Prediction of Atomic Web Services Reliability Based on K-means Clustering

Marin Silic
University of Zagreb, Faculty of Electrical Engineering and Computing, Unska 3, Zagreb
marin.silic@gmail.com

Goran Delac
University of Zagreb, Faculty of Electrical Engineering and Computing, Unska 3, Zagreb
goran.delac@gmail.com

Sinisa Srbljic
University of Zagreb, Faculty of Electrical Engineering and Computing, Unska 3, Zagreb
sinisa.srbljic@fer.hr

ABSTRACT

Contemporary web applications are often designed as composite services built by coordinating atomic services with the aim of providing the appropriate functionality. Although the functional properties of each atomic service assure correct functionality of the entire application, the nonfunctional properties such as availability, reliability, or security might significantly influence the user-perceived quality of the application. In this paper, we present *CLUS*, a model for reliability prediction of atomic web services that improves state-of-the-art approaches used in modern recommendation systems. *CLUS* predicts the reliability for the ongoing service invocation using the data collected from previous invocations. We improve the accuracy of the current state-of-the-art prediction models by considering user–, service– and environment–specific parameters of the invocation context. To address the computational performance related to scalability issues, we aggregate the available previous invocation data using K-means clustering algorithm. We evaluated our model by conducting experiments on services deployed in different regions of the Amazon cloud. The evaluation results suggest that our model improves both performance and accuracy of the prediction when compared to the current state-of-the-art models.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification

Keywords

Reliability, Prediction model, Clustering, Web services, Cloud computing

1. INTRODUCTION

The majority of modern web applications pull content and functionalities through atomic modules embedded from various sources located worldwide. Although the atomic modules are usually built using different technologies, the implementation is often guided by the set of rules and principles defined in Service-oriented architecture (SOA) [1]. The SOA is an architectural style mostly used for design and implementation of information systems that support various enterprise solutions, but is also commonly applied for more general applications. The basic application functionalities (e.g. *process-payment*, *new-order*, *fetch-video*, *retrieve-social-feed*), can be implemented as reusable atomic services accessible through publicly available interfaces. To support more advanced application functionalities, SOA provides the ability to compose the atomic services into composite ones. The researchers have already proposed a variety of service composition languages such as: *BPEL* [2], *SSCL* [3].

The process of composite application construction requires qualitative selection of atomic service candidates [4]. The perceived performance of the entire application depends on both functional and nonfunctional application attributes, which are highly influenced by the selected atomic candidates' functional and nonfunctional qualities. To create an efficient composite application, the developer should gain reliable information on both atomic candidates' functionalities and their nonfunctional attributes such as security, reliability, availability [5].

In this paper, we focus on atomic web services reliability as one of the most important nonfunctional application properties. We define service reliability as the probability that a service invocation will be completed *successfully*. This means a *correct* response to the service invocation is successfully received under the specified conditions and time constraints. In this case, the reliability can be computed from the history invocation data as the ratio of the number of successful invocations against the total number of performed invocations. In the related literature this definition of reliability also appears as *successful delivery rate* [6] and *user-perceived reliability* [7]. It should be noted that the applied *user-centric* definition of reliability differs from the traditional *system-centric* reliability definition used for "never ending" systems [8]. However, we find the definition of "reliability on demand" is more suitable for web services since service invocations are discrete and relatively sparse events.

The accuracy and relevance of the computed reliability value depends on the quality and quantity of the past invocation

sample. However, gaining a numerous and diverse past invocation sample proves to be a very difficult task in practice. There is a difference in reliability perception from the user's and service provider's perspective [8]. The reliability value computed considering exclusively the data obtained by the service provider could be incorrect for a specific user due to oscillations caused by a variety of parameters that influence the invocation context. For example, users in different geographic locations might experience different reliabilities while using the same service. From the service user's perspective, further obstacles are related with the service usage cost and performance issues. For example, collecting the invocation sample by performing service reliability testing can be extremely expensive for the services that are not free of charge. On the other hand, conducting "stress testing" can significantly impact the performance of the service and also make the measured data irrelevant [9].

One possible approach to overcome these obstacles is to obtain *partial* but relevant history invocation sample (1) and to utilize *prediction* methods for the estimation of missing reliability records (2). The partial invocation sample can be gained by leveraging human feedback regarding service usage and collecting as much data as possible from the service providers. The researchers have proposed several prediction models that leverage the available past invocation records to estimate predictions for the ongoing invocations [10–13]. The proposed *state-of-the-art* prediction models are based on collaborative filtering technique, often used in recommendation systems [14]. Collaborative filtering based models compute statistical similarities among different entities and estimate the missing reliability values by employing the existing data from the statistically most similar entities.

Although the existing collaborative filtering based approaches achieve promising performance, they demonstrate disadvantages primarily related to the prediction *accuracy* in dynamic environments and *scalability* issues influenced 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. *movie 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 [15]. 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 present, *CLUS (CLUStering)*, the model for reliability prediction of atomic web services based on *K-means clustering* algorithm [16]. The proposed model aims to improve performance of the state-of-the-art prediction models by addressing the following issues:

- To improve the prediction *accuracy*, we consider user–, service– and environment– specific parameters that determine the service invocation context. While existing approaches implicitly consider only user– and service–specific parameters, we introduce the environment specific parameters that describe current load conditions in the environment. We disperse the collected past invocation data across the additional dimension that corresponds with the respected environment conditions.
- To address the *scalability* issues, we reduce the redundant data by grouping users and services into respected user and service clusters according to their reliability performance using K-means clustering algorithm. The authors in [17, 18] show that different nonfunctional qualities of a service are influenced by specific service characteristics such as internal complexity or service location. In fact, the existing approaches [10, 11] support claims that similar users and services obtain similar reliability values, which can be utilized to aggregate the redundant data and improve scalability.

