Predicting Web Service Response Time Percentiles

Yasaman Amannejad, Diwakar Krishnamurthy, Behrouz Far
Department of Electrical and Computer Engineering, University of Calgary, Canada
{yamannej,dkrishna,far}@ucalgary.ca

***Abstract*—Predicting Web service response time percentiles is often an important aspect of service level management exercises. Existing techniques can be very time consuming since they involve the manual construction of complex analytic or simulation models. To address this problem, we propose Prospective, a fully automated and data-driven approach for predicting Web service response time percentiles. Prospective relies on historical response time data collected from a Web service. Given a specification for workload expected at the Web service over a planning horizon, Prospective uses this 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 is able to predict various response time percentiles of interest with high accuracy for a wide variety of workloads.**

***Index terms*— Software performance engineering, Performance prediction, Response time percentile, Machine learning**

I. Introduction

Web services need to respond quickly to transactions issued by their users. Often, Web service operators have Service Level Agreements (SLA) with end users that specify 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 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.

Load tests combined with performance modeling has been widely used to support such SLA management exercises. With this approach, an operator conducts a load testing campaign to establish the performance behaviour of their service under some sample user workloads. 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, an analytic Queueing Network Model (QNM) or a discrete event simulation model may be used to predict the response time behaviour of the service under workloads and settings that were not explored during the testing phase. Measurements from the load tests are used to parametrize as well as validate the predictive model.

While performance models can be very effective, developing such models can be time consuming. QNM based techniques have been applied widely for Web services [1]–[3]. However, these models focus on predicting mean response times. Models capable of predicting percentiles [4]–[6] are significantly more complex and hence require a lot more effort to develop. Similar effort is also needed for constructing and validating queueing simulation models.

To address these problems, we propose Prospective, a fully automated and data-driven approach for predicting Web service response time percentiles. Prospective exploits historical performance trace data collected either from load testing campaigns or from a live system deployment. Given a specification for workload expected at the Web service over a planning horizon, Prospective uses this historical data to offer predictions for response time percentiles of interest over this period. At the core of Prospective is a lightweight simulator that interacts with a collaborative filtering (CF) [7] based response time prediction module to estimate the service's response time behaviour.

Prospective works as follows. We consider a Web service that supports many different types of transactions. Each transaction type refers to a specific function invoked by the end user, e.g. browse and buy. We define the load at any given time instant as a vector that records the numbers of concurrent transactions of each type at the service at that time instant. The central idea behind Prospective is the combined use of load and transaction type for predicting a transaction's response time. Prospective's simulator traverses an input synthetic workload trace of transactions, representing the expected workload, to produce initial estimates of the loads experienced by these transactions. Given the load estimated for a synthetic transaction and that transaction's type, the response time prediction module then estimates the transaction's response time. This prediction is carried out by analyzing the historical data to obtain past response times detected for the desired transaction type and load combination. Prospective uses a novel CF based method for scenarios where the historical data has not encountered this combination. Specifically, the CF method infers the response time for this previously unseen transaction type and load combination 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. We note that Prospective's simulator is fully automated and does not require the construction of a simulation model.

Results from a benchmark Web service testbed show that Prospective achieves high accuracy for a wide variety of workloads. In particular, it is 16 times more accurate than a technique that is agnostic to transaction types. In over 80% of the experimented scenarios, the 95^{th} and the 99^{th} percentile prediction errors are below 15%. Results also indicate that Prospective is robust and can continually improve its predictions by incorporating new response time data.

II. Related work

Analytic queueing modeling techniques are often used in SLA management. Such exercises [1]–[3] typically leverage Mean Value Analysis (MVA) [8], which is a very efficient technique for analytically estimating mean values of performance metrics. For example, Krishnamurthy *et al.* propose a technique called the Weighted Average Method (WAM), which combines a trace-driven simulator and MVA to predict the mean response time of a Web service under any given workload [2]. 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 [4]–[6]. Casale proposes a numeric approximation algorithm for estimating response time distributions of queueing models satisfying a certain set of assumptions [4]. Others have proposed techniques based on fluid analysis that work for certain types of queueing models [5], [6]. While queueing models are powerful, they require considerable effort and expertise to construct. Furthermore, such models often make simplifying assumptions that can impact accuracy [9].

