IoTPredict: Collaborative QoS Prediction in IoT

Gary White, Andrei Paladea, Christian Cabrera, Siobhán Clarke
School of Computer Science and Statistics,
Trinity College Dublin, Dublin, Ireland
{whiteg5, paladea, cabrerac, siobhan.clarke}@scss.tcd.ie

Abstract—Internet of Things (IoT) applications can be built from a number of heterogeneous services provided by a range of devices, which are potentially resource constrained and/or mobile. As these services and applications continue to be more widespread, a key research question is how to predict user-side quality of service (QoS), to ensure the optimal selection, composition and adaptation of IoT services. The exponential growth in the number of these services means that it is not practical to invoke all candidate services to test their QoS, especially during runtime service adaptation. QoS can vary by time and location, which makes it difficult for service providers to give accurate estimates of how the service will perform for users located in changing network topologies. We propose IoTPredict, a novel neighbourhood-based prediction approach for the IoT, which uses an alternative similarity computation mechanism. Our collaborative approach requires no additional invocation of services, which is a key requirement for resource constrained devices in the IoT. We evaluate our algorithm on a QoS dataset and show that it achieves higher QoS prediction accuracy than other state of the art approaches.

I. INTRODUCTION

The Internet of Things (IoT) has developed from recent advancements in Internet technologies, Wireless Sensor Networks (WSN) and a new generation of cheaper and smaller wireless devices. The number of devices connected to the IoT is predicted to grow at an exponential rate, with latest forecasts predicting that there will be between 26 and 50 billion connected devices by 2020 [1], [2]. The management of these devices is recognised as one of the most important areas of future technology and has gained wide attention from research institutes and industry in a number of different domains including transportation, healthcare, industrial automation and emergency response [3], [4].

The devices that provide software services in these domains are usually resource-constrained and/or mobile, which can lead to the QoS degrading or the service not responding [5]. A middleware can be used to handle these challenges and choose the optimal service at design time and conduct a dynamic service adaptation if the QoS begins to degrade [6]. Applications can have a range of QoS requirements based on the domain and sensitivity of the information. With additional services available from a variety of devices many providers offer services that provide equivalent functionalities. Such redundant services can be utilised for service adaptation by replacing the current working services with a candidate service in response to unexpected QoS changes (e.g. unacceptable response time). To achieve this, knowledge about QoS values of the services is required to make timely and accurate adaptation decisions, such as when to trigger an adaptation action, which working services to be replaced and which candidate service to choose. For applications in critical domains such as healthcare and emergency response, it is especially important for a middleware to react to degrading service quality and handle runtime service adaptation by selecting suitable replacement services.

Selecting the best IoT service for a user is a difficult task as the QoS of a service takes into account a number of factors, such as, response time, location and cost, which may be influenced by user preference [7]. There are also a number of QoS factors, which are difficult to quantify, such as, the security and functional stability of a service [8]. Quantifiable QoS factors, such as response time and throughput, are not stable and vary by time and location of invocation [9]. Traditional approaches have focused on time-series analysis to make predictions of future QoS values [10], [11], [12]. These approaches rely on the user executing the service to generate the values, which can be used for time series prediction, but they make no estimates for uninvoked services. This is a problem in the IoT as there will be a large number of candidate services and it would be too time consuming to invoke even a subset of the functionally equivalent services and as the services may be charged it can also increase the cost to the service user. The invocation of services just to obtain their QoS values is a waste of network resources, especially in a resource-constrained environment.

An alternative approach to predicting QoS is inspired from recommender systems, where the QoS information of similar users is used to make predictions about the QoS from possible services, using collaborative filtering [13], [14]. These collaborative filtering approaches use matrix factorisation (MF) to allow the user to receive QoS predictions of services that they have not invoked, based on QoS values from similar users. Using the QoS from similar users gives more information about candidate services, which can be chosen either at design time or during runtime service execution. These approaches have traditionally been used only in cloud based services, but recent results have shown they can be used in an IoT environment but require further changes to improve accuracy [15], [16]. They have an additional benefit that they can gather QoS information about possible candidate services in the service composition without harming the performance of the network by using the predicted values based on other users rather than invoking the services directly. They are also able to scale well through the use of distributed gradient descent, which makes them suitable for handling the large scale users
and services that we would expect in an IoT environment [17].

