A Study of Web Services Performance Prediction: A Client’s Perspective

Leslie Cheung, Leana Golubchik, Fei Sha
Computer Science Department, University of Southern California, Los Angeles, California 90089
Email: {lccheung,leana,feisha}@usc.edu

Abstract—The Web service (WS) paradigm is an emerging approach to building Web applications, in which software designers typically build new WSs by leveraging existing, third-party WSs. Understanding performance characteristics of third-party WSs is critical to the overall system performance. Although such performance evaluation can be done through testing of third-party WSs, it is quite an expensive process. This is especially the case when testing at high workloads, because performance degradations are likely to occur, which may render the WS under testing unusable during the tests’ duration. Avoiding testing at high workloads by applying standard extrapolation approaches from data collected at low workloads (e.g., using regression analysis) results in a lack of accuracy. To address this challenge, in this paper, we propose a framework that utilizes the benefits of queueing models to guide the extrapolation process, while achieving accuracy in both regimes – low and high workloads. Our extensive experiments show that our approach gives accurate results as compared to standard techniques (i.e., use of regression analysis alone).

Keywords—Web services; performance prediction; queueing theory; regression;

I. INTRODUCTION

Web services (WSs) are emerging as a paradigm to build complex distributed systems across different organizations. The WS architecture allows loosely-coupled services, potentially implemented on different platforms, to communicate via the Internet using standard protocols. Evaluating the performance of a third-party WS is therefore important in building high quality WS-based systems. This is because in a WS-based system, the performance a user experiences is highly dependent on the performance of other WSs, as well as the performance of the underlying network. For example, if a WS has limited resources and clients experience long queueing times, it not only affects that WS’s reputation, but also the reputation of its client WSs.

However, existing work on WS performance evaluation has not addressed the problem of performance estimation from a client’s perspective. Some existing work has focused on evaluating WSs from a system administrator’s or designer’s perspective. For example, [1] assumes the system’s architecture is known and models a WS-based system using a multi-tiered architecture. Other works assume the system’s architecture (e.g., how the third-party WSs are connected [2]), and/or the system’s parameters (e.g., the amount of I/O time needed to complete a service [3]) are known. We argue that such assumptions are (typically) not reasonable: it is not clear how such information can be obtained by a client, whereas the service providers may be reluctant to provide it. While Business Process Execution Language (BPEL) [4] may provide information about what other WSs may be invoked, it is unavailable in many cases, because WSs are not required to provide BPEL specifications. Even if it is available, the information about the internal structure of the WS would still be missing, which is essential in performance prediction. This information is needed in the approaches mentioned above, but it is unclear how such information can be obtained from a client’s perspective.

In this paper, we focus on evaluating the performance of third-party WSs from a client’s perspective; specifically, we focus on the average response time estimation. Our major challenge is the lack of information about the WS being tested. This includes (1) the structure of the WS: as a client, we do not know how often the WS being tested requests services from other WSs; and (2) the parameters of each WS that provides service to complete a client’s request.

Our proposed approach makes use of data collected from performance testing [5], which involves making requests to the WS of interest and collecting corresponding performance data. Once such data is collected, a typical approach is to apply regression analysis [6] to this data for response time prediction. In applying such techniques one typically encounters two types of problems (1) interpolation (i.e., predicting response time within the ranges of the parameters used during performance testing), and (2) extrapolation (i.e., predicting response time outside of the range of the parameters). It is often the case that extrapolation tends to be less reliable and in some cases, is advised not to be undertaken [7]. (Our experiments have confirmed this as well.) We also note that interpolation-based performance testing can be quite an expensive process, particularly at high workloads – performance degradations are likely to occur, which may render the WS under testing unusable during the tests’ duration. Thus, avoiding testing at high workloads and instead extrapolating from data collected at low workloads (e.g., using regression analysis) is highly desirable.

To address this challenge, we propose a framework that leverages queueing models to guide the extrapolation process, while maintaining accuracy. Queueing models have been widely used in performance modeling of computer systems. As shown by our experimental results, under simplifying assumptions, queueing networks (QNs) are adept at predicting performance for systems under high workloads. Thus, it is desirable to combine these two approaches: regression techniques which...
Thus, Step 2 (Section II-B) involves extrapolating response time using data collected in Step 1. We propose to use queueing models for response time prediction (Section II-B2), which, however, may give less accurate interpolation results (Section II-B3). This motivates us to derive a hybrid approach that combines the more accurate interpolation results when using standard regression approaches, with the more accurate extrapolation results when using queueing models (Section II-B4).

