Statistical Detection of QoS Violations Based on CUSUM Control Charts

Ayman Amin
Faculty of Information and Communication Technologies
Swinburne University of Technology
Hawthorn, VIC 3122, Australia
aabdellah@swin.edu.au

Alan Colman
Faculty of Information and Communication Technologies
Swinburne University of Technology
Hawthorn, VIC 3122, Australia
acolman@swin.edu.au

Lars Grunske
Software Engineering: AG AQUA
University of Kaiserslautern
Kaiserslautern,67653, Germany
grunske@cs.uni-kl.de

ABSTRACT

Currently software systems operate in highly dynamic contexts, and consequently they have to adapt their behavior in response to changes in their contexts or/and requirements. Existing approaches trigger adaptations after detecting violations in quality of service (QoS) requirements by just comparing observed QoS values to predefined thresholds without any statistical confidence or certainty. These threshold-based adaptation approaches may perform unnecessary adaptations, which can lead to severe shortcomings such as follow-up failures or increased costs. In this paper we introduce a statistical approach based on CUSUM control charts called AuDeQAV - Automated Detection of QoS Attributes Violations. This approach estimates at runtime a current status of the running system, and monitors its QoS attributes and provides early detection of violations in its requirements with a defined level of confidence. This enables timely intervention preventing undesired consequences from the violation or from inappropriate remediation. We validated our approach using a series of experiments and response time datasets from real-world web services.

Categories and Subject Descriptors

D.2 [**Software Engineering**]: Subjects—*Software configuration management*

General Terms

Algorithms, Performance

Keywords

Quality of Service (QoS), Runtime Monitoring, QoS Violation, Runtime Adaptation, CUSUM Control Charts

1. INTRODUCTION

Software systems are being ever-increasingly used in our daily life and have become a crucial part of various critical and non-critical applications. Consequently, there is an increasing need for software systems that are more reliable, with higher performance, and support more users [4]. At runtime, software systems may suffer from changes in their operational environment or/and requirements specification, so they need to be adapted to satisfy the changed environment or/and specifications [28, 6, 3]. The research community has developed a number of approaches to building adaptive systems that respond to such changes, for example Rainbow [10], MUSIC [26], and StarMX [2], QoSMOS [3], just to name a few.

Currently, several approaches have been proposed either to monitor QoS attributes at runtime with the goal of detecting QoS violations in order to trigger adaptations (e.g. [13, 19, 23, 32, 33, 35, 37]), or to verify whether QoS values meet the desired level to detect violations of Service Level Agreements (SLAs) (e.g. [18, 27, 29, 14]). These approaches detect violations of QoS requirements by observing the running system and determining QoS values. If these values exceed a predefined threshold, they are considered to be QoS violations. These approaches have two important drawbacks.

- First, the approaches rely on detecting QoS violations based on an observation or sample. If there is a background volatility in measured values, there is no indication of statistical confidence that the sample in fact indicates a systemic problem. Consequently, unnecessary adaptations maybe performed leading to failures or increased costs associated with adaptation.

- Second, a running software system's typical performance capability is not characterized, so any unusual deviations from that behavior cannot be detected other than gross violations of QoS requirements. As a result we use a capability estimation technique that characterizes the current status of the software system behavior, which may help to improve detecting violations.

In our previous work [1] addressing these limitations we have proposed CREQA approach based on Shewhart and CUSUM control charts for the runtime evaluation of the software system's behavior to detect out-of-control situations, where the system's behavior has significantly changed.

However, the CREQA approach evaluates the software system's behavior *without* assuming any predefined QoS requirements. This approach may be useful in some cases but in most of the real cases there are predefined QoS requirements that need to be monitored and evaluated at runtime.

To address the drawbacks of the existing approaches and the limitation of our previous work, we propose a statistical approach called AuDeQAV (Automated Detection of QoS Attributes Violations) based on CUSUM control charts for the runtime detection of QoS attributes violations. This approach consists of four phases: (1) Estimating the running software system capability in terms of descriptive statistics, i.e. mean, standard deviation, and confidence interval of QoS attributes, in order to describe the current normal behavior; (2) Building a CUSUM control chart using the given QoS requirements; (3) After getting each new QoS observation, updating the CUSUM chart statistic and checking for statistically significant violations; (4) In case of detecting violations, providing warning signals to the management layer to take the required adaptation actions.

