Workload Patterns for Quality-driven Dynamic Cloud Service Configuration and Auto-Scaling

Li Zhang, Yichuan Zhang
Northeastern University,
Shenyang, China
{zhangl,zhangyc}@swc.neu.edu.cn

Pooyan Jamshidi, Lei Xu, Claus Pahl
IC4, School of Computing, Dublin City University,
Dublin, Ireland
{pjamshidi,lxu,cpahl}@computing.dcu.ie

***Abstract*— Cloud service providers negotiate SLAs for customer services they offer based on the reliability of performance and availability of their lower-level platform infrastructure. While availability management is more mature, performance management is less reliable. In order to support an iterative approach that supports the initial static infrastructure configuration as well as dynamic reconfiguration and auto-scaling, an accurate and efficient solution is required. We propose a prediction-based technique that combines a pattern matching approach with a traditional collaborative filtering solution to meet the accuracy and efficiency requirements. Service workload patterns abstract common infrastructure workloads from monitoring logs and act as a part of a first-stage high-performant configuration mechanism before more complex traditional methods are considered. This enhances current reactive rule-based scalability approaches and basic prediction techniques based on for example exponential smoothing.**

Keywords — Quality of Service, Cloud Configuration, Auto-scaling, Web and Cloud Services, QoS Prediction, Workload Pattern Mining, Collaborative Filtering.

I. INTRODUCTION

Quality of Service (QoS) is the basis of web and cloud service configuration management and deployment [1,2]. Cloud service providers (CSPs) – whether at infrastructure, platform or software level – provide quality guarantees usually in terms of availability and performance to their customers in the form of service-level agreements (SLAs) [4]. Internally, the respective service configuration in terms of available resources then needs to make sure that the SLA obligations are met [10]. To facilitate SLA conformance, virtual machines (VMs) can be configured and scaled up/down in terms of CPU cores and memory, deployed with storage and network capabilities. Some current cloud infrastructure solutions allow users to define rules manually to scale up or down to maintain performance levels.

QoS like service performance in terms of response time or availability may vary depending on network, service execution environment and user requirements, making it hard for providers to choose an initial configuration and scale this up/down to maintain the SLA guarantees, but also optimising resource utilisation at the same time. We utilise QoS prediction techniques here, but rather than bottom-up predicting QoS from monitored infrastructure metrics [12,13,25], we reverse the idea, resulting in a novel technique for pattern-based resource configuration. We extract service workload patterns (SWPs) that correspond to typical workloads of the infrastructure and map these to QoS values. A pattern consists of narrow range of metrics measured for each infrastructure concern such as compute, storage and network under which the QoS concern is stable. In a top-down approach, we then take a QoS requirement and determine suitable workload-oriented configurations that maintain required values. Furthermore, we enhance this with a cost-based selection function, applicable if many candidate configurations emerge.

We specifically look at performance as the QoS concern here since dealing with availability in cloud environments is considered as easier to achieve, but performance is currently neglected in practice due to less mature resource management techniques [10]. We introduce pattern detection mechanisms and, based on a QoS-SWP matrix, we define SWP workload configurations for required QoS. The accuracy of the solution to guarantee that the chosen (initially predicted) resource configurations meet the QoS requirements is of utmost importance. An appropriate scaling approach is required in order to allow this to be utilised in dynamic environments. In this paper, we show that the pattern-based approach improves the efficiency of the solution in comparison with traditional prediction approaches, e.g. based on collaborative filtering. This enhance existing solutions by automating current manual rule-based reactive scalability mechanisms and also advances prediction approaches for QoS, making them applicable in the cloud with its accuracy and performance requirements.

Section II outlines the solution and justifies its practical relevance. Section III introduces SWPs and how they can be derived. Section IV discusses the selection of patterns as workload specifications for resource configuration. The application of the solution for SLA-compliant cloud resource configuration is described in Section V. Section VI contains an evaluation in terms of accuracy and performance of the solution and Section VII contains a discussion of related work.

II. APPROACH OUTLINE - QUALITY-DRIVEN CONFIGURATION AND SCALING

We now briefly discuss the state-of-the-art in cloud resource configuration and its relevance to the solution . An SLA is typically defined based on availability. Customers expect that the services they acquire will be always available. Thus, providers usually make extensive claims here. The

consensus in the industry is that cloud computing providers generally have solutions to manage availability. Response time guarantees, on the other hand, are harder to guarantee [10]. These types of obligations are more carefully phrased or fully ignored. A quote to illustrate this is "We are putting a lot of thought into how we can offer predictable, reliable and specific performance metrics to a customer that we can then [build an] SLA around," [C. Drumgoole, vice president global operations, Verizon Terremark, 2013]. Thus, we specifically focus on performance, although our solution is in principle applicable to availability as well.

From a provider's perspective, the question is how to initially configure and later scale VMs and other resources for a service such that the QoS (specifically response time) is guaranteed and, if additionally possible, cost is optimised. From an infrastructure perspective, data/storage, network conditions and CPU utilisation impact on QoS such as performance and availability significantly. We consider data size, network throughput and CPU utilization as representatives of data, network and computation characteristics. Common definitions, e.g. throughput as the rate of successful message delivery over a communication channel or bandwidth, shall be assumed. Fig. 1 illustrates in a simple example that values of the three resource configuration factors can be linked to the respective measured performance.

Service name	1st invocation	2nd invocation	3rd invocation	4th invocation	5th Invocation
s_1	[2,10,0.2] -> 0.5	[1.5,20,0.5] -> 2.0	[2.5,10,0.1] -> 0.2	[2,30,0.3] -> 0.8	[2,8,0.2] -> 1.2
s_2	[1.2,11,0.1] -> 0.3	[2,20,0.4] -> 1.8			
s_3	[2,20,0.3] -> 3.0	[1,20,0.2] -> 6.0	[1.5,20,0.3] -> 4.0	[2,15,0.2] -> 2.4	

Fig. 1. Measured QoS mappings from Infrastructure to Service (format: [data, network, CPU] → performance)

The first step is to monitor and record these input metrics in system logs. The second step is pattern extraction. From repeated service invocations records (the logs), an association to service QoS values based on prediction techniques can be made. An observation based on experiments that we made (see evaluation section) is that most services have relatively fixed service workload patterns (SWP):

- The patterns are defined here as ranges of data, network and CPU processing characteristics that reflect stable, small acceptable variations of a QoS value.
- Generally, service QoS keeps steady under a SWP, allowing this stable mapping between infrastructure input and QoS to be used further.