A. Step 1: Performance Testing

Performance testing has been used in evaluating software performance to ensure the system performs as expected [5]. The goal of performance testing is to understand the system’s properties, such as system throughput and response time, given a controlled workload.

Performance testing may assume an open model, in which clients arrive to the system at a pre-specified arrival rate \( \hat{\lambda}_E \), and leave the system once the request has been served. It may also assume a closed model, in which the number of clients is fixed. In either case, we are interested in observing the response time when we vary the arrival rate in an open model, or when we vary the number of clients in a closed model.

In the remainder of the paper, we assume the use of an open model, and generate arrivals accordingly to a Poisson process. (We note that, as a client of a third-party WS, we can control the arrival process.) Specifically, we generate \( D^j \) requests at rate \( \hat{\lambda}_E^j \), and measure the response time to each request \( k \), \( T^{j,k} \). We can then compute the average response time, \( \bar{T}^j = \frac{1}{D^j} \sum_1^{D^j} T^{j,k} \). We repeat the test at different values of \( \hat{\lambda}_E^j \), and compute the corresponding \( \bar{T}^j \).

A shortcoming of performance testing is the assumption that the system does not change over the duration of the test. This includes the WS being tested, any other third-party WSs involved, as well as network conditions. In real-world applications, this may not be the case. For example, making a large number of requests to a WS may be perceived as an attack. Thus, administrators may block the testing traffic, and, as a result, we would not be able to gather performance data. This again motivates the need to limit performance testing, particularly at higher workloads, and devise approaches for accurate extrapolation.

B. Step 2: Regression Analysis

The goal of regression analysis is to model and estimate the input-output relationship between random variables based on observed data, and then use the model for prediction. In our context, we apply regression analysis to model the relationship between the arrival rate and the WS response time, and predict WS response times at arrival rates that are not sampled during performance testing. The stage of modeling is often referred as “training”. We often need to assess the effectiveness of a trained model before we deploy it to real-world environments.

1We consider WSs with one request type in this paper. Incorporating multiple request types and other workload parameters is part of our ongoing work.
to make prediction. The stage of assessment is often referred as “testing” (or “evaluation” to avoid being confused with performance testing). The assessment is accomplished by comparing the model’s prediction on data with known arrival rates and responses times. However, such data should have no overlap with the data used in the training stage so that the model is not over-optimistic.

As noted earlier, we differentiate two different types of predictions: interpolation when the arrival dates are within the range of those being collected during performance testing, and extrapolation when the arrival dates are outside the range.

Statistical models for regression analysis can be broadly classified into parametric and nonparametric:

**Parametric regression:** In parametric regression, we specify a regression function with unknown parameters to capture the relationship between the arrival rate and the response time. One can leverage prior knowledge about the relationships among variables to determine a suitable regression function. An example regression function is an $n^{th}$-degree polynomial, i.e., $T(\lambda_E, \alpha) = \sum_{i=0}^{n} \alpha_i (\lambda_E)^i$, where $\lambda_E$ is the average customer arrival rate to the WS, and $\alpha$ is the unknown parameter vector, representing the coefficients of the polynomial. We estimate it using performance testing data.

More specifically, given a regression function and data from performance testing (pairs of values of $\lambda_E$ and $T$), we would like to find $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ such that the mean squared error between the measured response time and the model’s prediction is minimized. This problem can be formulated as the following optimization problem:

$$\alpha^* = \arg\min_{\alpha} \sum_j (T_j - T(\lambda_E^j, \alpha))^2$$

where $T(\lambda_E^j, \alpha)$ is the predicted response time when the external arrival rate is $\lambda_E^j$. This problem can be solved using standard optimization techniques [6].

Once we have estimated the unknown parameters, we predict response time, by plugging in the arrival rate of interest, $\lambda_E$, and parameters estimated from regression, $\alpha^*$. In this work, the GP encodes similarity among data (i.e., pairs of arrival rates and response times) with kernel functions and makes predictions by combining (nonlinear) response times from observed data. Intuitively, a closer training data at $\lambda_E$ to $\lambda_E$ contributes more to the final prediction on $\lambda_E$. Our experiments use the so-called “neural network tanh kernel” as it performs the best when compared to a few other alternatives.

1) A shortcoming of standard regression analysis: To illustrate a shortcoming of applying standard regression analysis for WS performance estimation, we show how poorly these approaches extrapolate. A more comprehensive validation is presented in Section III.