For evaluation purposes, we perform series of experiments varying different parameters that describe the service invocation context. We measure different performance aspects of our model and compare them with the prediction performance of the current state-of-the-art models. The evaluation results suggest that our model significantly improves the prediction accuracy with 56% lower *RMSE (Root Mean Square Error)* than the current state-of-the-art approaches based on collaborative filtering [13]. Furthermore, the evaluation results support our claims of scalability improvement. The presented *CLUS* model reduces the *prediction execution time* for two orders of magnitude compared to other state-of-the-art approaches.

The rest of the paper is organized as follows. Section 2 assembles the related work. Section 3 overviews prediction process and describes parameters used in our model. Section 4 formally describes each particular *CLUS* step. Section 5 describes the performed experiments and analyzes the evaluation results. Section 6 concludes the paper by summarizing advantages and disadvantages of the presented approach.

2. RELATED WORK

The researchers have proposed a variety of different approaches for modeling the reliability of traditional software systems [19–27]. As stated in section 1, web services provide their functionalities in a dynamic environment (interfaces accessible over the Internet) where the service invocation outcome depends on numerous different impacts determining the invocation context. Hence, these traditional approaches are not suitable for modeling the reliability of web services.

When considering the reliability properties of service-oriented systems, most of the researchers commonly focus on studying the reliability of service compositions. In such manner, plenty of different approaches for prediction of the composite services reliability have been proposed [28–34]. These existing approaches usually assume the reliability values of the atomic services are available or scarcely indicate how they can be acquired. However, the arguments mentioned in section 1 indicate that collecting a comprehensive sample of atomic services reliability values is a nontrivial task.

The most successful approaches for prediction of atomic services reliability [10–13] are based on the collaborative filtering technique [14]. There are three types of collaborative filtering defined in literature [14], *memory-based*, *model-based* and *hybrid*. The model-based and hybrid collaborative filtering are more computationally complex and difficult to implement. In fact, these types often require some additional domain specific information to be applied in a particular field. In the paper, we refer to the memory-based collaborative filtering used in current state-of-the-art recommendation systems [35–39]. The memory-based collaborative filtering extracts the information or patterns by statistically correlating the data obtained from multiple entities like agents, viewpoints or data sources. The advantage of collaborating filtering is that lacking information for a particular entity can be predicted using the available data from the most statistically similar entities.

The collaborative filtering technique 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, having new user-service pairs arise in real time. Furthermore, each user often consumes only a small subset of services. Hence, the user-item matrix is very sparsely filled and contains numerous empty cells presenting reliability values that need to be predicted.

Collaborative filtering can be applied in two different ways. The *UPCC* approach combines the information collected from different users and predicts missing reliability values using the available data from the most statistically similar users [10]. The *IPCC* approach collects the data from different services and predicts missing reliability values based on available values from the most statistically similar services [11]. The *Hybrid* approach [12, 13] 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* approaches. As noted in section 1, these approaches suffer from potential accuracy as well as scalability issues.

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.

To address this issue, we propose a new model that clusters users, services and time windows according to the perceived reliability performance. In this way, the lack of input parameters does not degrade the prediction performance, since the parameters are extracted from the past invocation sample based on the reliability performance.

3. RELIABILITY PREDICTION OVERVIEW

In this section we provide the overview of the *CLUS*, model for prediction of atomic web services reliability. With the aim to improve the prediction accuracy and scalability, we define user–, service– and environment specific parameters that determine service invocation context in greater detail then the related prediction models. We group the collected history invocation data across three different dimensions associated with the defined parameters. The rest of the section describes the parameters that determine service invocation context (Section 3.1) and present the reliability prediction process in *CLUS* model (Section 3.2).

3.1 Invocation context parameters

In our model we distinguish three groups of parameters that impact the reliability performance of the service: 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 s service such as user's location, network and device capabilities, usage profiles. To incorporate user-specific parameters in the process of prediction, we group users into clusters according to their reliability performance gained from the past invocation sample using K-means clustering algorithm.

3.1.2 Service-specific parameters

Service-specific parameters are related with 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 in 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 nonfunctional 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 day [15], 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 Figure 1. Prior to reliability prediction, we perform the 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. Then,

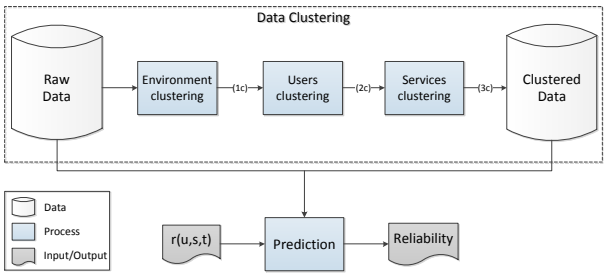


Figure 1: Reliability prediction overview

we cluster users (2c) and services (3c) considering their reliability performance for each time window cluster. Once the data is clustered, the prediction of the atomic service reliability can be performed.

4. RELIABILITY PREDICTION MODEL

In this section we present *CLUS*, the model for atomic web services reliability prediction. We separately describe each step of data clustering process that is crucial for the reliability prediction.

We formally define a service invocation as:

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

In the presented equation (1), u is the user executing the invocation, s is the service to be invoked and t is the actual time of the service invocation.

The history invocation sample contains data addressed as in the equation (1). In order to make scalable and accurate reliability predictions for future service invocations, we need to transform the data into a more structured and compact form. The idea is to store the data into a three-dimensional space $D[u, s, e]$ according to the defined groups of parameters. Each dimension u, s and e in space D is associated with one group of parameters respectively. In the following sections we will describe how particular records are clustered and associated with environment–, user– and service–specific 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:

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

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 aim 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, the analyses of the collected data from different service providers can pinpoint the regularities in the load distribution for certain time periods [15, 41, 42]. 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 the same time window. Once the time windows are determined, we calculate the average reliability value $\overline{r_{w_i}}$ 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.

