mathbf{L_i}$)
11             $pres_j^{\mathbf{L_i}} \leftarrow$ getValue($r$, $\mathbf{H_j^{L_i}}$)
12         end
13         $pres_i^{\mathbf{L_i}} \leftarrow$ adjusted value of all $pres_j^{\mathbf{L_i}}$ (Eq. 2)
14     else
15         $pres_i^{\mathbf{L_i}} \leftarrow$ mean response time of $k$
16     end
17 end
```

of transaction types assigned to that cluster. In this manner, transaction types with similar mean response times get clustered to the same transaction group.

With the clustered approach, the response time prediction module first maps the input transaction type k to its transaction group g. It also translates the input load vector $\mathbf{L_i}$ into a vector that records the numbers of concurrent transactions belonging to each transaction group. The rest of the prediction process is similar to what is outlined in Algorithm 2 except that the module operates at the granularity of transaction groups as opposed to transaction types. The clustered approach could benefit scenarios where it is possible to come up with a more compact grouping of transactions than that suggested by the service's URLs. We evaluate this feature in detail in Sec. V.

III. USING PROSPECTIVE FOR HORIZONTAL SCALING

In the previous sections, we describe how Prospective can be used for predicting the response times of transactions submitted to a single instance of a Web service. If a predicted response time percentile exceeds its corresponding threshold, the service operator would typically be interested in exploring horizontal scaling strategies that employ multiple server instances. With horizontal scaling, incoming transactions are distributed among a set of server instances based on a load balancing policy. To support evaluation of horizontal scaling strategies, we implement a load balancing module in Prospective that simulates the common load balancing policies, such as Weighted Round Robin (WRR) and the Least Connection (LC). The number of instances of each instance type is specified as an input to the simulation.

With the WRR policy, each server instance m has associated with it a weight w_m that determines the fraction of incoming transactions assigned to that instance. Before starting the simulation process described in Sec. II-B, Prospective uses the weights to assign each transaction in the input trace $\mathbf{T}$ to a specific server instance. As a result, each instance m gets assigned an input trace of transactions $\mathbf{T_m}$. Next, Prospective simulates the behaviour of each instance using the traces generated in this manner. Specifically, for each instance m Prospective uses the historical repository pertaining to that instance to estimate the response times in $\mathbf{T_m}$. Finally, response times estimated in this manner are collected from all instances to calculate the response time percentile of interest.

The LC policy assigns an incoming transaction to the instance that is currently handling the least number of active transactions ν. Active transactions at an instance are those transactions that are currently being executed at the instance. Since the ν values depend on the response time behaviour of individual instances, it is more challenging to simulate LC than WRR. To realize LC, Prospective maintains separate transaction arrival time lines for the instances. Consider transaction i in $\mathbf{T}$ with an arrival instant of t. Prospective assigns this transaction to the arrival time line of the instance with the least ν at time t. We note that in contrast to the example outlined in Fig. 2, at this point Prospective has no knowledge about the transactions that would follow i. This is because future arrivals at the instance depend on instantaneous values of ν at the instances. Consequently, response time and completion time of transaction i is estimated based only on the transactions that arrived before i at this instance. This estimate as well as the response time estimates of the other active transactions at this instance are continually adjusted based on the transactions that arrive subsequently to the instance. Response time percentiles of interest are calculated after all transactions in $\mathbf{T}$ have been assigned to the instances in this manner.

TABLE 1. Experiment settings.

Host specification	Two 6-core Intel Xeon 1.6 GHz E5645, 32GB RAM per socket
Large VM specification	vCPU: 2, Memory: 2GB
Small VM specification	vCPU: 1, Memory: 1GB
OS (Guest and Host)	Ubuntu 12.04
Web server	Apache 2
Workload generator	httperf

IV. EXPERIMENT SETUP

A. EXPERIMENT TESTBED

Table 1 provides details of our test environment. We use virtual machines (VM) running on a multicore server to host our Web servers. We use the httperf tool [14] for generating workloads. httperf is installed on a different multicore server. Both servers are connected by an Ethernet switch that provides dedicated 1 Gbps connectivity. Consequently, response times measured by httperf are a reliable indicator of the server's performance. Finally, we implement Prospective using a set of Python scripts.

B. EXPERIMENT FACTORS

Table 2 lists the experiment factors and their levels. Default levels are shown in bold.

TABLE 2. Experiment factors.

Factor	Levels
Web Application	RUBiS
VM Instances	**Large instance**, Small instance
Percentile	25, 50, 75, 95, 99
Arrival Patterns	1)Low-Exp, 2)**Medium-Exp**, 3)High-Exp, 4)Low-High, 5)High-Low, 6)Periodic-Gradual, 7)Periodic-Rapid, 8)HypoExp, 9)HyperExp, 10)VHigh-Exp, 11)XHigh-Exp
Mix	**Mix 40S-60D**, Mix 60S-40D, Mix 100S-0D
Estimation Approaches	**Prospective**, Avg-Tr, Avg-Load, Avg-Tr-Load Reg-CPU, Reg-Rate

We run the RUBiS benchmark version 1.0 [6] as our Web service. RUBiS is a popular benchmark for multi-tiered applications and has been used extensively by others [7], [15]–[19]. It emulates the behavior of an auction server that allows users to browse and bid for items. The service supports 14 distinct transaction types. Examples of the supported transactions are browsing products, searching, and viewing bid histories.

We use the Large VM instance by default. The target response time percentiles that we are interested in predicting are 25^{th} , 50^{th} , 75^{th} , 95^{th}, and 99^{th} percentiles. The 95^{th} and 99^{th} percentiles are commonly used in SLAs. They capture the tail behaviour of the response time distribution and are challenging to predict.

We study Prospective's ability to predict the behaviour of different types of transaction arrival patterns. Fig. 3 shows the inter arrival time between successive transactions of a subset of these patterns. Patterns 1 to 3 use the exponential distribution to generate inter arrival times. Their mean inter arrival times are selected to cause low (30%), medium (50%), and high (70%) mean per-core utilization on a large VM instance of the RUBiS server.

As shown in Fig. 3, patterns 4 to 7 emulate time varying arrivals, which are typical of many real Web services [20], [21]. Patterns 4 and 5 use exponential distributions with different means to achieve the time varying behaviour. In pattern 4, the mean inter arrival times are chosen such that the RUBiS server's mean per-core utilization increases from 10% to 80%, and then decreases from 80%, to 10%. In pattern 5, the mean per-core utilization decreases from 80% to 10% and then increases from 10% to 80%. By alternating between very low and very high utilizations, these patterns mimic bursty behaviour. Inter arrival times of pattern 6 and pattern 7 do not follow the exponential distribution. Pattern 6 causes mean per-core utilization to gradually vary between 10% to 65%. Pattern 7 utilizations vary similarly but the variations are more rapid.

Patterns 8 and 9 are used to study the impact of distributions with lower and higher variability than the exponential distribution. While pattern 8 has half the coefficient of variation (CV) of exponential, pattern 9 has two times the CV of exponential. For these patterns we use the same mean inter arrival time as in the default $Medium - Exp$ pattern, i.e., Pattern 2, to facilitate comparisons.

