Prediction of Atomic Web Services Reliability for QoS-Aware Recommendation

Marin Silic, *Student Member, IEEE*, Goran Delac, *Student Member, IEEE*, and Sinisa Srbljic, *Senior Member, IEEE*

Abstract—While constructing QoS-aware composite work-flows based on service oriented systems, it is necessary to assess nonfunctional properties of potential service selection candidates. In this paper, we present *CLUS*, a model for reliability prediction of atomic web services that estimates the reliability for an ongoing service invocation based on the data assembled from previous invocations. With the aim to improve the accuracy of the current state-of-the-art prediction models, we incorporate user-service-, and environment-specific parameters of the invocation context. To reduce the scalability issues present in the state-of-the-art approaches, we aggregate the past invocation data using K-means clustering algorithm. In order to evaluate different quality aspects of our model, we conducted experiments on services deployed in different regions of the Amazon cloud. The evaluation results confirm that our model produces more scalable and accurate predictions when compared to the current state-of-the-art approaches.

Index Terms—K-means clustering, prediction, QoS, recommendation, reliability, web services

1 Introduction

The majority of contemporary web applications often adheres to the set of rules and principles defined by the service-oriented architecture (SOA). SOA is an architectural style mostly used for design of information systems for various enterprise solutions, but is also commonly applied in a more general context. The basic functionalities are usually created as reusable atomic services accessible through publicly available interfaces. To provide more advanced functionalities, SOA allows designers to compose the atomic services into more complex ones. While constructing composite services, it is essential for the developer to select high quality atomic service candidates [1], [2], as the application quality relies on both functional and nonfunctional qualities of the selected candidates. Hence, to create an efficient composite application, the developer should be provided with reliable information on both atomic services' functionalities and their nonfunctional dependability attributes, such as response time, reliability, availability [3].

This paper is focused on atomic service reliability, as one of the most important nonfunctional properties. We define service reliability as the probability that a service invocation gets completed *successfully*—i.e. *correct* response to the service invocation gets successfully retrieved under the specified conditions and time constraints. In the related literature, this definition of reliability also appears as *successful delivery rate*[1] [4], *user-perceived reliability* [5] and *reliability on demand* [6]. It should be noted that the adopted *user-centric* definition of reliability differs from the traditional *system-centric* reliability definition used for "never ending" systems [7]. However, *reliability on demand* definition is more convenient for web services since service invocations are discrete and relatively rare events. According to the adopted definition, the reliability value can be computed from the past invocation sample as the ratio of the number of successful against the number of total performed invocations.

However, acquiring a comprehensive past invocation sample proves to be a very challenging task for several reasons. There is a familiar variability characteristic for service oriented systems in reliability perception from the user's and service provider's standpoint [6]. In general, the reliability value computed considering exclusively the data obtained by the service provider can be unrepresentative for a specific user because of the oscillations introduced by a variety of invocation context parameters. From user's perspective, further obstacles while collecting data are related to the service usage cost and performance issues. For example, gaining the invocation sample by performing service reliability testing can be enormously costly, while "stress testing" can intensely impact the service performance and also make the measured data worthless [8].

A promising strategy to overcome the aforementioned reliability assessment challenges is to obtain *partial* but relevant history invocation sample, and to utilize *prediction* algorithms to estimate reliability for the missing records. Thus, the following scenario can be employed to recommend the most suitable service candidates. While accessing various services on the Internet, users perceive different reliability properties depending on the given invocation context (1). The partial invocation sample can be gained by gathering live feedback regarding service usage through *collaborative feedback* [9], [10] (2*a*), and assembling as many data records as possible from the service providers by performing service *monitoring* [11] (2*b*). Prediction algorithms can

1. The terms reliability and availability are often used interchangeably in the literature.

• *The authors are with the Faculty of Electrical Engineering and Computing, Consumer Computing Lab, University of Zagreb, Unska 3, 10000 Zagreb, Croatia. E-mail: {marin.silic, goran.delac, sinisa.srbljic}@fer.hr.*

Manuscript received 25 Nov. 2013; revised 17 July 2014; accepted 1 Aug. 2014. Date of publication 7 Aug. 2014; date of current version 12 June 2015.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSC.2014.2346492

then be used to estimate reliability for future service invocations based on the collected past invocation sample (3). Finally, the most reliable service candidates can be recommended according to the predicted reliability values (4).

The researchers have proposed several prediction models based on *collaborative filtering* technique often used in modern recommendation systems [12]. Even though the existing collaborative filtering based approaches achieve promising performance, they demonstrate disadvantages primarily related to the prediction *accuracy* in dynamic environments and *scalability* issues caused by the invocation sample size. Regarding the prediction accuracy, collaborative filtering provides accurate recommendations in static environments where the collected data records are relatively stable. This means that the records remain up-to-date for a reasonably long period of time (e.g. *song ratings*, *product recommendations*). However, service-oriented systems are deployed on the Internet, which is a very dynamic environment where service providers register significant load variations during the day [13], [14], [15], [16]. In such a dynamic environment, user perceived service reliability may considerably differ depending of the actual time of invocation. Furthermore, collaborative filtering approaches store reliability values for each user and service pair. Having millions of users and a substantially large number of services, these approaches do not scale.

In this paper we advance the state-of-the-art in *collaborative filtering* based reliability prediction approaches through following contributions:

- A model-based collaborative filtering approach *CLUS (CLUStering)* is introduced. The model considers user-, service- and environment-specific parameters to provide a more accurate description of the service invocation context. The environment-specific parameters, not present in the related approaches, are used to model the effect of varying load conditions on service reliability. Such an approach results in more accurate reliability predictions. Furthermore, the model addresses *scalability* issues by aggregating users and services into respective user and service clusters according to their reliability values using K-means clustering.
- A novel strategy for assembly of most recent service usage feedback is presented. The strategy enables discovery of deviations from the presumed load distributions and is applied to increase CLUS accuracy.
- A novel model-based collaborative filtering approach that utilizes linear regression, an unsupervised machine learning technique, is presented.

For evaluation purposes, we perform a series of experiments on services deployed in the *Amazon EC2* cloud varying different parameters that describe the service invocation context. We measure different quality aspects of our models and compare them with the prediction performance of the state-of-the-art collaborative filtering approaches. The obtained evaluation results confirm that our models meet the expectations: significantly improve prediction *accuracy*, and also reduce prediction *execution time* in comparison to the related approaches.

The rest of the paper is organized as follows. Section 2 presents the related work. Section 3 overviews the prediction process and describes model parameters. Section 4 formally describes steps in *CLUS* approach. Section 5 introduces model-based collaborative filtering approach *LinReg*. Section 6 describes the conducted experiments and presents extensive evaluation results. Section 7 concludes the paper by summarizing advantages and disadvantages of the presented approach.

2 Related Work

A myriad of different approaches for modeling the reliability of traditional software systems have been proposed in the literature [7], [17], [18], [19]. Still, web services are dynamic software artifacts that provide their functionalities via publicly accessible interfaces over the Internet. The Internet is a very dynamic environment in which the service invocation outcome depends on a variety of different impacts that determine the invocation context. As a consequence, the traditional approaches for modeling software reliability are not suitable for assessing the reliability of web services.

While designing new models for service oriented systems, the majority of researchers usually focus on studying the reliability of service compositions. Various approaches for predicting the reliability of composite services have been proposed [20], [21], [22], [23], [24], [25], [26]. All these approaches usually assume the atomic service reliability values are already known or rarely suggest how can they be acquired. However, the arguments stated in Section 1 indicate that collecting a comprehensive sample of reliability values is a very difficult task in practice.

The most successful approaches for prediction of atomic service reliability are based on the collaborative filtering technique [12]. According to the related literature [12], the basic types of collaborative filtering are: *memory-based*, *model-based* and *hybrid*. The following sections briefly describe each type of collaborative filtering.

2.1 The Memory-Based Collaborative Filtering

The memory-based collaborative filtering is a commonly used technique in nowadays state-of-the-art recommendation systems [27], [28], [29], [30], [31]. This filtering technique extracts the information or patterns by statistically correlating the data obtained from multiple entities like agents, viewpoints or data sources. The benefit of memory-based collaborative filtering is that lacking information for a particular entity can be predicted using the available data from the most statistically similar entities.

This collaborative filtering type uses *user-item* matrix to store the data for reliability prediction. Each p_{ui} value in the matrix represents the reliability of the service i perceived by the user u. In real service-oriented systems, matrices can contain millions of user and services, while new user-service pairs arise in real time. Furthermore, each user accesses only a small subset of services. Consequently, the user-item matrix is extremely sparse and contains a significant amount of empty cells reflecting missing reliability values that need to be predicted.

Memory-based collaborative filtering can be applied in two different ways. The *UPCC* combines the information collected from different users and predicts missing reliability values using the available data from the most statistically similar users [32]. The *IPCC* collects the data from different services and predicts missing reliability values based on available values from the most statistically similar services [33]. The *Hybrid* approach [9], [10] achieves better prediction performance by employing both the data retrieved from similar users and services and predicting missing reliability values as a linear combination of *UPCC* and *IPCC*.