The average reliability value $\overline{p_{w_i}}$ is assigned to each respected time window w_i. Each particular time window is clustered using K-means clustering algorithm into an appropriate environment condition e_i according to its average reliability value. Now we can correlate each particular record from the history invocation sample with the environment conditions at the time it was performed. By applying a 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, than 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 different user groups U as follows:

$$U = \{u_1, u_2, ..., u_i, ..., u_m\}, \tag{4}$$

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:

$$\boldsymbol{p_u} = \{\overline{p_{e_1}}, \overline{p_{e_2}}, ..., \overline{p_{e_i}}, ..., \overline{p_{e_n}}\}, \tag{5}$$

where each vector dimension $\overline{p_{e_i}}$ represents the average reliability value perceived by the given user u during the environment conditions e_i. Once the 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. 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:

$$S = \{s_1, s_2, ..., s_i, ..., s_l\}, \tag{6}$$

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:

$$\boldsymbol{p_s} = \{\overline{p_{e_1}}, \overline{p_{e_2}}, ..., \overline{p_{e_i}}, ..., \overline{p_{e_n}}\}, \tag{7}$$

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 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. 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 prediction

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

$$D[u_k, s_j, e_i] = \frac{1}{|R|} \sum_{r \in R} p_r. \tag{8}$$

where p_r is user perceived reliability for invocation r and:

$$R = \{r(u, s, t) | r \in u_k \wedge r \in s_j \wedge r \in e_i\} \tag{9}$$

Now, let us assume we have the ongoing service invocation $r_c(u_c, s_c, t_c)$ whose reliability p_c needs to be predicted. First, we check if there is a set H in the past invocation sample containing record with the same invocation context parameters as r_c:

$$H = \{r_h | u_h = u_c \wedge s_h = s_c \wedge t_h, t_c \in w_i\}. \tag{10}$$

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

$$p_c = \frac{1}{|H|} \sum_{r \in H} p_r. \tag{11}$$

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

$$p_c = D[u_k, s_j, e_i], \tag{12}$$

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.

5. EVALUATION

In this section we present the evaluation results that support our claims that the *CLUS* model improves the prediction accuracy and overcomes scalability issues present in the state-of-the-art models used in modern recommendation systems. To prove our claims we conducted series of experiments analyzing several quality aspects of the *CLUS* model. We compare our model with three approaches based on the collaborating filtering technique: *user-based* approach (*UPCC*) [10], *item-based* approach (*IPCC*) [11] and the *Hybrid* approach [12, 13]. To predict the missing reliability values, collaborative filtering based approaches calculate most similar entities using *Pearson Correlation Coefficient (PCC)*. The *UPCC* approach employs the available information from most similar users, while *IPCC* approach employs the available information from the most similar services. The *Hybrid* approach employs both the available information from similar users and similar services. Our results suggest that *CLUS* provides best prediction performance among the competing approaches considering both prediction accuracy and computational performance.

To evaluate prediction accuracy for atomic services, we use standard error measures: *Mean Absolute Error*, (*MAE*) and *Root Mean Square Error* (*RMSE*). The *MAE* represents the average magnitude of errors for the predicted reliability values:

$$MAE = \frac{\sum_j^N |p_j - \hat{p_j}|}{N} \tag{13}$$

where N is the cardinal number of the prediction set, p_j the reliability fetched from the data collected in experiment while $\hat{p_j}$ is the predicted reliability value. The *RMSE* is a quadratic scoring rule which also represents average magnitude of errors:

$$RMSE = \sqrt{\frac{\sum_j^N (p_j - \hat{p_j})^2}{N}} \tag{14}$$

To evaluate the impact of propagation of atomic service prediction errors on the reliability of the composite service, we apply a more suitable percentage measure *Root Mean Square Percentage Error* (*RMSPE*). The *RMSPE* is a quadratic scoring rule and it measures average percentage error magnitude:

$$RMSPE = \sqrt{\frac{1}{N} \sum_j^N \frac{\left(P_j - \hat{P_j}\right)^2}{P_j}} \tag{15}$$

where P_j is the composite service reliability calculated as described in [43] using reliability values measured in the experiment, while $\hat{P_j}$ is the composite reliability calculated using predicted reliability values.

All, *MAE*, *RMSE* and *RMSPE*, can range from 0 to ∞. They are negatively-oriented scores, which means that lower values are better.

The rest of the section is organized as follows. Section 5.1 describes the experimental setup. Section 5.2 analyses the impact of the amount of collected data on the accuracy and computational performance of the prediction for each competing approach. Section 5.3 studies the impact of the number of clusters on prediction accuracy and timing performance of the *CLUS* approach. Section 5.4 examines how errors in prediction of atomic services reliabilities impact the estimation of the composite service reliability. Section 5.5 provides the analysis of theoretical worst case complexity as well as the expected practical complexity for the different prediction approaches.

5.1 Experimental setup

In order to evaluate different properties of *CLUS*, we constructed a controlled environment containing a set of web services. We implemented web services that create two random matrices and perform the operation of matrix multiplication. By implementing our own services, we were able to reduce the external noise and control the service specific parameters such as service locations and complexity as well as the environment conditions such as different loads.

In our experiments we achieved different service-specific pa-

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

Table 1: Matrix ranks in different service classes

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

Table 2: Time intervals in different load levels

rameters by implementing services with different computational complexity and by placing services on different geographic locations worldwide. Although a variety of parameters influence the computational complexity of services, for the reasons of simplicity, we chose the amount of memory to distinguish services by their computational complexity. Any other parameter could be considered in similar manner and this does not reduce the generality of our experiments. We created seven different classes of services for matrix multiplication having each class operate with matrices of different rank as shown in Table 1.

To introduce another service-specific parameter we placed 49 web services in seven available *Amazon EC2 (Elastic Compute Cloud)* [44] geographic regions: *Ireland*, *Virginia*, *Oregon*, *California*, *Singapore*, *Japan* and *Brazil*, having each service class in each region. Each service was deployed on an Amazon machine image running *Microsoft Windows Server 2008 R2 SP1 Datacenter edition*, 64-bit architecture, *Microsoft SQLServer*, *Internet Information Services 7* and *ASP.NET 3.5*.

To introduce user-specific parameters, we simulate users by placing instances of *loadUI* [45] tool, running as distributed agents waiting for the tasks to be delivered and executed, on different locations within the cloud. The tool *loadUI* is an open source tool intended for stress and performance testing of web services which enables creation of various test cases.