To demonstrate the ability of Prospective to support system sizing exercises, we consider two additional patterns. Patterns 10 and 11, $VHigh - Exp$ and $XHigh - Exp$, are exponential patterns (not shown in Fig. 3) that cause very high and extreme utilizations, respectively. $VHigh - Exp$ is the maximum load that can be sustained on the large instance. This in turn leads to extremely long response times on the service. The mean arrival rate of $XHigh - Exp$ is almost twice that of $VHigh - Exp$ and hence this pattern cannot be handled using one VM instance alone. Results involving these patterns are discussed in Sec. VI.

We also study the sensitivity of Prospective to transaction mix. We use three different mixes that differ in the percentages of static transactions (S), i.e., transactions that do not require dynamically generated content, and dynamic transactions (D). $Mix40S - 60D$ is the default browsing mix of RUBiS. It contains 40% static transactions and 60% dynamic transactions. This mix is used in creating the repository. $Mix60S - 40D$ contains 60% static and 40% dynamic transactions. $Mix100S - 0D$ contains only static transactions.

We evaluate Prospective under two different modes. Under the basic mode, Prospective offers predictions at the granularity of transaction types and uses arrival independent selection of O an I. With the enhanced mode, it clusters transaction types into transaction groups and also uses arrival dependent parameter selection. In this mode, we consider 5 second transaction windows. As described in Sec. II, we compute λ_w for each such window and use that in combination with the look up table τ to obtain the O and I values to use for that window.

Several different baseline response time estimation strategies are compared. The *Avg-Tr* approach ignores the dependency of transaction response times on the load vector. It estimates the response time of a transaction by computing the mean of the response times recorded for that transaction's type in the historical repository. The *Avg-Load* approach considers load but ignores transaction type. For each transaction, it estimates the load, i.e., total number of concurrent transactions. It then obtains the prediction as the mean response time of all transactions, regardless of transaction type, that have experienced the estimated load. We note that this approach is similar to the approach used by Spicuglia *et al.* [22]. The *Avg-Tr-Load* approach considers both load and transaction type similar to Prospective. However, unlike Prospective it merely estimates the response time of a transaction as the mean response time of all records with the desired load and transaction type. We note that these baseline methods all use arrival independent parameter selection.

We also compare Prospective with two different quantile regression approaches. The *Reg-CPU* approach suggested by Watson *et al.* [7] relates the response time percentile to the mean per-core utilization U of the server, as shown by

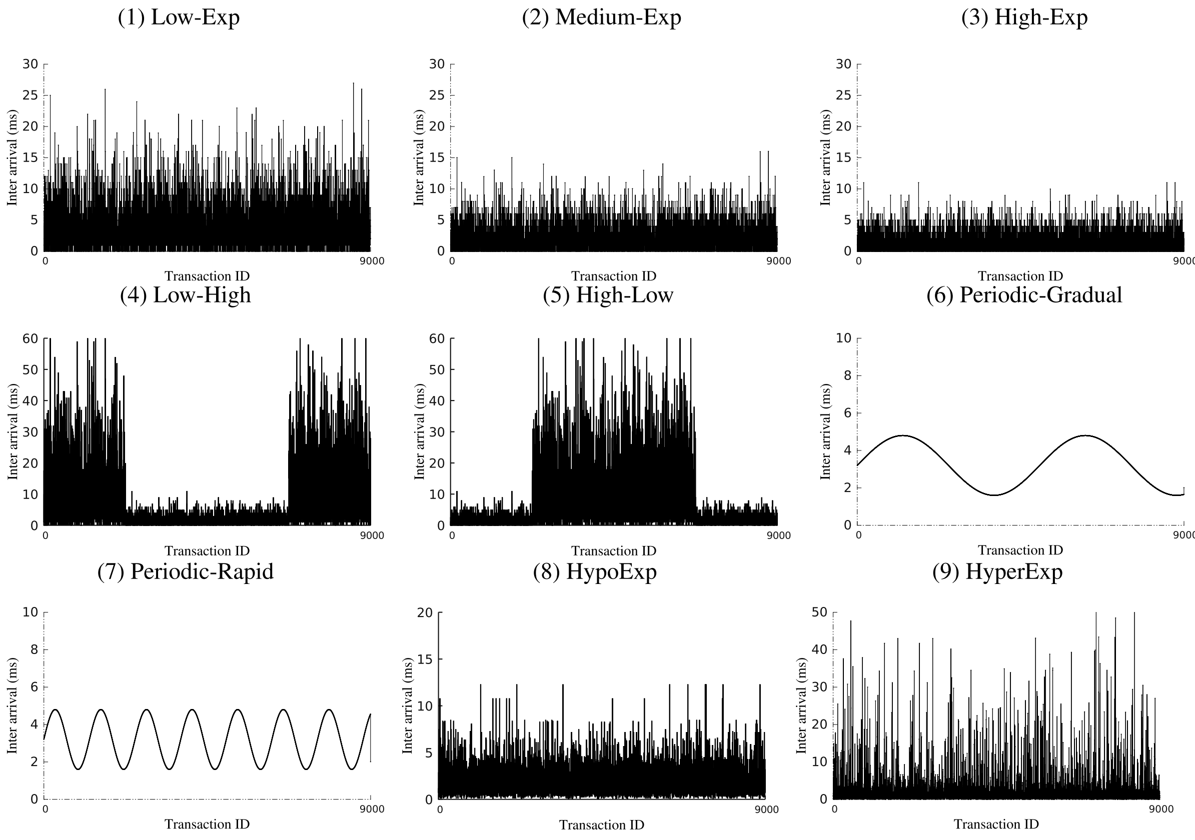


FIGURE 3. Arrival patterns used in experiments.

the performance function in Eq. 3. The *Reg-Rate* approach used by Bodik *et al.* [8] is similar but uses mean transaction arrival rate λ as the independent variable, as shown in Eq. 4. To facilitate a fair comparison, we use the same historical data contained in Prospective's repository to train and obtain the β coefficients in the models. The historical data is divided into 1-second windows. Each window provides training observations that relate the dependent variable, i.e., response time of a transaction in that window, to the independent variable, i.e., either U or λ of the window. Quantile regression uses the training data to obtain different sets of β coefficients for the different percentiles of interest.

$$res^P_{predicted}(U) = \beta_0 + \beta_1 \times U + exp(\beta_2 + \beta_3 \times U) \quad (3)$$

$$res^P_{predicted}(\lambda) = \beta_0 + \beta_1 \times \lambda + exp(\beta_2 + \beta_3 \times \lambda) \quad (4)$$

C. EXPERIMENT PROCESS

Before starting the evaluation, we need to create a historical repository. Our intent is to create a repository that covers a diversity of transaction types and per-core utilizations. To create such a repository, we use the default mix, $Mix40S - 60D$, that contains all the RUBiS transaction types. We then submit to the service traces of transactions conforming to this mix and having exponential inter arrival time distributions. The mean inter arrival times are selected to generate various per-core utilization levels ($10\% <$ per-core utilization $< 90\%$ in steps of 5%). For each utilization level, we repeat the experiment using 5 different samples drawn from the relevant exponential distribution. Each run contains 9,000 transactions. As mentioned previously, a transaction trace $\mathbf{T_{rep}}$ is collected from system logs under these experiments and is used for creating the repository. After creating the repository, the pre-processing steps discussed previously are carried out.