The main contributions of this paper include:

1. We propose an approach based on CUSUM control charts that characterizes a software system's capability and then monitors QoS attributes to detect QoS requirement violations before they lead to undesired consequences.

2. We use a case study and perform experiments to evaluate the accuracy and performance of the approach. In addition, we compare the approach with the threshold based method to highlight the former's advantages.

3. We apply the proposed approach to QoS response time data sets of real web services [5] to evaluate the applicability of the proposed approach.

The rest of the paper is organized as follows. Section 2 describes the research problem through a presentation of a motivating scenario and a discussion of related work. Section 3 provides some background on statistical control charts. The proposed approach AuDeQAV is discussed in Section 4. Evaluation of the proposed approach is reported in Section 5. Section 6 concludes the paper and outlines directions for future work.

2. RESEARCH PROBLEM

In this section we first present a scenario that illustrates QoS requirements of a real application. We then investigate the extent to which solutions presented in the literature can meet these requirements. The identified shortcomings are used to motivate our work.

2.1 Motivating Scenario

Scenario: A hospital has elderly patients who return home after a treatment but still need a daily follow up to complete the treatment course. The patients live in remote areas, and it is inconvenient for both the doctors and the patients to interact directly. To address this problem, the hospital will develop a Patient Assistance (PA) system [36, 12], which is a software- and telecommunication-based service, to enable the patients and the doctors to communicate in a timely manner. The patients' situations have different criticality levels. As such, there is a need to define a limit on the PA system's response time as QoS attribute based on the urgency of a patient's need. For example, when the patient's criticality level is high, the PA system should respond within two seconds to each request with a probability of 95% or higher.

Scenario Analysis: To assure the QoS requirements of the PA system, the hospital needs a management framework that monitors the PA system's QoS metrics. Then, based on the monitored QoS data, the management framework determines violations of the above requirements and gives warning signals to trigger the required adaptation actions, e.g. allocating more resources. This needs to be achieved before the violations lead to undesired consequences, e.g. putting a patient life in danger.

Consequently, to get a software system that is able to adapt itself in order to avoid QoS requirement violations its management framework must be able to monitor QoS attributes and detect in a timely manner QoS violations before they lead to undesired consequences. This can be achieved by continuously monitoring and collecting QoS values of the running system, and then using the collected data to infer unwanted systemic behavior.

2.2 Related Work

The above scenario shows that in order to fulfil the application requirements, the management framework needs the ability to monitor QoS attributes. In this subsection we investigate the literature and discuss how existing techniques/frameworks address the problem.

The Rainbow framework [10, 8], which has been developed to dynamically monitor and adapt a running software system, uses monitoring techniques to detect QoS violations after they have occurred by simply comparing the collected QoS values with the predefined threshold. It then uses condition-action scenarios (tactics) to fix these QoS violations. Other approaches [7, 9] that build on the Rainbow framework use the same mechanism to detect and address QoS violations.

The work in [31] proposes DySOA (Dynamic Service Oriented Architecture) that uses monitoring to track and collect information regarding a set of predefined QoS parameters (e.g. response time and failure rates), infrastructure characteristics (e.g. processor load and network bandwidth), and even context information (e.g. user GPS coordinates). The collected QoS information is directly compared to the QoS requirements; and in case of a deviation, the reconfiguration of the application is triggered.

In [16], a middleware architecture is proposed to enable SLA-driven clustering of QoS-aware application servers. This middleware consists of three components: The Configuration Service is responsible for managing the QoS-aware cluster, the Monitoring Service observes at runtime the application and verifies whether the QoS values meet the desired level to detect violations of SLAs, and the Load Balancing Service intercepts client requests to balance them among different cluster nodes. If the QoS values delivered by the cluster deviate from the desired level (e.g., the response time breaches the predefined threshold), the middleware reconfigures that cluster by adding clustered nodes.