We achieved different environment-specific parameters by creating test cases with different load generators defined by the *time interval* between sending each invocation. We burdened the services with seven different levels of load by altering the time interval between sending each request as defined in Table 2. We created a special test case for each load level and delivered it to all agents running in the cloud. During the test case, each agent sent 150 requests to each deployed service. Finally, we collected measured data from the agents and restarted the machines hosting web services to recover for the next test case. As part of our experiments, we performed overall 360.150 distinct web service invocations.

5.2 The impact of collected data density

In this section we study the impact of data density on the prediction performance for all considered approaches. Section 5.2.1 analyses the impact of data density on prediction accuracy while section 5.2.2 analyses 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% of all the data collected in experiments with the step value of 5%. We distinguish two different cases regarding the environment-specific parameters. In the first case we consider dynamic environment with different loads having request frequencies from *req/3 sec* to *req/9 sec*. In the second case we consider a static environment with constant load having request frequency of *req/3 sec*.

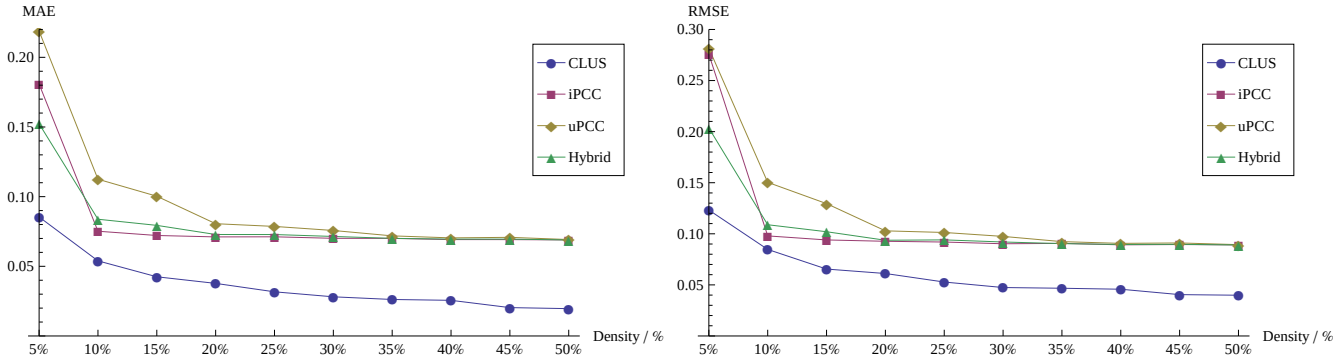
While describing the *CLUS* model (Section 4), we indicated an arbitrary number of user, service and environment conditions clusters. However, in this experiment we assume that each parameter space has 7 clusters, which is related to the number of service classes and load settings in the experimental setup.

To evaluate different aspects of prediction performance we create the testing set consistent of all the collected data for both cases. We randomly include 5% of the collected data into the prediction data set. Once the data is included into the prediction set, we predict the reliability for all the missing records for all competing approaches. Since the actual reliability values have been measured during the experiments, we calculate the *MAE* and *RMSE* values. In addition, we measure the time that it takes to compute the predictions for all approaches. In the next step, we include another 5% of the collected data i to the prediction data set and we recalculate the reliability predictions and prediction performance measures. We repeat this procedure until the calculation is done for the data density of 50%.

5.2.1 Prediction accuracy

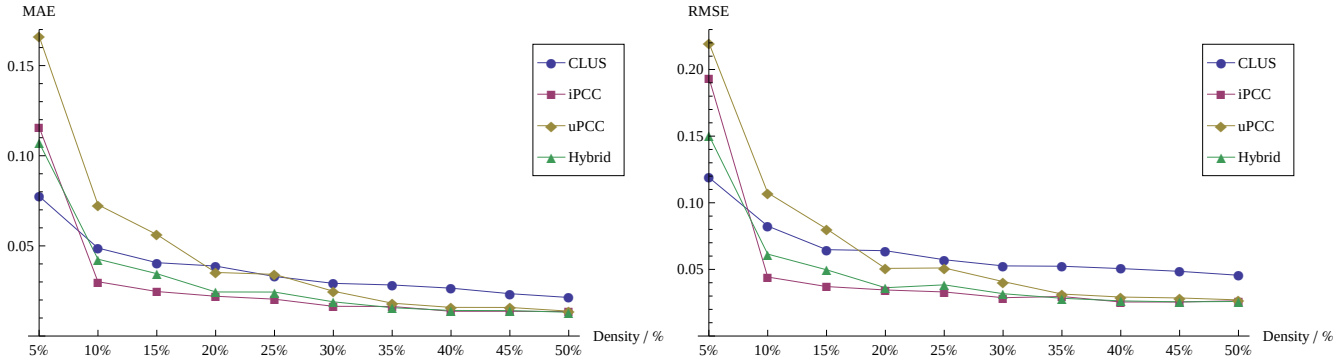
The evaluation results in the case of dynamic environment are depicted in Figures 2a and 2b. The Figure 2a captures the *MAE* values, while the Figure 2b captures the *RMSE* values in relation to the data density for all the analyzed approaches. The results show that the prediction accuracy highly depends on the data density (e.g. for the *IPCC* approach the *RMSE* value varies between 0.281 and 0.089). It is obvious from the presented figures that *CLUS* approach provides the best prediction accuracy. As can be expected, *CLUS* improves its prediction accuracy as the density increases lowering the *MAE* values from 0.085 to 0.019 and *RMSE* values from 0.123 to 0.039. It should be noted that our model achieves a slightly better *RMSE* value of 0.084 for the density of 10%, than the competing *Hybrid* approach for the density of 50% with *RMSE* value of 0.089. All collaborative filtering based approaches achieve similar prediction accuracy that cannot be considerably improved with the increase of the collected data due to negligence of environment's dynamic nature and inefficient usage of the collected data.