In this experiment, we use extrapolation error as our metric. We collected performance testing data by varying the arrival rates, until the system has been saturated (i.e., when the system has started returning errors because of resource saturation). We then divide the data into two sets: the training set and the validation set. Data in the training set, consisting of the data points in the bottom 60% of the arrival rates sampled, was supplied to the regression algorithm. Then, we compute the extrapolation error by comparing the predicted response time and data in the validation set, which corresponds to the data points in the upper 40% of the arrival rates sampled.

Here, we show the results on the Java Adventure Builder (AB) application [11]. This simple travel agent WS is provided by Sun to demonstrate the development and deployment of a WS. It is an atomic WS (i.e., one that does not make requests to other WSs), that makes requests to a local database server. Our system has 54 customers and 1,022 bookings.

The extrapolation results are depicted in Figure 2. Data in the training set and the validation set are depicted as circles and squares, respectively. We depict results based on an $8^{th}$-degree polynomial in Figure 2(a) as, in this experiment, the results of using an $8^{th}$-degree polynomial were more accurate than those using polynomials of other degrees. We observe, from Figure 2, that standard regression techniques are unable to predict response time when the arrival rates are outside of the data used as input to regression analysis. Specifically, all four approaches we studied predict the response time to
remain flat when the arrival rate increases beyond the sampled arrival rates, instead of increasing rapidly as the system nears saturation. Indeed, the fact that standard regression approaches may give poor extrapolation results is a well-known problem in the regression literature.

2) A Queueing Model-Based Framework: To address the shortcoming that standard regression approaches tend to perform poorly at extrapolation, we propose a queueing network-based framework to estimate the response time of black-box WSs. More specifically, we use queueing models to derive a function that describes the relationship between arrival rates and response time; this function is then used as the regression function in parametric regression for response time prediction. The challenge is, however, that we do not know the structure of the WS being tested. For example, we do not know if it is deployed on a server, using a three-tier architecture as in [1], or if it makes use of other WSs. In the absence of structural information, we approach this problem by using a suite of queueing models, and as shown in Section III, this provides us with insight about the performance of the WS. For example, we can determine the stability conditions of the WS using the most pessimistic model.

In presenting our queueing model-based framework, we first discuss single-queue models, followed by queueing network models. We also give several instantiations of the queueing models we have considered in our evaluation in Section III.

Single-Queue Models: A single-queue model is characterized by: (1) the arrival process, which describes the workload characteristics; (2) the service time distribution, which describes the characteristics of the servers; and (3) the number of servers, which describes the degree of concurrency. As a client to a WS, we can control the arrival process by adjusting the performance testing parameters. Parameters related to the service time distribution are estimated using regression, while the number of servers is determined by the system modelers. Given this information, we can derive the average response time as a function of arrival rate and other model parameters, and estimate model parameters by applying standard regression analysis using data collected from performance testing.

Since information about the WS being tested is limited, in general, it is challenging to determine the number of servers and the service time distribution. However, in our validation in Section III, we show that even with simple queueing models (as detailed below) one can gain valuable insight into the WS being tested. For instance, we can determine the stability conditions of the WS, which are useful in, for instance, determining how much workload one should send to that WS.

M/M/1 Model: As an example, let us consider the M/M/1 model (i.e., with a Poisson arrival process and exponential service time distribution). The corresponding average response time is then $T(\lambda_E, \mu) = 1/(\mu - \lambda_E)$ [12], where $\lambda_E$ and $\mu$ are the average customer arrival and service rates, respectively. We apply regression analysis to estimate $\mu$ using performance testing data. In applying regression analysis, we need to specify constraints to ensure that the resulting system is stable, i.e., in the case of the M/M/1 model, that $\mu > \lambda_E$.

We apply regression analysis to predict the response time of the AB WS using the M/M/1 model, with results depicted in Figure 3(a). Even though the M/M/1 model can predict the rapid increase in response time (beyond a certain load), it does so pessimistically in this case, i.e., this increase occurs much sooner than in the actual system. One reason for this is that the exponential service time distribution assumption is unlikely to hold in a real system. Thus, the M/M/1 model illustrates the basic idea and motivates the use of more complex models, as we do next.

M/G/1 Model: The M/G/1 model allows a general service time distribution that is characterized by its mean and variance. The corresponding average response time is [12]:

$$T(\lambda_E, \mu, \sigma) = \frac{1}{\mu} + \frac{\lambda_E \sigma^2}{2(1 - \frac{\lambda_E}{\mu})}$$

where $\sigma^2$ is the variance of the service time distribution. We apply regression analysis to estimate both $\mu$ and $\sigma$ using performance testing data.