2.2 The Model-Based Collaborative Filtering

The model-based collaborative filtering approaches are known to be more computationally complex and difficult to implement. These approaches often combine more complex techniques such as machine learning or data mining algorithms to learn the prediction model by recognizing complex patterns using the training data, and then use the model to make predictions on the real data [12]. For instance, Yu et al. propose a trace norm regularized matrix factorization based approach for prediction of web services reliability [34]. Similarly, Zheng and Lyu also utilize matrix factorization in their approach that assumes a small number of factors that impact the user perceived reliability [35]. However, the mentioned approaches do not include any environment-specific parameters into the prediction process. Typical representatives of model-based collaborative filtering are the approaches based on the *linear regression* technique [36], such as [37], [38]. Linear regression is also found most suitable for numerical prediction domains. Since there are no existing linear regression models that incorporate environmental parameters in the field of web services reliability prediction, we additionally contribute by constructing our own *LinReg* approach, described in Section 5. In addition, we present *CLUS*, a model-based prediction approach based on the k-means clustering, as described in Sections 3 and 4.

2.3 The Hybrid Collaborative Filtering

The hybrid collaborative filtering approaches can be very effective in addressing disadvantages of basic memory-based collaborative filtering [39]. However, the main disadvantage of these approaches is that their prediction capability often relies on additional domain specific data describing the internals of a system. It proves to be a challenging task to obtain such data in practice.

In our recent work [40], we addressed the disadvantages of the collaborative filtering based approaches by improving prediction accuracy and scalability. However, the model we proposed (*LUCS*) is applicable in the environments where the model's input parameters are highly available. For example, we group services into service classes considering service's computational complexity and we assume each service's class is explicitly known as the input parameter. As the amount of services with missing input parameters increases the prediction accuracy deteriorates. These deficiencies are addressed by *CLUS* and linear regression approaches described in the following sections.

3 CLUS Prediction Overview

In this section we present overview of *CLUS*, a model for prediction of atomic web services reliability. With the aim of improving the prediction accuracy, we define user-, service- and environment-specific parameters that determine service invocation context more precisely than the related prediction models. Furthermore, to achieve scalability, we group the collected invocation sample across three different dimensions associated with the defined parameters using the K-means clustering algorithm. The comparative benefit of K-means among other clustering algorithms [41] is that it converges accurately and very quickly when performed on the available service reliability data.

3.1 Invocation Context Parameters

In our model we distinguish three groups of parameters: user-, service- and environment-specific parameters.

3.1.1 *User-Specific Parameters*

We associate user-specific parameters with user-introduced fluctuations in the service reliability performance. User-specific parameters include a variety of factors that might impact the reliability of a service such as user's location, network and device capabilities, usage profiles. To incorporate user-specific parameters into the prediction process, we group users into clusters according to their reliability performance, gained from the past invocation sample, using the K-means clustering algorithm.

3.1.2 *Service-Specific Parameters*

Service-specific parameters are related to the impact of service characteristics on the reliability performance. Numerous factors influence service-specific parameters such as service's location, computational complexity and system resources (e.g. CPU, RAM, disk and I/O operations). We include the service-specific parameters into the prediction process by grouping services into clusters according to their reliability performance obtained from the past invocation sample using K-means clustering algorithm.

3.1.3 *Environment-Specific Parameters*

Environment-specific parameters relate to the current conditions in the environment such as service provider load or network performance at the time of the invocation. For the purposes of evaluation, we only consider service load as the environment parameter. We define the service load as the number of requests received per second. The non-functional qualities of a service, such as availability and reliability are significantly influenced by fluctuations in the service load. Since web servers register considerable load variations during the course of a day [13], [14], [15], [16], we divide the day into an arbitrary number of *time windows*. In order to improve the prediction accuracy we disperse the past invocation data among different time windows. Finally, we group time windows into clusters according to the reliability performance computed from the past invocation sample using K-means clustering algorithm.

3.2 Reliability Prediction Process

The high level overview of *CLUS*, model for prediction of atomic web services is depicted in Fig. 1. As can be seen in the figure, reliability prediction process is consisted of two separate phases: *data clustering phase* and *prediction phase*.

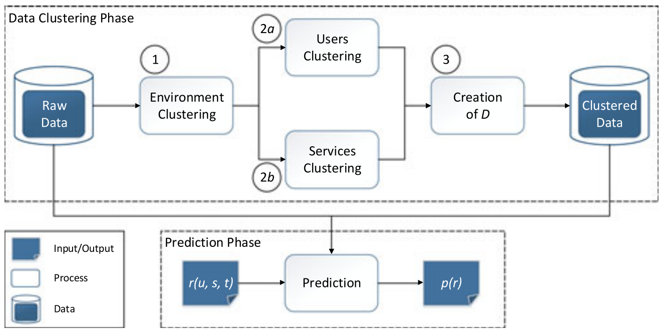


Fig. 1. Reliability prediction overview.

Prior to reliability prediction, we perform clustering of the history invocation sample. First, we cluster the time windows associated with the environment conditions according to the reliability performance fetched from the past invocation sample (1). Then, we cluster users ($2a$) and services ($2b$) considering their reliability performance within each time window cluster. Finally we create a three-dimensional space D containing clustered data (3). Once the clustering phase is done, prediction of the atomic services reliability can be performed in the prediction phase.

4 CLUS Prediction Model

This section describes each step of the data clustering process used in *CLUS* and defines how predictions are calculated form the clustered data.

A service invocation is formally defined as follows:

$$r(u, s, t), \tag{1}$$

where u is the user executing the invocation, s is the invoked service and t is the actual time of the invocation.

The past invocation sample contains data addressed as in the Equation (1). In order to make scalable and accurate reliability predictions for future service invocations, the data needs to be transformed into a more structured and compact form. Hence, we store the data into a three-dimensional space $D[u, s, e]$ where each dimension u, s and e is associated with one group of parameters. In the following sections we describe how particular records are clustered and associated with the corresponding parameters. Finally, we describe the creation of space D, i.e. how each entry in D is calculated and how the reliability is predicted for an ongoing service invocation.

4.1 Environment-Specific Data Clustering

First, we define the set of different environment conditions E as follows:

$$E = \{e_1, e_2, \ldots, e_i, \ldots, e_n\}, \tag{2}$$

where e_i refers to a specific environment condition determined by service provider load and n is an arbitrary number of distinct environment conditions.

The intention is to correlate each available history invocation record with the service provider load at the time it was performed. As already stated in Section 1, analyses of the collected data from different service providers can discover regularities in the load distribution for certain time periods [13], [14], [15], [16]. Thus, we divide the day in an arbitrary number of time windows, where each time window w_i is determined with its start time t_i and end time t_{i+1}. We assume that the environment-specific parameters are stable during a particular time window. Once the time windows are determined, we calculate the average reliability value $\overline{p_{w_i}}$ for each time window w_i:

$$\overline{p_{w_i}} = \frac{1}{|W_i|} \sum_{r \in W_i} p_r, \tag{3}$$

where W_i is the set of records within the time window w_i, r is the record from the past invocation sample and p_r is user perceived reliability for that invocation.

Each particular time window is clustered using K-means clustering algorithm [36] into an appropriate environment condition e_i according to its average reliability value $\overline{p_{w_i}}$ as described in the following paragraph.

In K-means algorithm, the goal is to partition the data points into K clusters. In case of time window clustering, K is equal to the cardinality number of the environment conditions set $|E|$. For each environment condition cluster, we introduce a *centroid* value μ_{e_k} which represents the given cluster e_k. Moreover, for each time window, we introduce a corresponding binary indicator variable $b_{w_i,e_k} \in \{0, 1\}$. The indicator variable b_{w_i,e_k} has a value 1 if the associated time window W_i is assigned to the cluster e_k, otherwise it has a value 0. Now, we can define an objective function as follows:

$$J = \sum_{W_i \in W} \sum_{e_k \in E} b_{w_i,e_k} \|\overline{p_{w_i}} - \mu_{e_k}\|^2. \tag{4}$$

The function J represents the sum of the squares of distances of each time window to its assigned centroid μ_{e_k}. The goal of K-means clustering algorithm is to minimize the objective function J with respect to variables $\{b_{w_i,e_k}\}$ and $\{\mu_{e_k}\}$.

K-means algorithm performs an iterative procedure in which each iteration involves two successive steps corresponding to optimizations with respect to b_{w_i,e_k} and μ_{e_k}. In the first step, we minimize J with the respect to the b_{w_i,e_k}, while keeping the μ_{e_k} fixed. Then in the second step, we minimize J with respect to the μ_{e_k}, keeping the b_{w_i,e_k}. This two-stage optimization is repeated until convergence. While performing the first stage optimization, we simply assign each time window to the closest cluster centroid as follows:

$$b_{w_i,e_k} = \begin{cases} 1, & \text{if } k = min_j \|\overline{p_{w_i}} - \mu_{e_j}\|^2 \\ 0, & otherwise. \end{cases} \tag{5}$$

Regarding the second stage of optimization, J is a quadratic function of μ_{e_k}, and it can be minimized by setting the derivative with respect to μ_{e_k} to zero which can be easily solved for μ_{e_k}:

$$\mu_{e_k} = \frac{\sum_i b_{w_i,e_k} \times \overline{p_{w_i}}}{\sum_i b_{w_i,e_k}}. \tag{6}$$

The denominator in the above equation is equal to the number of time windows that belong to the respective cluster e_k.