The Figures 2c and 2d depict the evaluation results in a static environment, capturing the *MAE* and *RMSE* values for all the competing approaches in relation to the data density. As can be expected, all the approaches improve prediction accuracy with the increase of data density. The *CLUS* approach provides better prediction accuracy for the lower data densities (e.g. the *RMSE* value of 0.119 against *Hybrid RMSE* value of 0.150 for the density of 5%). As the data density increases, the collaborative filtering approaches provide better prediction accuracy (e.g. *CLUS RMSE* value of 0.045 against *Hybrid RMSE* value of 0.025 for the density of 50%). This result is unsurprising since our *K-means clustering* based approach aggregates the collected data by clustering users and services which degrades the prediction accuracy. Note that lower data densities are more realistic on the Internet, the environment with substantially large number of users and services.



(a) MAE, with load intensity

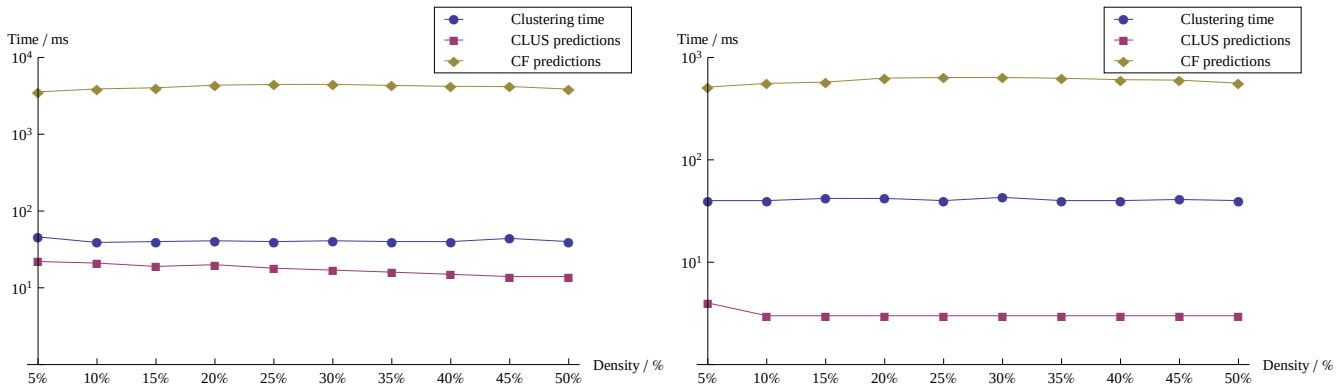
(b) RMSE, with load intensity



(c) MAE, without load intenisty

(d) RMSE, without load intenisty

Figure 2: The impact of density on prediction accuracy



(a) Prediction time, with load intensity

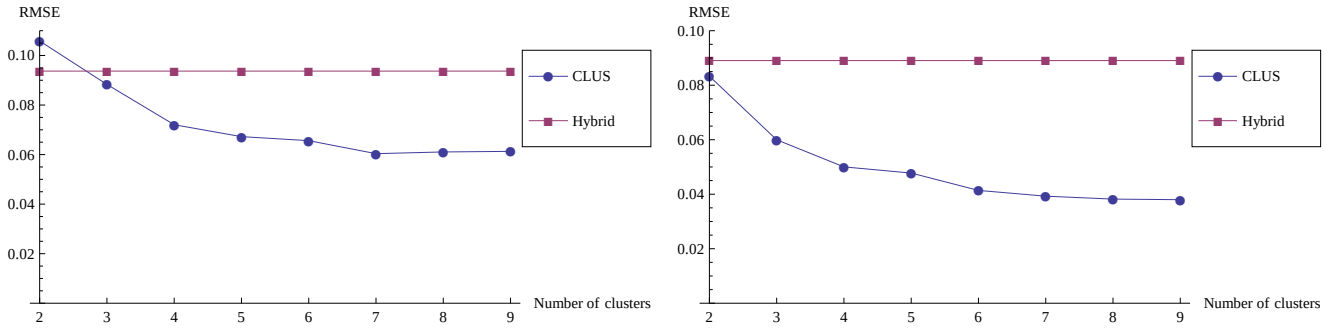
(b) Prediction time, without load intensity

Figure 3: The impact of density on timing performance of the prediction

5.2.2 *Computational performance*

The evaluation results for computational performance are captured in Figures 3a and 3b. We chose the execution time as the measure for computational performance. The figures depict aggregated *prediction time* for the whole testing set in *milliseconds* in relation to the data density for the *collaborative filtering* and *CLUS* approaches in the logarithmic scale. Since all the collaborative filtering based approaches have similar analytical complexity (see section 5.5), we choose the *Hybrid* approach as the representative for the computational performance. The *CLUS* prediction process is done in two phases *data clustering* and *prediction calculation* as depicted in figure (1). Thus, we present both *clustering time* and *prediction time* for the *CLUS* approach. Note that the data clustering phase is done only once prior to the prediction phase.

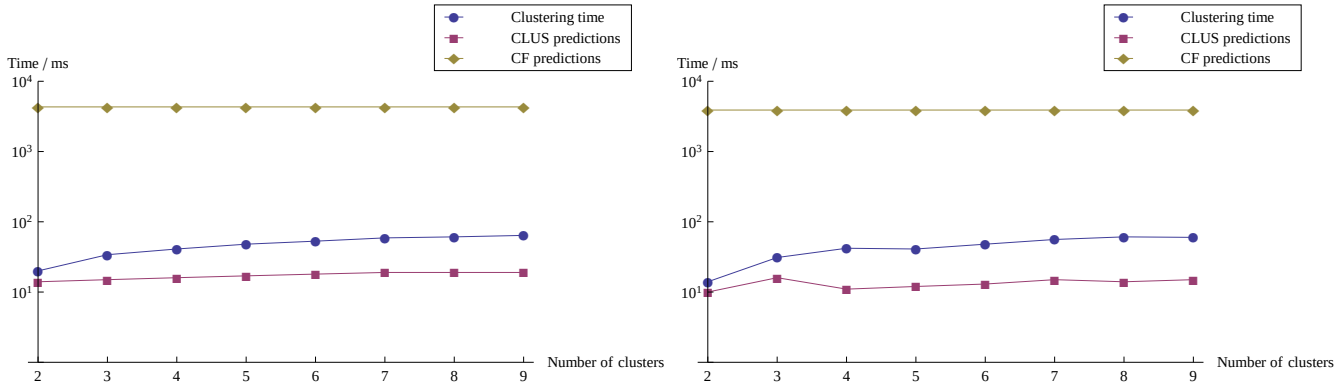
Figure 3a depicts computational performance evaluation for