Regression based approaches provide an alternative to queueing analysis. Several studies have used quantile regression to predict a response time percentile as a function of input parameters such as workload and system settings [10], [11]. Regression models can be time consuming to develop since they require the manual selection of an appropriate function that relates a response time percentile to input parameters.

Queueing simulations are used in situations where it is difficult to abstract a system using an analytic model. Unfortunately, the process of constructing a simulation model and validating it is often time consuming. This motivates the need for automated simulation approaches such as Prospective that do not require explicit model construction. Spicuglia *et al.* propose an automated simulation method to predict response time percentiles of cloud applications [12]. Similar to our work, the paper uses results from controlled load tests to drive a simulation that estimates transaction response times. However, unlike Prospective the technique presented in the paper can only handle a single transaction type. We show in Section V that not distinguishing between transaction types can cause significant errors.

III. 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 Section III-A. Sections III-B to III-D discuss in detail the implementation of Prospective.

A. Overview

Fig. 1 shows the architecture of Prospective. Prospective predicts response time percentiles of an application for a given workload. The target application is a Web service with n distinct types of transactions with each transaction type assigned a unique id in the range 1 to n. To use Prospective, a Web service operator should specify 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. $\mathbf{W}$ can be obtained by exploiting workload prediction techniques [13], [14] 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 the time at which the transaction arrives at the Web service and an id indicating the transaction's type.

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. As mentioned previously, $\mathbf{D}$ 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 historical 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. 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. These parameters are automatically calculated from a pre-processing phase described in Section III-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

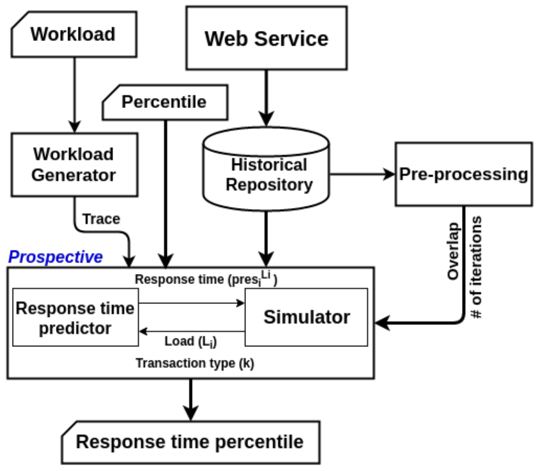

Fig. 1: High level architecture of Prospective

response times are then used to calculate the P^{th} percentile of response times. Furthermore, multiple independent simulation runs can be carried out to calculate a mean of such predictions along with a confidence interval for the mean. The Web service operator can use the predictions to take remedial decisions about system sizing and application deployment.

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. The simulator takes as input the trace $\mathbf{T}$, the desired percentile value P, and the input parameters derived from the pre-processing phase namely, the overlap parameter O and the number of iterations parameter I. We will defer the description of the pre-processing phase to Section III-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 of 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 $\mathbf{D}$. To calculate the response time of any given transaction i the simulator requires the load over the execution of that transaction $\mathbf{L_i}$. However, $\mathbf{L_i}$ is unknown when the simulator

Algorithm 1: Main Steps of Prospective

```
1  Input: $\mathbf{T}$, $P$, $O$, $I$,  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  i ← 0
9  foreach event in Timeline do
10     i ← i + 1
11     k ← $event$.transactionType
12     $arrival_i \leftarrow event$.arrivalTime
13     $\mathbf{L_i}$ ← calcLoad(i, $arrival_i$)
14     $pres_i^{\mathbf{L_i}}$ ← PredictResponseTime($i$, $k$, $\mathbf{L_i}$)
15     $departure_i \leftarrow arrival_i + pres_i^{\mathbf{L_i}}$
16     foreach iteration in (1, I) do
17         $\mathbf{L_i}$ ← calcLoad($arrival_i$, $departure_i$, $O$)
18         $pres_i^{\mathbf{L_i}}$ ← PredictResponseTime($i$, $k$, $\mathbf{L_i}$)
19         $departure_i \leftarrow arrival_i + pres_i^{\mathbf{L_i}}$
20     end
21     $\mathbf{RESPs}$.add($pres_i^{\mathbf{L_i}}$)
22 end
23 $res^P_{predicted}$ ← calculatePercentile($\mathbf{RESPs}$, $P$)
```

processes the arrival instant. As a result, the simulator has to guess $\mathbf{L_i}$ and iteratively refine that estimate.