In this paper we outline a collaborative framework for the IoT to support collaborative QoS prediction and show how it can be used in a small-scale example. We also propose a novel algorithm, which uses an alternative similarity comparison to reduce the error in the predicted QoS values compared to state of the art approaches. The remainder of the paper is organised as follows: Section II describes the collaborative framework in IoT and a middleware, which can implement this framework. Section III shows how this framework can be used in the IoT to predict QoS and introduces the alternative similarity computation method. Section IV describes the experiments that were conducted and Section V presents the results of these experiments. Section VI outlines the related work and Section VII concludes and presents future work.

II. COLLABORATIVE FRAMEWORK IN IOT

Figure 1 shows a small-scale IoT scenario, with services provided from different service types including web services (WS), residing on resource rich devices, wireless sensor networks (WSN), which may be resource-constrained and controlled by a software defined network, and autonomous service providers (ASP), who are independent mobile users in the environment with intermittent availability. Even in a small-scale scenario such as this, traditional time-series approaches can cause a number of additional service requests for each user, which can flood the network [18]. A collaborative framework allows for users to share local QoS usage experiences and combine this information to get a global view of the QoS of the services in the environment. Each user keeps a record of their QoS experiences in the service registry. If a service consumer would like to get personalised QoS predictions for alternative services, they can allow their QoS records to be used for other users predictions, which encourages collaboration. We can then use our algorithm to make predictions of QoS values for new users based on local information from similar users and forward the recommended services to the service composition and execution engine.

The services provided by devices in the environment can be used in a number of applications in a variety of different domains. One way to categorise these applications is by their QoS requirements, which is based on the sensitivity and criticality of the application. IoT applications can typically be categorised as best effort (no QoS), differentiated services (soft QoS) and guaranteed services (hard QoS) [8]. In the hard QoS case, there are strict hard real-time QoS guarantees. This would be for safety critical applications such as monitoring patients in a hospital or collision avoidance in a self-driving car system. Soft QoS doesn’t require hard real time guarantees but needs to be able to reconfigure and replace services that fail. This could be a routing application, which uses air quality, flooding and noise information to provide the best route through the city. If one of the services is about to fail, the middleware should re-compose the application using suitable replacement services. The final case is best effort, where there are no guarantees when a service fails. The collaborative framework that we define can provide soft QoS guarantees by using collaborative filtering process to select suitable replacement services based on the QoS experiences from other users in the environment.

Figure 2 shows our middleware architecture, including the QoS monitor used to collect QoS information from a number of different user requests. The middleware is implemented in the gateways to register, compose, monitor and execute the services in the environment. The services have a range of different QoS properties including response-time, location and energy consumption. The communication links for invoking
these services are diverse, which will affect the personal QoS experience of the users, so they are monitored using the monitoring engine. These devices are resource constrained and mobile, which adds additional complexity compared to traditional web services that are static and more reliable. The monitored results are used by the prediction engine to provide personalised QoS prediction to service consumers, through their prediction algorithm.

The main components of the middleware are the Request Handler (RH), the Service Registration Engine (SRE), the Service Discovery Engine (SDE), the QoS Monitor and the Service Composition & Execution Engine (SCEE). The RH establishes a request/response communication channel with the user and forwards the request to the other middleware components. The SRE registers the available services in the environment. The SDE uses the backward-planning algorithm to identify the concrete services, which can be used to satisfy the request and sends this list of services to the SCEE. The QoS monitor is used to monitor these services and can predict possible candidate services to switch to if one of the services begins to degrade, using the prediction engine. The prediction engine is the main focus of this paper and the design is discussed in detail in Section III. The SCEE uses these services and the predicted QoS values to create a response for the request.

The SCEE is responsible for the composition and execution of services discovered by the SDE. Figure 3 illustrates a list of available services in the environment identified to satisfy a user request, which was received from the RH. The SCEE creates a list of service flows based on the concrete service providers received from the SDE. The flows are then merged based on the service description. If two or more services in the flow have the same input, the SCEE creates a guidepost to enable the invocation of one of these services based on QoS requirements. As some of the QoS values can be missing from the registries the goal of the collaborative framework is to make predictions for the missing values.