(a) RMSE, density of 20%

(b) RMSE, density of 50%

Figure 4: The impact of number of clusters on prediction accuracy



(a) Prediction time, density of 20%

(b) Prediction time, density of 50%

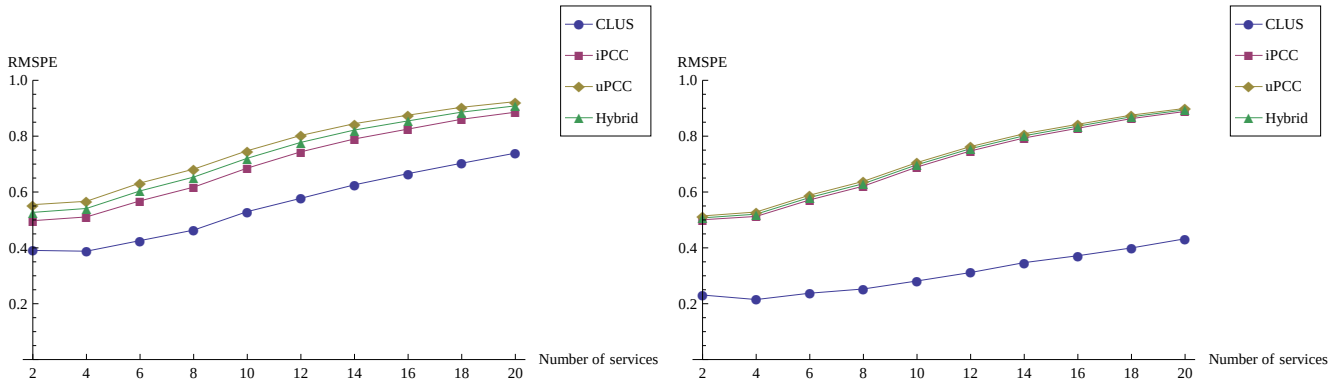
Figure 5: The impact of number of clusters on timing performance of the prediction

the competing approaches in the case of a dynamic environment. It is obvious from the presented graphs that the *CLUS* approach provides better performance for the two orders of magnitude (e.g. prediction time of 4454 *ms* for collaborative filtering approach against *CLUS* clustering time of 41 *ms* and prediction time of 17 *ms* for the data density of 30%). Note that both *CLUS* clustering and prediction times are relatively stable as the data density changes. On the other hand, the collaborative filtering approach prediction time depends on the data density. For the lower data densities the computational performance is better (prediction time of 3562 *ms* for the data density of 5%) since the low amounts of the collected data require less computation. With the increase of the collected data, the computation time increases as can be expected (e.g. prediction time of 4454 *ms* for the density of 30%). As the amount of the collected data continues to increase, the number of records with available reliability values grows. Thus, the reliability value needs to be predicted for fewer records which in turn results in decrease of the prediction time (e.g. prediction time of 3872 *ms* for the density of 50%).

The computational performance of the prediction in the case of a static environment is presented in Figure 3b. Like in the case of a dynamic environment, the performance is presented in the logarithmic scale. The evaluation results show that *CLUS* approach provides better performance for the order of magnitude (e.g. prediction time of 638 *ms* for collaborative filtering approach against *CLUS* clustering time of 41 *ms* and prediction time of 3 *ms* for the data density of 30%). The *CLUS* approach provides almost constant clustering and prediction time as the data density changes, while collaborative filtering approaches manifest similar behavior like in the case of a dynamic environment.

5.3 The impact of the number of clusters

As described in Section 4, *CLUS* supports an arbitrary number of user, service and environment conditions clusters. The *number of clusters* is a model parameter which can be adjusted to a specific environment. In this section we analyze how number of clusters impacts different aspects of the prediction performance. To evaluate the impact of number of clusters we consider dynamic environment conditions with various load levels. Similarly like with the density evaluation, we include all the collected data in the testing set for evaluation. In the evaluation process, we vary the number of user and service clusters keeping the number of environment conditions clusters constant at value of 7. We choose the initial value of 2 for the number of user and service clusters. Then, we calculate the reliability predictions and prediction performance measures. In the next step, we increase the



(a) Density 20%

(b) Density 50%

Figure 6: The propagation of prediction error on the service composition reliability in relation with number of atomic services

number of clusters for the step value of 1 and we recalculate the predictions and measures. The procedure is repeated until the number of clusters reaches the value of 9. We separately evaluate the impact of the number of clusters for the data densities of 20% and 50%.

5.3.1 Prediction accuracy

The evaluation results capturing the impact of clusters number on prediction accuracy are shown in Figure 4. Each subfigure presents prediction accuracy for the *CLUS* approach and a collaborative filtering representative - the *Hybrid* approach. Figure 4a depicts the *RMSE* values in relation to the number of clusters for the data density of 20%. The performance of the *Hybrid* approach is not influenced by altering the number of clusters. It achieves the constant *RMSE* value of 0.072 for the density of 20%. As can be expected, the prediction accuracy of the *CLUS* is increased as the number of clusters grows (the *RMSE* value of 0.105 for 2 clusters and the *RMSE* of 0.060 for 7 clusters). In fact, the greater number of clusters means less aggregation which improves the prediction accuracy. Note that further increase in the number of clusters after the value of 7 does not improve accuracy. This behavior can be explained by the experimental setup with 7 highly distinct service classes. The evaluation results of the prediction accuracy for the density of 50% are depicted in Figure 4b. Similarly, like for the density of 20%, the prediction accuracy is increased as the number of clusters grows (the *RMSE* value of 0.083 for 2 clusters and the *RMSE* of 0.037 for 9 clusters). The *Hybrid* approach achieves the *RMSE* value of 0.890.