Hence, the result has a simple interpretation, the algorithm assigns mean of all time windows belonging to the cluster e_k to cluster centroid μ_{e_k}.

Note that each phase reduces the value of the objective function J, which means that the convergence of the algorithm is assured. However, there is a caveat that the algorithm converges to the local rather than global minimum. We assure convergence to an acceptable minimum by selecting good initial values for centroids μ_{e_k}. The reason we choose K-means among so many clustering algorithms [41] is that it quickly converges for the service reliability records, which is closely related to the algorithm's computational complexity (see Section 6.5).

Once the K-means algorithm converges, we can correlate each particular record from the history invocation sample with the environment conditions at the time it was performed, by applying the following transitive relation. If the record about previous invocation $r(u, s, t)$ belongs to the time window w_i and the time window w_i is clustered into environment condition e_i, then the invocation $r(u, s, t)$ is performed during the environment condition e_i.

4.2 User-Specific Data Clustering

Next, we define the set of user groups U as follows:

$$U = \{u_1, u_2, \ldots, u_i, \ldots, u_m\}, \tag{7}$$

where each user group u_i contains users that achieve similar reliability performance, while m is the number of different user groups.

For each user u in the past invocation sample, we calculate the n-dimensional reliability vector $\boldsymbol{p_u}$ as follows:

$$\boldsymbol{p_u} = \{\overline{p_{e_1}}, \overline{p_{e_2}}, \ldots, \overline{p_{e_i}}, \ldots, \overline{p_{e_n}}\}, \tag{8}$$

where each vector dimension $\overline{p_{e_i}}$ represents the average reliability value perceived by the given user u during the environment condition e_i. Once the average reliability n-dimensional vector is calculated and assigned to each user, we perform K-means clustering of users into different user groups according to their reliability vector's $\boldsymbol{p_u}$ values. The K-means clustering is performed similarly like in the case of time windows clustering. The only difference is that data points in this case are multi-dimensional vectors $\boldsymbol{p_u}$ instead of values $\overline{p_{w_i}}$ in time windows clustering. Now we can easily correlate each available previous invocation record $r(u, s, t)$ with an appropriate user group u_i.

4.3 Service-Specific Data Clustering

Finally, we define the set of service groups S as follows:

$$S = \{s_1, s_2, \ldots, s_i, \ldots, s_l\}, \tag{9}$$

where each service group s contains services that achieve similar reliability performance, while l is an arbitrary number of different service groups.

For each service s in the past invocation sample, we calculate the n-dimensional reliability vector $\boldsymbol{p_s}$ as follows:

$$\boldsymbol{p_s} = \{\overline{p_{e_1}}, \overline{p_{e_2}}, \ldots, \overline{p_{e_i}}, \ldots, \overline{p_{e_n}}\}, \tag{10}$$

where each vector dimension $\overline{p_{e_i}}$ represents the achieved average reliability for invoking service s during the environment conditions e_i. Once the average reliability n-dimensional vector is calculated for each service, we perform K-means clustering of services into different service groups according to their reliability vector's $\boldsymbol{p_s}$ values. Again, the clustering is performed like in the case of time windows clustering, except the data points in this case are multi-dimensional vectors $\boldsymbol{p_s}$ instead of values $\overline{p_{w_i}}$ in time windows clustering. Now we can easily correlate each available previous invocation record $r(u, s, t)$ with appropriate service group s_i.

4.4 Creation of Space D and Reliability Prediction

Once each available history invocation record $r(u, s, t)$ is associated with the respective data clusters u_k, s_j and e_i, we can generate the space D. We calculate each entry in D as follows:

$$D[u_k, s_j, e_i] = \frac{1}{|R|} \sum_{r \in R} p_r, \tag{11}$$

where p_r is user perceived reliability for invocation r and:

$$R = \{r(u, s, t) \mid r \in u_k \wedge r \in s_j \wedge r \in e_i\}. \tag{12}$$

Now, let us assume we have an ongoing service invocation $r_c(u_c, s_c, t_c)$ whose reliability p_c needs to be predicted.

In *CLUS* approach we assume regular daily load distributions which adhere to the results and studies presented by other researchers [13], [14], [15], [16]. However, certain deviations regarding the expected load may negatively impact the quality of prediction. Unlike in our recent work [42], we propose a strategy of assembling the most recent feedback on the achieved reliability with the aim of preventing less accurate predictions. Hence, we take into account all the records from K, a set of recent feedback records collected during the time interval $[t_c - \Delta t, t_c]$, where Δt is an arbitrary long interval which can be adjusted to a specific environment. Each reported feedback record contains the outcome information for some specific service invocation from the client's perspective. For instance, the client can report successful or failed service invocations. With the aim to inspect current load conditions, we calculate the average reliability value $\overline{p_K}$ for the time interval $[t_c - \Delta t, t_c]$ considering feedback records from the set K.

Then, we calculate the closest environment conditions cluster for the average reliability value $\overline{p_K}$ among all the environment conditions clusters obtained in the environment-specific clustering phase (see Section 4.1, Equation (5)). In this manner, we associate the average reliability value $\overline{p_K}$ with the actual load conditions in the environment.

Once the actual load conditions in the environment are associated with the respective environment conditions cluster w_i, we check if there is a set H in the past invocation sample containing records with the same invocation context parameters as r_c:

$$H = \{r_h \mid u_h = u_c \wedge s_h = s_c \wedge t_h, t_c \in w_i\}. \tag{13}$$

If the set H is not empty, than we calculate the reliability p_c by using the existing reliabilities in the set H:

$$p_c = \frac{1}{|H|}\sum_{r\in H} p_r. \tag{14}$$

Otherwise, if the set H is empty we calculate the reliability p_c using the data stored in the space D as follows:

$$p_c = D[u_k, s_j, e_i], \tag{15}$$

where current user u_c belongs to the user group u_k, current service s_c belongs to the service group s_j and the actual time t_c belongs to the time window w_i associated with the environment conditions e_i.

In order to improve prediction accuracy, past invocation sample should be updated with newly experienced reliability records and obsolete records should be removed. Past invocation sample update may be performed periodically depending on how dynamically the environment changes. Also, each time past invocation sample gets updated, the space D should be recreated.

5 Linear Regression Prediction Model

In this section we present a model-based collaborative filtering approach *LinReg* which applies the *linear regression* method to learn the prediction model, and then utilizes the model to produce the missing reliability values. The *LinReg* model considers past invocation training data classified according to the following input parameters: *user location* (u), *service location* (s), *service load* (l) (describes load conditions at the time of the invocation) and *service class* (c) (describes the internal complexity of the service). According to the input parameters, the model considers the following *hypothesis* function containing $M = 4^{m+1}$ features:

$$h_{\boldsymbol{w}}(r) = \sum_{i,j,k,v}^{m} w_{i,j,k,v} \times \left({u_r}^i \times {s_r}^j \times {l_r}^k \times {c_r}^v\right) = \boldsymbol{w}^T\boldsymbol{\phi}(r), \tag{16}$$

where r is a record from the past invocation sample with its properties u_r, s_r, l_r, c_r, representing model input parameters, m is a model parameter that determines the total number of features M in the hypothesis function and the complexity of the model (we used $m = 3$ for the environment with load intensity, and $m = 2$ for the environment without load intensity), while $w_{i,j,k,v}$ are model parameters that need to be computed in the *learning phase*. The hypothesis function constructed in this manner enables creation of multiple features using only 4 input parameters. Hence, each combination of i, j, k, v represents a new feature. For instance, if we have a combination $i = 2$, $j = 3$, $k = 2$ and $v = 1$, we obtain the feature ${u_r}^2 \times {s_r}^3 \times {l_r}^2 \times c_r$. The hypothesis function can be written in a vectorized form where $\boldsymbol{w}$ is a parameter vector and $\boldsymbol{\phi}$ is a feature vector.

We define the cost function $J(\boldsymbol{w})$ as follows:

$$J(\boldsymbol{w}) = \frac{1}{2}\sum_{r=1}^{|N|}[\hat{p_r} - h_w(r)]^2 + \frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w}, \tag{17}$$

where $\hat{p_r}$ represents measured reliability value, and $h_w(r)$ reliability value computed using the hypothesis. The right part of the equation is a *regularization* term included to prevent the *overfitting* problem and defined with the regularization parameter λ [36]. The cost function calculates the accumulated squared prediction error over the entire set of training examples N. The general idea is to minimize the cost function $J(\boldsymbol{w})$ by setting the gradient of Equation 17 with respect to $\boldsymbol{w}$ to zero. Then, by solving the obtained equation for $\boldsymbol{w}$, we get model parameters $\boldsymbol{w}$ using following matrix equation:

$$\boldsymbol{w} = (\lambda \boldsymbol{I} + \boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\boldsymbol{p}, \tag{18}$$

where $\boldsymbol{I}$ is a $M \times M$ identity matrix, $\boldsymbol{\Phi}$ is a $N \times M$ matrix, called *design matrix* [36] and $\boldsymbol{p}$ is a N-dimensional vector that contains reliability values $\hat{p_r}$ from the past invocation sample used as training examples.