An execution guidepost \( G = \{ R_{id}, D \} \) is a split-choice control element of the composition process and maintains a set of execution directions \( D \) for a composition request \( R_{id} \). These execution directions will be referred to as branches. Each element in the set \( D \) is defined \( d_j = \{ id, w, q \} \) where \( j \leq |D| \). The set \( w \) represents the services in the branch and \( id \) represents the identifier of the branch. The value \( q \) reflects the branch’s aggregated QoS values [19], which can be calculated according to predefined formulas [20]. The branch that maximises/minimises an objective function will be selected by the guidepost during execution. This objective function can contain a number of different non-functional QoS factors that can be specified by the user in the service request.

As an example we consider the response time for each branch. The formula in Equation 1 calculates the response time by aggregating the response time value of each component service in a sequential flow [20]. In this formula, \( r_{ti} \) is the response time of service \( i \). However, it is possible that this value could not be calculated because of a missing QoS value from an individual service candidate in the flow, or the value being out-of-date.

\[
\text{Response Time (RT)} = \sum_{i=1}^{n} r_{ti} \quad \text{(1)}
\]

To address this problem, the QoS monitor uses QoS prediction to predict QoS values across each branch stored in the guidepost. Figure 3 shows the flows created by the SCEE for User 4 (\( U_4 \)) in Figure 4. The response time values recorded during service discovery phase were 0.34s for service provider \( WSN_1 \), 0.34s for \( WSN_1 \) and 0.23s for \( ASP_1 \). The response time values for \( WSN_1 \) and \( ASP_3 \) were not recorded. In Figure 3a, when the execution reaches Guidepost G, we can only aggregate the response time values for Branch 1, which is not optimal. If the composition selects the branch with the lowest reported response time, it will select Branch 3, which is also not optimal. Only when we use the predicted values for the missing service QoS does the composition choose the optimal Branch 2, which can be seen in Figure 3b.

This example shows that we have been able to make a number of predictions for services without directly invoking them, which reduces the load on the network. The effectiveness of the example relies on accurate QoS predictions to ensure that the correct branch is selected. Recent studies have shown some accuracy issues with current state of the art collaborative filtering algorithms for reliable service composition, with some instances where the composition was worse using predicted values for the flows [16]. This motivated the need to improve the accuracy of current state of the art approaches.
III. COLLABORATIVE QoS PREDICTION

A. Small-Scale Demonstration

To provide QoS values on \( m \) IoT services for \( n \) users, you need to invoke at least \( n \times m \) services. This is almost impossible in an IoT environment where we expect a large number of services and users. Without this QoS information, the service composition and execution engine cannot select the optimal components based on their QoS and must make a choice based on whatever QoS information is available. This leads to choosing potentially non-optimal services, which can cause service degradation at runtime and execution errors.

The QoS value of IoT service \( s \) observed by user \( u \) can be predicted by exploring the QoS experiences from a user similar to \( u \). A user is similar to \( u \) if they share similar characteristics, which can be extracted from their QoS experiences with different services. By sharing local QoS experience among users, these approaches can predict the QoS value of a range of IoT services including ASPs, web services and WSNs even if the user \( u \) has never invoked the service \( s \) before. Figure 1 can be modelled as a bipartite graph \( G = (U \cup S, E) \), such that each edge in \( E \) connects a vertex in \( U \) to \( S \). Let \( U = \{u_1, u_2, ..., u_4\} \) be the set of component users and \( S = \{ASP_1, ASP_2, ..., WSN_2\} \) denote the set of IoT services and \( E \) (solid lines) represent the set of invocations between \( U \) and \( S \). Given a pair \((i, j), u_i \in U \) and \( c_j \in S \), edge \( e_{ij} \) corresponds to the QoS value of that invocation. Given the set \( E \) the task is to predict the weight of potential invocations (broken lines).

We visualise the process of matrix factorisation for the flows generated in Figure 3, in Figure 4b, where each table entry shows an observed weight in Figure 4a. The task can be framed as a matrix completion problem, where we want to fill in the remaining values to have a fully completed matrix as shown in Figure 4c. There are a number of steps to make these predictions, which are discussed in detail in the following sections. In the remainder of this section we present the overall framework used to generate the predicted values at a higher level based on a specific example.