If we can abstract SWPs from service logs or the respective resource usage logs, the associated service quality can be based on usage information using pattern matching and prediction techniques. Even if there is no or insufficient usage information for a given service, quality values can be calculated using log information of other similar services, e.g. through collaborative filtering. These two step can be carried out offline. The next, first online step, is pattern matching, where dynamically a pattern is matched in the matrix against performance requirements. The final step is the (if necessary dynamic) configuration of the infrastructure in the cloud.

The hypothesis behind our workload pattern-driven resource configuration based on required service-level quality is the stability of variations of quality under SWPs. We assume SLA definitions to establish QoS requirements and the charged costs for a service to be decided between provider and consumer. Service-specific workload pattern are mined and constructed – which considers environmental characteristics of a service (in a VM) deployment. We experimentally demonstrate that the hybrid technique for QoS-to-SWP mappings (based on pattern matching and collaborative filtering for missing information) enhances accuracy and computational performance and makes it applicable in the cloud. Traditional prediction techniques can be computationally expensive and unsuitable for the cloud.

We limit this investigation to services and infrastructure with some reasonably deterministic behaviour, e.g. classical business or technology management applications. Larger substantial uncertainties arising from the environment shall be neglected – we will however discuss this context later. We also focus on single cloud environment, ignoring uncertainties arising from multi-cloud environments.

III. Workload Patterns

The core concept of our solution is a *Service Workload Pattern* (SWP). A SWP is a group of service invocation characteristics reflected by the utilised resources. In a SWP, the value of workload characteristics is a range. The QoS is meant to be steady under a SWP. We describe a SWP *M* as a triple of ranges low to high:

$$M = [\ Data_{low} \sim Data_{high},\ Network_{low} \sim Network_{high},\ CPU_{low} \sim CPU_{high}\ ] \quad (3.1)$$

Data, *Network* and *CPU* are common data/storage, network and server/computation characteristics that we have chosen for this investigation [25].

A. SWP Pattern Mining and Construction

We assume service-level execution quality logs in the format $< q_1, \ldots, q_n >$ and infrastructure-level resource monitoring logs $< r^i_1, \ldots, r^i_m >$ with $i=1,..,j$ for quality aspects (e.g., data, network, server CPU utilisation) of the past invocations of the services under consideration, as illustrated in Fig. 1. For each service, the resource metrics and the associated measured performance are recorded. The challenge is now to determine or mine combinations of value ranges for input parameters *r* that result in stable, i.e. only slightly varying performances. The solution is a SWP extraction process that constructs the workloadpatterns.

- A SWP is composed of data, network and computation characteristics. For these, we take throughput, data size and CPU utilization as representatives, resp.
- We consider the execution (response) time as the representative of QoS here.

An execution log records the input data size and execution QoS; a monitoring log records the network status and Web server status. We reorganize these two logs to find the SWP under which QoS keeps steady.

Our *SWP extraction algorithm* is based on a generic algorithm type, DBSCAN (density-based spatial clustering of application with noise). DBSCAN [14] analyses the density of data and allocates the data into a cluster if the spatial density is greater than a threshold. The DBSCAN algorithm has two parameters: the threshold ε and the minimum number of points *MinPts*. Two points can be in the same cluster if their distance is less than ε. The minimum number of points is also given. We also need a parameter *MaxTimeRange*, the max time range of a cluster. We expect the range of time is a cluster that can be steady and has a size limit. When the cluster is too large, e.g. if the range exceed a threshold, the cluster construction should be stopped. The main steps:

- Select any object *p* from the object set *S* and find the objects set *D* in which the object is density-reachable from object *p* with respect to ε and *MinPts*.
- Choose another object without cluster and repeat the first step.

The pattern extraction algorithm is presented below:

Algorithm: SWP Extraction Algorithm based on DBSCAN
Input: Service Usage *InforSet* (exec+monitor log), *ε, MinPts, MaxTimeRange*
Output: SWP *PatternBase*, Pattern-QoS information *PatternQoS*

```
for(Infor_i<DataSize,ThroughPut,CPU,Performance>∈InforSet){
  if (Infor_i does not belong to any exist cluster) {
    P_j=newPattern(Infor_i)
         // create a new pattern with Infor_i as seed
    Add( P_j, PatternBase )
    InforSet = InforSet - Infor_i
    SimInfor = SimilarInfor(InforSet, Infor_i, ε)
        // SimInfor is the information set which includes
           all the similar usage information of Infor_i.
           Differences between the information in SimInfor
           and Infor_i on the characteristics value except
           execution time are less than ε. n is the number
           of information items in SimInfor.
    InforSet = InforSet - SimInfor
    if ( n>MinPts ) {
        // MinPts is min number of exec info in cluster
       (S_1, S_2, … ,S_m) = Divide(SimInfor)
        // Divide SimInfor into different groups.
           Group S_1 includes all information of service s_1
       for(k=1; k≤m; k++) {
         for(Infor_j∈S_k) {
           if (MaxTime-MinTime<MaxTimeRange) {
             SimInfor =
              SimilarInfor(InforSet,Infor_j,time,MinPts,ε)
           // Search similar info of S_k in execution
           information set.  If the number of similar
           information item is less than MinPts, then the
           density will turn low and top the loop.
             S_k = S_k + SimInfor
             InforSet = InforSet - SimInfor
           }
         }
         PatternCharacteristics(S_k)
         // Organizes the information in the cluster and
         statistics for the ranges of characteristics -
         completes matrix
       }
    }
  }
}
```

We give higher precedence to more recent log entries. Exponential smoothing can be applied to any discrete set of sequential observations x_i. Let the sequence of observations begin at time *t=0*, then simple exponential smoothing is defined as follows:

$$\begin{aligned} y_0 &= x_0 \\ y_t &= \alpha x_t + (1-\alpha) y_{t-1}, t > 0, 0 < \alpha < 1 \end{aligned} \tag{3.2}$$

The choice of α is important. Close to 1 has no smoothing effect and gives higher weight to recent changes and as a result the estimate may fluctuate dramatically. Values of α closer to 0 have a better smoothing effect and the estimate is less responsive to recent changes. We propose 0.8 as the default, which is relatively high, but reflects the most recent multi-tenancy situation (which can undergo short-term changes).