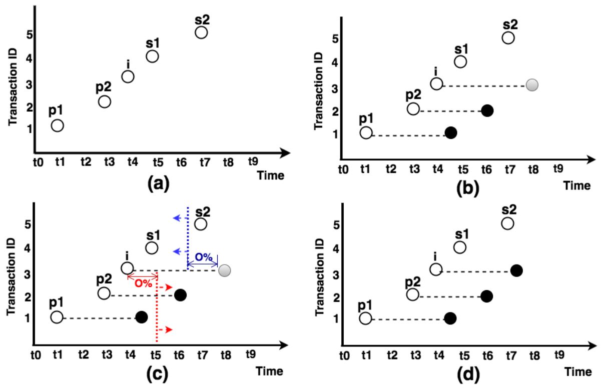

Fig. 2: Example of load and response time estimation

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. As shown in Fig. 1, 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. This is shown in lines 16-20 of Algorithm 1.

We first refine $\mathbf{L_i}$ using O. O allows the simulator to only include those transactions that have a significant overlap with the execution of i. For any given transaction i and any other 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. The pre-processing step 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 predict the measured response times of transactions in the historical trace data $\mathbf{T_{rep}}$ using the simulator. The simulator employs various O and I values and selects the values that most accurately predict the historically measured response time percentiles embodied by $\mathbf{T_{rep}}$.

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. The simulator also takes as input the percentiles for which predictions are desired. 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 and selects the value that yields the least error between the simulator predicted percentiles and the corresponding measured percentiles as calculated from $\mathbf{T_{rep}}$. The simulator then uses this value of O and searches similarly for the best values of I for each of the percentiles of interest. We note that multiple independent runs are carried out for each parameter setting and the average of prediction errors over these runs is used for selecting the final values.

D. Response time prediction module

Response time prediction involves a machine learning approach and is an extension of a method we proposed earlier [15]. The earlier work focused on predicting abnormal response times of cloud-based Web services. In this work, we modify that algorithm to predict response time percentiles.

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 $\mathbf{H_k^{L_i}}$ containing past transaction response times observed for this combination. Finally, a response time value is selected at random from this list and returned as $pres_i^{\mathbf{L_i}}$, the response time prediction for transaction i. This process is depicted in lines 2-4 of 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. While several similarity measures have been proposed in literature [16], [17], we use a formulation based on the commonly used Pearson correlation coefficient. The exact formulation of the similarity measure is not discussed further here due to space constraints but is similar to what we used in our earlier work [15]. We note that the similarity measures 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 0.0 and 1.0. It then identifies $\mathbf{N_k^{L_i}}$ as the set of *other* transaction types similar to k that have encountered the load $\mathbf{L_i}$ in D. For simplicity, we only consider the top 5 most similar transaction types in $\mathbf{N_k^{L_i}}$. For each similar transaction type j in $\mathbf{N_k^{L_i}}$, the module then constructs a list $\mathbf{H_j^{L_i}}$ 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 $pres_j^{\mathbf{L_i}}$. This process is depicted in lines 6-12 of Algorithm 2.

The final step of the CF method is to aggregate all the $pres_j^{\mathbf{L_i}}$ values pertaining to the similar transaction types into a single response time prediction $pres_i^{\mathbf{L_i}}$ for the transaction i. This aggregation uses the similarity measures between k and the similar transaction types recorded in $\mathbf{N_k^{L_i}}$. Specifically, we use the aggregation approach proposed by Breese *et al.* [18] implemented as Eq. 1 in our context. In Eq.1, res_k^* and res_j^* are the mean response times of transaction types k and j under all system loads recorded in D, respectively. $S[k,j]$ is the similarity between transaction types k and j. This formula ensures that response times drawn from highly similar transaction types contribute more to the final response time prediction than those from less similar transaction types. We note that any negative values of the weighted sum in the equation are set to 0.

Lines 14 to 16 of Algorithm 2 handle the case when no transactions in D have experienced the load $\mathbf{L_i}$. In this case, the module simply returns the predicted response time $pres_i^{\mathbf{L_i}}$ as the mean response time of all transactions of type k in the repository $\mathbf{D}$. We refer to the percentage likelihood of this scenario as the miss rate.