One of the problems that emerges due to the large number of services in the IoT is that there are a number of functionally similar services for users to choose, which reduces the possibility of having commonly invoked services. In the small-scale example in Figure 4b, with half the values reported there are few services common to users. For example, \( U_1 \) has only one service in common with \( U_2 \) and \( U_3 \) and two services in common with \( U_4 \). The values come from the public dataset released by Zheng et al. [9], which consists of a matrix of the response time and throughput of 339 users for 5,825 web services. As this dataset is for web services we use the HetHetNets traffic model to add heterogeneous IoT traffic data to the existing dataset [21], which is described in more detail in Section IV-A. The original dataset only has the user service matrix as features, which makes it difficult to calculate the similarity between users with few commonly invoked services. The first step in the algorithm, discussed in more detail in Section III-B, is to factorise the sparse user-service matrix and then use \( V^TH \) to approximate the original matrix, where the low-dimensional matrix \( V \) denotes the user latent feature space and the low-dimensional matrix \( H \) represents the service latent feature space, using the latent factor model [22]. The additional features in the latent feature space represents the underlying structure in the data, computed from the original dataset using matrix factorisation. As these matrices are dense they allow for similarity computation between all the users and services in the matrix, which solves the original problem of having sparse matrices.

Once we have access to the low dimensional dense matrices we can then compute the similarity between different users and services using their latent features (Section III-C). Traditional approaches for QoS prediction have used Persons Correlation Coefficient (PCC), to calculate the similarity between users and services. However after conducted some experiment on the IoT data we found that there were a number of assumptions, which PCC makes that are not satisfied, such as having no outliers and the variables being approximately normally distributed. We propose an alternative non-parametric similarity computation mechanism that does not make these assumptions called Kendall’s Tau.

Once we have computed the similarity between users and services we can then make predictions for the missing values by using the Top-K largest tau values for the users and services (Section III-D). The predictions for the users and services are weighted based on how similar they are and using an equal weighting of user and service based prediction.

B. Latent Features Learning

To learn the latent features of the users and services matrix we employ matrix factorisation, which fits a model to the user-service matrix from the original dataset. The original QoS matrix is factorised into two low rank matrices \( V \) and \( H \). The QoS usage experience of a user is typically determined by a small number of factors such as the network load, location of invocation and provider resources. The latent feature model ensures an accurate and low-dimensional representation of the original matrix.

Let \( \Omega \) be the set of all pairs \( \{i, j\} \) and \( \Lambda \) be the set of all known pairs \( \{i, j\} \) in \( \Omega \). Consider the matrix \( W \in \mathbb{R}^{m \times n} \) consisting of \( m \) users and \( n \) services. Let \( V \in \mathbb{R}^{l \times m} \) and \( H \in \mathbb{R}^{l \times n} \) be the latent user and service feature matrices. Each column in \( V \) represents the \( l \)-dimensional user-specified latent feature vector of a user and each column in \( H \) represents the \( l \)-dimensional service-specific latent feature of a service.

We employ an approximating matrix \( \tilde{W} = V^TH \) to learn the user-service relationship \( W \) [23]:

\[
 w_{ij} \approx \tilde{w}_{ij} = \sum_{k=1}^{l} v_{ki}h_{ki} \tag{2}
\]

To learn matrices \( V \) and \( H \) from the obtained QoS values in the original matrix \( W \), we need to construct a cost function to evaluate the accuracy of the approximation. We use the
standard Euclidean distance between the two matrices as the cost function.

\[ F(W, \tilde{W}) = \| W - \tilde{W} \|_F^2 = \sum_{ij} (w_{ij} - \tilde{w}_{ij})^2 \]  \hspace{1cm} (3)\]

where \( \| \cdot \|_F^2 \) denotes the Frobenius norm.