B. Pattern-Quality Matrix

The input value ranges form a pattern that is linked to the stable performance ranges in a *Quality Matrix MS(M,S)* based on patterns *M* and services *S*. *MS* associates a service quality $QoS_P(S_i,M_i)$ (with *P* standing for performance) of service S_i in *S* under a pattern M_j in *M*.

$$\boldsymbol{MS} = \begin{array}{c} \\ \boldsymbol{M_1} \\ \boldsymbol{M_2} \\ \cdots \\ \boldsymbol{M_l} \end{array} \begin{array}{c} \begin{array}{cccc} s_1 & s_2 & \cdots & s_m \end{array} \\ \begin{bmatrix} q_{1,1} & q_{1,2} & \cdots & q_{1,m} \\ q_{2,1} & q_{2,2} & \cdots & q_{2,m} \\ \cdots & \cdots & \cdots & \cdots \\ q_{l,1} & q_{l,2} & \cdots & q_{l,m} \end{bmatrix} \end{array}$$

Fig. 1 at the beginning illustrated monitoring and execution logs that capture low-level metrics (CPU, network, storage) and the related service response time performance. SWPs M_i then result from the log mining process using clustering.

$$\begin{array}{c} \\ \boldsymbol{M_1} \\ \boldsymbol{M_2} \\ \boldsymbol{M_3} \\ \boldsymbol{M_4} \end{array} \begin{array}{c} \begin{array}{cccc} s_1 & s_2 & s_3 & s_4 \end{array} \\ \begin{bmatrix} 0.2 \sim 0.5s & & 1 \sim 1.3s & \\ 0.8 \sim 1.1s & 1.1 \sim 1.5s & & \\ 0.4 \sim 0.5s & & 2 \sim 2.4s & 0.3 \sim 0.5s \\ & 3 \sim 4s & & \end{bmatrix} \end{array}$$

The matrix *MS* above shows the QoS in this example for performance information of all services *s* for all patterns *M*.

The quality q_{ij} ($1 \le j \le l$, $1 \le i \le m$) is the quality of service s_j under pattern M_i with the quality value q_{ij} defined as follows:

- as ϕ if the service s_j has no invocation history under pattern m_i and
- as $low_{ij} \sim high_{ij}$ if the service s_j has an invocation history under m_i with range ~.

For a pattern M_1 = [*0.5-0.6* , *0.2-0.4* , *30-40MB*] the CPU utilization rate is *0.5-0.6*, memory utilization is *0.2-0.4* and network throughput is *30-40MB*. The sample matrix illustrates the workload pattern to QoS association for services. Empty spaces (undetermined *null* values) for a service indicate lacking data. In that case, a prediction based on similar services is necessary, for which we use collaborative filtering.

C. Pattern Matching

For monitored resource metrics (data, network, CPU), we need to determine which of these influences performance the most. This determines the matched pattern. Let the usage information of service s be a sequence x_k of data storage D, network throughput N and CPU utilisation C values mapped to response time R for $k = 1, .. ,n$:

$$\begin{aligned} &[< x_D^1 , x_N^1 , x_C^1 > , x_R^1] \\ &\vdots \\ &[< x_D^k , x_N^k , x_C^k > , x_R^k] \\ &\vdots \\ &[< x_D^n , x_N^n , x_C^n > , x_R^n] \end{aligned} \tag{3.3}$$

We use response time performance in the log as the reference sequence $x_R(k)$, $k = 1,\ldots, n$, and other configuration metrics as comparative sequences. Then, we calculate the association degree of other characteristics with response time and use characteristics of an invocation as standard and carry out a *normalization* of the other metrics. Thus, the normalized usage information is (schematically) for any invocation k:

$$[< y_D(x_D^k) , y_N(x_N^k) , y_C(x_C^k) > , 1] \tag{3.4}$$

Next, we calculate *absolute differences* for the table above using

$$\Delta_{0i}(k) = |y_0(k) - y_i(k)| \tag{3.5}$$

With Oi here ranging over the quality aspects $O1 = D$, $O2 = N$ and $O3 = C$. The resulting absolute difference sequence is for our 3 quality aspects the following:

$$\begin{aligned} &\Delta_{01} = (0, y_{01}(1), \cdots, y_{01}(n)) , \\ &\Delta_{02} = (0, y_{02}(1), \cdots, y_{02}(n)) , \\ &\Delta_{03} = (0, y_{03}(1), \cdots, y_{03}(n)) \end{aligned} \tag{3.6}$$

In the next step, we determine a *correlation coefficient* between reference and comparative sequence (using here the correlation coefficient of the Gray relevance):

$$\zeta_{0i}(k) = \frac{\Delta_{\min} + \rho\Delta_{\max}}{\Delta_{0i}(k) + \rho\Delta_{\max}} \tag{3.7}$$

Here $\Delta_{0i}(k) = |y_0(k) - y_i(k)|$ is the absolute difference, $\Delta_{\min} = \min_i \min_k \Delta_{0i}(k)$ is the minimum difference between two poles, $\Delta_{\max} = \max_i \max_k \Delta_{0i}(k)$ is the maximum difference, $\rho \in (0,1)$ is a distinguishing factor. Afterwards, we use the formula

$$r_{0i} = \frac{1}{n}\sum_{k=1}^{n} \zeta_{01}(k) \tag{3.8}$$

to calculate the *correlation degree* between the metrics. Then, we sort the metrics based on the correlation degree. If r_0 is the largest, it has the greatest impact on response time and will be matched prior to others in the pattern matching process.

Clouds are shared multi-user environments where users and applications require different quality settings. A multi-valued utility function $\tilde{U}$ can be added representing the user weighting of a vector $\tilde{Q}$ of quality attributes $q \in Q$ for a matrix $m_i \in M$ as a weighting. This utility function allows a user to customise the matching with user-specific weightings:

$$\tilde{U}_{p,m,q} : \mathrm{rng}(\tilde{Q}_{m,q}) \to [0,1] \tag{3.9}$$

The overall utility can be defined, taking into account the importance or severity of the quality attributes ω_i for each $q \in Q$:

$$\begin{aligned} &\tilde{U}_m \stackrel{\text{def}}{=} \sum_{\forall q_i \in Q} \omega_i \tilde{U}_{m,q_i}\left(\tilde{Q}_{m,q_i}(MS)\right) \\ &\tilde{U}_{m,q} = \sum_{\forall p \in P} \tilde{U}_{p,m,q} \\ &\sum_i \omega_i = 1, \omega_i \ge 0 \end{aligned} \tag{3.10}$$

Finally, the pattern that optimizes the overall configuration utility is determined through the maximum utility calculated as:

$$max_{m \in M_S}\ \tilde{U}_m \tag{3.11}$$

Note, that the utility is based on the three quality concerns, but could potentially be extended to take other factors into account. Furthermore, costs for the infrastructure can also be taken into account to determine the best configuration. We will define an additional cost function in the cloud configuration Section V.