Michlmayr et al. [18] have presented a framework that observes the QoS values and checks whether they meet the required levels to detect possible violations of SLAs. Once an SLA violation is detected, adaptive behavior can be triggered such as the hosting of new service instances. Also, in [27] a QoS requirements satisfaction is analyzed in terms of the SLA fulfillment, which is considered as the main quality criterion of a service.

Recent studies [19, 35] propose autonomic frameworks to monitor QoS attributes and dynamically adapt service-based systems in an automated manner in response to requirements violations. In [19], Mirandola and Potena propose a framework that dynamically adapts a service based system while minimizing the adaptation costs and guaranteeing a required level of QoS. This framework triggers adaptation actions automatically in response to runtime violation of system QoS constraints, or the availability/non-availability of services in the environment. Thongtra and Aagesen [35] present a framework for service configuration that has *goals*, which express required performance and income measures, and *policies*, which define actions in states with unwanted performance and income measures. This framework monitors quality attributes constraints and in the case of violations it triggers pre-defined policies.

All these approaches detect QoS violations by just comparing observed QoS values to predefined thresholds. Using such an approach has critical limitations:

- First, the current approaches collect QoS attribute values of a running system at specific times or periodically, i.e. second, minute, or hour; and if one (or more) of these collected values exceeds a predefined threshold they are considered to be QoS violations of the running system. However, in that sense these collected QoS values are considered to be samples, rather than a collection of all the experienced QoS values. This implies that false classifications are likely, and to correctly and significantly generalize to the running system over time one needs to either: (1) Collect all the experienced QoS values, which is inapplicable in most real-world cases, or (2) Use a statistical method that can generalize the behavior with a specified confidence level, e.g. 95%.

- Second, a software system capability, in terms of its different QoS attributes, can not be estimated. Where, it just verifies whether the QoS values exceed the given threshold and does not provide any information about the software system current status (e.g. the average and variance of the system response time). This updated capability estimation can help software system engineers to continuously obtain the current status of software system behavior, which enables them to detect a change or deviation in that behavior before undesired events occur.

The proposed AuDeQAV approach addresses these drawbacks. It uses collected QoS values to estimate the capability of the software system, and uses the predefined QoS requirements to build CUSUM chart to detect significant violations in these requirements with a specified confidence level.

3. CONTROL CHARTS BACKGROUND

Statistical control charts are an efficient quality control technique for online monitoring and detecting of changes and violations in a given statistical process [20, 11, 34, 15]. Control charts are constructed by taking sample readings of the variable to be controlled from the process, then plotting values of the quality characteristic of interest in time order on a chart. This chart contains a center line (CL), which represents the average value of the quality characteristic, and two other horizontal lines called the upper control limit (UCL) and the lower control limit (LCL). These control limits are chosen such that if the process is in-control, which means there is no change or shift in the process, nearly all of the sample points will fall between them. In general, if the values plot within the control limits, the process is assumed to be in-control, and no action is necessary. However, a value that falls outside of the control limits is taken as a signal that a change has occurred, the process is out-of-control, and investigation and corrective action are required.

The general model for a control chart proposed first by Shewhart [30] can be given as follows. Let X be a sample statistic that measures some quality characteristic of interest, and suppose that μ_X and σ_X are the mean and the standard deviation of X, respectively. Then chart parameters become:

$$UCL = \mu_X + D \cdot \sigma_X,\ CL = \mu_X,\ LCL = \mu_X - D \cdot \sigma_X \quad (1)$$

where D is the distance expressed in standard deviation units of the control limits from the center line. Usually D is chosen in practice to be 3 standard deviations to give a statistical confidence level of about 99.7%. Thus, this control chart is called three sigma (3-σ) chart.

In the cases that the quality characteristic of interest is in the form of required probability, i.e. proportion, the Shewhart chart is called P-chart and its parameters become:

$$UCL = P_0 + D \cdot \sigma_{P_0},\ CL = P_0,\ LCL = P_0 - D \cdot \sigma_{P_0} \quad (2)$$

where P_0 is the required proportion to be monitored and its standard deviation σ_{P_0} is computed as $\sigma_{P_0} = \sqrt{(}P_0(1 - P_0)/n)$.