The optimisation problem can then be solved by using the optimisation objective function in [23]:

\[
\min_{V,H} \; f(V,H) = \sum_{(i,j) \in \Lambda} [\tilde{w}_{ij} - w_{ij} \log \tilde{w}_{ij}],
\]

\[
\text{s.t.} \quad \tilde{w}_{ij} = \sum_{k=1}^{l} v_{ik} \bar{h}_{ki},
\]

\[
V \geq 0,
\]

\[
H \geq 0.
\]  \hspace{1cm} (4)

The objective function in Eq. 4 can then be minimised using incremental gradient descent to find a local minimum, where one gradient descent step intends to decrease the square of the prediction error of only one rating, that is \( \tilde{w}_{ij} - w_{ij} \log \tilde{w}_{ij} \). We update the \( V \) and \( H \) in the opposite direction of the gradient descent in each iteration [13].

C. Similarity Computation

The similarities of the latent features for different users and services in matrix \( V \) and \( H \) can be calculated using a correlation coefficient to measure the similarity. Pearson Correlation Coefficient (PCC) is widely used for correlation computation in collaborative filtering [13], [24], [25]. The correlation between users \( u_i \) and \( u_j \) is defined by performing PCC computation on their l-dimensional latent feature vectors \( V_i \) and \( V_j \) with the following equation [24]:

\[
S(u_i, u_j) = \frac{\sum_{k=1}^{l} (v_{ik} - \bar{v}_i)(v_{jk} - \bar{v}_j)}{\sqrt{\sum_{k=1}^{l} (v_{ik} - \bar{v}_i)^2} \sqrt{\sum_{k=1}^{l} (v_{jk} - \bar{v}_j)^2}}
\]  \hspace{1cm} (5)

where \( v_i = (v_{i1}, v_{i2}, \ldots, v_{il}) \) is the latent feature vector of user \( u_i \) and \( v_{ik} \) is the weight on the \( k^{th} \) feature. \( \bar{v}_i \) is the average weight on the l-dimensional latent features for user \( u_i \). The similarity between two users \( S(i,j) \) falls into the interval [-1, 1], where a larger value indicates higher similarity.

PCC is also employed to compute the similarity between service \( s_i \) and service \( s_j \) as follows:

\[
S(s_i, s_j) = \frac{\sum_{k=1}^{l} (h_{ik} - \bar{h}_i)(h_{jk} - \bar{h}_j)}{\sqrt{\sum_{k=1}^{l} (h_{ik} - \bar{h}_i)^2} \sqrt{\sum_{k=1}^{l} (h_{jk} - \bar{h}_j)^2}}
\]  \hspace{1cm} (6)

where \( h_i = (h_{i1}, h_{i2}, \ldots, h_{il}) \) is the latent feature vector of service \( s_i \) and \( h_{ik} \) is the weight on the \( k^{th} \) feature. \( \bar{h}_i \) is the average weight on l-dimensional latent features for service \( s_i \).

PCC makes a number of assumptions that must be taken into consideration [26]. The variables, which are being correlated, must be approximately normally distributed. However, as can be seen in Figure 5, which shows the histogram of latent feature values for the users and services after conducting matrix factorisation, it is clear that the distribution is not normal. PCC also assumes that outliers are kept to a minimum or removed entirely. Figure 6 is a scatter plot that shows the correlation between two users and services. The figure shows that in the latent features there can be a number of outliers, which effects PCC as it assumes a linear relationship.

As it is difficult to justify removing the latent outliers, we can use an alternative approach, in particular, a non-parametric correlation coefficient such as Kendall’s Tau, which is much less sensitive to outliers [27]. Kendall’s tau measures the degree of monotonic relationship between variables, and calculates the dependence between ranked variables, which makes it feasible for non-normally distributed data. The similarity between two users \( u_i \) and \( u_j \) is defined by performing Kendall’s tau on their l-dimensional latent feature vectors \( V_i \) and \( V_j \). Let \( (x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \) be a set of
observations from $V_i$ and $V_j$. Any pair of observations from $V_i$ and $V_j$ are said to be concordant if the ranks for both elements agree i.e. if both $x_i < x_j$ and $y_i < y_j$, or if both $x_i > x_j$ and $y_i > y_j$. They are said to be discordant, if $x_i > x_j$ and $y_i < y_j$, or $x_i < x_j$ and $y_i < y_j$. If $x_i = x_j$, or $y_i = y_j$, the pair is a tie [27].

$$\tau = \frac{c - d}{c + d} = \frac{S}{\binom{n}{2}} = \frac{2S}{n(n-1)} \tag{7}$$

where,

$c$ = the number of concordant pairs,

d = the number of discordant pairs.