5.3.2 Computational performance

The impact of cluster number on computational performance of the prediction is presented in Figure 5. Each subfigure depicts prediction time aggregated on the whole testing set for both *CLUS* approach and collaborative filtering representative - the *Hybrid* approach. The results are shown in the logarithmic scale. Note that figures separately capture *clustering* and *prediction* time for each phase of the *CLUS* approach. Figure 5a depicts prediction time for the density of 20% in relation to the number of clusters. The *Hybrid* approach timing performance is not influenced by the number of clusters and it achieves the prediction time of 4317 ms. The *CLUS* prediction time is relatively stable and ti is not influenced by altering the number of clusters. On the other hand, the clustering time increases as the number of clusters grows (from 20 ms for 2 clusters to 64 ms for 9 clusters). Knowing the computational complexity of the *K-means clustering* (see section 5.5), this behavior is quite expected. Similar results are obtained for the data density of 50% as shown in Figure 5b. Collaborative filtering approach achieves slightly better performance compared to prediction time for the density of 20% which is an expected result (see Section 5.2.2 for explanation).

5.4 Impact of error propagation on composite service reliability

In previous sections we studied the prediction accuracy and performance of the competing approaches considering exclusively atomic services. The real world examples of service usage imply orchestration and coordination of more atomic services into composite services to support more advanced functionalities. The reliability of a composite service highly depends on reliability of each atomic service building block. Knowing the atomic services reliability values, composite service reliability can be computed as described in [43]. In this section we analyze the impact of errors in atomic services reliability prediction on the computation of composite service reliability. We consider the case of a dynamic environment with varying loads and the constant number of *CLUS* clusters, set to the value of 7.

In order to analyze the propagation of errors on composite service reliability estimation, we begin with number of 1000 randomly generated service compositions containing 2 atomic services composed into different basic structural patterns as described in [43]. We compute each service composition reliability value using reliability predictions of the underlaying atomic services and we calculate the *RMSPE* value for each considered prediction approach. In the next step we increase the number of atomic services in service compositions for the step value of 2 and we recalculate the predictions and the *RMSPE* values. We repeat this procedure until the calculation is done for 20 of atomic services in the compositions.

The Figures 6a and 6b depict the impact of error propagation on composite service reliability in relation to the number of atomic services within a service composition. The Figure 6a captures the *RMSPE* values for each competing approach for the density of 20%, while Figure 6b captures the *RMSPE* values for the density of 50%. As can be expected, the accuracy of prediction for the service compositions decreases as the number of atomic services within a composition increases (e.g. the *CLUS* approach at the density of 20% has the *RMSPE* value between 0.390 for 2 atomic services, and 0.739 for 20 atomic services). It is obvious from the graphs that *CLUS* approach outperforms all the other considered approaches. However, the analysis demonstrates that even minor errors in prediction of atomic services reliability result in significant errors when calculating the reliability of a service composition.

5.5 The analysis of complexity

We propose the *CLUS* approach with two main goals: to improve the *accuracy* of the reliability prediction (1), and to improve the *scalability* of the prediction (2). The evaluation results presented in previous sections confirm that our approach achieves both of the goals. In this section, we provide the analysis of complexity for the *CLUS* approach and collaborative filtering based approaches: *IPCC*, *UPCC* and the *Hybrid* approach.

The computational complexity required in the *IPCC* approach is $O(n^2 \times m)$, where n is the number of services and m is the number of users. Similarly, *UPCC* approach has the computational complexity of $O(m^2 \times n)$. Thus, the computational complexity required in the *Hybrid* approach is $O(n^2 \times m + m^2 \times n)$. Note that real service oriented systems on the Internet may comprise out of millions of different users and services which results in serious *scalability* and *data sparsity* issues for collaborative filtering based approaches.

In general, the computational complexity required for *K-means clustering* is $O(i \times c \times d \times n)$, where i is the number of iterations performed by the procedure, c is the number of clusters, n is the number of vectors to be clustered and d is the dimensionality of vectors [16]. Although the theoretical worst case may take exponential time for the algorithm to converge [46], in practical case with data points representing service reliabilities, the algorithm converges very quickly. However, in the K-means clustering implementation used in the evaluation process, we limited the number of iterations to the value of 10.

In the data clustering phase of our approach we perform K-means clustering three times. 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 here), where i is the number of iterations, $|E|$ is the number of environment conditions clusters and $|W|$ is the number of time windows the day is divided to. Then, we separately cluster users and services. The computational complexity required in user clustering is $O(i \times |U| \times |E| \times m)$, where $|U|$ is the number of user clusters, and m is the number of users. Similarly, the computational complexity of services clustering takes $O(i \times |S| \times |E| \times n)$, where $|S|$ is the number of service clusters and n is the number of services.

The values i, $|W|$ and $|E|$ are constant and do not impact the computational complexity. In our approach we assume the number of clusters is relatively small when compared to the number of users and services ($|U|, |S| << n, m$). Hence, the practical case computational complexity in the *CLUS* approach requires $O(m+n)$. The presented analysis of complexity strongly supports our claims in favor of the *CLUS* approach scalability improvement.

6. CONCLUSION

In this paper, we proposed a novel approach for reliability prediction of atomic web services based on the collected past invocation data. Our prediction model, called *CLUS*, improves the existing state-of-the-art approaches based on the collaborative filtering technique. While existing approaches implicitly consider only user- and service-specific parameters, we incorporate the *environment-specific* parameters in the prediction. In such manner, we significantly improve the accuracy of the prediction in a dynamic environment with 56% lower *RMSE* value than the *Hybrid*, current state-of-the-art approach.

Further, we reduce the redundant data by applying the *principle of aggregation*. We group similar users and services considering their reliability performance using K-means clustering algorithm. In this way, we accomplish to improve the computational complexity while not degrading the accuracy. We reduce the execution time for *two orders of magnitude* when compared to the current state-of-the-art approaches. Another benefit of our approach is that it can be applied in different environments due to flexibility reflected in a balanced *trade-off* between accuracy and scalability. To improve the accuracy, we increase the number of clusters. On the other hand, reducing the number of clusters makes the approach more scalable.