Example ▷ For the sake of illustrating how a P-chart can be used to monitor QoS requirement and detect violations, suppose that there is a running software system and the requirement is "No more than 2 seconds response time for 95% of the requests". In other words, the requirement is that the proportion of response time that is greater than 2 seconds is 0.05. To use a P-chart to monitor this software system performance the following steps are required: First, samples of size n (e.g. $n = 30$) from the system response time are taken. Second, a variable called rt is created, and it takes the value 1 if the response time is greater than 2 seconds and 0 otherwise. Third, the proportion of the response time (P_{rt}) that is greater than 2 seconds is computed, and the 3-σ control limits of P-chart are constructed using equation (2) and response time requirement as follows: UCL = 0.05 + 3($\sqrt{(}0.05 * 0.95/30)$) = 0.17, CL = 0.05, and LCL = 445 - 3(($\sqrt{(}0.05 * 0.95/30)$) = -0.17. Fourth, the values of computed proportions P_{rt} are checked and compared to the control limits; and if there is any value that falls outside of these control limits, it is taken as a warning signal that a violation in the response time requirement has occurred. ◁

P-charts have a critical limitation when used at runtime because they wait until all the n sample observations are collected before computing the proportion. This process implies late detection of violations. Therefore, our proposed approach adopts advanced control charts that continuously monitor response time observations, and once the value is obtained it is checked and compared to the built chart limits. These control charts and their performance measures are discussed in detail in the following subsections.

3.1 CUSUM Control Charts

Page [22] has proposed CUSUM control charts as an effective alternative to the Shewhart control charts. These charts compute and plot the cumulative deviations sum of the sample values from a target (required) value. For example, suppose that samples of size $n = 1$ are collected, and X_j is the value of the j^{th} sample. Then, the CUSUM control chart is formed by plotting the quantity $C_i = \sum_{j=1}^{i}(X_j - \mu_0)$ against the sample number, where μ_0 is the target value for the quality characteristic of interest. Because the CUSUM charts combine information from several samples, they are more effective than Shewhart charts for detecting small process shifts. Furthermore, they are a good candidate for situations where an automatic measurement of the quality characteristic is economically feasible as they are particularly effective with samples of size $n = 1$ [15].

If the cumulative sum statistic C_i fluctuates around zero, it indicates that the process remains in-control at the target value. However, if it has a positive (or negative) drift, it signals that the quality characteristic values have shifted upward (or downward). Therefore, if the CUSUM has either positive or negative trend in the plotted points, this should be considered as a signal that the quality characteristic values have deviated from the target and required value.

Particularly, if the quality characteristic of interest is proportion (P) and samples observations (X_t) are binary data (0 and 1 values), the CUSUM statistic is called Bernoulli CUSUM. This statistic, to be used to detect an increase in P, is computed using a tabular procedure as follows:

$$B_t = max[0,\ B_{t-1} + X_t - K], \quad t = 1, 2, \ldots, \tag{3}$$

where the starting value is $B_0 = 0$. The parameter K is called the reference value for the CUSUM chart. For the value of K to be determined, it is necessary to specify the values P_0 and P_1, where P_0 represents the target or required value (in our work, it is a QoS requirement) and $P_1 > P_0$ is an out-of-control (violation) value of P that is required to be quickly detected. In other words, $(P_1 - P_0)$ represents the violation size in P that is required to be quickly detected. After specifying P_0 and P_1, K can be computed as follows:

$$K = r_1/r_2, \tag{4}$$

where, r_1 and r_2 are constants computed based on likelihood ratio test [15] as follows:

$$r_1 = -\ln[\frac{1 - P_1}{1 - P_0}]\ and\ r_2 = \ln[\frac{P_1(1 - P_0)}{P_0(1 - P_1)}] \tag{5}$$

The lower and upper control limits of CUSUM are computed as follows [25]:

$$LCL = -\frac{\ln\frac{(1-\alpha)}{\beta}}{2\ln\frac{P_1(1-P_0)}{P_0(1-P_1)}} \quad and \quad UCL = \frac{\ln\frac{(1-\beta)}{\alpha}}{2\ln\frac{P_1(1-P_0)}{P_0(1-P_1)}} \tag{6}$$

where α is an upper bound for an acceptable type I error (or a false positive) which means that the CUSUM chart decides that there is a violation in the QoS requirements of the running system even when the system is at the required level and does not have any violation, and β is an upper bound for an acceptable type II error (or a false negative) which means that the chart decides that there is no violation when the system has indeed violated its requirements. To improve the performance of CUSUM control charts, some researchers [17] have proposed the fast initial response (FIR) feature which permits a more rapid response to an initial out-of-control, especially at start-up or after the CUSUM chart has given an out-of-control signal.