Suppose we have an ongoing service invocation r_c defined by its properties u_{r_c}, s_{r_c}, l_{r_c}, c_{r_c} whose reliability p_{r_c} needs to be predicted. According to our model-based collaborative filtering approach *LinReg*, reliability value p_{r_c} is estimated using the hypothesis function as follows:

$$p_{r_c} = \begin{cases} 0 & \text{if } h_{\boldsymbol{w}}(r_c) < 0 \\ 1 & \text{if } h_{\boldsymbol{w}}(r_c) \geq 1 \\ h_{\boldsymbol{w}}(r_c) & \text{otherwise.} \end{cases} \tag{19}$$

6 Evaluation

This section presents exhaustive evaluation results that confirm our claim that *CLUS* and *LinReg* improve prediction accuracy and overcome scalability issues present in the related approaches. To prove our claims we conducted a series of experiments to collect the data and analyze several model quality aspects. We compare the models with three *memory-based* collaborating filtering approaches: *UPCC* [32], *IPCC* [33] and the *Hybrid* approach [9], [10]. Furthermore, we compare the models with the *hybrid* collaborative filtering approach *LUCS* proposed in our recent work [40]. The evaluation results confirm that *CLUS* model provides best prediction performance among the competing approaches considering both prediction accuracy and computational performance.

To evaluate prediction accuracy, we use standard error measure *root mean square error* (*RMSE*). It computes a quadratic scoring rule which represents average magnitude of errors:

$$RMSE = \sqrt{\frac{\sum_j^N (p_j - \hat{p_j})^2}{N}}, \tag{20}$$

where N is the cardinal number of the prediction set, p_j an existing reliability value in the prediction set, while $\hat{p_j}$ is the predicted reliability value. Note that *RMSE* can range from 0 to ∞. It is a negatively-oriented score, which means that lower values are better.

6.1 Experiment Setup

To evaluate different model quality aspects, we created a controlled environment containing our own implemented web services. In this way we reduced the external noise and controlled the service-specific parameters such as service locations and complexity as well as the environment conditions such as different loads.

TABLE 1
Matrix Ranks in Different Service Classes

Service class	1	2	3	4	5	6	7
Matrix rank	350	310	280	250	210	180	150

TABLE 2
Time Intervals in Different Load Levels

Load level	1	2	3	4	5	6	7
Time interval/sec	3	4	5	6	7	8	9

In our experiments we introduced different user-, service- and environment-specific parameters. We ensured different service-specific parameters by implementing RESTful services with different computational complexity and by placing services in different geographic locations worldwide. Although a myriad of other parameters influence the computational complexity of services, for practical reasons we chose the amount of memory as a parameter that separates services by their computational complexity. Note that the generality of our experiments is preserved and that any other parameter can be chosen as well. Hence, we created seven different service classes that perform matrix multiplication operations on randomly generated matrices, having each class operate on matrices of different rank as shown in Table 1.

To introduce another service-specific parameter we placed 49 web services in seven available *Amazon EC* regions: *Ireland*, *Virginia*, *Oregon*, *California*, *Singapore*, *Japan* and *Brazil*, having one service class in each region. Each service was deployed on an Amazon machine image running *Microsoft Windows Server 2008*, 64-bit, *IIS 7* and *ASP.NET 3.5*.

To incorporate user-specific parameters in experiments, we simulated users by placing the amount of 50 instances of *loadUI* tool [43] in different locations within the cloud. The instances were running as agents in the cloud waiting for the test cases to be delivered and committed. The used tool is an open source tool designed for web services “stress-testing” and it supports creation of various test cases.

Finally, we introduced different environment-specific parameters by creating test cases with different load generators defined by the *time interval* between subsequent invocations. We encumbered services with seven different load levels by altering the time interval as defined in Table 2. For each particular *load*, a special *test case* was created and delivered to all agents in the cloud. During every single test case, each agent sent 150 requests to each deployed service. Based on the number of successful requests against the number of total 150 sent requests, the *measured* reliability value for that particular agent (user), service and load is computed. Upon the test case completion, we collected measured reliability data from agents and restarted the machines hosting services to recover for the next test case. As part of our experiments, the overall amount of around 2.5 million distinct web service invocations was performed.

The input parameters values in *LinReg* approach used in our evaluations are: $u_r \in [1, 50]$ and $s_r, l_r, c_r \in [1, 7]$.

6.2 The Impact of the Data Density

In this section we analyze how the amount of collected data, i.e. data density, impacts different qualities of prediction performance. Section 6.2.1 investigates the impact of data density on prediction accuracy while Section 6.2.2 examines how the data density impacts computational performance of the prediction.

We simulate different amounts of the collected data by altering data densities between 5 and 50 percent of the data collected in experiments with the step value of 5 percent. We create two different environments by varying environment-specific parameters. In the first case, we assume a dynamic environment with different loads having request frequencies from *req/3 sec* to *req/9 sec*. In the second case, we assume a static environment with a constant load having request frequency value of *req/3 sec*. Furthermore, for density evaluation purposes we group the data into seven different clusters. Although an arbitrary number of clusters can be applied, we selected seven as it is the number of service classes, user locations on the Amazon cloud and loads in the experiment setup.

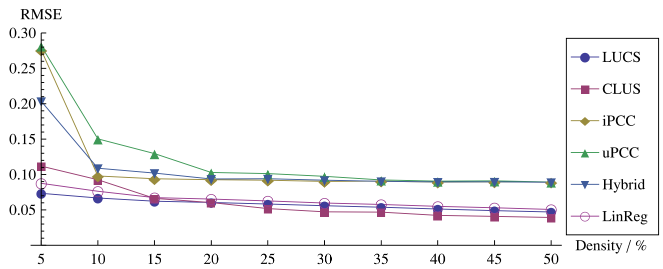
To evaluate the impact of data density on prediction performance we vary the percentage of the collected data used for training. In the first step, we randomly include the amount of 5 percent of the collected data for training. Then, we predict the reliability for the remaining data for all competing approaches based on the training data. At the same time, we calculate *RMSE* values using reliability values that have been measured in the experiments. We also measure the time that it takes to produce the predictions for all approaches. In the next step, we randomly include another 5 percent of the collected data into training data and we recalculate the predictions and performance measures. This process is repeated until the calculation is done for the density of 50 percent.

6.2.1 The Impact on Prediction Accuracy

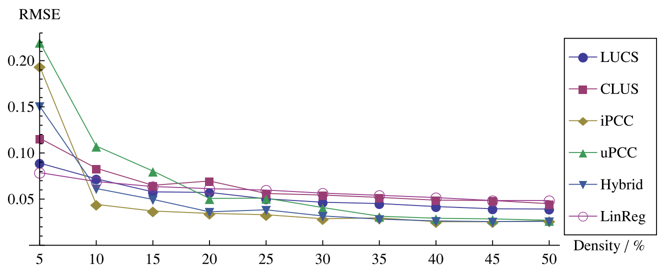
In this section we present the impact of the collected data density on prediction accuracy. First, we analyze the evaluation results obtained in a dynamic environment, and then we discuss the evaluation results for a static environment.

Dynamic environment. The evaluation results in the case of dynamic environment are depicted in Fig. 2a. The figure captures *RMSE* values with respect to the data density for all analyzed approaches. The results show that the prediction accuracy highly depends on the data density (e.g. for *UPCC* the *RMSE* value drops from 0.281 for density of 5 percent to 0.0893 for density of 50 percent). As can be seen in the figures, *CLUS*, *LinReg* and *LUCS* approaches significantly outperform memory-based collaborative filtering approaches.

For low densities, *LUCS* approach provides the best prediction accuracy with the *RMSE* value of 0.073 for density of 5 percent. The second score for low densities belongs to the *LinReg* approach with the *RMSE* value of 0.087 for density of 5 percent. *CLUS* takes the third place with *RMSE* value of 0.111 for density of 5 percent, but as the density increases its *RMSE* value drops and approaches the *RMSE* values of *LUCS* and *LinReg* (e.g. for density of 15 percent



(a) RMSE, in dynamic environment



(b) RMSE, in static environment

Fig. 2. The impact of density on prediction accuracy.

CLUS approach already takes the second place with the *RMSE* value of 0.066). It is obvious from the graphs that prediction accuracy of *UPCC*, *IPCC* and *Hybrid* approach is far from the *RMSE* scores of *CLUS*, *LinReg* and *LUCS*.

For high data densities, *CLUS* provides the best prediction accuracy among all the competing approaches (e.g. the *RMSE* value of 0.039 for density of 50 percent). Approaches *LUCS* and *LinReg* achieve similar prediction accuracy, but *LUCS* approach provides slightly better prediction accuracy with the *RMSE* value of 0.047 compared to the *LinReg*'s *RMSE* value of 0.050 for density of 50 percent. Note that all memory-based collaborative filtering approaches achieve similar prediction accuracy that cannot be significantly improved with the increase of the collected data due to negligence of environment's dynamic nature and inefficient usage of the collected data.

Static environment. The evaluation results for static environment without load intensity are depicted in Fig. 2b. The figure presents *RMSE* values in relation to the data density for every considered approach. As can be expected, all the approaches improve prediction accuracy with the increase of data density (e.g. for *IPCC* the *RMSE* varies from the value of 0.19 for density of 5 percent to the value of 0.025 for density of 50 percent).