Algorithm 2: Response Time Prediction

```
1  Input: $\mathbf{D}$, $i$, $k$, $\mathbf{L_i}$,  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 $pres_j^{\mathbf{L_i}}$ (using Eq.1)
14     else
15         $pres_i^{\mathbf{L_i}} \leftarrow$ mean response time of $k$
16     end
17 end
```

$$pres_k^{\mathbf{L_i}} = res_k^* + \frac{\sum_{j \in \mathbf{N_k^{L_i}}} (pres_j^{\mathbf{L_i}} - res_j^*) \times S[k,j]}{\sum_{j \in \mathbf{N_k^{L_i}}} |S[k,j]|} \qquad (1)$$

IV. Experiment setup

A. Experiment Testbed

We use the RUBiS benchmark [19] as our Web service. RUBiS emulates the behavior of an auction server. The service supports 26 distinct types of transactions. We use the httperf tool [20] for generating workloads to RUBiS. RUBiS and httperf are installed on separate multicore servers. The servers are connected by an Ethernet switch that provides dedicated 1 Gbps connectivity. Finally, Prospective is implemented as a collection of Python scripts.

B. Experiment Factors

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

TABLE I: Experiment factors

Factor	Value
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) Erlang, (9) HypoExp, (10) HyperExp-CV2, (11) HyperExp-CV5
Mix	**Mix 40S-60D**, Mix 60S-40D, Mix 100S-0D
Estimation	**Prospective**, Avg-Tr, Avg-Load, Avg-Tr-Load

We study Prospective's ability to predict the behaviour of different types of transaction arrival 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 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. 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 40%, to 60%, and then decreases from 60%, to 40%, to 10%. In pattern 5, the mean per-core utilization decreases from 60% to 10% and then increases from 10% to 60%. 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 to 11 are used to study the impact of distributions with lower and higher variability than the exponential distribution. While patterns 8 and 9 have lower coefficients of variation (CV) than exponential, patterns 10 and 11 have 2 times and 5 times the CV of exponential, respectively. For patterns 8 to 11 we use the same mean inter arrival time as in the default Medium-Exp pattern to facilitate comparisons.

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 don't require dynamically generated content, and dynamic transactions (D).

Finally, several different response time estimation strategies are compared. The Avg-Tr approach ignores the dependency of transaction response times on load. It estimates the response time of a transaction by computing the average 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 average 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.* [12]. Finally, 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 average response time of all records with the desired load and transaction type. We note that a comparison with the regression approaches in literature [10], [11] is not presented since they are not designed for predicting the impact of changes in arrival distributions and mixes.

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 CPU utiliza-

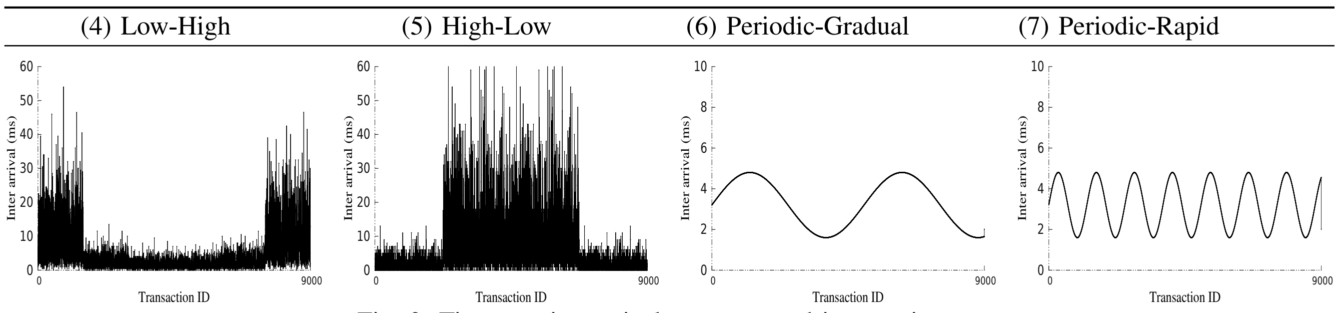

Fig. 3: Time varying arrival patterns used in experiments

tions. 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 of the distributions are selected to generate various levels of intensity ($10\% <$ per-core utilization $< 75\%$). For each utilization level, we repeat the experiment using 5 different samples drawn from the relevant exponential distribution.

Each run contains 9000 transactions. As mentioned previously, transaction traces are collected from the system and used for creating the repository. After creating the repository, the pre-processing steps discussed in Section III-C are carried out.