The results of predicting response time of the AB WS using M/G/1 model is depicted in Figure 3(a). The M/G/1 model is more accurate than the M/M/1 model, due to the more general model of the service time distribution.

M/M/m Model: The M/M/m model relaxes the single-server assumption of the M/M/1 model, i.e., we have a single queue with $m$ servers. The corresponding average response time is [12]:

$$T(\lambda_E, \mu, m) = P_Q \frac{\rho}{\lambda_E (1 - \rho)} + \frac{1}{\mu}$$

where $\rho = \lambda_E / (m\mu)$ and $P_Q$, the probability of queueing, is given by [12]:

$$P_Q = \frac{(m\rho)^m}{m!} \frac{p_0}{1 - \rho}$$

$$p_0 = \left( \sum_{k=0}^{m-1} \frac{(m\rho)^k}{k!} + \frac{(m\rho)^m}{m! (1 - \rho)} \right)^{-1}$$

We applied the M/M/m model on the AB WS. However, since the AB WS is a single-queue system, the M/M/m model degenerated into an M/M/1 model and hence both models gave the same results. Thus, to illustrate the use of multi-server queueing models, we present results using the TPC-App benchmark [13] we deployed, which we refer to as TPC WS in the remainder of the paper. This benchmark emulates a bookstore WS environment, in which customers can create
an account, search for books, place an order, and retrieve the status of an order. The WS makes use of several internal WSs: an Order WS, an Item WS, and a Customer WS. Each of the three internal WSs runs on a separate physical machine, and queries a local database. Our system has 100 customers, 500 books, and 30,000 order records.

The extrapolation results\(^4\) using the TPC WS are depicted in Figure 3(b). While the results based on the M/M/m model are more accurate than those based on the M/M/1 model, they are still pessimistic. One reason is that the TPC WS was deployed on four machines, and each machine has its own queue. Therefore, a single-queue, multi-server model, such as the M/M/m model, may not be as accurate as a model with multiple queues, which motivates consideration of queueing network models.

**Queueing Network Models:** To simplify our discussion, we assume an open\(^3\) QN of M/M/1 queues with Markovian routing [14]. We also assume there is only one class of customers: the arrival and service processes for all customers are the same. In such a QN, a queue may, e.g., represent an internal server (such as a Web server or a database server), another WS, or the underlying network.\(^4\) With these assumptions, our QN is a product-form network [14].\(^5\)

The first piece of information needed in addition to single-queue models is the *number of queues*, which is estimated by the system modelers. This corresponds to the number of physical servers (e.g., database and application servers) that serve a client’s request. One approach is to try different number of queues, and determine which gives the most accurate results. We use a two-queue QN to model the TPC WS in Section III, as it generates the most accurate results among QNs with different number of queues. For each queue, in addition to the parameters specified in single-queue models, we need to determine its *visit ratio*, using regression (see below).

We now define a QN model more formally. Let \(K\) be the number of queues, \(p_{i,j}\) be the probability of going to queue \(j\) upon leaving queue \(i\); \(p_{E,i}\) be the probability that an external arrival goes to queue \(i\), and \(p_{E}\) be the probability that a customer leaves the system upon leaving queue \(i\), where \(\sum_{j} p_{i,j} + p_{E,i} = 1\). Note that in a WS, a customer always arrives at the WS being tested (e.g., a customer cannot send requests directly to an internal database server). If we assume Queue 1 is the WS being tested, then \(p_{E,1} = 1\), and \(p_{E,i} = 0\) for all \(i \neq 1\).

The visit ratio of queue \(i\) is given by \(v_i = p_{E,i} + \sum_j v_j p_{i,j}\) [12], where the total arrival rate at queue \(i\) is \(\lambda_i = \lambda_E v_i\). Given \(\lambda_i\) for each M/M/1 queue \(i\), the average number of customers in queue \(i\), \(N_i\), is [12]:

\[
N_i = \frac{\lambda_i}{\mu_i - \lambda_i} = \frac{\lambda_E v_i}{\mu_i - \lambda_E v_i} \quad (6)
\]

Since the QN is product-form, the joint probability of having \(n_i\) customers in queue \(i\), \(1 \leq i \leq K\), is \(P(n_1, n_2, \ldots, n_K) = \prod_i P(n_i)\), where \(P(n_i)\) is the probability that there are \(n_i\) customers in queue \(i\) [14]. Here, the average number of customers in the system, \(N\), is

\[
N = \sum_i N_i = \sum_i \frac{\lambda_E v_i}{\mu_i - \lambda_E v_i} \quad (7)
\]