If ties are present among the two ranked variables, the following equation shall be used instead:

$$\tau = \frac{S}{\sqrt{n(n-1)/2 - T \sqrt{n(n-1)/2 - U}}} \tag{8}$$

$$T = \sum t(t - 1)/2 \tag{9}$$

$$U = \sum u(u - 1)/2 \tag{10}$$

where,

$t$ = number of observations of variable $x$ that are tied,

$u$ = number of observations of variable $y$ that are tied.

The same formula can also be used to compute the similarity between the services through the use of concordant and discordant pairs.

D. Missing QoS Value Prediction

The similarity values can then be used to identify similar neighbours to the current user by ordering the values. Kendall’s tau falls into the interval [-1, 1], where a positive value denotes similarity and a negative value denotes dissimilarity. QoS usage experience from dissimilar users negatively affects the prediction accuracy, so users with a negative tau value are excluded from the similarity set. We employ only the QoS usage experience of users with the Top-K largest tau values for predicting the QoS of the user. The Top-K similar users for user $u_i$ are defined as $\Psi_i$:

$$\Psi_i = \{u_k|S(u_i, u_k) > 0, rank_k(k) \leq K, k \neq i\} \tag{11}$$

where $rank_k(k)$ is the ranking position of user $u_k$ based on user $u_i$ and $K$ denotes the size of set $\Psi_i$. We can similarly denote the Top-K IoT services for service $s_j$ as $\Phi_j$ by:

$$\Phi_j = \{s_k|S(s_j, s_k) > 0, rank_k(k) \leq K, k \neq j\} \tag{12}$$

where $rank_j(k)$ is the ranking position of service $s_k$ based on service $s_j$ and $K$ denotes the size of set $\Psi_j$. To predict missing values for $w_{ij}$ in the user service matrix, user-based approaches use the values from the Top-K similar users as follows:

$$w_{ij} = \bar{w}_i + \sum_{k \in \Psi_i} \frac{S(u_i, u_k)}{\sum_{a \in \Psi_i} S(u_i, u_a)} (w_{kj} - \bar{w}_k) \tag{13}$$

where $\bar{w}_i$ and $\bar{w}_k$ are the average observed QoS values of different services by users $u_i$ and $u_k$ respectively. For service based approaches, entry values of Top-K similar services are employed for predicting the missing entry $w_{ij}$ in a similar way:

$$w_{ij} = \bar{w}_j + \sum_{k \in \Phi_j} \frac{S(i_j, i_k)}{\sum_{a \in \Phi_j} S(i_j, i_a)} (w_{ik} - \bar{w}_k) \tag{14}$$

where $\bar{w}_j$ and $\bar{w}_k$ are the average observed QoS values of different services $s_i$ and $s_k$ by different users respectively. The predicted values using Eq. 13 and Eq. 14 are combined for a more precise prediction in the following equation [28]:

$$w_{ij}^* = \lambda \times w_{ij}^u + (1 - \lambda) \times w_{ij}^s \tag{15}$$

where $w_{ij}^u$ denotes the predicted value by user based approach and $w_{ij}^s$ denotes the predicted value by the service based approach. The parameter $\lambda$ controls how much the hybrid prediction relies on user based or service based approach. We summarize the proposed algorithm in Algorithm 1.

IV. EXPERIMENTAL SETUP

A. Dataset Description

To test our algorithm, we use an established dataset to control for any differences in QoS that can be caused by invoking services at different times. We use the dataset released by Zheng et al. [9], which consists of a matrix of the response time and throughput of 339 users for 5,825 web services. Response time stands for the time duration between a user sending a request and receiving a response, while throughput denotes the data transmission rate (e.g. kbps) of a user invoking a service. The dataset has 1,974,675 observations for both the response time and throughput dataset, which allows for a comprehensive evaluation of the algorithms. The reason for the use of a dataset instead of a testbed is due to the time varying nature of QoS, which can change due to congestion on the network. This would mean that the algorithms would be evaluated using different values. The dataset also allows more users and services to be evaluated as there are no available testbeds that have access to 339 users and 5825 service, that
we know of. An established dataset not only allows for a comprehensive evaluation, but also allows future algorithms to easily be compared to the existing state of the art.