In practice when the monitored proportion is small the LCL will be negative, and that means the lower limit is ineffective [24]. Consequently, our approach concentrates only on the upper limit (UCL), which in literature is normally called the decision interval H. It is clear from equation 6 that the value of this decision interval is determined by P_0, P_1, and the required α and β values. It is worth mentioning that statistically in practice $(1 - \alpha)$ and $(1 - \beta)$ refer to the statistical confidence and power levels of the chart, respectively.

3.2 Control Charts performance

The control chart performance is measured by the ability of the chart to detect changes in the process. The average run length (ARL), which is defined as the average number of samples that must be plotted on the chart before a sample indicates an out-of-control condition, is a commonly used measure for the performance of a control chart.

In the case of Shewhart chats, the ARL can be calculated from the mean of a geometric random variable [20]. For illustration, suppose that p is the probability that any point falls outside of the control limits, then, the ARL is computed as the inverse of that probability, $ARL = 1/p$. In the case of three sigma limits and an in-control process, we have $p = 0.0027$ based on the normal distribution and the in-control ARL (ARL_0) = 370. That means on the average a false alarm will be generated every 370 data points. In the case of out-of-control, to compute the out-of-control ARL (ARL_1); the probability p of a point falls outside of the control limits is needed and it depends on the shift size. Suppose the actual mean of process shifts by three standard deviations, then it is straightforward to show that $p = 0.5$ and $ARL_1 = 1/0.5 = 2$.

The ARL for the CUSUM chart depends on the decision interval H, which determines the width of control limits, and on the selected reference value K. Several authors investigated the optimal values of K and H, and they concluded that a CUSUM chart with smaller K is more sensitive to small shifts and violations [15]. Practically, it is advisable to choose the decision interval H that corresponds to an in-control ARL of roughly 740, which is equivalent to a 3σ Shewhart chart [15].

4. THE AUDEQAV APPROACH

The AuDeQAV is a statistical approach we introduce for the automated detection of QoS attributes violations. This approach uses CUSUM control charts. It has mainly two tasks. First, it uses collected QoS data to estimate at runtime the capability of a running system. This capability estimation is to obtain the current status of the software system's behavior. The capability is represented in terms of mean, standard deviation, and confidence interval of QoS attributes. Second, the AuDeQAV uses predefined QoS requirements and the software system capability estimates to construct a CUSUM model for monitoring the QoS values and detecting violations in their requirements.

To achieve these tasks, the AuDeQAV approach has four steps as depicted in Figure 1. In the following, we explain in detail these steps and illustrate the procedure based on the response time data of the Patient Assistance (PA) system.

Phase 1 (P1): Estimating Capability

In the first phase, the software system's capability is estimated in terms of descriptive statistics of specific QoS attributes. This estimation is performed after the software system has started, and it can be updated in different schedules:

1. After an adaptation has been triggered.
2. Every specific predefined period of time (including after an adaptation has been triggered), e.g. after every one day and also after any adaptation action has been triggered.
3. After getting each new QoS attribute value.

Basically, the main task of the approach in that phase is to use collected QoS values to estimate mean and standard deviation of the given QoS attributes, and then it uses these estimates to build QoS confidence interval, i.e. 95% confidence interval. In cases where the software system engineers do not have precise QoS requirements, they can use these QoS estimates and confidence intervals to update and refine these requirements. On the other hand, if there are precise requirements, the approach can initially estimate how these requirements are fulfilled from the collected QoS data. This initial estimation of requirements fulfilment can help the software system engineers later, when they have requirement violations, in triggering adaptation actions, e.g. by how much they need to increase the resources, or in refining the QoS requirements.