Thus, using Little’s result [12], the average response time is

\[
T = \frac{N}{\lambda_E} = \sum_i \frac{\lambda_E v_i}{\mu_i - \lambda_E v_i} = \sum_i \frac{v_i}{\mu_i - \lambda_E v_i} \quad (8)
\]

We can simplify our process as follows: instead of estimating the entire routing matrix (i.e., the \(p_{i,j}\)’s) and compute the visit ratios, we choose to estimate the visit ratio \(v_i\)’s directly. Furthermore, if we multiply Eq. (8) by \((1/v_i)/(1/v_i)\), we obtain:

\[
T = \sum_i \frac{v_i}{\mu_i - \lambda_E v_i} \times \frac{1/v_i}{1/v_i} = \sum_i \frac{1}{\alpha_i - \lambda_E} \quad (9)
\]

where \(\alpha_i = \mu_i/v_i\). Rewriting Eq. (8) as Eq. (9) allows us to simplify the response time estimation process by using regression analysis to estimate \(\mu_i/v_i\) directly, instead of their individual values.

We apply this QN model to the data collected from the TPC WS, with the results depicted in Figure 3(b). We observe that the QN model is more accurate than the M/M/1 and M/M/m models, because of its more accurate description of the TPC WS’s structure. This QN model, however, is too optimistic when the arrival rate is high. This suggests that we should use a suite of queueing models to understand the behavior of a WS, rather than a single model.

3) A shortcoming of queueing models: While the extrapolation results using queueing models are better than those of standard regression approaches, their interpolation results are not as good. This can be explained as follows. System response time increases rapidly when the system is close to being saturated, and hence the slope of the response time function is very steep when \(\lambda_E\) is high. This property causes the regression algorithm to overemphasize fitting data at high workload intensity, because a slight error in the estimated parameters results in very large errors in these data points. Given that the queueing models usually have few parameters to fit (e.g., the M/M/1 model only has one parameter), the regression algorithm cannot adjust the parameters to fit data at low workload intensity, and hence the response time estimates at low workload intensity are not as good as queueing models. On the other hand, standard regression approaches are usually more flexible in fitting data at both low and high workload intensities, and hence are able to produce more accurate interpolation results.

As an illustrative example, consider the TPC WS we used earlier. We sampled the arrival rates of TPC WS at \(0.5 \leq \lambda_E \leq 2.3\), and provided every other data point collected during performance testing as training data, and the remaining data points were used to compute interpolation errors. We show
results using the QN model and NN, because these results are most accurate among queueing models and standard regression approaches, respectively. Note that we present the results here as a motivation for the hybrid approach in Section II-B4; we will present a more comprehensive validation with other aforementioned models and WSs in Section III. The results are depicted in Table I.

We can see that the interpolation errors of QN are higher than those of NN, e.g., when $\lambda_e = 0.7$, the error of NN is 0.01292 (or 1.136%), while the error of QN is 0.10358 (or 9.11%). These results have motivated us to derive a hybrid approach, that takes advantage of the low interpolation errors of standard regression approaches at low workload intensity, and more accurate extrapolation results of the queueing models at high workload intensity.

4) A hybrid approach: How do we take advantage of the better interpolation accuracy of standard regression approaches at low workload intensity, and the better extrapolation results of the queueing models at high workload intensity? Figure 4 illustrates our proposed hybrid approach. Here, let $\lambda_e^{max}$ be the highest arrival rate sampled during performance testing.

The main idea is to first fit queueing models with performance testing data at the sampled arrival rates ($\lambda_e \leq \lambda_e^{max}$, Step 2a), and then generate new performance data points at higher arrival rates ($\lambda_e > \lambda_e^{max}$) using the fitted queueing model (Step 2b). In the final Step 2c, we augment the real performance testing data with the QN-predicted performance testing data. We then apply standard regression approaches to the augmented data to build a new prediction model which fuses knowledge from the queueing model.

We hypothesize that the resulting model has low interpolation errors at low workload intensity, as compared to using queueing models alone, while being able to extrapolate response time at high workload intensity, as compared to using standard regression approaches alone. The following example supports the hypothesis. More detailed validation results are given in the next section.