For each experiment, we use a sequence of inter arrival times sampled from one of the 11 arrival models. We submit to the Web service a trace of 9000 transactions conforming to this arrival pattern and exhibiting the desired mix. Response times are measured and recorded for transactions in the trace. Next, for the same trace, we run Prospective and record the predicted response time values. Each measurement and simulation experiment has 10 independent replications. The average and the 95% confidence interval for each response time percentile value is recorded for both the actual measurements and their predicted values.

D. *Evaluation Metrics*

To evaluate the accuracy of Prospective, we use the absolute relative error of the predicted response time percentile values as shown in Eq. 2.

$$err_p = |\frac{res^P_{predicted} - res^P_{measured}}{res^P_{measured}}| \times 100 \qquad (2)$$

In Eq. 2, 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

A. *General observations*

The pre-processing to deduce O and I is carried out once and took approximately 20 minutes for this study. The maximum duration of each test run to measure actual response times is about 5 minutes. A simulation run using Prospective is very fast. In our experiments, the simulation run times are in the range 45 seconds to 90 seconds depending on the choice of I. We also verify that the number of transactions simulated is large enough to yield consistent results across independent runs of a simulation.

B. *Pre-processing*

Fig. 4 shows the selection of the overlap parameter O. From the figure, values around 40% to 50% yield the lowest errors considering all percentiles of interest for this study. With lower values, transactions that have very low overlap are counted towards the load leading to overly pessimistic response time estimates. Values higher than 50% ignore the effects of some transactions that have quite a bit of overlap leading to optimistic estimates.

As mentioned previously, the tuning of I is specific for each percentile of interest. We omit presenting detailed results due to space constraints. We notice in general that the value of I needs to be greater than 1 for the tail percentiles, e.g., the 99^{th} percentile, for better accuracy.

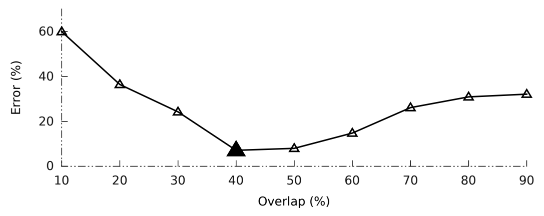

Fig. 4: Selecting the value of overlap parameter

C. *Effect of estimation approaches*

This experiment compares the average error over all percentiles of interest of the estimation approaches introduced in Section IV-B. AVG-TR, which ignores load, produces a large error of 39.8%. AVG-LOAD, which considers load but ignores transaction type, also does poorly with an error of 72.9%. These results establish the importance of incorporating both load and transaction type into the prediction process.

Among the two methods that consider both load and transaction type, Prospective, which has an error of 4.6%, outperforms AVG-TR-LOAD, which has an error of 12.5%. Recalling from Section IV-B, AVG-TR-LOAD uses average 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

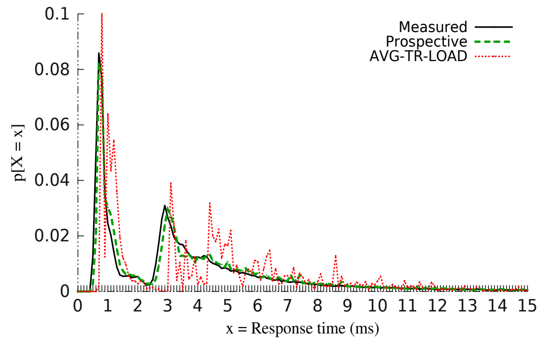

Fig. 5: Comparison of response time distributions

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. The Prospective method avoids this problem and follows the measured distribution more closely.

D. *Effect of arrival patterns*

Table II compares Prospective's accuracy for the different arrival patterns. Patterns 1 to 5 use either exponential or time varying exponential distributions. While the repository is constructed using exponential, the parameters of the repository's arrival model are different from those of patterns 1 to 5. Results show that Prospective yields very good accuracy with the maximum average error being 15.7%.

The worst errors are 14.6% and 26.8% for the 95^{th} and 99^{th} percentiles of the High-Exp pattern (pattern 3), respectively. This pattern causes the heaviest load on the system. We notice that at very high loads there is more diversity in the number of unique loads experienced 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. We find that defining a lower value of O is beneficial for high load scenarios since it improves Prospective's sensitivity to load. Enhancing the simulator to incorporate a load-dependent choice of O is deferred to future work.

Table II also shows that Prospective is very accurate for the Periodic-Gradual and Periodic-Rapid patterns, which have non-exponential, time varying arrivals. This indicates 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 seen by the simulator.