IV. Quality Pattern-driven Configuration Determination

The QoS-SWP matrix is the tool to determine SLA requirements-compliant SWPs as workload specifications for the resource configuration and re-configuration/re-scaling. For quality-driven configuration, the question is: *for a given service S_i and a given performance requirement QoS_P, what are suitable SWPs to configure the execution environment?* The execution environment is assumed to be a VM image configuration with storage and network services – samples are discussed in Section 5.

We first determine a few configuration determination use cases to get a comprehensive picture where the pattern technique can be used and then discuss the core solutions in turn.

A. Use Cases

In general, there is a possibly empty set of patterns $MS(s_i)$ for each service s_i, i.e. some services have usage information, others have no usage information in the matrix itself. Consider the sample matrix from the previous section. Three use cases emerge that indicate how the matrix can be used:

- *Configuration Determination – Existing Patterns*: For a service s with monitoring history: Since s_1 has an invocation history for various patterns for a requested response time of *0.45s*, we can return this set of patterns including *M1* and *M3*.
- *Configuration Determination – Non-existing Patterns*: For a given service s without history: Since s_2 has no invocation history for a required response time of *2s*, we can utilise collaborative filtering for the prediction of settings – i.e. use similar services to determine patterns for the given service [8,9].
- *Configuration Test*: For a given triple of SWP values and a service s: If a given s_1 has an invocation history for a required response time of *2s* and we have a given workload configuration, we can test the compliance of the configuration with a pattern using the matrix.

B. Pattern-based Configuration Determination

If patterns exist that satisfy the performance requirements, then these are returned as candidate configurations. In the next step, a cost-oriented ranking of these can be done. We use quality level to cost mappings that will be explained in Section V below. If no patterns exist for a particular service (which reflects the second use case above), then these can be determined by prediction through collaborative filtering, see [25].

QoS Prediction Process. For any service s, if there is information of s_v under pattern m_i, then calculate the similarity between other services s_j and s_v. We can get the k neighbouring services of service s_j through a similarity calculation. The set of these k services is $S = \{s_1', s_2', \ldots, s_k'\}$. We fill the null (empty) QoS values for the target invocation using the information in this set. Using the information in S, we then calculate the similarity of m_i with other patterns that have the information for target service s_j. We choose the most similar k' patterns of m_i, and use the information across the k' patterns and S to predict the quality of service s_j.

Service Similarity Computation. If there is no information of s_j in a pattern m_i, we need to predict the response time $q_{i,j}$ for s_j. Firstly, we calculate the similarity of s_j and services which have information within pattern m_i ranges. For a service s_v in I_i – where I_i is the set of services that have usage information within pattern m_i – we calculate the similarity of s_j and s_v. We need to consider the impact of configuration environment differences, i.e. redefine common similarity definitions. M_{vj} is the set of workload patterns which have the usage information of services s_v and s_j.

$$sim_S(s_v, s_j) = \frac{\sum_{\mathbf{m}_c \in M_{vj}} (q_{c,v} - \overline{q_v})(q_{c,j} - \overline{q_j})}{\sqrt{\sum_{\mathbf{m}_c \in Mvj} (q_{c,v} - \overline{q_v})^2} \sqrt{\sum_{\mathbf{m}_c \in Mvj} (q_{c,j} - \overline{q_j})^2}} \quad (4.1)$$

Here, $\overline{q_v}$ is the average quality value for service s_v and $\overline{q_j}$ the respective value for s_j. From this, we can obtain all similarities between s_j and others services which have usage information within pattern m_i. The more similar the service is to s_j, the more valuable its data is.

Predicting Missing Data. Missing or unreliable data can have a negative impact on prediction accuracy. In [26], we considered noise up to 10% to be acceptable. In order to deal with uncertainty beyond this, we calculate the similarity between two services and get the k neighbouring services. Then, we establish the k-neighbour matrix N_{sim} and complete the missing data. We add $s_{i,p}$ as the data of service s_p under pattern m_i if required:

$$s_{i,p} = \bar{q}'_p + \frac{\sum_{n \in S'} sim_{n,p} \times (q'_{i,n} - \bar{q}'_n)}{\sum_{n \in S'} (|sim_{n,p}|)} \quad (4.2)$$

Again, $\bar{q}'_p$ is the average quality value of s_p, and $sim_{n,p}$ is the similarity between s_n and s_p. Now every service $s \in S'$ has usage information within all pattern ranges in m_i.

Calculating Pattern Similarity and Prediction. There is QoS information of k neighbouring services of s_j in matrix N_{sim}. Some of them are prediction values. We can calculate the similarity of pattern m_i and other patterns using the correction cosine similarity method:

$$sim_M(m_i, m_j) = \frac{\sum_{s_k \in S} (t'_{i,k} - \overline{t'_i})(t'_{j,k} - \overline{t'_j})}{\sqrt{\sum_{s_k \in S} (t'_{i,k} - \overline{t'_i})^2} \sqrt{\sum_{s_k \in S} (t'_{j,k} - \overline{t'_j})^2}} \quad (4.3)$$

After determining the pattern similarity, the data of patterns with low similarity are removed from N_{sim}, the set of the first k patterns. The data of these patterns are retained for prediction, if $p_{i,j}$ is the data to be predicted as the usage data of service s_j within pattern m_i, it can be calculated.

$$p_{i,j} = \bar{q}'_i + \frac{\sum_{\mathbf{n} \in \mathbf{M}'} sim_{n,i} \times (q'_{n,j} - \bar{q}'_n)}{\sum_{\mathbf{n} \in \mathbf{M}'} (|sim_{n,i}|)} \quad (4.4)$$

The average QoS of data related to pattern m_i *is* $\bar{q}'_i$ and $sim_{n,i}$ is the similarity between patterns m_n and m_p.

C. Pattern-based Configuration Testing

We can use the pattern-QoS matrix to test a standard or any known resource configuration in SWP format (i.e., three concrete values rather than value ranges for the infrastructure aspects) – for instance in the situation outlined above for a service s_i for which its performance is uncertain. This can also be done instead of collaborative filtering, as indicated above, if the returned set of patterns is empty and a candidate configuration is available. Then, the matrix can be used to determine the respective QoS values, i.e., to predict quality such as performance in our case through testing as well.

This situation shall be supported by an algorithm that matches candidate configurations with stored workload patterns based on their expected service quality. The algorithm

takes into account whether or not possibly matching workload patterns exist.