As an illustrative example, consider applying this hybrid approach to the TPC WS. Since the interpolation results using NN are most accurate among standard approaches, and the extrapolation results using QN are most accurate among queueing models (Figure 3(b)), we use QN in Steps 2a and 2b, and NN in Step 2c in the results to be presented here and in Section III. In the remainder of the paper, we refer to this approach as $QN^3$.

**Step 2a:** We fit data collected during performance testing at the sampled arrival rates, depicted as circles in Figure 5, using a QN with 2 queues (introduced in Section II-B2). In this example, the parameters of the QN, obtained using regression analysis by supplying Eq. (9) as the regression function, are $\alpha_1 = 2.5908$ and $\alpha_2 = 2.5912$.

**Step 2b:** The next step is to generate new data points using this QN model by plugging in $\lambda_e > \lambda_e^{max}, \alpha_1$ and $\alpha_2$ into Eq. (9). In our example, the new data points are depicted as triangles in Figure 5.

**Step 2c:** Finally, we take the data from Steps 2a and 2b as inputs to a standard regression approach (in our example NN), with results depicted in Table II and Figure 5.

The results in Table II indicate that the interpolation errors of $QN^3$ are comparable to using NN alone and are lower than using QN alone. At the same time, the extrapolation errors of $QN^3$ are very close to using QN, and are lower than using NN alone (which produces poor extrapolation results). These results illustrate that $QN^3$ is more accurate than using either QN or NN alone. A more comprehensive validation is presented next.

### III. Validation

We perform an extensive evaluation and comparison of the approaches described in Section II, i.e., standard regression techniques, queueing models (QN, M/M/1, M/M/m, and M/G/1), and $QN^3$. Concretely, we analyzed four WSs with different configurations. We predict response times using above stated approaches and report their errors.

The four WSs which have been analyzed are the AB WS and the TPC WS that we deployed in a controlled environment (both described earlier), and the Weather WS [15] and the
Geocoding WS [16] that are “live”. Analysis on other “live” WSs and synthetic WSs yielded similar conclusions and are thus omitted for brevity.

We report RMSE – (squared) root of mean squared errors – a commonly used evaluation metric in regression analysis. The errors are defined as the differences between the predicted values and the measurements (ground-truth). For each WS, we sent 10000 requests at a fixed arrival rate according to a Poisson process, and computed the average response time. This process was repeated at 9 - 11 different arrival rate values. The data was then split into two sets (with details given below): data in the training set was supplied as input to each approach, and we computed the approach’s RMSE using its predictions and data in the validation set.

In what follows, we report first results on interpolation. In this setting, parameters of our models are estimated on training data (i.e., different arrival rates) whose value ranges are the same as validation data. Then, we report results on extrapolation, where the ranges of training data and validation data are disjoint. Our evaluation results show that, while other techniques perform well on either interpolation or extrapolation, $QN^3$ performs the best in both cases.

### A. Interpolation Errors

In this set of experiments, we choose an odd number of data points. An example is the data in the first two columns in Table II. We sort them according to the corresponding arrival rates and then select the data points, alternating between the training and the validation data sets.

Note that since the first and the last data points are always selected for training data, we are guaranteed that the arrival rates in the validation data are always within the range of the rates in the training data.

In Figure 6, we illustrate the fitted regression curves (draw in blue) along with the training data (using circles) and the validation data (using squares). In Table III, we report the errors of the TPC WS at different arrival rates, and in Table IV, we report the average interpolation errors across all arrival rates for each of the four WSs – best performing techniques are shown in bold. Detailed results for the other three WSs have similar patterns to those reported in Table III and are thus omitted.

From Table III, we observe that the M/M/1 and M/M/m models give higher interpolation errors than the QN model in the TPC WS. This illustrates that the QN model is a better description of the TPC WS than the M/M/1 and M/M/m models, because the TPC WS was deployed on four physical servers, and hence the QN model, which assumes a multi-queue system, describes the TPC WS more accurately than the single-queue systems.

From Table IV, we observe that while applying the QN model to the TPC, Geocoding, and Weather WSs had lower interpolation errors, it had higher interpolation errors than the $M/G/1$ model when it was applied to the AB WS. This is because (1) the AB WS was deployed on a single machine, in which the $M/G/1$ model had accurately described as a single-queue system; and (2) the QN model uses exponential service times, which is unlikely the case in our performance testing. The $M/G/1$ model, on the other hand, is able to more accurately capture the service time distribution, as it assumes a general service time distribution. This illustrates that the $M/G/1$ model is more accurate if the WS is a single-server WS. Since we do not know if a WS being tested is a single-server or multi-server system, these results indicate that we should use a combination of queueing models, because none of the queueing models outperformed the others.