For low densities, *CLUS*, *LUCS* and *LinReg* approaches provide better prediction accuracy in comparison to the memory-based collaborative filtering approaches. More specifically, *LinReg* approach achieves best prediction accuracy with *RMSE* value of 0.078 against *Hybrid*'s *RMSE* value of 0.150 for density of 5 percent. For the same density, *LUCS* and *CLUS* approaches also outperform memory-based collaborative filtering having *RMSE* values of 0.089 and 0.116 respectively.

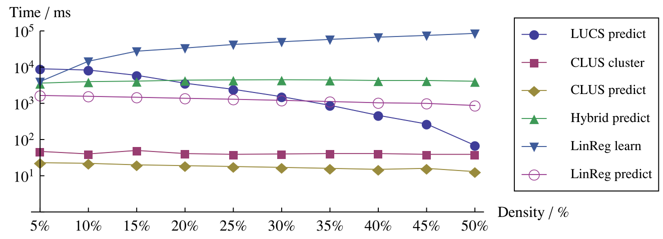
As the density increases, memory-based collaborative filtering approaches provide better prediction accuracy (e.g. *LUCS* with *RMSE* value of 0.039 and *CLUS* with *RMSE* value of 0.045 against *Hybrid*'s *RMSE* value of 0.025 for density of 50 percent). This result is unsurprising since both our approaches aggregate the data either by clustering users and services into respected clusters in *CLUS*, or by grouping past invocations by their locations, classes and loads as it is done in *LUCS* [40]. Either way, by aggregating the collected data, prediction accuracy expectedly decreases. Note that lower densities are more realistic for web-based system since the number of users and services is substantially large.

To summarize, approaches *LUCS*, *CLUS* and *LinReg* achieve better prediction accuracy in a dynamic environment, while approaches *IPCC*, *UPCC* and *Hybrid* produce more accurate predictions in a static environment.

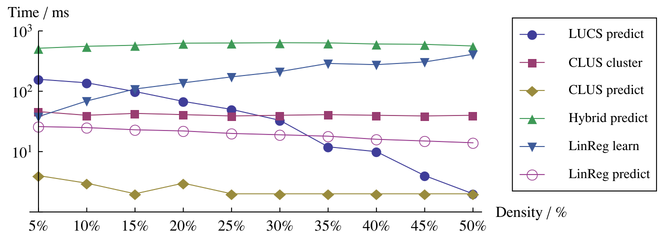
6.2.2 The Impact on Computational Performance

The evaluation results for computational performance of the prediction in *dynamic* and *static* environments are depicted in Figs. 3a and 3b. We use the execution time that is required to compute the predictions as the measure of prediction's computational performance. The figures depict aggregated *prediction time* for the whole testing set in *milliseconds* in relation to the data density in the logarithmic scale. All memory-based collaborative filtering approaches induce similar computational complexity (see details in Section 6.5). Hence, to depict the evaluation results more clearly, we choose the *Hybrid* approach [9], [10] as their representative (labeled as *Hybrid predict* in Figs. 3a and 3b).

Recall Section 3 and note that *CLUS* prediction process is done in two phases, *data clustering phase* and *prediction phase*. Thus, we present both *data clustering phase* (labeled as *CLUS cluster*) and *prediction phase* (labeled as *CLUS predict*). Similarly, the prediction process in *LinReg* is performed in two phases: *learning phase* and *prediction phase*, as explained in Section 5. Note that the clustering phase in *CLUS* and the learning phase in *LinReg* are performed only once prior to prediction phase.



(a) Prediction time, in dynamic environment



(b) Prediction time, in static environment

Fig. 3. The impact of density on computational performance of the prediction.

Dynamic environment. Fig. 3a captures prediction's computational performance in a dynamic environment. It is clear from the presented graphs that *CLUS* approach provides the best computational performance among all the competing approaches.

More specifically, it achieves two orders of magnitude lower prediction time when compared to *Hybrid* (e.g. *CLUS* clustering time of 44 ms and prediction time of 19 ms against *Hybrid* prediction time of 4,461 ms for density of 30 percent). In comparison to the model-based approach *LinReg*, *CLUS* clustering phase takes three orders of magnitude less time than learning phase in *LinReg* approach (e.g. learning phase takes 52,487 ms for density of 30 percent) while *CLUS* prediction phase takes two orders of magnitude less time than is required for the prediction phase in *LinReg* (e.g. prediction time of 1,144 ms for density of 30 percent). *CLUS* also clearly outperforms *LUCS*, when computational performance is regarded. Specifically, *CLUS* requires at least two orders of magnitude less time to compute the predictions than *LUCS* (e.g. prediction time of 1,554 ms for density of 30 percent).

Additional benefit of *CLUS* approach is that its clustering and prediction phases are relatively stable as the data density changes. By contrast, learning and prediction phases in *LinReg* approach, and prediction time in *Hybrid* and *LUCS* as well, depend on the data density.

The duration of the learning phase in *LinReg* grows linearly as the data density increases (e.g. from 3,820 ms for density of 5 percent to 85,166 ms for density of 50 percent, recall that graphs are in the logarithmic scale). Knowing the computational complexity entailed by linear regression, this result is quite expected (see details in Section 6.5). On the other hand, the duration of prediction phase in *LinReg* slightly drops with the increase of data density. This behavior is expected since the testing set contains a fewer amount of records to be predicted. The prediction time required in *LUCS* significantly drops (e.g. from 8,987 ms for density of 5 percent to 68 ms for density of 50 percent) as the amount of collected data grows for two main reasons. The first reason is the reduction of testing set size. The second reason is the fact that the 4-dimensional space used in *LUCS* [40], gets densely populated fast as the density increases, which results in less computation required to produce the predictions. The computational performance of *Hybrid* is better for lower densities (3,623 ms for density of 5 percent), since low amounts of data require less computation. With the increase of density, computation time grows (e.g. 4,461 ms for density of 30 percent). However, as density continues to increase, reliability needs to be predicted for fewer records which requires less computation (e.g. 3,870 ms for density of 50 percent).

Static environment. The evaluation results capturing impact of data density on prediction's computational performance in a static environment are depicted in Fig. 3b. The results are presented in the logarithmic scale for all the analyzed approaches. The evaluation results show that *CLUS* approach provides best computational performance among the competing approaches.

Specifically, *CLUS* achieves for at least an order of magnitude lower prediction time than *Hybrid* (e.g. *CLUS* clustering time of 41 ms and prediction time of 3 ms against *Hybrid* prediction time of 634 ms for density of 30 percent). Furthermore, *CLUS* clearly outperforms *LinReg* approach since it provides an order of magnitude lower clustering phase than is the learning phase in *LinReg* (e.g. learning phase of 205 ms for density of 30 percent), and the order of magnitude lower prediction phase than is required in *LinReg* (e.g. prediction phase of 22 ms for density of 30 percent). Finally, in comparison to *LUCS*, *CLUS* approach achieves better prediction time (e.g. prediction time of 25 ms required by *LUCS* for density of 30 percent).

CLUS also provides almost constant clustering and prediction time as the data density changes, while *LinReg*, *LUCS* and *Hybrid* manifest similar behavior like in the dynamic environment. Note, however, that in this case *LinReg* approach provides better prediction time than *Hybrid* which is not the case in the dynamic environment. This improvement in *LinReg*'s computational performance is caused by reduction of the number of training examples, and also by reduction of the number of used features (recall section 5, $m = 3$).

6.3 The Impact of Number of Clusters

We stated in Section 4 that *CLUS* approach tolerates an arbitrary number of user, service and environment condition clusters. Thus, each *number of clusters* in model parameters can be fine-tuned to a specific environment. In this section we inspect how number of clusters impacts prediction accuracy and computational performance. To examine this impact we assume a dynamic environment with different load levels. Like in the density impact evaluation, we include all the collected data in the testing set. During the evaluations, we simultaneously alter both number of user and service clusters, while the number of environment condition clusters is preserved at the constant value of 7. We start the evaluation with the initial value of two clusters. Next, we compute the reliability predictions and prediction performance measures. Then, we increase the number of clusters for the step value of 1 and we recalculate the predictions and measures. This procedure is repeated until the number of clusters reaches the value of 9. We provide the evaluation results separately for data densities of 10 and 25 percent.

6.3.1 The Impact on Prediction Accuracy

The impact of number of clusters on prediction accuracy is depicted in Figs. 4a and 4b. The figures capture prediction's accuracy results for the competing approaches: *CLUS*, *LUCS*, *LinReg* and a memory-based collaborative filtering representative—the *Hybrid* approach.