Algorithm: Matching Candidate Configurations
Input: Service Usage Information of a Service
Output: Metrics Sorted by Correlation Degree

```
1   Match [ candidate configuration Config = < y1, y2, y3 >
    of target service s_i ] with  [characteristics (ranges)
    < low1~high1, low2~high2, low3~high3 > ] of stored pat-
    terns M_i.
2   If  there is a pattern that can be matched
        a.  Then  return it
        b.  Else  use Gray relevance analysis (formula
            (3.7)) to match a pattern
3   Let the matched pattern be m_i
        a.  Search information about matched pattern m_i in
            matrix M
        b.  If there is QoS information of service s_i in
            m_i,
                i.  Then  return it as expected QoS for
                          candidate configuration
               ii.  Else  If    no related QoS information
                                exists
                          Then  predict QoS by collaborat.
                                filtering (4.1)-(4.4)
4   Return                                                .
```

If no patterns exist, existing candidate configurations can be tested – to enable always a solution, at least one default configuration should be provided. Alternatively, similar services can be considered; these can be determined through collaborative filtering and then we would start again.

V. Cloud Service SLA, VM Configuration and Auto-Scaling

This section shall illustrate how the approach can be used in a cloud setting for resource (VM) configuration and auto-scaling. Predefined configurations for VMs and other resources offered by providers as part of standard SLAs could be the following that relate to the CPU, data/storage and network utilisation criteria < *Data*, *Network*, *CPU* > we used in Sections 3 and 4 for the SWPs:

32-bit VM	Bronze	Silver	Gold
Virtual CPU @ 1.25 GHz	1	2	4
Virtual Memory (GB)	2	4	4
Virtual Storage (GB)	60	120	240
Network Bandwidth (GB)	350	700	1400

Gold, Silver and Bronze are names for the different configurations. We can add pricing for Pay-as-you-Go (PAYG) and monthly subscription fees to the above scheme to take cost-based selection of configurations into account:

32-bit VM	Pay-as-you-Go	Bronze	Silver	Gold
CPU Hours	1/hr	100	150	200
Virtual Memory	0.05/hr	200	300	450
Virtual Storage	0.1/hr	60	120	240
Network Bandwidth	10/TB	35	50	75

We define below a *cost function C : Config -> Cost* to formalise such a table. The categories based on the resource workload configurations can now be aligned by the provider with QoS values that are promised in the SLA – here with response time and availability guarantees filled in the *Configuration-Quality* matrix *CQ*:

$$\begin{array}{r} Gold \\ Silver \\ Bronze \end{array} \begin{bmatrix} 0.75 & 99.99 \\ 1.0 & 99.9 \\ 1.5 & 99 \end{bmatrix} \tag{5.1}$$

In general, the *Configuration-Quality Matrix* is defined by

$$CQ = [\, c_{ij} \,] \quad \text{with} \quad i : \textit{configuration category} \quad \text{and} \quad j : \textit{quality attribute} \tag{5.2}$$

A selection function σ determines suitable workload patterns M_i for a given quality target q as defined in the *Configuration-Quality* matrix and a service s_j:

$$\sigma(q,s) = \{\, M_i \in M \mid MS(q,s_j) \in q_{ij} \,\} \tag{5.3}$$

From this set of workload patterns $\{M_1, \ldots, M_n\}$, we determine the most optimal one in terms of resource utilisation. For minimum and maximum utilisation thresholds min_U and max_U, the best pattern is selected based on a minimum deviation of pattern ranges $M_i(q)$ across all quality factors (based on the overall mean value) from the threshold average value, defined as the *mean average deviation* (where $\bar{x}$ indicates the mean value for any expression x):

$$\min_i \sqrt{\sum \left(\overline{(maxU - minU)} - \overline{Mi(q)} \right)^2} \tag{5.4}$$

The thresholds can be set at 60% and 80% of the pattern range averages to achieve a good utilisation with some remaining capacity for spikes.

Based on the best selected SWP M_i with the given key metrics, a VM image can be configured accordingly in terms of CPU, data and network parameters and deployed with the service in question. If several SWPs apply to meet performance requirements, then costs can be considered to select the cheapest offer (if the cost in the table reflects in some way the real cost of provisioned resources and not only charged costs)

$$\sigma^{Cost}(q,s) = \min_i C(\sigma(q,s)) \tag{5.5}$$

for a *cost function C* that maps a pattern in $\sigma(q,s)$ to its cost value. The cost function can create a ranking of otherwise equally suitable patterns or configurations.

The service-based framework presented in Sections III and IV was here applied to the cloud context by linking it to standard configuration and payment models. Specific challenges arose from the cloud context that we have addressed are:

- Standard cloud payment models allow an explicit costing, which we took into account here through the cost function. Essentially, the cost function can be used to generate a ranked list of candidate patterns for a required QoS value in terms of the operational cost. In [30], we have demonstrated that different performance result, but also costs vary for a given configuration pattern.

- Cloud solutions are subject to (dynamic) configurations, generally both at IaaS and PaaS level. While our configuration here is geared towards typical IaaS attributes, our implementation work with Microsoft Azure (see Section VI) also demonstrates the possibility and benefit of PaaS-level configuration. In [30], we have discussed different PaaS-level storage configurations and their cost and performance implications.
- User-driven scalability mechanisms such as CloudScale or CloudWatch or the AWS Autoscaling typically work on scaling rules defined on the granularity of VMs added/removed. Our solution is based on similar metrics, e.g. GB for storage or network bandwidth, i.e. further automates these solutions.

We have briefly mentioned uncertainties that arise from cloud environments in Section II. While this aspect is neglected here, in [26], we have presented an approach that adds uncertainty handling on top of prediction for VM (re-) configuration. Uncertainties arise for instance from incomplete or potentially untrusted monitoring data or from varying needs and interpretations of stakeholders regarding quality aspects. The approach in [26] adds a fuzzy logic processing on top of a prediction approach.

VI. Implementation and Evaluation

Implementation. The implementation of the prediction and configuration technique covers different parts – offline and online components:

- The pattern determination and the construction of the patterns-quality matrix is done off-line based on monitoring logs. The matrix is needed for dynamic configuration and can be updated as required in the cloud system. For the prediction, the accuracy is central. As the construction is offline, performance overhead for the cloud environment is, as we will demonstrate, neglectable.
- The actual prediction through accessing the matrix is done in a dynamic cloud setting as part of a scaling engine that combines prediction and configuration. Here the acceptable performance overhead for the prediction needs to be demonstrated.

For both the accuracy and performance concern, we use a standard prediction solution, collaborative filtering (CF) as the benchmark, which is widely used and analysed in terms of these properties, cf. [8,9] or [12,13].