Now let us study the accuracy of using polynomials. We experiment with polynomials of different degrees to fit the results of the four WSs, and present the results with the lowest interpolation errors in Figure 6: an $8^{th}$-degree polynomial for the TPC WS, a $12^{th}$-degree polynomial for the AB WS, a $4^{th}$-degree polynomial for the Geocoding WS, and a $3^{rd}$-degree polynomial for the Weather WS. From Table IV, the interpolation errors of using polynomials are similar to the queueing models, and outperform all four queueing models in the Geocoding WS. We conclude that the use of polynomials gives similar interpolation results as the queueing models.

In our experiments, splines exhibited overfitting, which is characterized by decreases in response times even when the arrival rates increase (e.g., in Figure 6(b)(vii)). This undesirable property makes it not a good approach for interpolation.

In general, from Table IV, NN and GP had lower interpolation errors than the queueing models, and NN had lower errors than GP. For example, in the Geocoding WS, the interpolation errors of NN and GP (0.0513 and 0.0659, respectively) were lower than the most accurate queueing model (QN, whose error is 0.1847). However, we observed that the interpolation errors of GP were noticeably higher than those of the queueing models in the TPC WS. This is because GP used a straight line to connect data points at high workload intensities, causing high interpolation errors when $\lambda_W = 2.1$ in Figure 6(a)(viii). Despite the possibility of overestimation at high workload intensities, the results have indicated that NN and GP are better approaches than using queueing models for interpolation. We
Fig. 6. Interpolation

Fig. 7. Extrapolation
TABLE V

<table>
<thead>
<tr>
<th>λ</th>
<th>measured</th>
<th>QN</th>
<th>M/M/m</th>
<th>QN 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPC</td>
<td>1.00000</td>
<td>2.50000</td>
<td>0.3444</td>
<td>0.59427</td>
</tr>
<tr>
<td></td>
<td>1.50000</td>
<td>3.94810</td>
<td>0.45419</td>
<td>1.767352</td>
</tr>
<tr>
<td></td>
<td>2.00000</td>
<td>5.05620</td>
<td>0.962797</td>
<td>3.012225</td>
</tr>
<tr>
<td>AB</td>
<td>13.50000</td>
<td>1.83289</td>
<td>2.66497</td>
<td>17.90088</td>
</tr>
<tr>
<td></td>
<td>14.00000</td>
<td>4.46033</td>
<td>-</td>
<td>58.95611</td>
</tr>
<tr>
<td></td>
<td>14.50000</td>
<td>10.85322</td>
<td>-</td>
<td>676.94406</td>
</tr>
<tr>
<td></td>
<td>15.00000</td>
<td>26.67467</td>
<td>-</td>
<td>2495.16183</td>
</tr>
<tr>
<td>Geocoding</td>
<td>3.63640</td>
<td>2.74360</td>
<td>0.39774</td>
<td>1.81043</td>
</tr>
<tr>
<td></td>
<td>4.44440</td>
<td>6.72810</td>
<td>-</td>
<td>4.229958</td>
</tr>
<tr>
<td></td>
<td>3.09790</td>
<td>0.60660</td>
<td>0.13125</td>
<td>0.10086</td>
</tr>
<tr>
<td></td>
<td>3.63640</td>
<td>0.94270</td>
<td>0.33239</td>
<td>0.01556</td>
</tr>
<tr>
<td></td>
<td>4.44440</td>
<td>1.68640</td>
<td>0.19728</td>
<td>1.23738</td>
</tr>
<tr>
<td></td>
<td>5.00000</td>
<td>4.09690</td>
<td>1.55715</td>
<td>-</td>
</tr>
</tbody>
</table>

Note that the results of $QN^3$ were the same as NN in this experiment. This can be explained as follows: since we supplied data at high workload intensities (i.e., $\lambda_E \approx \lambda_{E,\text{max}}$), little or no new data is generated in Step 2b. Hence, the data supplied to NN in $QN^3$ in Step 2c was the same as the data supplied to NN when it was to be used alone.

B. Extrapolation Errors

The next experiment studies how well the models predict response times beyond the range of arrival rates used in performance testing. As in the results presented in Sections II-B1 and II-B2, the training set consists of data points corresponding to arrival rates in the lower 60%, and the evaluation set consisted of data points corresponding to arrival rates in the upper 40%.