Our ongoing research is focused on other nonfunctional qualities of atomic services and integration of different nonfunctional aspects as part of the recommendation system for appropriate service selection.

7. ACKNOWLEDGMENTS

The authors acknowledge the support of 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.

8. REFERENCES

[1] D. Krafzig, K. Banke, and D. Slama, *Enterprise SOA: Service-Oriented Architecture Best Practices (The Coad Series)*. Prentice Hall PTR, 2004.

[2] OASIS, "Web services business process execution language version 2.0," April 2007. OASIS Standard.

[3] S. Sinisa, S. Dejan, and S. Daniel, "Programming language design for event-driven service composition.," *AUTOMATIKA - Journal for Control, Measurement, Electronics, Computing and Communications*, 2011.

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

[5] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of

dependable and secure computing," *Dependable and Secure Computing, IEEE Transactions on*, 2004.

[6] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "Qos-aware middleware for web services composition," *Software Engineering, IEEE Transactions on*, 2004.

[7] D. Wang and S. T. KISHOR, "Modeling user-perceived reliability based on user behavior graphs," *International Journal of Reliability, Quality and Safety Engineering*, 2009.

[8] V. Cortellessa and V. Grassi, "Reliability modeling and analysis of service-oriented architectures," pp. 339–362.

[9] L. Cheung, L. Golubchik, and F. Sha, "A study of web services performance prediction: A client's perspective,"

[10] J. S. Breese, D. Heckerman, and C. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering,"

[11] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms,"

[12] Z. Zheng, H. Ma, M. R. Lyu, and I. King, "Qos-aware web service recommendation by collaborative filtering," *IEEE Transactions on Services Computing*, 2011.

[13] Z. Zheng and M. R. Lyu, "Collaborative reliability prediction of service-oriented systems,"

[14] X. Su and T. M. Khoshgoftaar, "A survey of collaborative filtering techniques," *Adv. in Artif. Intell.*, vol. 2009.

[15] Y. Wang, W. M. Lively, and D. B. Simmons, "Web software traffic characteristics and failure prediction model selection," *J. Comp. Methods in Sci. and Eng.*, 2009.

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

[17] N. B. Mabrouk, S. Beauche, E. Kuznetsova, N. Georgantas, and V. Issarny, "Qos-aware service composition in dynamic service oriented environments,"

[18] Z. Zheng, Y. Zhang, and M. Lyu

[19] M. R. Lyu, ed., *Handbook of software reliability engineering*. Hightstown, NJ, USA: McGraw-Hill, Inc., 1996.

[20] J. D. Musa, A. Iannino, and K. Okumoto, *Software reliability: measurement, prediction, application (professional ed.)*. New York, NY, USA: McGraw-Hill, Inc., 1990.

[21] Z. Jelinski and P. Moranda., "Software reliability research.,"

[22] M. R. Lyu, "Software reliability engineering: A roadmap,"

[23] L. H. Putnam and W. Myers, *Measures for Excellence: Reliable Software on Time, within Budget*. Prentice Hall Professional Technical Reference, 1991.

[24] M. Friedman and P. Tran

[25] A. A. Abdel-Ghaly, P. Y. Chan, and B. Littlewood, "Evaluation of competing software reliability predictions," *IEEE Trans. Softw. Eng.*, 1986.

[26] L. Cheung, R. Roshandel, N. Medvidovic, and L. Golubchik, "Early prediction of software component reliability,"

[27] L. Cheung, I. Krka, L. Golubchik, and N. Medvidovic, "Architecture-level reliability prediction of concurrent systems,"

[28] B. Zhou, K. Yin, S. Zhang, H. Jiang, and A. J. Kavs, "A tree-based reliability model for composite web service with common-cause failures,"

[29] V. Grassi and S. Patella, "Reliability prediction for service-oriented computing environments," *IEEE Internet Computing*, 2006.

[30] W. T. Tsai, D. Zhang, Y. Chen, H. Huang, R. Paul, and N. Liao, "A software reliability model for web services,"

[31] J. Ma and H.-p. Chen, "A reliability evaluation framework on composite web service,"

[32] F. Mahdian, V. Rafe, R. Rafeh, and A. T. Rahmani, "Modeling fault tolerant services in service-oriented architecture,"

[33] 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,"

[34] L. Coppolino, L. Romano, and V. Vianello, "Security engineering of soa applications via reliability patterns.," *JSEA*, 2011.

[35] A. S. Das, M. Datar, A. Garg, and S. Rajaram, "Google news personalization: scalable online collaborative filtering,"

[36] H. Guan, H. Li, and M. Guo, "Semi-sparse algorithm based on multi-layer optimization for recommendation system,"

[37] C. Wei, W. Hsu, and M. L. Lee, "A unified framework for recommendations based on quaternary semantic analysis,"

[38] R. Burke, "Hybrid recommender systems: Survey and experiments," *User Modeling and User-Adapted Interaction*, 2002.

[39] H. Ma, I. King, and M. R. Lyu, "Effective missing data prediction for collaborative filtering,"

[40] M. Silic, G. Delac, I. Krka, and S. Srbljic, "Scalable and accurate prediction of availability of atomic web services," *IEEE Transactions on Services Computing*.

[41] Y. Baryshnikov, E. Coffman, G. Pierre, D. Rubenstein, M. Squillante, and T. Yimwadsana, "Predictability of web-server traffic congestion," *Web Content Caching and Distribution, International Workshop on*, 2005.

[42] M. Andreolini and S. Casolari, "Load prediction models in web-based systems,"

[43] "Architecture-based software reliability modeling," *Journal of Systems and Software*, 2006.

[44] A. W. Services, "Amazon ec2," March 2012. Elastic Compute Cloud.

[45] S. software, "Loadui." `http://www.loadui.org/`, 2012. Open source load and stress testing tool.

[46] A. Vattani, "k-means requires exponentially many iterations even in the plane,"