For the arrival independent selection of O and I, we vary O in steps of 10% from 10% to 90% and I in steps of 1 from 1 to 3. Since we search for O and I independently, this translates to 11 simulation experiments to identify these parameters. For the arrival dependent selection, we split the trace used in the arrival independent approach into 9 smaller sub-traces having equal number of transactions. For each sub-trace, we use the same ranges for O and I as in the arrival independent selection. However, since we employ a full factorial parameter sweep, parameter selection involves 27 experiments for each sub-trace. Accounting for the smaller traces, the arrival dependent approach needs approximately 2.5 times more experiments than the arrival independent approach for deducing the values of O and I.

For each measurement and simulation experiment, we use multiple statistically similar samples from the relevant arrival model and transaction mix. Each of them involves a trace of 9,000 transactions. We calculate the mean and the 95% CI for both the measured and predicted response time percentiles by replicating the experiments.

D. EVALUATION METRIC

To evaluate the accuracy of Prospective, we use the absolute relative error of the predicted response time percentile values as shown in Eq. 5.

$$err_P = |\frac{res^P_{predicted} - res^P_{measured}}{res^P_{measured}}| \times 100 \quad (5)$$

In Eq. 5, P represents the desired percentile level and err_P is the error for the P^{th} percentile response time prediction. $res^P_{predicted}$ and $res^P_{measured}$ are the mean of the predicted and measured P^{th} percentile values.

V. RESULTS

In this section, we report the result of experiments designed to validate Prospective.

A. PRE-PROCESSING

We first focus on selecting the O and I values. Fig. 4 shows the selection of the overlap parameter O for three different sub-traces with progressively increasing arrival rates namely, Low, Medium, and High. We only show the 95th percentile with I set to 1. Similar trends are observed for other cases.

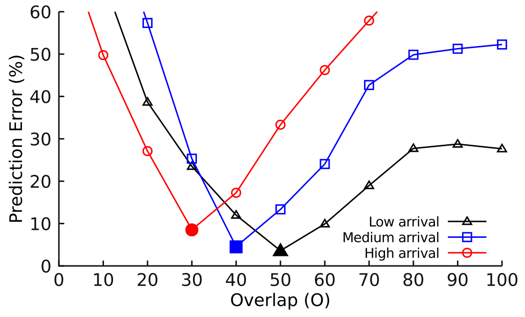


FIGURE 4. Selecting the overlap value.

From Fig. 4, O values around 50%, 40%, and 30% yield the lowest errors for Low, Medium, and High, respectively. For any given arrival rate, very low values of O cause transactions that have very low overlap to be counted towards the load leading to overly pessimistic response time estimates. Very high values ignore the effects of some transactions that have quite a bit of overlap leading to optimistic estimates. Furthermore, Fig. 4 shows that higher arrival rates benefit from lower values of O. This suggests that a transaction's response time at higher loads is sensitive to even transactions that only have a small overlap. These results motivate the arrival dependent selection of O for scenarios experiencing high arrival rates.

Results suggest that the tuning of I is specific for each percentile of interest. In general the value of I needs to be greater than 1 for the extreme tail percentiles, e.g., the 99th percentile, for better accuracy. We omit presenting detailed results for the sake of brevity.

Finally, we focus on the clustered approach where Prospective assigns transaction types to transaction groups using K-means clustering. As discussed previously, the selection of the number of clusters, i.e., transaction groups, and the clustering is done automatically in the pre-processing step. In our case, when increasing the number of clusters the mean distance of a point from its cluster centroid decreases and stabilizes with 5 clusters. As a result, we group RUBiS transaction types into 5 transaction groups.

B. COMPARISON WITH BASELINE AND REGRESSION APPROACHES

In this section, we compare the baseline estimation approaches and the state-of-the-art quantile regression techniques introduced in Sec. IV-B with Prospective. We use the basic operation mode of Prospective, which uses arrival independent O and I and operates at the granularity of transaction types. We consider the default levels for the experiment factors as shown in Table 2 and report mean prediction errors of all percentile values.

Avg-Tr, which ignores load, produces a large mean error of 39.8%. Avg-Load, which considers load but ignores transaction type, also does poorly with a mean error of 72.9%. In contrast, Prospective yields a mean error of 4.6%. These results establish the importance of incorporating both load and transaction type into the prediction process.

Similar to Prospective, Avg-Tr-Load considers both transaction type and load. Recalling from Sec. IV-B, Avg-Tr-Load uses mean of the response times recorded for a transaction type and load combination. In contrast, Prospective randomly draws from a list of historical response times recorded for that combination. Fig. 5 compares their predicted response time distributions. From the figure, the averaging done by Avg-Tr-Load causes probability masses to bunch together over a few values causing the distribution to deviate significantly from the actual measured distribution. Prospective avoids this problem and follows the measured distribution more closely.

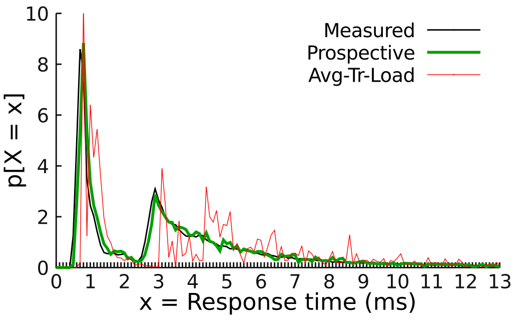


FIGURE 5. Comparison of response time distributions.

Next, we compare the performance of Prospective with the two quantile regression approaches, $Reg - CPU$ and $Reg - Rate$. Table 3 shows the prediction errors of the three approaches when we vary the mix while keeping other factors at their default levels. From the table, all three techniques perform comparably when predicting for the default mix, i.e., $Mix - 40S - 60D$. Recalling from Sec. IV, this is the mix used to build the historical repository and train the regression

TABLE 3. Mean prediction errors of regression models.

	40S-60D	60S-40D	100S-0D
Reg-CPU	7.0	51.2	165.6
Reg-Rate	3.6	92.9	511.9
Prospective	4.6	9.9	8.8

models. From Table 3, as expected, the accuracies of the regression techniques suffer significantly while predicting for the other two mixes, i.e., $Mix-60S-40D$ and $Mix-100S-0D$. This is because these regression techniques do not consider the mix in their models. $Reg-CPU$ approach performs better than $Reg-Rate$, as CPU utilization can capture the change of mix better than arrival rate. However, $Reg-CPU$ requires a method to predict the CPU utilization of the workload for which prediction is desired. In contrast to the regression techniques, Prospective provides accurate estimates for these non-default mixes. These results indicate Prospective's ability to generalize well for workloads not directly captured by the historical repository. We explore the impact of mix in more detail in Sec. V-D.

We also compare the sensitivity of the three approaches to arrival patterns different than the one used to build the repository. Recalling from Sec. IV, the repository is built by using a set of time varying exponential transaction inter-arrival time distributions. We use the three techniques to offer predictions for the four non-exponential workloads, i.e., Patterns 6-9, and default levels for other factors. For these four patterns, $Reg-CPU$ and $Reg-Rate$ perform comparably and result in mean errors of 23% and 25%, respectively. This is about 3 times higher than the mean prediction error of 7.2% that we obtain with Prospective for these four patterns. This result further corroborates the ability of Prospective to generalize better than the regression approaches. The accuracy of the regression approaches could be improved by exploring other choices for performance functions, regression techniques, and featurization, e.g., independent variables that explicitly capture the mix and arrival pattern. Prospective offers an alternative that does not require an operator to spend time investigating such choices.