We have implemented a simulation environment with a workload generator to evaluate accuracy of the prediction approach and the performance of the prediction-based configuration. We provided 100 application services from three different categories, each category either sensitive to data size, network throughput or CPU utilization. Figs. 2 and 3 describe the testbed with the monitoring and SWP extraction solution for Web services. The primary concern is the accuracy of the pattern extraction and pattern-based prediction of performance for deployed services. Furthermore, as dynamic reconfiguration, i.e. auto-scaling, is an aim, also performance needs to be looked at.

We have tested our scalability management in Microsoft Azure. We have implemented a range of standard applications, including an online shopping application, a services management solution and a video processing feature to determine the quality metrics for different service and infrastructure configuration types. For the first two, we used the Azure Diagnostics to collect monitoring data (Fig. 2). We also created an additional simulation environment to gather a reliable dataset without interference from uncontrollable cloud factors such as the network (Fig. 3).

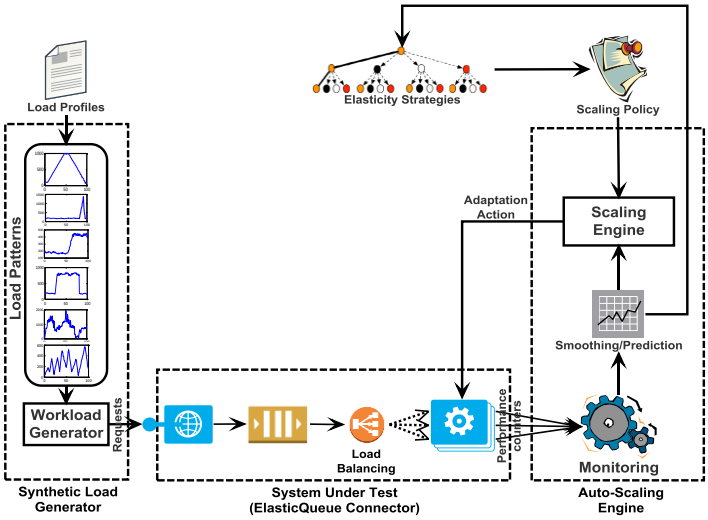

Fig. 2. Evaluation Architecture.

Elsewhere, we have implemented our prediction mechanism in other platforms. Work we described in [26] deals with how to implement this in a scalability solution such as Amazon AWS where monitored workload and performance metrics are considered together with a prediction of anticipated behaviour to configure the compute capabilities.

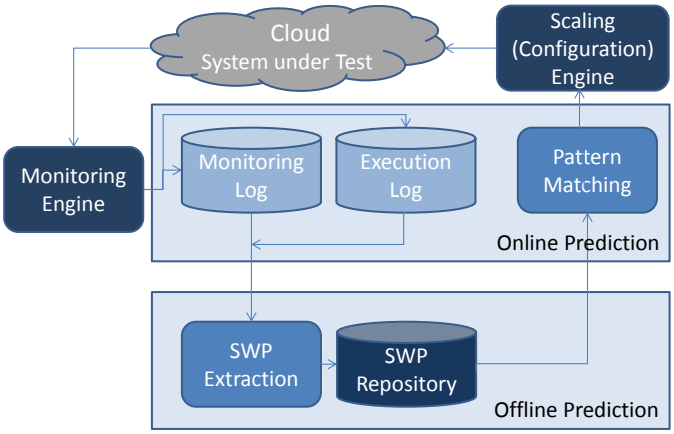

Fig. 3. Scaling Engine Components – Pattern and Prediction

Accuracy. Reliable performance guarantees based on configuration parameters is the key aim – real performance needs to match the expected or promised one for the provider to fulfil the SLA obligations. Accuracy in virtualisation environments is specifically challenging [28] due to the layered architecture, shared resources and distribution.

Accuracy of prediction is measured in terms of deviation from the real behaviour. The metric we use here is based on the *mean absolute error* (MAE) between prediction (SLA imposed) and real response time, which is the normal choice to measure prediction accuracy. Different characteristics of QoS have different ranges. Consequently, we use NMAE (the Normalized Mean Absolute Error) instead of the MAE. The smaller the NMAE, the more accurate is the prediction. We compare our solution with similar methods based on traditional prediction methods in terms of matrix density. This covers different situations from situations where little is known about the services (density is low) and situations where there is a reliable base of historic data for pattern extraction and prediction (high density).

In earlier work, we included average-based predications and classical collaborative filtering (CF) in a comparison with our own hybrid method (MCF) of matrix-based matching and collaborative filtering [25]. The NMAE of $k=15$ and $k=18$ (higher or lower ks are not interesting as lower values are statistically not significant and higher ones only show a stabilisation of the trend) shows an accuracy improvement for our solution compared to standard prediction techniques, even without utility function and exponential smoothing, see Fig. 4. For this evaluation here, we also include time series (TS) into the comparison. For the evaluation, we considered some noisy data which cannot be in any pattern. We also removed invocation data and then predicted it using the CF, MCF and also the TS time series method from [22].

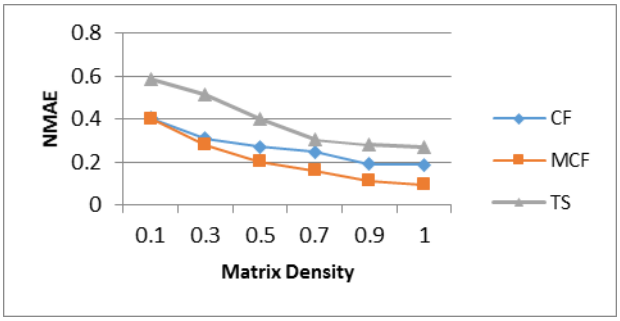

Fig. 4. Accuracy Evaluation

We can observe that an increase of the dataset size improves the accuracy significantly. In all cases, our MCF approach outperforms the other ones.

Efficiency Overhead (Runtime). For automated service management – in the context of cloud auto-configuration and auto-scaling – we need sufficient performance of the extraction and matching approach itself. To be tested in this context are the performance of three components:

1) SWP Extraction from Logs (Matrix Determination)
2) Configuration-Pattern Matching (Existing Patterns)
3) Collaborative Filtering (Non-Existing Patterns)

For cases 1 and 2, we determined 150 workload patterns from 2400 usage recordings. We tested the algorithm on a range of different datasets extracted from a number of documented benchmarks and test cases. Compared to other work based on the TS and CF solutions, the matrix for collaborative computation is reduced from 2400*100 to 150*100, which reduces execution time significantly by the factor 16. For case 3, only when a matched pattern provides no information for a target service, the calculation for collaboration prediction is required – see Figure 5 where we compare prediction with and without the pattern-based matrix utilisation.