The results are depicted in Figure 7 and Table V. As in the results in Section II-B1, the standard regression approaches predicted increases in response time at much slower rates in many cases. For example, the standard regression approaches predicted an increase in response time at $\lambda = 2.3$, which is much lower than the M/M/m model. This indicates that the extrapolation results of $QN^3$ are comparable to those of using NN alone.

Summary: Combining our results in Tables IV and V, we observe that $QN^3$ can perform well at both, interpolation and extrapolation tasks and is better than using standard regression approaches or queueing models alone.

IV. RELATED WORK

There is a vast literature on software performance evaluation, going back to [17], which proposed the software performance engineering process that has been in wide use; it examines issues in software performance evaluation, e.g., information gathering, model construction, and performance measurements. More recently, research has focused on performance evaluation using software architectural models, e.g., [18] provides a representative survey on the topic. These works leverage software architectural models of their choice to generate performance models and focus on performance evaluation from a system designers’ perspective – this allows early performance evaluation which aids in avoiding costly design problems. Given the scope of our paper, here we discuss works that have focused on performance evaluation of third-party WSs. Although there has been significant interest in this topic, the main shortcomings of existing techniques (as considered accuracy in interpolation and extrapolation of standard regression approaches over queueing models.

Note that the results of $QN^3$ were the same as NN in this experiment. This can be explained as follows: since we supplied data at high workload intensities (i.e., $\lambda_E \approx \lambda_{E,\text{max}}$), little or no new data is generated in Step 2b. Hence, the data supplied to NN in $QN^3$ in Step 2c was the same as the data supplied to NN when it was to be used alone.
detailed below) include (a) high cost of measurements at high workloads (needed by those techniques to estimate system response time) and (b) assumptions made by those techniques about availability of information about third-party WSs.

Several black-box approaches consider predicting performance of third-party components, where the performance model is built from the component’s documentation [19], or by examining the component’s binary code (e.g., Java bytecodes) [20]. However, these approaches assume the availability of design models, documentation, or binaries of a third-party component, which are typically unavailable in the case of a third-party WSs. Thus, they are not readily applicable.

In [21] an approach to WS performance evaluation is proposed; however, it requires testing WSs at high workloads, which is expensive. In [22], a simulation-based approach to estimate WS response time is proposed, where results from performance testing are used when the WS being tested is lightly-loaded, to obtain simulation parameters, and predicting response time for heavier loads is done using simulation. However, a shortcoming of this work is the assumption (when generating the simulation model) of knowledge of the architecture of the WS being tested. Moreover, simulations could take a fairly long time to converge, and thus at design time, analytic techniques may be more desirable. In [2] a QN-based model of a composite WS is generated, in which each WS is modeled as a server in the QN. However, if the WS being tested is a third-party WS, it is not clear how information about the structure of a composite WS can be gathered (e.g., to what other WSs the WS under testing makes requests).

Another approach is to include performance information in a WS’s service description, so that their clients can use such information for performance evaluation. For example, [23] proposes that P-WSDL includes service performance characteristics of the system (e.g., utilization and/or throughput), network information (e.g., network bandwidth), and workload characteristics (e.g., request arrival rate). We argue that service providers may be reluctant to provide such information, and it is not clear how this information can describe a composite WS, in which the service performance depends on other WSs. Lastly, [3] proposes to include demands on server resources for each interfaces (e.g., a service requires X units of CPU time and Y units of I/O). Unfortunately, it is not clear how the service demand can be obtained, as it is difficult to map a high-level service to low-level hardware demands.

V. CONCLUSION

The WS paradigm allows integration of third-party WSs for creation of new services; hence it is important to understand performance characteristics of third-party WSs. To reduce the cost of performance testing, we estimate the performance of third-party WSs during high workloads using data collected at low workloads. Our hybrid approach combines the low interpolation errors of standard regression analysis with the low extrapolation errors of queueing models for response time prediction. Our validation results indicate that the hybrid technique is accurate, as compared to using standard regression approaches or queueing models alone. Thus, we believe that our technique can be used to improve WS-based system designs. For instance, our approach can be utilized by service selection techniques [24], [25]. As discussed before, a WS can be composed dynamically, where performance characteristics can be part of the selection criteria. Our approach can support such techniques by proving performance estimation information for a given WS, i.e., so that such approaches can make more informed decisions.

ACKNOWLEDGMENTS

This work is supported in part by the NSF 0905665 and 0917340 awards. The authors would like to thank GJ Halfond for the useful discussions on an earlier version of this work, and the anonymous reviewers for their valuable comments.

REFERENCES