C. EFFECT OF ARRIVAL PATTERNS

Table 4 shows Prospective's percentile prediction errors and their mean for the different arrival patterns in the *basic mode*. Patterns 1 to 5 use either exponential or time varying exponential distributions. While the repository is constructed using exponential distributions, the exact sequence and inter-arrival times of transactions in the repository are different from those of patterns 1 to 5. For example, unlike Patterns 4 and 5, the repository does not encounter situations where the system switches between very high and very low utilizations. Results show that Prospective yields very good accuracy for these patterns with the maximum mean error being 13.2%.

Among patterns 1 to 5, the worst errors occur for cases where the system encounters very high per-core utilizations. For example, the $High-Exp$ pattern imposes a mean per-core utilization of 70% with peak utilizations reaching up to 90%. The prediction errors for the 95^{th} and 99^{th} percentiles for this case are 18.5% and 30.1%, respectively. Similarly, the $Low-High$ and $High-Low$ represent bursty workload patterns where the mean per-core utilization can reach an extremely high value of 80%. The mean prediction errors for the 95^{th} and 99^{th} percentiles for these two case are 15.0% and 23.5%, respectively. We notice that at very high utilizations there is more diversity in the number of unique load vectors that can be encountered by the system. As a result, there is a higher likelihood that Prospective will have a miss, i.e., not find a match for a transaction type and load combination in the repository either by using historical response times directly or by applying the CF method. As discussed previously, Prospective merely uses the mean historical response time for the relevant transaction type as its prediction when there is a miss. This leads to optimistic predictions for the 95^{th} and 99^{th} percentiles.

TABLE 4. Prediction errors - basic prospective.

	25	50	75	95	99	Mean
Low-Exp	6.2	2.4	8.6	10.0	13.0	**8.0**
Medium-Exp	4.9	2.3	4.8	5.5	5.8	**4.6**
High-Exp	5.5	2.6	9.5	18.5	30.1	**13.2**
Low-High	2.1	1.0	9.6	16.0	25.0	**10.8**
High-Low	2.3	3.4	8.7	14.0	22.0	**10.1**
Periodic-Gradual	5.7	2.8	9.7	7.1	6.5	**6.4**
Periodic-Rapid	11.1	3.7	9.0	1.9	6.4	**6.4**
HypoExp	5.0	2.3	10.6	8.1	3.3	**5.9**
HyperExp	7.1	0.9	8.0	18.7	16.2	**10.2**

Table 4 also shows that Prospective is very accurate for the $Periodic-Gradual$ and $Periodic-Rapid$ patterns, which have non-exponential, time varying arrivals. These results indicate that Prospective can offer predictions for inter arrival time distributions different than that governing the repository. Specifically, it can provide good predictions as long as there is a good match in the repository for the transaction type and load combinations.

Next, we focus on the impact of the variability of the inter-arrival time distribution. From Table 4, Prospective is very accurate for the *HypoExp* pattern, which imposes the same per-core utilization as $Medium-Exp$ but has half the CV of exponential. The *HyperExp* pattern has twice the CV as exponential and hence triggers phases with intense transaction arrivals where the per-core utilization of the system is very high. As a result, Prospective is less accurate for this case than the $Medium-Exp$ and *HypoExp* cases. However, the mean error is still only 10.2% for this case.

Summarizing, the basic operation mode of Prospective is very accurate for cases that correspond to the utilization ranges of real life servers [23]. Errors increase for scenarios with very high utilizations. We next show how such high utilization scenarios can benefit from the enhanced operation mode of Prospective, i.e., using arrival dependent O and I and operating at the granularity of transaction groups.

TABLE 5. Prediction errors - enhanced prospective.

	25	50	75	95	99	Mean
(1) Low-Exp	2.3	0.4	6.6	3.1	3.3	**3.1**
(2) Medium-Exp	5.0	3.8	3.9	5.6	4.7	**4.6**
(3) High-Exp	5.2	2.8	5.3	8.1	5.5	**5.4**
(4) Low-High	4.0	3.2	6.1	10.1	11.3	**6.9**
(5) High-Low	5.8	2.5	2.8	10.9	14.1	**7.2**
(6) Periodic-Gradual	4.0	1.7	5.5	4.2	5.5	**4.1**
(7) Periodic-Rapid	7.8	4.5	5.3	2.2	4.3	**4.8**
(8) HypoExp	3.2	3.9	6.2	8.5	7.2	**5.8**
(9) HyperExp	7.0	8.6	7.1	2.2	9.1	**6.8**

Table 5 compares Prospective's accuracy for the different arrival patterns while operating in the *enhanced mode*. In general, Prospective is even more accurate under the enhanced mode. The mean error over all patterns and all percentiles decreases from 8.4% to 5.5%. The baseline operation of Prospective has mean errors of 11.0% and 14.2% for the 95th and 99th percentiles, respectively. In contrast, the enhanced mode reduces the mean errors for the 95th and 99th percentiles to 6.1% and 7.2%, respectively.

From Table 5, the enhanced mode is particularly effective for scenarios that cause high per-core utilizations. For the $High-Exp$ pattern, the errors for the 95th and 99th percentiles decrease from 18.5% and 30.1% to 8.1% and 5.5%, respectively. Similar improvements are also seen for the extremely bursty $Low - High$ and $High - Low$ patterns as well as the highly variable *HyperExp* pattern.

The reason for the effectiveness of the enhanced mode at high utilizations is two fold. First, as noted previously, at high utilizations there is a greater diversity in the number of possible load vectors seen by the system. Clustering reduces this diversity by grouping similar transaction types into the same transaction group. This decreases the number of misses thereby improving accuracy. Second, as shown in Sec. V-A, high load scenarios benefit from smaller values of O when compared to low load scenarios. By adapting the choice of O to load, i.e., transaction arrival rate, the enhanced mode is able to further improve accuracy. For the remainder of the paper we use the enhanced mode of Prospective.

D. EFFECT OF TRANSACTION MIX, CF, AND REPOSITORY EVOLUTION

To study Prospective's sensitivity to mix, we submit each of the mixes shown in Table 2 while using the default $Medium - Exp$ arrival pattern. The mixes have very different response time behaviours. For example the 99th percentile of the response time with $Mix40S - 60D$ is 10 times larger than the response time of the $Mix100S - 0D$. Table 6 shows the prediction errors. As expected, Prospective offers the most accurate predictions for the default $Mix40S - 60D$, which also corresponds to the mix of the historical repository. Predictions degrade slightly for the non-default mixes $Mix60S-40D$ and $Mix100S-0D$ but are still quite accurate. Errors correlate well with the miss rates seen by the simulator. The miss rates for $Mix60S - 40D$ and $Mix100S - 0D$ are

TABLE 6. Prediction errors(%) for different mixes.

	25	50	75	95	99	Mean
Mix 40S-60D	5.0	3.8	3.9	5.6	4.7	**4.6**
Mix 60S-40D	3.8	11.4	2.3	8.3	11.8	**7.5**
Mix 100S-0D	5.2	7.3	4.4	5.9	5.8	**5.7**

12% and 4%, respectively. This suggests that the miss rate reported by Prospective can be used by the operator to assess predictive accuracy.