Fig. 4a captures *RMSE* values with respect to the number of clusters for the density of 10 percent. The performance of the *LUCS*, *LinReg* and *Hybrid* approaches is not influenced by varying the number of clusters, they achieve constant *RMSE* values of 0.066 (*LUCS*), 0.077 (*LinReg*), and 0.108 (*Hybrid*). Expectedly, prediction accuracy of *CLUS* is increased as the number of clusters grows (*RMSE* value of 0.121 for two clusters and *RMSE* of 0.091 for nine clusters). This behavior is expected since a greater number of clusters reduces the extent of aggregation, which in turn improves the prediction accuracy. It is evident that *CLUS* almost constantly (except for the initial number of clusters)

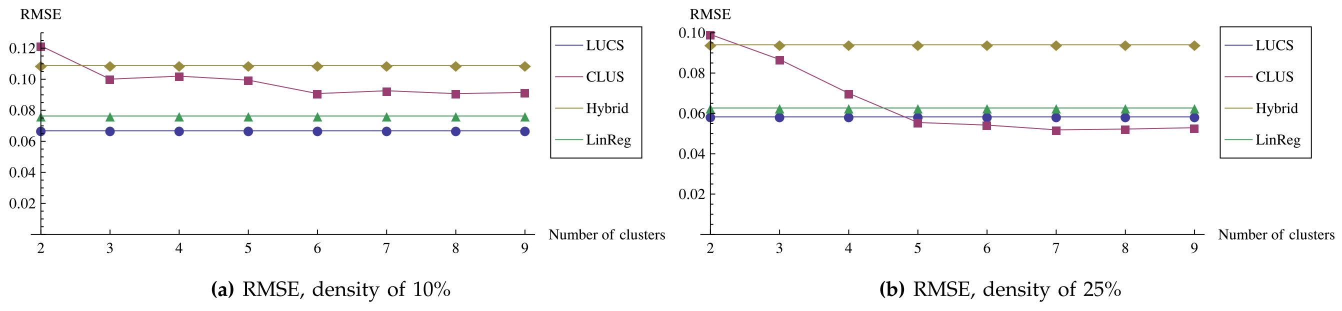


Fig. 4. The impact of number of clusters on prediction accuracy.

outperforms *Hybrid*. By contrast, it can not achieve the prediction accuracy of *LinReg* and *LUCS* at this density which adheres to the results presented in Section 6.2.1. Note that prediction accuracy is not improved by further increasing the number of clusters beyond the value of 7, which is attributed to seven highly distinct service classes in experimental settings.

The evaluation results for prediction accuracy at density of 25 percent are depicted in Fig. 4 b. Expectedly, prediction accuracy of *CLUS* approach increases as the number of clusters grows. Approaches *LUCS*, *LinReg* and *Hybrid* obtain constant *RMSE* values of 0.058, 0.062 and 0.094 respectively. In comparison to *Hybrid* approach, *CLUS* almost constantly (except for the initial number of clusters) produces more accurate predictions. As the number of clusters grows, *CLUS* outperforms *LUCS* and *LinReg* which complies to the results presented in Section 6.2.1.

6.3.2 The Impact on Computational Performance

The impact of number of clusters on prediction's computational performance is depicted in Figs. 5a and 5b. Each figure captures the aggregated prediction time for all considered approaches: *CLUS*, *LUCS*, *LinReg* and *Hybrid* in the logarithmic scale. The figures separately depict *clustering phase* and *prediction phase* in *CLUS*, and *learning phase* and *prediction phase* in *LinReg*.

Fig. 5a captures prediction time for the density of 10 percent in relation to the number of clusters. The performance of *LUCS* (prediction time of 7,786 ms), *LinReg* (learning time of 15,340 ms and prediction time of 1,338 ms) and *Hybrid* (prediction time of 3,883 ms) in not influenced by the number of clusters. Although the prediction time required in *CLUS* is slightly increasing (e.g. from 15 ms for two clusters to 21 ms for nine clusters), it is not significantly influenced by alterations in the number of clusters. By contrast, the clustering time increases as the number of clusters grows (from 14 ms for two clusters to 67 ms for nine clusters). For the density of 25 percent we obtain similar results, as shown in Fig. 5b. Expectedly, *Hybrid* and *LinReg* degrade, while *LUCS* improves its performance.

6.4 The Impact of Collaborative Feedback in Dynamic Environment

The researchers have shown that distributions of load conditions assembled over a longer period of time manifest clear regularities on a daily basis as described in [13], [14], [15], [16]. In *CLUS* we employ this fact to accurately predict the reliability for ongoing service invocations. Although assumed regularities in load distributions are certainly present, eventual deviations from the presumed load may cause the algorithm to produce less accurate predictions. To prevent potentially inaccurate predictions, we introduce the analysis of most recent feedback while determining the actual load conditions in the environment (see details in Section 4.4). To evaluate our strategy, we conducted two different experiments. In the first experiment, described in Section 6.4.1, we analyze how different amount of collected collaborative feedback improves the accuracy of prediction, while in the second experiment, described Section 6.4.2, we analyze how the size of the time window Δt for feedback assembly impacts the accuracy of prediction.

6.4.1 The Impact of Different Amount of Collaborative Feedback

In this section we assess how potential variabilities in the assumed load can be detected using our strategy of monitoring most recent feedback. We simulate real-time predictions by constructing a special test suite consisted out of 70 test cases, where each test case contains an amount of 1,225 different service invocations whose reliability values need to be predicted. We create a hypothetic load distribution that represents the *assumed* load during the entire test

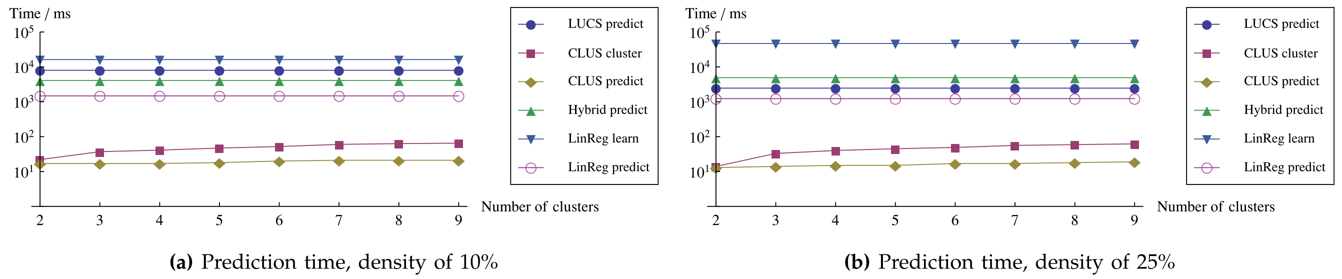
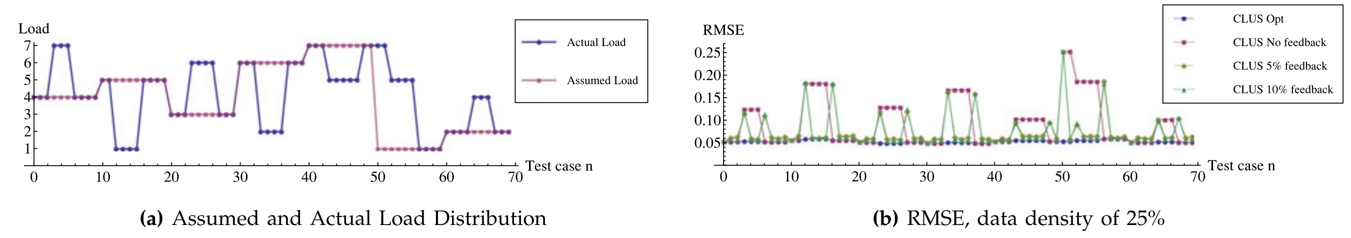


Fig. 5. The impact of number of clusters on computational performance of the prediction.



(a) Assumed and Actual Load Distribution (b) RMSE, data density of 25%

Fig. 6. The impact of the collaborative feedback on prediction accuracy in dynamic environment.

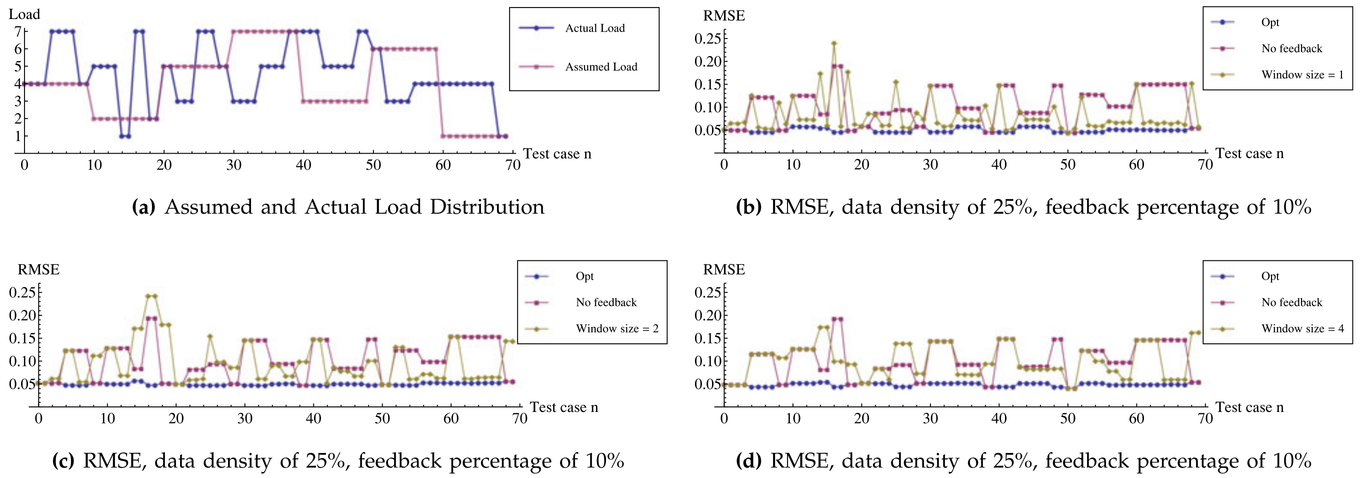
suite. However, to evaluate our strategy of real-time feedback assembling, we introduce random load deviations to define the *actual* load distribution during the entire test suite. Fig. 6a depicts both the *assumed* and *actual* load distributions for the testing suite. While producing predictions for ith test case, we utilize feedback assembled from the previous, $i-1$th test case to determine the actual load conditions (the exception is the first test case in which we preserve the *assumed* load). More specifically, we analyze how different percentages of randomly assembled feedback from the previous test case impact the accuracy of future predictions.