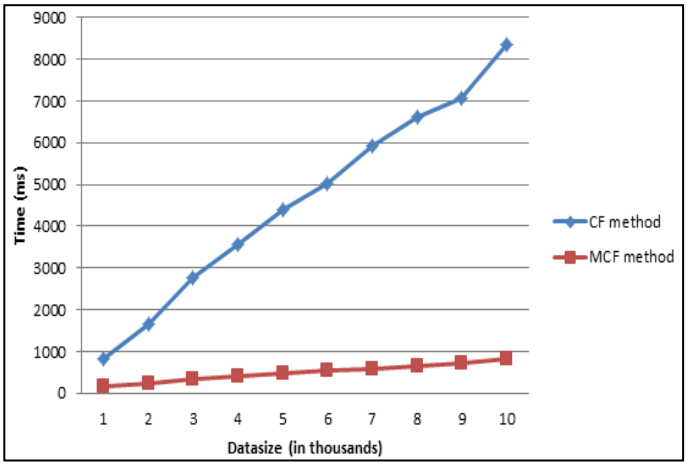

Fig. 5. Performance Evaluation

Thus, in conclusion, the computational effort for the dynamic prediction is decreased to a large extent due to the already partially filled matrix. As already explained, the performance of the pattern extraction and matrix construction (DBSCAN based clustering and collaborative filtering) can be computationally expensive, but can be done offline and only the matrix-based access (as demonstrated in the performance figure above) impacts on the runtime overhead for the configuration. However, as the figure shows, our method's overhead increases only slowly even if the data size increases substantially. Consequently, the solution in this setting is no more intrusive than a reactive rule-based scalability solution such as Amazon AWS Auto Scaling that would also follow the architecture in Fig.2.

VII. Related Work

QoS-based service selection in general has been widely covered. There are three main categories of prediction-based approaches for selection.

- The first one covers statistical methods, which is often adopted for simplicity [1,2,4,5,6]. These methods are simple and easy to implement.
- The second category selects services based on user feedback and reputation [16,17]. It can avoid malicious feedback, but does not consider the impact of SLA requirements and the environment and cannot customise prediction for users.
- The third category is based on collaborative filtering [8,9,11], which is a widely adopted recommendation method [18-20] – e.g., [19] summarizes the application of

collaborative filtering in different types of media recommendation. Here, we combine collaborative filtering with service workload patterns, user requirements and SLA obligations and preferences. This considers different user preferences and makes prediction personalized, while maintaining good performance results.

To demonstrate that our solution is an advancement compared to existing work on prediction accuracy, we had singled out two approaches for categories 1 and 3 for the evaluation above.

Some works integrate user preferences and user characteristics into QoS prediction [8,9,11,12], e.g. [8,9] propose prediction algorithms based on collaborative filtering. They calculate the similarity between users by their usage data and predict QoS based on user similarity. This method avoids the influence of the environment factor on prediction. However, even the same user will have different QoS experiences over time depending on the configuration of the execution environment or will work with different input data. Current work generally does not consider user requirements. Another current limitation of current solutions is low efficiency as we demonstrated. Our work in [26] is a direction based on fuzzy logic to take user scalability preferences into account for a cloud setting.

In [24,25], pattern approaches are proposed. [24] suggests pattern-based management for cloud configuration management, but without a detailed solution. [25] is about bottom-up QoS predication for standard service-based architectures, while in this paper QoS requirements are used to predict suitable workload-oriented configurations taking specifically cloud concerns into consideration. We added additionally exponential smoothing and utility functions and the cost analysis here, but draw on some evaluation results from [25] in comparison to standard statistical methods.

Supporting cloud service management can automatically scale the infrastructure to meet the user/SLA-specified performance requirements, even when multiple user applications are running concurrently. Jamshidi et al. [26] deal with multi-user requirements as part of an uncertainty management approach, which performs well based on a fuzzy-logic approach, but cannot in comparison demonstrate as accurate prediction as only exponent smoothing based on a few workload patterns is done. Ghandi et al. [27] also leverage application level metrics and resource usage metrics to accurately scale infrastructure. They use Kalman filtering to automatically learn changing system parameters and to proactively scale the infrastructure, but have less of a performance gain than through patterns in our solution. Another work in this direction is [29], where the solution aims to automatically adapt to unpredicted conditions by dynamically updating a Kriging behaviour model. These deal with uncertainty concerns that we have excluded. However, an integration of both directions would be beneficial in the cloud. These approaches can add the uncertainty management solutions required.

The proposed method in this paper takes full account of user requirements (reflected in SLA obligations for the provider), the network and computational factors. It abstracts the service workload pattern to keep the service QoS steady. When user/SLA requirements are known, prediction-base configuration can be done based on matched patterns. This approach is efficient and reduces the computational overhead.

VIII. Conclusions

Web or cloud services [23] usually differ with respect to QoS characteristics. Relevant service-level qualities are response time, execution cost, reliability, or availability. There are many factors that impact on QoS [15]. They depend not only on the service itself, but also how it is deployed. Some factors are static, some are run-time static, the others are dynamic. Run-time static and dynamic factors like client load, server load, network channel bandwidth or network channel delay are generally uncertain, but can be influenced by suitable configuration in virtualised environments such as the cloud. Most factors can be monitored, and their impact on service-level quality can be calculated as part of a service management solution. Service management in cloud environments requires SLAs for individual users to be managed continuously through dynamic platform and infrastructure configuration, based on monitored QoS data.

We provided a solution that links defined SLA obligations for the provider in terms of service performance with lower-level metrics from the infrastructure that facilitates the provisioning of the service. Our solution enables cloud workload patterns to be associated to performance requirements in order to allow the requirements to be met through appropriate configuration.

Performance management is still a problem in the cloud [10,31]. While availability is generally managed and, correspondingly, SLA guarantees are made, reliably guaranteeing performance is not yet solved. Through a mining approach we can extract resource workload patterns from past behaviour that match the performance requirement and allow a reliable prediction of a respective configuration for the future.

In order to further fine-tune the approach, in the future we will take more infrastructure metrics into account. More specific cloud infrastructure solutions and more different use cases shall be used on the experimental side to investigate whether different patterns emerge either for different resource provisioning environments or for different application domains and consumer customisations [32]. Another crucial direction is the incorporation of uncertainty into the approach. Uncertainty, as discussed, manifests itself through incomplete and untrustworthy data or the consequences of multiple stakeholders in the cloud. We propose to follow [26] and use a fuzzy logic approach to incorporate this.