We explore $Mix60S - 40D$ further to show the value of incorporating the CF method into Prospective. Disabling CF doubles the miss rate. As a result, the prediction error for the 95th percentile and the mean error increase from 8.3% and 7.5% to 16.0% and 14.1%, respectively.

We now study the impact of evolving the repository to include a new mix. Prospective can continually expand its repository to include new measurements, e.g., data from the live system. To verify this, we expose the system to a workload that uses $Mix60S - 40D$. We use a time varying exponential inter arrival time distribution similar to the process described in Sec. IV-C. Response time information from this workload is then integrated into the repository. We then conduct a simulation to predict for $Mix60S - 40D$ with the default $Medium - Exp$ arrival pattern. Due to the expanded repository, the error for the 95th percentile and the mean error decrease from 8.3% and 7.5% to 5.7% and 6.0%, respectively. These prediction errors are close to the error of the default mix, $Mix40S-60D$, and shows that Prospective is robust and can continually refine its predictions based on new data. The expansion of the repository happens in the same way when new arrival patterns are observed.

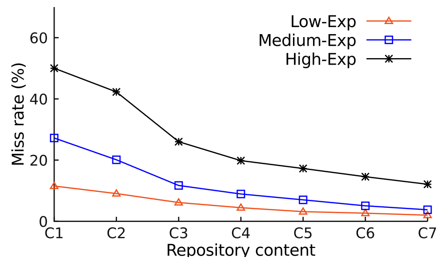
E. SENSITIVITY TO REPOSITORY CONTENT

In this experiment, we study the sensitivity of Prospective to the contents of the repository in more detail. For this purpose, we compare the prediction accuracy for the $Low - Exp$, $Medium - Exp$, and $High - Exp$ patterns when the repository contains different sets of historical data. Recalling from Sec. IV, these workloads cause per-core utilizations of 30%, 50%, and 70%, respectively. All other factors are set at their default levels. For this experiment, as shown in Table 7, we use 7 different controlled sets of traces to populate the repository. All traces are generated using exponential inter-arrival times and the default mix. The sets differ in the range of per-core utilizations they cause on the system. For example, $C3$ causes two distinct periods of activity with mean per-core utilizations of 10% and 90%. $C7$ is the default trace used to generate the repository for the experiments presented in the previous sections.

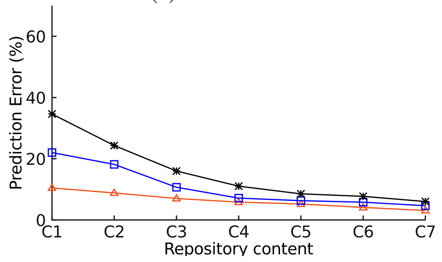
For the various repositories we consider, Fig. 6a and Fig. 6b show the miss rate and mean prediction error, respectively. From the figures, the miss rate and prediction error are high with $C1$ when the repository only contains data for low per-core utilization. Interestingly, miss rate decreases and accuracy improves for all three workloads for $C2$ even though

TABLE 7. Repository content.

Cases	Included data
C1	$\{U10\}$
C2	$\{U90\}$
C3	$\{U10, U90\}$
C4	$\{U10, U50, U90\}$
C5	$\{U10, U30, U50, U70, U90\}$
C6	$\{U10, U20, ..., U90\}$
C7	$\{U10, U15, U20, ..., U90\}$



(a) Miss rates



(b) Prediction errors

FIGURE 6. Results with different repository content.

the repository only contains data pertaining to the extreme 90% per-core utilization. As stated previously, the number of unique load vectors increases with utilization. As a result, $C2$ yields more load vectors than $C1$ thereby helping Prospective. From Fig. 6a and Fig. 6b, miss rate decreases and accuracy improves significantly with $C3$, which captures behaviour under both low, i.e., 10%, and extreme, i.e., 90%, per-core utilizations. Covering a wide range of utilizations yields a diverse set of load vectors to the repository which in turn helps improve accuracy.

F. DISCUSSION

From the results, Prospective significantly outperforms other baseline techniques. In particular, unlike the existing quantile regression-based approaches, Prospective is able to offer accurate predictions even for arrival patterns and mixes not directly captured in the repository.

The use of load vectors in combination with CF helps offer accurate predictions for a given arrival pattern and mix even if that pattern and mix have not been explicitly observed in the past. Specifically, Prospective modulates the progression of load vectors during a simulation to implicitly consider the impact of both the arrival pattern and the mix. For example, a synthetic input trace with a highly variable arrival pattern causes Prospective to generate a highly variable progression of load vectors thereby helping to capture the impact of the variability. Similarly, a mix that favours one particular transaction type will cause Prospective to generate a progression of load vectors where that transaction type dominates.

Prospective is a data-driven technique and therefore, similar to any other data-driven technique depends on the quality of the historical repository. Data-driven techniques are usually sensitive to the quality of their data. The superiority of Prospective is in mitigating this sensitivity through the CF technique. Where possible, Prospective exploits existing historical response time measurements for the transaction type and load vector combinations generated during simulation. For example, while predicting for $Mix60S-40D$ in Sec. V-D, 76% of the combinations generated by Prospective during the simulation are already present in the repository even though the repository is generated using a different mix. For combinations not observed previously, it exploits a novel CF based method to estimate response times. Considering the same $Mix60S-40D$ example from Sec. V-D, our CF method allowed predictions to be generated for approximately half of the 24% of combinations not present in the repository.

The main reasons for the superiority of the technique can be summarized as follows:

- Prospective's response time prediction process considers the transaction type. This improves accuracy since different transaction types can stress system resources differently.
- The load vector used during prediction records the numbers and types of concurrent transactions observed in the system. The mix of the incoming workload can affect the mix of the transactions observed in the population vector. Therefore, it can capture the impact of the workload mix.
- The population vectors can also capture the arrival rate of transactions. The higher the transaction arrival rate is, there is more chance to observe vectors with a higher number of transactions in them.
- Moreover, the population vectors can capture the effect of arrival patterns. For example, a high variability in a bursty workload is reflected as higher variability in the vectors experienced by the system.
- The use of the population vectors in conjunction with CF, helps our system to offer predictions for a given workload even if that workload has not been explicitly observed in the past.

VI. USING PROSPECTIVE FOR SYSTEM SIZING

We now demonstrate how Prospective can be used for horizontal scaling to mitigate violation of a response time percentile target. Specifically, we show how one can estimate the number and type of server instances with minimum cost to achieve a target response time percentile. This feature can enable self-organization in could services.

We consider two different types of VM instances for the RUBiS service namely, Large (L) and Small (S). We assume that the cost of acquiring one Large instance is greater than the cost of acquiring two Small instances and lower than the cost of three Small instances. Given this assumption, Prospective searches for the system configuration with a minimum cost that meets the SLA requirements. Accordingly, we create a repository for Large and another repository for Small using the methodology described in Sec. IV-C.

We use an iterative sizing approach to determine a horizontal scaling strategy. We start by simulating the service deployed on a single Large instance. If there is a violation, we explore the addition of another instance. We define this as *scaling step*. At each scaling step, we explore first the addition of a Small instance. If simulation of this configuration suggests that the violation persists, we remove the Small instance from the configuration and explore instead the addition of a Large instance. If the violation still persists even with the addition of the Large instance, we initiate one more scaling step to provision an additional instance. The process is repeated until Prospective indicates no violation.