There is an important issue related to this phase: how many samples are needed to be used to accurately estimate the system capability? To address this issue, we use a sequential estimation method based on sampling theory. According to parametric estimation methods [21], the absolute relative error of estimating a population mean is less than a specified amount which can be evaluated as follows:

$$E = \frac{Z_{\alpha/2}\sigma}{\bar{x}\sqrt{n}} \tag{7}$$

where n, $\bar{x}$ and σ are the sample size, mean and standard deviation respectively. $Z_{\alpha/2}$ is a coefficient of the confidence level 100(1-α)%, and it is computed from the standard normal distribution. This equation can be used only when $\bar{x}$ is normally distributed. To address this normality issue we use the central limit theorem [21], which simply reports that a mean of sample of size at least 30 observations driven from any distributed population can be approximated to be normally distributed.

Therefore, our sequential estimation method works as follows:

1. Specify the maximum absolute relative error (E$_{req}$) that is required to estimate the software system's capability. The required confidence level is also specified, e.g. 95%, to compute the coefficient $Z_{\alpha/2}$.
2. Once 30 QoS observations are obtained, the approach computes QoS mean and standard deviation, $\bar{x}$ and σ, and then computes E as in equation 7.
3. The inequality $E_{req} \leq E$ is evaluated. If it is satisfied, use these estimates; otherwise wait until new QoS observation are received and repeat 2 and 3 until capability estimates with a required accuracy are obtained.

Example $\triangleright$ Suppose that the proposed approach is being used to monitor the PA system's response time values (referred as PA(RT)) and to detect a violation in its requirement which is: *"No response time is more than 2 seconds for 95% of the requests"*. We specify the confidence level = 95% (accordingly, $Z_{\alpha/2} = 1.96$) and E$_{req}$ = 0.01 to estimate the capability of PA system. Once the approach gets 30 response time observations, it computes mean (= 1621.6) and standard deviation (= 172.1), and then computes E (= 0.038). The approach concludes that $E_{req} < E$ and then estimates the capability in terms of mean, standard deviation, and 95% confidence interval of PA(RT). In addition, as we have assumed that we have a precise response time requirement (i.e. PA(RT) $\leq$ 2 seconds with 95%), the approach can initially estimate the fulfilment of this requirement (*ReqFul (%)*) by computing the proportion of response time values that are equal to or less than 2 seconds. These results are depicted in Table 1. These estimates in practice can be updated after getting each new response time observation. $\triangleleft$

Metric (in ms)	Mean	Std. Dev.	Lower 95% limit	Upper 95% limit	ReqFul (%)
Value	1621.6	172.1	1468.9	1782.5	96.7

Table 1: Response time capability estimates of PA system

Phase 2 (P2): Building CUSUM Control Charts

After estimating software capability and getting the predefined QoS requirements, the approach builds the CUSUM control chart model to monitor QoS values and directly detect violations. This building of control chart is achieved by using the methodology of CUSUM control charts that are explained and discussed in Section 3. However, some additional issues need to be addressed in order to build CUSUM charts at runtime.

These issues are as follows. Initially, there are some parameters that need to be specified as discussed in Section 3. These parameters are P_0, P_1 and H. P_0 can be easily computed from the QoS requirement or from the capability estimates. However, software system engineers need to specify the other two parameters P_1 and H. As H and P_1 determine the violation size that needs to be quickly detected and the control chart limits, the performance of CUSUM charts is dominated by the specified values of these parameters. Therefore, software system engineers need to carefully specify these parameters values to design and build a control chart with much better performance to detect the preferred violation size with smaller false alarms. In the evaluation section we investigate in detail how to specify the parameters of CUSUM charts to help engineers to design control charts with better performance and fewer false alarms.