From Table II, Prospective is very accurate for the Erlang and HypoExp patterns, which both have lower CVs than the exponential distribution. It also performs well for HyperExp-CV2 whose CV is twice that of exponential. However, accuracy suffers for the extremely variable HyperExp-CV5 pattern, whose CV is five times that of exponential. In this scenario, there are periods where the system is subjected to extremely high loads where the per-core utilization exceeds 80%. Such extreme loads are not observed in the repository. The miss rate (defined in Section III-D) encountered by the simulator is about 31% leading to errors. We note that the miss rate information provided by Prospective can be exploited to infer the quality of predictions. Also, predictions can improve if the repository is expanded with load test or live system response time data pertaining to such extreme load scenarios.

From Table II, the errors are somewhat high for the 25^{th} percentile predictions. The measured 25^{th} percentiles are very small and as a result even small deviations in predictions result in large errors. We note that SLA compliance typically focuses on higher percentiles.

TABLE II: Percentile prediction errors(%)

	$\mathbf{25^{th}}$	$\mathbf{50^{th}}$	$\mathbf{75^{th}}$	$\mathbf{95^{th}}$	$\mathbf{99^{th}}$	**Avg**
Low-Exp	6.4	2.1	5.8	7.1	11.1	**6.5**
Medium-Exp	11.0	2.4	6.2	1.1	2.4	**4.6**
High-Exp	1.1	3.4	6.5	14.6	26.9	**15.7**
Low-High	4.7	3.1	9.6	4.1	4.1	**5.1**
High-Low	2.3	2.6	5.6	8.6	8.0	**5.4**
Periodic-Gradual	11.3	3.5	9.7	3.5	5.6	**6.7**
Periodic-Rapid	11.1	3.7	9.0	1.9	6.4	**6.4**
Erlang	13.5	3.2	9.1	3.2	5.3	**6.9**
HypoExp	13.3	3.4	9.2	1.3	5.8	**6.6**
HyperExp-CV2	2.0	4.0	1.5	15.8	9.7	**6.6**
HyperExp-CV5	30.8	11.9	48.5	66.9	72.9	**46.2**

Fig. 6 compares the measured and predicted confidence intervals for all the patterns. For a vast majority of cases that correspond to the utilization ranges of real life servers [21] the measured and predicted confidence intervals overlap or are very close to one another. The largest gaps between the measured and predicted confidence intervals are for cases such as HyperExp-CV5 where the system experiences extreme bursts that are not captured by the repository.

E. *Effect of transaction mix*

We submit each of the mixes shown in Table I with the same arrival pattern (Medium-Exp). Fig. 7 shows that they have very different response time behaviours. Table III shows the prediction errors. 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 16% and 4%, respectively.

TABLE III: Percentile errors(%) for different mixes

	$\mathbf{25^{th}}$	$\mathbf{50^{th}}$	$\mathbf{75^{th}}$	$\mathbf{95^{th}}$	$\mathbf{99^{th}}$	**Avg**
Mix 40S-60D	11.0	2.4	6.2	1.1	2.4	**4.6**
Mix 60S-40D	9.8	7.1	2.1	14.7	16.2	**9.9**
Mix 100S-0D	9.1	12.9	10.1	5.9	6.0	**8.8**

We explore $Mix60S-40D$ further to show the value of incorporating the CF method into Prospective. Disabling CF