We study both the WRR and LC load balancing policies. From Table 1, the Large instance has twice the number of cores and memory as the Small instance. Consequently, a Large instance in a configuration will handle twice the number of transactions as the Small instance in the configuration.

Our goal is to realize a system that meets a 95th percentile response time target of 10 ms. We consider three workloads namely, $High - Exp$, $VHigh - Exp$, and $XHigh - Exp$. Both $High - Exp$ and $VHigh - Exp$ violate the target while using a single Large instance. As mentioned in Sec. IV, $XHigh - Exp$ cannot be sustained by a single Large instance alone.

Fig. 7 shows the configurations suggested by Prospective while using WRR. Prospective indicates that the 95th percentile for the $High - Exp$ workload is just under the target of 10.0 ms when using a configuration with one instance each of Large and Small, i.e., L-S in Fig. 7. The $VHigh - Exp$ and $XHigh - Exp$ workloads require 2 and 3 Large instances, respectively. Note that in Fig. 7, the measured and predicted values for $XHigh - Exp$ are shown only for configurations that can sustain this workload.

To validate Prospective's predictions, we measure the performance of all configurations evaluated by Prospective for all three workloads. We use the HAProxy [24] load balancer configured with WRR. From Fig. 7, the 95th percentiles obtained from Prospective are close to the measured percentiles for every configuration explored. Measurements show that the configurations suggested by Prospective are the configurations with the minimum cost that will not violate the target 95th percentile. We repeated the same experiments with the LC policy. As with WRR, predictions closely match measurements.

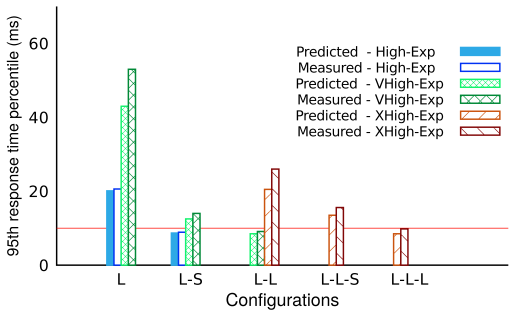


FIGURE 7. 95th percentiles of WRR for different configurations.

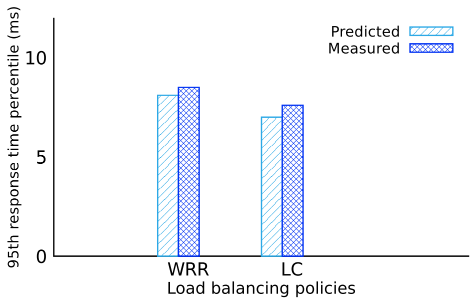


FIGURE 8. Response times under WRR and LC.

Detailed analysis also reveals that LC yields lower measured response times than WRR and that Prospective is able to capture this behaviour. As an example, we refer to Fig. 8, which focuses on the predicted and measured 95th percentile response times of both policies for the $High - Exp$ workload under the $L - S$ configuration. From the figure, both the measured and predicted response times with LC are lower than their corresponding values with WRR. This shows that Prospective is able to successfully predict the measured behaviour of improved response times with LC. Overall, Prospective has comparable response time prediction accuracies for WRR and LC with the accuracy of LC being slightly lower due to its higher complexity. Over the scaling steps shown in Fig. 7, predictions under LC have only 1.9% higher error on average than those under WRR.

VII. RELATED WORK

A large body of work has focused on the problem of selecting services that meet end user response time requirements [25]–[34]. Such selection approaches can be used to recommend Web services [25]–[31], automate the service composition [32] and runtime management [33], [35] of complex service chains to meet cost and response time requirements. In contrast to our work, these studies do not focus on predicting the impact of workload characteristics and system configurations on Web service performance.

Several studies have proposed novel instrumentation techniques to characterize the performance of Web services [36], [37]. Li et al. propose WebProphet to predict the page load time percentiles of popular services such as Google Maps [36]. Jiang et al. develop WebPerf to estimate a cloud-based Web service's response time distribution under various cloud configurations and runtime conditions [37]. Prospective differs from WebProphet and WebPerf in that it does not require specialized binary instrumentation to offer performance predictions.

Analytic queuing modeling techniques have been frequently used in literature for performance prediction [1], [38], [38]. With this approach, key workload characteristics and system configuration parameters are reflected into a Queuing Network Model (QNM) [39]. The QNM is often solved using Mean Value Analysis (MVA) [39]. While MVA has been used extensively, it is not intended for predicting response time percentiles.

Modeling techniques that are considerably more complex than MVA have been proposed for predicting response time percentiles [40]–[46]. The models involved in these techniques require considerable effort and expertise to construct. Furthermore, in general, techniques based on analytic queuing models often make simplifying assumptions. These assumptions impact accuracy and limit applicability [47].

Queuing modeling techniques require an operator to manually construct and validate a model. Typically, the process of model building and validation is iterative. This process is time consuming and requires experts in performance modeling. To address this problem, several researchers have proposed techniques to automatically build queuing models based on historical system response time measurements [48], [49]. However, the models resulting from these techniques are capable of only predicting mean response times.

Queuing simulations are used in situations where it is difficult to abstract a system using an analytic model. Spicuglia et al. propose an automated simulation approach that does not require explicit model construction to predict response time percentiles of cloud applications [22]. Similar to our work, the authors uses results from controlled load tests to drive a simulation that estimates transaction response times. The technique presented in their paper can only handle a single transaction type. As we show in Sec. V, not distinguishing between transaction types can cause significant prediction errors.

Machine learning provides an alternative to analytic and simulation based queuing modeling techniques. Techniques such as regression analysis [7], [8], [50], [51], Support Vector Machines (SVM) [15], [52], Artificial Neural Networks (ANN) [15], and Bayesian networks [53] have been used in literature for performance prediction.

Watson et al. use quantile regression to relate percentiles of response time metrics as a function of system resource utilizations [7]. Bodik *et al.* also use quantile regression to predict the 99^{th} percentile of response times of a Web application as a function of the mean transaction arrival rate [8]. Uttamchandani et al. propose a black box performance modeling technique for storage systems that exploits SVM regression [52]. These regression approaches require an expert to identify appropriate workload parameters, e.g., resource utilization and mean transaction arrival rate, as part of a featurization process. Furthermore, one has to specify an appropriate performance function, e.g., a kernel in the case of SVM, that captures how the response time percentile of interest relates to the workload parameters.

Several researchers have used regression approaches that do not require the specification of a performance function [50], [51]. For example, Wang et al. use the Classification And Regression Trees (CART) technique to predict the mean and 90^{th} percentile response times of a storage system [50]. The authors show that the accuracy of CART models is acceptable when the workload for which prediction is offered is similar to the workload used to train the model.

Kundu et al. explore the use of an ANN to predict the response time percentiles of a virtualized RUBiS Web service as a function of the hardware resources assigned to that service [15]. The authors show that splitting the feature space into different regions and building a sub-model for each region yields better accuracy than using a single global model. However, needs excessive training data. Furthermore, they state that careful tuning of the ANN, e.g., selecting an appropriate activation function, is required for obtaining good prediction accuracy.