Fig. 6b depicts evaluation results presenting *RMSE* values over the entire test suite for different strategies regarding feedback in *CLUS* approach: strategy that assembles no recent feedback (labeled as *CLUS No feedback*), strategy that acquires 5 percent of the feedback from the previous test case, strategy that utilizes 10 percent of feedback from the previous test case, and for the *optimal* strategy that produces predictions according to the *actual* load. Note that *CLUS Opt* is a hypothetic strategy depicted only as a reference, it can not be reached in reality. It is obvious from the presented graphs that our strategy of assembling most recent feedback achieves promising performance. *CLUS Opt* always "knows" the *actual* load and it achieves the average *RMSE* value of 0.054 for the entire test suite. By contrast, *CLUS No feedback* produces predictions based on the *assumed* load with an average *RMSE* value of 0.0933 for the entire test suite. Expectedly, strategies that assemble feedback significantly improve their prediction accuracy compared to *CLUS No feedback* strategy having average *RMSE* values of 0.078 (for *CLUS 5 percent feedback*) and 0.075 (for *CLUS 10 percent feedback*). Moreover, it is visible from the graphs that strategy that assembles no feedback produces less accurate predictions for the test cases in which the *actual* load offsets from the *assumed* load, while strategies that incorporate feedback revise their predictions.

6.4.2 The Impact of Collaborative Feedback Time Window Size

In this section we analyze how the time window size, applied while acquiring collaborative feedback, effects the prediction accuracy. Similarly like in the previous experiment, we simulate real-time predictions by creating a dedicated test suite of 70 test cases, where each test case contains an amount of 400 service invocations whose reliability is to be predicted. In this experiment we set the data density to a value of 25 percent and we choose the feedback percentage strategy that utilizes 10 percent of feedback. As in previous experiment, a hypothetic load distribution is generated along with its random load deviations. Fig. 7a depicts both the *assumed* and *actual* load distributions for the testing suite. In this experiment, we vary the size of the time window during which the collaborative feedback is collected. The size of the time window is expressed as the amount of test cases during which the feedback is acquired. For instance, the time window size of k test cases means that the actual load for the following k test cases is determined based on feedback collected in the previous k test cases. The time window size simply determines the frequency of computing the actual load based on assembled feedback.

We provide the results for three different time window sizes: size of 1, 2 and 4 test cases which are presented in Figs. 7b, 7c and 7d respectively. As in the previous



(a) Assumed and Actual Load Distribution (b) RMSE, data density of 25%, feedback percentage of 10%

(c) RMSE, data density of 25%, feedback percentage of 10% (d) RMSE, data density of 25%, feedback percentage of 10%

Fig. 7. The impact of the time window size on prediction accuracy in dynamic environment.

experiment, we depict the *RMSE* scores over the entire test suite. Apart from the results for the strategy that uses the respected time window size, each figure depicts the results for the strategy that assembles no recent feedback (labeled as *No feedback*) and *optimal* strategy that produces predictions according to the *actual* load (labeled as *Opt*) which are depicted only as references. The strategy that uses window size of 1 test case achieves an average *RMSE* value of 0.081 for the entire test suite, which is at least 22 percent lower *RMSE* value when compared to the strategy that assembles no feedback. The results for the strategy that utilizes the window size value of two test cases show that the strategy that assembles feedback still outperforms the basic approach having an average *RMSE* value of 0.098 over the entire test suite. However, this results reveal a potential caveat that is present while predicting the actual load based on collaborative feedback. Specifically, the strategy that uses the window size value of four subsequent test cases provides slightly more accurate predictions (an average *RMSE* value of 0.102) than the basic approach that assembles no feedback. These results are quite expected knowing the actual load distribution depicted in Fig. 7a. As can be seen in the figure, the actual load is mainly changing more frequently than is the frequency of computing the load which is determined by the size time window size.

The evaluation results confirm that the most recent collaborative feedback can be utilized to improve the accuracy of prediction. However, it is important to choose the appropriate size of the time window during which the feedback is collected. In the environment in which the load conditions change very frequently, it is better to select a shorter time window, while a longer time window is more convenient for the environment in which the actual load conditions do not vary so frequently.

6.5 Computational Complexity Analysis

The evaluation results presented in preceding sections demonstrate that *CLUS* outperforms the competition regarding prediction's computational performance. However, to further support the obtained results, we provide the analysis of complexity for *CLUS*, *LUCS*, *LinReg* and memory-based collaborative filtering approaches: *IPCC*, *UPCC* and *Hybrid*.

The computational complexity implied by the *IPCC* is $O(n^2 \times m)$, while the *UPCC* has the complexity of $O(m^2 \times n)$ (n is the number of services and m is the number of users). Hence, the computational complexity required in the *Hybrid* is $O(n^2 \times m + m^2 \times n)$.

The complexity required in *LUCS* [40] is $O(n_u \times n_l \times n_c \times n_s \times (n_u + n_l + n_c + n_s))$, where n_u is the number of user locations, n_l is the number of loads, n_c is the number of services classes, and n_s is the number of service locations. Note that the expected number of groups in *LUCS* is significantly smaller than the individual number of services and users ($n_u, n_l, n_c, n_s \ll n, m$).

The computational complexity required in *LinReg* approach corresponds to the computational complexity of linear regression. From the complexity standpoint, most dominant operations in linear regression (recall equation (18)) are the computation of matrix $\boldsymbol{\Phi}^T \boldsymbol{\Phi}$ and the computation of its inverse. The matrix computation requires computational complexity of $O(M^2 \times N)$ and matrix inversion has the computational complexity of $O(M^3)$, where M is the number of features while N is the number of training examples. The number of training examples is proportional to the product of number of services and users $n \times m$, while the number of features M is constant and significantly smaller than the number of services and users ($n, m \gg M$). Hence, the practical case complexity required in *LinReg* is $O(n \times m)$.

In the data clustering phase of *CLUS*, we use K-means clustering algorithm [36]. First, we cluster different time windows, which requires the computational complexity of $O(i \times |E| \times |W| \times 1)$ (the vectors are one-dimensional), where i is the number of iterations, $|E|$ is the number of environment clusters and $|W|$ is the number of time windows. Then, we separately cluster users and services. The computational complexity required in user clustering is $O(i \times |U| \times |E| \times m)$ ($|U|$ and m are the numbers of user clusters and users), while the computational complexity of services clustering takes $O(i \times |S| \times |E| \times n)$ ($|S|$ and n are the numbers of service clusters and services). Note that values i, $|W|$ and $|E|$ are constant and we assume the number of clusters is relatively small compared to the number of users and services ($|U|, |S| \ll n, m$). Hence, we obtain the practical case computational complexity required in *CLUS* is $O(m + n)$. The presented analysis of complexity strongly supports our claims in favor of *CLUS* approach scalability improvement.

7 Conclusion

In this paper, we introduce *CLUS* and *LinReg*, novel approaches for prediction of atomic web services reliability, having two clear objectives: (1) to improve prediction's *accuracy*, and (2) achieve better *scalability* with respect to the state-of-the-art approaches.

The existing approaches implicitly consider only user- and service-specific parameters of the prediction. On the other hand, we incorporate the *environment-specific* parameters into the prediction which in turn significantly reduces the *RMSE* value. This can especially be observed for *CLUS* approach at higher data densities (e.g. it achieves 17 percent lower *RMSE* then *LUCS*). At lower densities *LUCS* outperforms *CLUS* and *LinReg*, but *LinReg* achieves 21 percent lower *RMSE* then *CLUS*.

Regarding scalability, in case of *CLUS* approach, it is achieved by grouping similar users and services considering their reliability performance using the K-means clustering algorithm. In this way *CLUS* reduces the execution time for at least two *orders of magnitude* when compared to the competing approaches. In case of *LinReg*, computationally demanding learning phase is performed only once, after which fast predictions are possible. This makes *LinReg* particularly applicable at lower data densities, despite the higher accuracy achieved by *LUCS*.

Finally, additional advantages of our approaches can be found in their flexibility which is manifested in a *compromise* between accuracy and scalability. Increasing the number of clusters in *CLUS* and complexity of the hypothesis function in *LinReg* yields more accurate predictions at the cost of computational performance.

ACKNOWLEDGMENTS

This work was supported by the Ministry of Science, Education, and Sports of the Republic of Croatia through the Computing Environments for Ubiquitous Distributed Systems (036-0362980-1921) research project.

REFERENCES

[1] T. Yu, Y. Zhang, and K.-J. Lin, "Efficient algorithms for web services selection with end-to-end qos constraints," *ACM Trans. Web*, vol. 1, 2007.