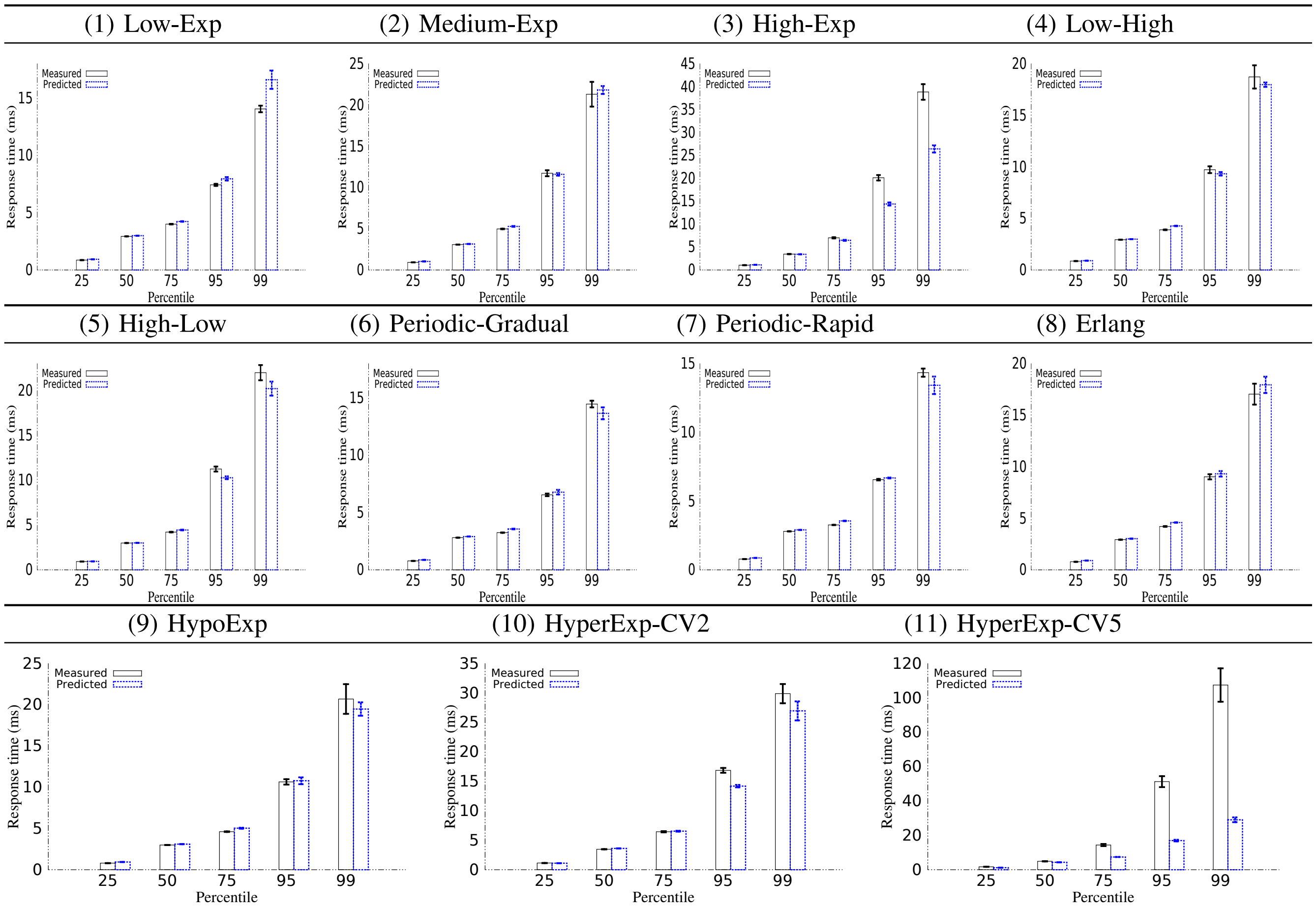

Fig. 6: Confidence intervals for measured and predicted response times

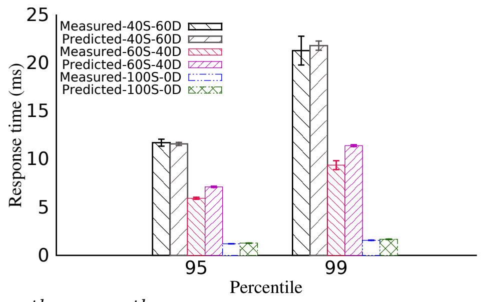

Fig. 7: 95th and 99th percentile confidence interval of mixes

increases the miss rate from 16% to 34%. As a result, the error for the 95th percentile and the average error increase from 14.7% and 9.9% to 19.9% and 13.7%, respectively.

The accuracy of Prospective depends on the quality of the historical repository. Prospective can continually expand its repository to include new measurements, e.g., data from the live system. We now study the impact of evolving the repository to include a new mix. We exposed the system to a workload that uses $Mix60S-40D$ and a time varying exponential inter arrival time distribution. Response time information from this workload is then integrated into the repository, which causes its size to almost double. 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 95^{th} percentile and the average error decrease from 14.7% and 9.9% to 5.8% and 5.5%, respectively. This shows that Prospective is robust and can learn to continually refine its predictions. Our future work will conduct a more extensive analysis to study the sensitivity of Prospective to the quality of the repository.