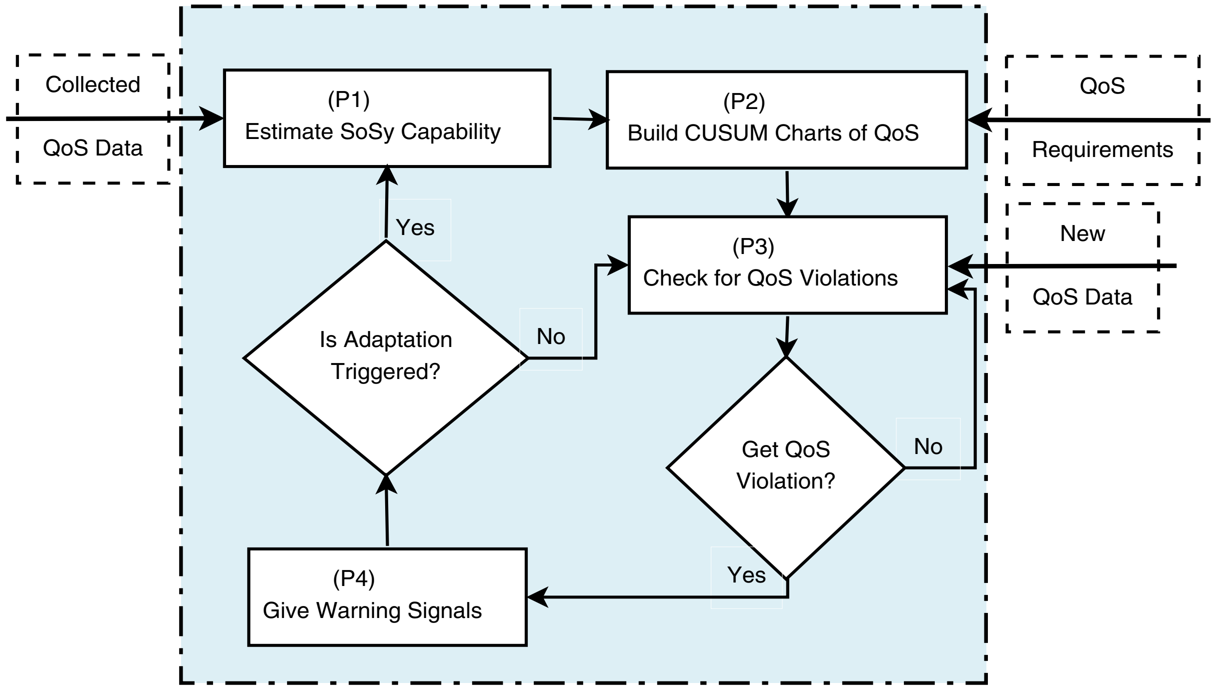

Figure 1: AuDeQAV approach process

Example ▷ To build a CUSUM chart for PA(RT), we first create a dummy variable X_t that takes the value 1 if the response time is greater than 2 seconds and 0 otherwise. Based on the response time requirement, it can be verified that $P_0 = 0.05$. In addition, we specify $P_1 = 0.10$, which means the violation size that we need to quickly detect is 0.05. Using P_0 and P_1 values and equations (4) and (5), K parameter value is computed ($= 0.0724$) and CUSUM statistic is rewritten as:

$$B_t = max[0,\ B_{t-1} + X_t - 0.0724], \quad t = 1, 2, \ldots, \qquad (8)$$

where $B_0 = 0$. In addition, we set the statistical confidence and power of the chart to be 99%, which means that $\alpha = 0.01$ and $\beta = 0.01$. H value is computed using equation (6) as follows:

$$H = \frac{\ln \frac{(1-\beta)}{\alpha}}{2 \ln \frac{P_1(1-P_0)}{P_0(1-P_1)}} = \frac{\ln \frac{(1-0.01)}{0.01}}{2 \ln \frac{0.10(1-0.05)}{0.05(1-0.10)}} \approx 3.0 \qquad (9)$$

◁

Phases 3 and 4 (P3&P4): Detecting QoS Violations and Giving Warning Signals

After building the CUSUM chart model (in P2) and getting each new QoS observation, the CUSUM statistic is recomputed; and once its value exceeds the decision interval H it is considered to be a requirement violation. Consequently a warning signal is sent to the management layer to trigger the necessary corrective adaptation.