[2] T. H. Tan, E. André, J. Sun, Y. Liu, J. S. Dong, and M. Chen, "Dynamic synthesis of local time requirement for service composition," in *Proc. Int. Conf. Softw. Eng.*, 2013, pp. 542–551.

[3] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Secure Comput.*, vol. 1, no. 1, pp. 11–33, Jan.–Mar. 2004.

[4] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "Qos-aware middleware for web services composition," *IEEE Trans. Softw. Eng.*, vol. 30, no. 5, pp. 311–327, May 2004.

[5] D. Wang and S. T. KISHOR, "Modeling user-perceived reliability based on user behavior graphs," *Int. J. Rel., Qual. Safety Eng.*, vol. 16, p. 303, 2009.

[6] V. Cortellessa and V. Grassi, "Reliability modeling and analysis of service-oriented architectures," in *Proc. Test Anal. Web Services*, 2007, pp. 339–362.

[7] M. R. Lyu, ed., *Handbook of Software Reliability Engineering*. New York, NY, USA: McGraw-Hill, 1996.

[8] L. Cheung, L. Golubchik, and F. Sha, "A study of web services performance prediction: A client's perspective," in *Proc. IEEE 19th Int. Symp. Model., Anal. Simul. Comput. Telecommun. Syst.*, 2011, pp. 75–84.

[9] Z. Zheng and M. R. Lyu, "Collaborative reliability prediction of service-oriented systems," in *Proc. ACM/IEEE Int. Conf. Softw. Eng.*, 2010, pp. 35–44.

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

[11] L. Baresi and S. Guinea, "Event-based multi-level service monitoring," in *Proc. IEEE Int. Conf. Web Services*, 2013, pp. 83–90,.

[12] X. Su and T. M. Khoshgoftaar, "A survey of collaborative filtering techniques," *Adv. Artif. Intell.*, vol. 2009, pp. 4:2–4:2, Jan. 2009.

[13] Y. Wang, W. M. Lively, and D. B. Simmons, "Web software traffic characteristics and failure prediction model selection," *J. Comp. Methods Sci. Eng.*, vol. 9, pp. 23–33, 2009.

[14] Y. Baryshnikov, E. Coffman, G. Pierre, D. Rubenstein, M. Squillante, and T. Yimwadsana, "Predictability of web-server traffic congestion," in *Proc. Int. Workshop Web Content Caching Distrib.*, 2005, pp. 97–103.

[15] M. Andreolini and S. Casolari, "Load prediction models in web-based systems," in *Proc. Int. Conf. Perform. Eval. Methodolgies Tools*, 2006.

[16] Y.-T. Lee and K.-T. Chen, "Is server consolidation beneficial to mmorpg? A case study of world of warcraft," in *Proc. IEEE 3rd Int. Conf. Cloud Comput.*, 2010, pp. 435–442.

[17] M. R. Lyu, "Software reliability engineering: A roadmap," in *Proc. 2007 Future Softw. Eng.* 2007, pp. 153–170.

[18] L. Cheung, R. Roshandel, N. Medvidovic, and L. Golubchik, "Early prediction of software component reliability," in *Proc. Int. Conf. Software Eng.*, 2008, pp. 111–120.

[19] L. Cheung, I. Krka, L. Golubchik, and N. Medvidovic, "Architecture-level reliability prediction of concurrent systems," in *pROC. WOSP/SIPEW Int. Conf. Perform. Eng.*, 2012, pp. 121–132.

[20] G. Delac, M. Silic, and S. Srbljic, "A reliability improvement method for SOA-based applications," *IEEE Trans. Dependable Secure Comput.*, vol. PP, no. 99, pp. 1–1, 2014.

[21] V. Grassi and S. Patella, "Reliability prediction for service-oriented computing environments," *IEEE Internet Comput.*, vol. 10, no. 3, pp. 43–49, May/Jun. 2006.

[22] W. T. Tsai, D. Zhang, Y. Chen, H. Huang, R. Paul, and N. Liao, "A software reliability model for web services," in *Proc. Int. Conf. Softw. Eng. Appl.*, 2004, pp. 144–149.

[23] J. Ma and H.-p. Chen, "A reliability evaluation framework on composite web service," in *Proc. IEEE Int. Symp. Serv.-Oriented Syst. Eng.*, 2008, pp. 123–128.

[24] B. Li, X. Fan, Y. Zhou, and Z. Su, "Evaluating the reliability of web services based on bpel code structure analysis and run-time information capture," in *Proc. Asia Pac. Softw. Eng. Conf.*, 2010, pp. 206–215.

[25] L. Coppolino, L. Romano, and V. Vianello, "Security engineering of soa applications via reliability patterns," *J. Softw. Eng. Appl.*, vol. 4, no. 1, pp. 1–8, 2011.

[26] G. Delac, M. Silic, and K. Vladimir, "Reliability sensitivity analysis for yahoo! pipes mashups," in *Proc. MIPRO*, 2013, pp. 851–856.

[27] A. S. Das, M. Datar, A. Garg, and S. Rajaram, "Google news personalization: Scalable online collaborative filtering," in *Proc. Int. Conf. World Wide Web*, 2007, pp. 271–280.

[28] H. Guan, H. Li, and M. Guo, "Semi-sparse algorithm based on multi-layer optimization for recommendation system," in *Proc. Int. Workshop Program. Models Appl. Multicores Manycores*, 2012, pp. 148–155.

[29] C. Wei, W. Hsu, and M. L. Lee, "A unified framework for recommendations based on quaternary semantic analysis," in *Proc. ACM SIGIR Conf. Res. Devel. Inform. Retrieval*, 2011, pp. 1023–1032.

[30] R. Burke, "Hybrid recommender systems: Survey and experiments," *User Modeling User-Adapted Interaction*, vol. 12, no. 4, pp. 331–370, 2002.

[31] H. Ma, I. King, and M. R. Lyu, "Effective missing data prediction for collaborative filtering," in *Proc. ACM SIGIR Conf. Res. Develop. Inform. Retrieval*, 2007, pp. 39–46.

[32] 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.

[33] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proc. Int. Conf. World Wide Web*, 2001, pp. 285–295.

[34] Q. Yu, Z. Zheng, and H. Wang, "Trace norm regularized matrix factorization for service recommendation," in *Proc. Int. Conf. Web Serv.*, 2013, pp. 34–41.

[35] Z. Zheng and M. R. Lyu, "Personalized reliability prediction of web services," *ACM Trans. Softw. Eng. Methodol.*, vol. 22, pp. 12:1–12:25, Mar. 2013.

[36] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. New York, NY, USA: Springer-Verlag, 2006.

[37] J. Canny, "Collaborative filtering with privacy via factor analysis," in *Proc. ACM SIGIR Conf. Res. Develop. Inform. Retrieval*, 2002, pp. 238–245.

[38] S. Vucetic and Z. Obradovic, "Collaborative filtering using a regression-based approach," *Knowl. Inf. Syst.*, vol. 7, pp. 1–22, Jan. 2005.

[39] L. Yao, Q. Z. Sheng, A. Segev, and J. Yu, "Recommending web services via combining collaborative filtering with content-based features," in *Proc. IEEE 20th Int. Conf. Web Serv.*, 2013, pp. 42–49.

[40] M. Silic, G. Delac, I. Krka, and S. Srbljic, "Scalable and accurate prediction of availability of atomic web services," *IEEE Trans. Serv. Comput.*, vol. 7, no. 2, pp. 252–264, Apr. 2014.

[41] R. Xu and I. Wunsch, D., "Survey of clustering algorithms," *IEEE Trans. Neural Netw.*, vol. 16, no. 3, pp. 645–678, May 2005.

[42] M. Silic, G. Delac, and S. Srbljic,, "Prediction of atomic web services reliability based on k-means clustering," in *Proc. Joint Meet. Found. Softw. Eng.*, 2013, pp. 70–80.

[43] SmartBear. (2012). Load ui. [Online]. Available: http://www.loadui.org/

Marin Silic received the PhD degree in computer science from the Faculty of Electrical Engineering and Computing, University of Zagreb, Zagreb, Croatia, in 2013. He is currently a research and teaching assistant at the Faculty of Electrical Engineering and Computing, University of Zagreb. His current research interests include distributed systems, service-oriented computing, software engineering, and software reliability. He is a graduate student member of the IEEE.

Goran Delac received the PhD degree in computer science from the Faculty of Electrical Engineering and Computing, University of Zagreb, Zagreb, Croatia, in 2014. He is currently a research and teaching assistant at the Faculty of Electrical Engineering and Computing, University of Zagreb. His current research interests include distributed systems, parallel computing, fault tolerant systems and service-oriented computing. He is a graduate student member of the IEEE.

Sinisa Srbljic is currently a full professor at the Faculty of Electrical Engineering and Computing, University of Zagreb, Zagreb, Croatia, and the head of Consumer Computing Lab. He was a visiting scientist at the Huawei, Santa Clara, the GlobalLogic, San Jose, CA, the UC Irvine, CA, the AT&T Labs, San Jose, CA, and the University of Toronto, Canada. He is a computing visionary with unique experience in consumerization of innovation. He is a senior member of the IEEE.