In summary, applying machine learning requires operator intervention for tasks that can have a big impact on accuracy such as selecting an appropriate learning technique, featurization, performance function selection, and model structure tuning. This motivates the need for approaches such as Prospective that do not burden the operator with such tasks.

VIII. CONCLUSION AND FUTURE WORK

In this paper we address the key requirement of building a prediction tool that does not require a Web service operator to perform time consuming tasks such as model building, featurization, performance function selection, and model structure tuning. Prospective applies a novel technique based on CF to predict workload scenarios not directly observed in the historical data. The use of load vectors in combination with CF helps offer accurate predictions for a given arrival pattern and mix even if that pattern and mix have not been explicitly observed in the past. Specifically, Prospective modulates the progression of load vectors during a simulation to implicitly consider the impact of both the arrival pattern and the mix.

Results show that Prospective outperforms other baseline techniques. It is effective for a wide variety of workloads including those that exhibit highly variable transaction arrival patterns. The technique is robust and is able to improve its predictions continually based on new data. Prospective also reports the miss rate metric, which helps an operator assess the accuracy of a prediction. We show how the load balancing module of Prospective can be leveraged in system sizing exercises to determine the numbers and types of server instances needed for meeting a response time percentile requirement. In future, our research will be complemented along several dimensions. Future research will validate Prospective for other types of applications. We will also explore different ways of featurizing arrival pattern and mix and compare the accuracy of these alternatives.

REFERENCES

[1] D. Krishnamurthy, J. Rolia, and M. Xu, "WAM—The weighted average method for predicting the performance of systems with bursts of customer sessions," *IEEE Trans. Softw. Eng.*, vol. 37, no. 5, pp. 718–735, Sep./Oct. 2011.

[2] G. Casale, N. Mi, L. Cherkasova, and E. Smirni, "Dealing with burstiness in multi-tier applications: Models and their parameterization," *IEEE Trans. Softw. Eng.*, vol. 38, no. 5, pp. 1040–1053, Sep. 2012.

[3] N. R. Draper, H. Smith, and E. Pownell, *Applied Regression Analysis*, vol. 3. New York, NY, USA: Wiley, 1966.

[4] R. Koenker, *Quantile Regression*, no. 38. Cambridge, U.K.: Cambridge Univ. Press, 2005.

[5] F. Ricci, L. Rokach, and B. Shapira, *Introduction to Recommender Systems Handbook*. Boston, MA, USA: Springer, 2011.

[6] *RUBiS: Rice University Bidding System*. Accessed: Aug. 1, 2019. [Online]. Available: http://rubis.ow2.org/

[7] B. J. Watson, M. Marwah, D. Gmach, Y. Chen, M. Arlitt, and Z. Wang, "Probabilistic performance modeling of virtualized resource allocation," in *Proc. 7th Int. Conf. Auton. Comput.*, 2010, pp. 99–108.

[8] P. Bodık, C. Sutton, A. Fox, D. Patterson, and M. Jordan, "Response-time modeling for resource allocation and energy-informed SLAs," in *Proc. Workshop Stat. Learn. Techn. Solving Syst. Problems (MLSys)*, Whistler, BC, Canada, 2007, pp. 1–4.

[9] Y. Amannejad, D. Krishnamurthy, and B. Far, "Predicting Web service response time percentiles," in *Proc. Int. Conf. Netw. Service Manage. (CNSM)*, Oct./Nov. 2016, pp. 73–81.

[10] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, "Workload prediction using ARIMA model and its impact on cloud applications' QoS," *IEEE Trans. Cloud Comput.*, vol. 3, no. 4, pp. 449–458, Aug. 2015.

[11] V. Debusschere and S. Bacha, "Hourly server workload forecasting up to 168 hours ahead using seasonal ARIMA model," in *Proc. IEEE Int. Conf. Ind. Technol.*, Mar. 2012, pp. 1127–1131.

[12] J. S. Breese, D. Heckerman, and C. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," in *Proc. 14th Conf. Uncertainty Artif. Intell.*, 1998, pp. 43–52.

[13] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. 5th Berkeley Symp. Math. Statist. Probab.*, Oakland, CA, USA, vol. 1, 1967, pp. 281–297.

[14] D. Mosberger and T. Jin, "Httperf—A tool for measuring Web server performance," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 26, no. 3, pp. 31–37, 1998.

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

[16] A. Bahga and V. K. Madisetti, "Synthetic workload generation for cloud computing applications," *J. Softw. Eng. Appl.*, vol. 4, no. 7, p. 396, 2011.

[17] Z. Gong, X. Gu, and J. Wilkes, "PRESS: Predictive elastic resource scaling for cloud systems," in *Proc. Int. Conf. Netw. Service Manage. (CNSM)*, Oct. 2010, pp. 9–16.

[18] P. Padala, X. Zhu, Z. Wang, S. Singhal, and K. G. Shin, "Performance evaluation of virtualization technologies for server consolidation," HP Labs, Palo Alto, CA, USA, Tech. Rep. HPL-2007-59R1, 2007, p. 137.

[19] W. Zhang, Y. Shi, Y. Zheng, L. Liu, and L. Cui, "Resource and performance prediction at high utilization for N-tier cloud-based service systems," in *Proc. Australas. Comput. Sci. Week Multiconf.*, 2017, p. 43.

[20] V. Almeida, M. F. Arlitt, and J. Rolia, "Analyzing a Web-based system's performance measures at multiple time scales," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 30, pp. 3–9, Sep. 2002.

[21] N. D. Mickulicz, P. Narasimhan, and R. Gandhi, "To auto scale or not to auto scale," in *Proc. 10th Int. Conf. Auton. Comput. (ICAC)*, 2013, pp. 145–151. [Online]. Available: https://www.usenix.org/conference/icac13/technical-sessions/presentation/mickulicz

[22] S. Spicuglia, M. Björkqvist, L. Y. Chen, and W. Binder, "Catching the response time tail in the cloud," in *Proc. Int. Symp. Integr. Netw. Manage. (IM)*, May 2015, pp. 572–577.

[23] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and QoS-aware cluster management," *ACM SIGPLAN Notices*, vol. 49, no. 4, pp. 127–144, 2014.

[24] *HAProxy: The Reliable, High Performance TCP/HTTP Load Balancer*. [Online]. Available: http://www.haproxy.org/

[25] Z. Zheng, H. Ma, M. R. Lyu, and I. King, "QoS-aware Web service recommendation by collaborative filtering," *IEEE Trans. Services Comput.*, vol. 4, no. 2, pp. 140–152, Apr./Jun. 2011.

[26] Z. Zheng, H. Ma, M. R. Lyu, and I. King, "Collaborative Web service QoS prediction via neighborhood integrated matrix factorization," *IEEE Trans. Services Comput.*, vol. 6, no. 3, pp. 289–299, Jan. 2013.

[27] Y. Hu, Q. Peng, X. Hu, and R. Yang, "Time aware and data sparsity tolerant Web service recommendation based on improved collaborative filtering," *IEEE Trans. Services Comput.*, vol. 8, no. 5, pp. 782–794, Sep. 2015.

[28] Y. Ma, S. Wang, P. C. K. Hung, C.-H. Hsu, Q. Sun, and F. Yang, "A highly accurate prediction algorithm for unknown Web service QoS values," *IEEE Trans. Services Comput.*, vol. 9, no. 4, pp. 511–523, Jul./Aug. 2016.