As this dataset is for web services, which are usually deployed in the cloud, they have better response time than might be expected from low power devices. We use the HetHetNets traffic model to add heterogeneous traffic data to the existing dataset [21]. The model provides a realistic and comprehensive evaluation, but also allows future algorithms to be compared to the existing state of the art. An established dataset not only allows for a

B. Metrics

To evaluate the prediction accuracy of the proposed algorithms, we use standard error metrics: the Mean Absolute Error (MAE), the Mean Relative Error (MRE), the Root Mean Square Error (RMSE) and the Ninety-percentile Relative Error (NPRE). The NPRE is the 90th percentile of all the pairwise relative errors.

MAE is defined as:

\[ MAE = \frac{1}{N} \sum_{i,j} |w_{ij} - w_{ij}^*| \]  

(16)

MRE is defined as:

\[ MRE = \text{median}_{i,j} \left| \frac{w_{ij} - w_{ij}^*}{w_{ij}^*} \right| \]  

(17)

and RMSE is defined as:

\[ RMSE = \sqrt{\frac{1}{N} \sum_{i,j} (w_{ij} - w_{ij}^*)^2} \]  

(18)

where \( w_{ij} \) is the QoS value of service \( c_j \) observed by user \( u_i \), \( w_{ij}^* \) denotes the predicted QoS value of the service \( c_j \) by user \( u_i \). \( N \) is the number of predicted QoS values. MAE gives equal weight to individual differences, RMSE on the other hand gives large weight to extreme errors due to the squaring term.

C. Performance Comparison

We compare the performance of our algorithm against two state of the art approaches: non-negative matrix factorisation (NMF) [23] and personalised neighbourhood matrix factorisation (PMF) [25]. These methods serve as a good comparison to our algorithm based on the results that were reported in their papers. These algorithms use PCC for similarity computation, which allows us to test the effect of Kendall’s Tau similarity measure.

V. Results

A. Impact of Matrix Density

Matrix density has an obvious effect on accuracy, with the error decreasing as the density of the matrix increases. Figure 7a and 7b show the results of the Response Time dataset for all algorithms. Figure 7a shows that our algorithm performs slightly worse for low matrix densities (< 10%), but shows a large improvement for densities greater than that in the MRE. Figure 7b shows the RMSE for the algorithms, which assigns more weight to outlying errors. In this figure, we can see that our algorithm has marginally reduced error for all of the matrix densities compared to the other approaches.

Figure 7c and 7d show the results for the throughput dataset, where we can see a greater difference in the results of the algorithms. Figure 7c shows a clear improvement in the IoTPredict algorithm compared to the two other approaches for all of the
matrix densities in the MRE. Figure 7d shows the NPRE for the algorithms and we can see that the IoTPredict algorithm has worse performance for the lower matrix densities. As the matrix density increases the error in the IoTPredict algorithm reduces dramatically and has better performance than the competing approaches.

B. Impact of Dimensionality

Dimensionality is the number of latent features used by the algorithm. In this section we investigate the impact this parameter has on the prediction error. Figure 8a and 8b show the impact of the latent features for 10% density on the response time dataset. The error is minimised using 10 latent features and as more latent features are added the error increases, indicating an overfitting problem.

Figure 8c and 8d show the impact of the latent features for 90% density. In this case, the MAE and RMSE are minimised with around 20-30 latent features. The results in Section V-A are conducted with 20 latent features, which reduces error for more dense matrices as shown by these results. The results also show that the number of latent features can be tuned to improve performance if the matrix density is already known, to avoid the under/overfitting problem. In this section we focus on detailed description of our own model parameters rather than a comparison against existing approaches.

C. Impact of Top-K

The Top-K is the number of similar users selected to make the predictions for the individual user, which can impact the prediction accuracy of the approaches. Figure 9 shows how the error changes for the 10% and 90% matrix densities as the top K users is increased from 2-50, with the dimensionality set to 20 latent features. Figure 9a and 9b show the impact of the number of Top-K users chosen on the response time dataset. In this case, we see that the the error drops considerably as K increases from 2 to 10 in both the MAE and RMSE. As K continues to increase, the error either remains constant or increases slightly. This can be seen when the matrix density is both at 10% and 90%.