VI. Conclusions and Future Work

We tackle the challenging problem of predicting Web service response time percentiles. Existing approaches require manual model building and also rely on simplifying assumptions about the Web service that may not be valid. Prospective addresses these problems. Prospective uses historical data to predict response time percentiles for any given workload. A novel aspect of this work is the use of CF to improve prediction accuracies for load scenarios not directly observed in the historical data. Results show that Prospective is effective for a wide variety of the workloads.

Future work will refine Prospective to better handle extreme load and bursty scenarios. This can for example be achieved through a load dependent overlap factor or by clustering transaction types to reduce the number of unique load vectors. We will also study Prospective's use in dynamic environments where activities such as load balancing can be triggered in response to the current load.

References

[1] J. Rolia, G. Casale, D. Krishnamurthy, S. Dawson, and S. Kraft, "Predictive modelling of sap erp applications: challenges and solutions," in *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*, p. 9, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009.

[2] D. Krishnamurthy, J. Rolia, and M. Xu, "Wam—the weighted average method for predicting the performance of systems with bursts of customer sessions," *IEEE Transactions on Software Engineering*, vol. 37, no. 5, pp. 718–735, 2011.

[3] L. Y. Chen, D. Ansaloni, E. Smirni, A. Yokokawa, and W. Binder, "Achieving application-centric performance targets via consolidation on multicores: myth or reality?," in *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing*, pp. 37–48, ACM, 2012.

[4] G. Casale, "Approximating passage time distributions in queueing models by bayesian expansion," *Performance Evaluation*, vol. 67, no. 11, pp. 1076–1091, 2010.

[5] R. A. Hayden, A. Stefanek, and J. T. Bradley, "Fluid computation of passage-time distributions in large markov models," *Theoretical Computer Science*, vol. 413, no. 1, pp. 106–141, 2012.

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

[7] F. Ricci, L. Rokach, and B. Shapira, *Introduction to recommender systems handbook*. Springer, 2011.

[8] M. Reiser and S. S. Lavenberg, "Mean-value analysis of closed multichain queuing networks," *Journal of the ACM (JACM)*, vol. 27, no. 2, pp. 313–322, 1980.

[9] U. N. Bhat, "Sixty years of queueing theory," *Management Science*, vol. 15, no. 6, pp. B–280, 1969.

[10] P. Bodık, C. Sutton, A. Fox, D. Patterson, and M. Jordan, "Response-time modeling for resource allocation and energy-informed slas," in *Proc. Workshop on Statistical Learning Techniques for Solving Systems Problems (MLSys' 07), Whistler, BC, Canada*, 2007.

[11] B. J. Watson, M. Marwah, D. Gmach, Y. Chen, M. Arlitt, and Z. Wang, "Probabilistic performance modeling of virtualized resource allocation," in *Proceedings of the 7th international conference on Autonomic computing*, pp. 99–108, ACM, 2010.

[12] S. Spicuglia, M. Bjorkqvist, L. Y. Chen, and W. Binder, "Catching the response time tail in the cloud," in *IFIP/IEEE International Symposium on Integrated Network Management (IM2015)*, pp. 572–577, IEEE, 2015.

[13] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, "Workload prediction using arima model and its impact on cloud applications' qos," *IEEE Transactions on Cloud Computing*, vol. 3, no. 4, pp. 449–458, 2015.

[14] V. Debusschere, S. Bacha, *et al.*, "Hourly server workload forecasting up to 168 hours ahead using seasonal arima model," in *2012 IEEE International Conference on Industrial Technology*, 2012.

[15] Y. Amannejad, D. Krishnamurthy, and B. Far, "Managing performance interference in cloud-based web services," *IEEE Transactions on Network and Service Management*, vol. 12, no. 3, pp. 320–333, 2015.

[16] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 6, pp. 734–749, 2005.

[17] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez, "Recommender systems survey," *Knowledge-Based Systems*, vol. 46, pp. 109–132, 2013.

[18] J. S. Breese, D. Heckerman, and C. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," in *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pp. 43–52, Morgan Kaufmann Publishers Inc., 1998.

[19] "Rubis rice university bidding system." [Online]. Available: http://rubis.ow2.org/.

[20] D. Mosberger and T. Jin, "httperf—a tool for measuring web server performance," *ACM SIGMETRICS Performance Evaluation Review*, vol. 26, no. 3, pp. 31–37, 1998.

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