Acknowledgment

This research has been supported by the Fundamental Research Funds for the Central Universities of China (grant N130317005), by the National Natural Science Foundation of China (grant 61402090), and the Irish Centre for Cloud Computing and Commerce IC4, an Irish national Technology Centre funded by Enterprise Ireland, the Irish Industrial Development Authority, and by Science Foundation Ireland (International Strategic Cooperation Award Grant Number SFI/13/ISCA/2845).

REFERENCES

[1] Cardoso J., Sheth A., Miller J., Arnold J., Kochut, K.: Quality of Service for Workflows and Web Service Processes. Jrnl of Web Sem 1(3):281-308 (2004)

[2] Kritikos, K., Plexousakis, D.: Requirements for QoS-based Web service description and discovery. IEEE Transact on Services Computing, 2(4), 320-337 (2009)

[3] Huang, A. F. M., Lan, C. W., Yang, S. J. H.: An optimal QoS-based Web service selection scheme. Information Sciences, 179(19): 3309-3322 (2009)

[4] Ye, Z., Bouguettaya, A., Zhou, X.: QoS-Aware Cloud Service Composition based on Economic Models. Service-Oriented Computing, Springer, 111-126 (2012)

[5] Alrifai, M., Skoutas, D., Risse, T.: Selecting skyline services for QoS-based web service composition. Proc. Intl Conf on World Wide Web, ACM, 11-20 (2010)

[6] Zeng, L., Benatallah, B., Ngu, A. H. H., et al.: QoS-Aware middleware for Web services composition. IEEE Transact on Softw Eng, 30(5), 311-327 (2004)

[7] Yu, T., Lin, K. J.: Service Selection Algorithms for Web Services with End-to-end QoS constraints. Inf Syst and E-Bus Management, 3(2):103-126 (2005)

[8] Shao, L., Zhang, J., Wei, Y., et al.: Personalized QoS prediction for Web services via collaborative filtering. Intl Conf on Web Services ICWS'07, 439-446 (2007)

[9] Zheng, Z., Ma, L. M. R., et al.: Qos-aware web service recommendation by collaborative filtering. IEEE Transact on Services Computing, 4(2), 140-152 (2011)

[10] Chaudhuri, S.: What next?: a half-dozen data management research goals for big data and the cloud. Proc. 31st Symp. Princ. of Database Systems (2012)

[11] Wu, G., Wei, J., Qiao, X., et al.: A Bayesian network based QoS assessment model for web services. IEEE Intl Conf on Service Computing, 498-505 (2007)

[12] Li, Z., Bin, Z., Ying, L., et al. A Web Service QoS Prediction Approach Based on Collaborative Filtering. Asia-Pacific Serv Comp Conf APSCC10,725-731 (2010)

[13] Li, Z., Bin, Z., Jun, N., et al.: An Approach for Web Service QoS prediction based on service using information. Intl Conference on Service Sciences ICSS'2010. 324-328 (2010)

[14] Ester, M., Kriegel, H. P., Sander, J., et al.: A density-based algorithm for discovering clusters in large spatial databases with noise. Intl Conf on Knowledge Discovery in Databases and Data Mining (KDD-96), 226-232 (1996)

[15] Lelli, F., Maron, G., Orlando, S.: Client Side Estimation of a Remote Service Execution, IEEE International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS) (2007)

[16] Vu, L. H., Hauswirth, M., Aberer, K.: QoS-based Service Selection and Ranking with Trust and Reputation Management. Comp Sci, 3760(2005) , 466-483 (2005)

[17] Yan, L., Minghui, Z., Duanchao, L., et al.: Service selection approach considering the trustworthiness of QoS data. Jrnl of Software 19(10), 2620-2627 (2008)

[18] Sarwar, B., Karypis, G., Konstan, J., et al.: Item-based collaborative filtering recommendation algorithms. World-Wide Web Conf. ACM, 285-295 (2001)

[19] Chun, Z., Chunxiao, X., Lizhu, Z.: A Survey of Personalization Technology. Journal of Software, 13(10), 1852-1861 (2002)

[20] Hailing, X., Xiao, W., Xiaodong, W., Baoping, Y.: Comparison study of Internet recommendation system. Journal of Software, 20(2):350-362 (2009)

[21] Ailing, D., Yangyong, Z., Bole, S.: A collaborative filtering recommendation algorithm based on item rating prediction. Jrnl of Software 14(9):1621-1628 (2003)

[22] Cavallo B, Di Penta M, Canfora G. An empirical comparison of methods to support QoS-aware service selection. 2nd International Workshop on Principles of Engineering Service-Oriented Systems. 64-70 (2010)

[23] Pahl, C., Xiong, H.: Migration to PaaS Clouds - Migration Process and Architectural Concerns. International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems MESOCA'2013 (2013)

[24] Srinivas, D.: A patterns/workload-based approach to the cloud. DIMS Lightning Talk. IBM (2013)

[25] Zhang, L., Zhang, B., Pahl, C., Xu, L., Zhu, Z: Personalized Quality Prediction for Dynamic Service Management based on Invocation Patterns. Intl Conference on Service Oriented Computing ICSOC (2013)

[26] Jamshidi, P., Ahmad, A., Pahl, C. Autonomic Resource Provisioning for Cloud-Based Software. 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems SEAMS'14 (2014)

[27] Gandhi, A., Harchol-Balter, H., Raghunathan, R., Kozuch M.A.: Autoscale: Dynamic, robust capacity management for multi-tier data centers. ACM Transactions on Computer Systems (TOCS) 30 (4), 14 (2012)

[28] Gmach, D., Rolia, J., Cherkasova, L., Kemper, A.: Workload Analysis and Demand Prediction of Enterprise Data Center Applications. Proceedings of the 2007 IEEE 10th International Symposium on Workload Characterization (2007)

[29] Gambi, A., Toffetti, G., Pautasso, C., Pezze, M.: Kriging controllers for cloud applications. Internet Computing, IEEE 17.4: 40-47 (2013)

[30] Xiong, H., Fowley, F., Pahl, C., Moran, N.: Scalable Architectures for Platform-as-a-Service Clouds: Performance and Cost Analysis. European Conference on Software Architecture ECSA'14 (2014)

[31] Jamshidi, P., Ahmad, A., Pahl, C. Cloud Migration Research: A Systematic Review. IEEE Transactions on Cloud Computing 1(2):142-157 (2013)

[32] Wang, Bandara, K.Y., Pahl, C. Process as a Service - Distributed Multi-tenant Policy-based Process Runtime Governance. IEEE International Conference on Services Computing SCC'2010. IEEE Press (2010)