In the throughput dataset, the response is slightly different. The 10% matrix density remains almost constant even as the number of Top-K users changes. The results for the 90% matrix density are interesting and show that the error increases with the number of Top-K users. The increased error is caused by using values from an increased number of users that are not similar, which decreases the prediction accuracy. Figure 9 also shows the problem, when trying to optimise the algorithm for one QoS factor, throughput, as it can make the algorithm less accurate for other QoS factors such as response time.

D. Overhead Introduced

The end-to-end response time from the user needs to take into account the overhead of the algorithm as well as the prediction accuracy. In this section we discuss the overhead introduced by the algorithms. For each of the matrix densities we measure the time to train the model and to generate the predictions for the missing values in the matrix. The experiment is repeated 20 times and the average value is taken. We use the same parameters for the overhead as the impact of matrix density results in Figure 7.

The results for the RT dataset are presented in Figure 10a and the results for the TP dataset are presented in Figure 10b. Both of the datasets follow similar patterns with NMF introducing the most overhead in both the RT and TP dataset. It can be seen that the overhead for NMF and PMF increase with density however IoTPredict remains constant. There is a trade-off between PMF and IoTPredict, with PMF introducing less overhead for matrices with densities between 5-25% and IoTPredict introducing less overhead for matrices between 25-35% density.

VI. RELATED WORK

A key research question in the IoT is how to increase the reliability of applications that use the services provided in the environment [5]. A number of approaches have been proposed for dynamic service composition [19], service selection [29], run-time QoS evaluation [30] and service ranking prediction [31]. These approaches assume that the QoS values are already known, however, in reality, user side QoS may vary significantly, given unpredictable communication links, mobile service providers and resource constrained devices.

Approaches such as collaborative filtering have been used in other domains such as commercial recommender systems [24], [32], [33]. There are two main approaches for predicting values using collaborative filtering, which can typically be classified as either memory or model based. Memory based approaches store the training data in memory and in the prediction phase similar users are sorted based on the current user. There
are a number of approaches that use neighbourhood based collaborative filtering, including user-based approaches [34], item-based approaches [35] and their combination [32]. VSS [34] and PCC [24] are often used in similarity computation methods, in this paper, we have shown justification for the use of Kendall’s Tau, with improved results against these approaches.

Model-based approaches, which employ a machine learning technique to train a predefined model from the training datasets have become increasingly popular. Several approaches have been studied, including clustering models [36], latent factor models [22] and aspect models [37]. Latent factor models, create a low-dimensional factor model, on the premise that there are only a small number of factors influencing the QoS [24]. The latent factor model has been extended to include neighbourhood integrated matrix factorisation, which fuses neighbourhood-based and model-based approaches to achieve higher accuracy [38].

Both approaches can be combined to predict missing QoS values by using global information from MF and local information from similar users and items [13], [14]. Previous approaches have focused on web services deployed in the cloud [39], [25], [13], [38], [40]. However these make up only one part of the IoT and we need to collect QoS information and make predictions for all service types to create a reliable applications that use all the services provided in the IoT.

VII. CONCLUSION AND FUTURE WORK

In the IoT, QoS predictions for individual users can be made by exploring past usage experience from the individual user and similar users in the environment. The collaborative filtering approach is useful in an IoT environment as it requires no additional invocations to generate the extra QoS information, which reduces the load on the network - a key challenge in the IoT as the devices are usually resource constrained.

We have proposed a framework and middleware to allow users to contribute QoS information to enable collaborative filtering. The values reported by users can then be used by our IoTPredict algorithm to make QoS predictions for candidate services. The prediction accuracy is higher compared to other state of the art approaches, while maintaining a low overhead. The algorithm can be used by a number of alternative service composition frameworks to compose recommended services or the goal oriented approach as shown in Section II.

The IoT is highly dynamic as it includes a number of mobile devices, which may be resource and power constrained. In current experiments the users and services are static, which is one of the threats to validity. In future work, we will add mobility to both the users and services for a more robust evaluation.
REFERENCES