[29] X. Wu, B. Cheng, and J. Chen, "Collaborative filtering service recommendation based on a novel similarity computation method," *IEEE Trans. Services Comput.*, vol. 10, no. 3, pp. 352–365, May 2017.

[30] D. Yu, M. Wu, and Y. Yin, "A combination approach to QoS prediction of Web services," in *Proc. Int. Conf. Service-Oriented Comput.* Berlin, Germany: Springer, 2012, pp. 99–106.

[31] R. Karim, C. Ding, and A. Miri, "End-to-end performance prediction for selecting cloud services solutions," in *Proc. IEEE Symp. Service-Oriented Syst. Eng. (SOSE)*, Mar. 2015, pp. 69–77.

[32] K. Xiong and H. Perros, "SLA-based service composition in enterprise computing," in *Proc. 16th Int. Workshop Qual. Service (IWQoS)*, Jun. 2008, pp. 30–39.

[33] V. Cardellini, E. Casalicchio, V. Grassi, and F. L. Presti, "Adaptive management of composite services under percentile-based service level agreements," in *Proc. Int. Conf. Service-Oriented Comput.* Berlin, Germany: Springer, 2010, pp. 381–395.

[34] S. Ding, Y. Li, D. Wu, Y. Zhang, and S. Yang, "Time-aware cloud service recommendation using similarity-enhanced collaborative filtering and ARIMA model," *Decis. Support Syst.*, vol. 107, pp. 103–115, Mar. 2018.

[35] G. H. Alférez and V. Pelechano, "Achieving autonomic Web service compositions with models at runtime," *Comput. Elect. Eng.*, vol. 63, pp. 332–352, Oct. 2017.

[36] Z. Li, M. Zhang, Z. Zhu, Y. Chen, A. G. Greenberg, and Y.-M. Wang, "WebProphet: Automating performance prediction for Web services," in *Proc. NSDI*, vol. 10, 2010, pp. 143–158.

[37] Y. Jiang, L. R. Sivalingam, S. Nath, and R. Govindan, "WebPerf: Evaluating what-if scenarios for cloud-hosted Web applications," in *Proc. ACM SIGCOMM*, 2016, pp. 258–271.

[38] J. Rolia, G. Casale, D. Krishnamurthy, S. Dawson, and S. Kraft, "Predictive modelling of SAP ERP applications: Challenges and solutions," in *Proc. 4th Int. ICST Conf. Perform. Eval. Methodol. Tools*, 2009, pp. 1–9.

[39] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance, Computer System Analysis Using Queuing Network Models*. Upper Saddle River, NJ, USA: Prentice-Hall, 1984.

[40] G. Casale, "Approximating passage time distributions in queueing models by Bayesian expansion," *Perform. Eval.*, vol. 67, no. 11, pp. 1076–1091, 2010.

[41] R. A. Hayden, A. Stefanek, and J. T. Bradley, "Fluid computation of passage-time distributions in large Markov models," *Theor. Comput. Sci.*, vol. 413, no. 1, pp. 106–141, 2012.

[42] J. F. Pérez and G. Casale, "Line: Evaluating software applications in unreliable environments," *IEEE Trans. Rel.*, vol. 66, no. 3, pp. 837–853, Feb. 2017.

[43] M. Grottke, V. Apte, K. S. Trivedi, and S. Woolet, "Response time distributions in networks of queues," in *Queueing Networks*. Boston, MA, USA: Springer, 2011, pp. 587–641. doi: 10.1007/978-1-4419-6472-4_14.

[44] M. Nguyen, Z. Li, F. Duan, H. Che, and H. Jiang, "The tail at scale: How to predict it?" in *Proc. 8th USENIX Workshop Hot Topics Cloud Comput. (HotCloud)*, 2016, pp. 1–6.

[45] U. Sharma, P. Shenoy, and D. F. Towsley, "Provisioning multi-tier cloud applications using statistical bounds on sojourn time," in *Proc. 9th Int. Conf. Auton. Comput.*, 2012, pp. 43–52.

[46] M. Björkqvist, N. Gautam, R. Birke, L. Y. Chen, and W. Binder, "Optimizing for tail sojourn times of cloud clusters," *IEEE Trans. Cloud Comput.*, vol. 6, no. 1, pp. 156–167, Aug. 2018.

[47] U. N. Bhat, "Sixty years of queueing theory," *Manage. Sci.*, vol. 15, no. 6, p. B-280, 1969.

[48] S. Kraft, S. Pacheco-Sanchez, G. Casale, and S. Dawson, "Estimating service resource consumption from response time measurements," in *Proc. 4th Int. ICST Conf. Perform. Eval. Methodol. Tools (VALUETOOLS)*, 2009, pp. 48:1–48:10. doi: 10.4108/ICST.VALUETOOLS2009.7526.

[49] T. Zheng, C. M. Woodside, and M. Litoiu, "Performance model estimation and tracking using optimal filters," *IEEE Trans. Softw. Eng.*, vol. 34, no. 3, pp. 391–406, May 2008.

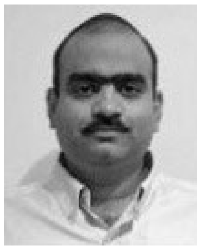
[50] M. Wang, K. Au, A. Ailamaki, A. Brockwell, C. Faloutsos, and G. R. Ganger, "Storage device performance prediction with CART models," in *Proc. Int. Symp. Modeling, Anal., Simulation Comput. Telecommun. Syst. (MASCOTS)*, Oct. 2004, pp. 588–595.

[51] M. Courtois and M. Woodside, "Using regression splines for software performance analysis," in *Proc. 2nd Int. Workshop Softw. Perform. (WOSP)*, 2000, pp. 105–114. [Online]. Available: http://doi.acm.org/10.1145/350391.350416

[52] S. Uttamchandani, L. Yin, G. A. Alvarez, J. Palmer, and G. Agha, "CHAMELEON: A self-evolving, fully-adaptive resource arbitrator for storage systems," in *Proc. USENIX Annu. Tech. Conf. (ATEC)*, 2005, pp. 1–6. [Online]. Available: http://dl.acm.org/citation.cfm?id=1247360.1247366

[53] I. Cohen, J. S. Chase, M. Goldszmidt, T. Kelly, and J. Symons, "Correlating instrumentation data to system states: A building block for automated diagnosis and control," in *Proc. OSDI*, vol. 4, 2004, p. 16.

YASAMAN AMANNEJAD received the Ph.D. degree in software engineering from the University of Calgary, Canada. She is currently an Assistant Professor with Mount Royal University. Her research interests include software engineering, data analytics, and machine learning.



DIWAKAR KRISHNAMURTHY is currently a Professor with the University of Calgary. His research interest includes the performance evaluation of software systems. He is also involved in research projects related to cloud computing, virtualization technologies, big data analytics, and healthcare simulation.

BEHROUZ FAR is currently a Professor with the University of Calgary. His research interests include engineering of intelligent, distributed and heterogeneous networked software systems, specifically design, and implementation and testing agent-based software systems. He is also involved with research projects related to distributed systems, intelligent transportation, cloud computing, and bio-informatics.