Example ▷ For each new value of PA(RT), the approach continuously updates CUSUM statistic and checks for violations. Once it gets one value that exceeds $H = 3$, it gives a warning signal. To illustrate, we compute the sequential proportions of the response time values that are greater than two seconds and the corresponding CUSUM statics and visualize this process in Figure 2. From this Figure, we can see that the approach detects violation of the response time requirement at sample 291, and this result is consistent with the sequentially computed proportion. ◁

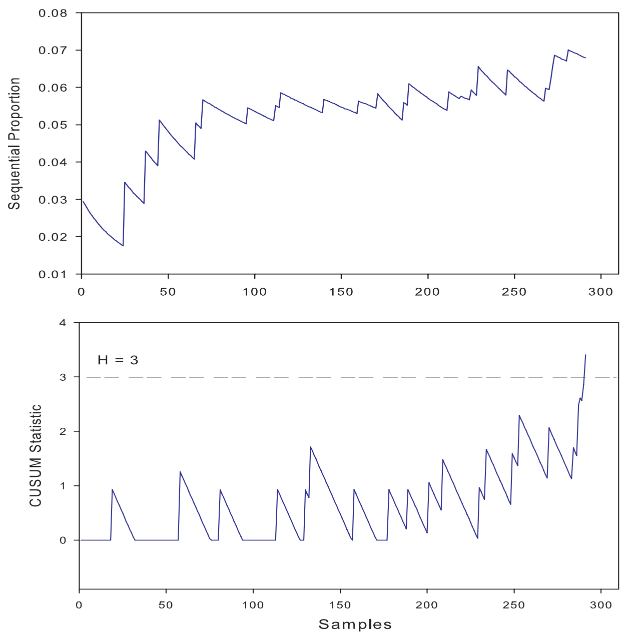

Figure 2: Applying AuDeQAV to monitor response time of PA system

5. EVALUATION

This section investigates various accuracy, performance, and applicability aspects of the proposed approach. First, the accuracy and performance of the approach are evaluated based on a series of experiments. Then, this approach is applied to datasets of response times from real-world web services in order to evaluate its applicability to analyzing and evaluating real situations. Finally, we discuss threats to the validity of our work.

5.1 Evaluating Accuracy and Performance

To investigate the performance and accuracy aspects of the proposed approach, it has been applied to the PA system case study. Based on this case study, a series of experiments have been performed in order to answer two basic research questions.

RQ1: How many samples are required for the approach to detect a violation in the QoS requirements, with respect to different violation sizes? And how many false alarms does the approach give?

RQ2: What is the real advantage of using the proposed approach over a simple threshold-based method to detect QoS violations?

RQ1: The number of samples required to detect a violation in the QoS requirements and the false alarm rate:

As discussed in the previous section, in order to apply the approach to monitor QoS attributes and detect violations in their requirements, the values of P_0, P_1, and H parameters need to be specified, and their specified values determine the values of ARL_0 and ARL_1. These ARL_0 and ARL_1 values refer to the expected false alarm rate and the expected number of samples to detect real occurred violations, respectively. Consequently, in our experiments we have a range of values for these parameters to analyze their effect on the performance of the proposed approach. These parameters and their specified values are discussed as follows:

- Response time requirement: To evaluate the performance of the approach in monitoring different requirements, we have five settings, i.e. that *no response time is more than 2 seconds for 99%, 95%, 90%, 85%, and 80% of the requests*. These requirements are expressed into required proportions of response times that are greater than 2 seconds (P_0 values) as inputs for CUSUM charts, i.e. 0.01, .05, 0.10, 0.15, and 0.20 respectively.

- Required violation size to be quickly detected ($[P_0 - P_1]$ values): In our experiments we specify four violation sizes; 0.050, 0.075, 0.100, and 0.150.

- Decision interval (H): Its value is determined by the required statistical confidence ($1-\alpha$) and power ($1-\beta$). Therefore, we specify different values for H that vary from one to ten with an increment of one to represent different levels of α and β. It is worth noting that an increasing value of H provides increasing values of statistical confidence and power levels.

To explain in detail one scenario of these experiments, suppose we specify $P_0 = 0.05$, $P_1 = 0.10$, and $H = 4$. After that, a violation has been systematically injected with a predefined violation size into the case study's response time. Then the case study has been invoked, the AuDeQAV was run until it signaled a violation condition, and the number of samples (i.e. run lengths) was computed. Simply, the ARL was computed as the average of the run lengths of all the experiments with specific injected violation.

The results of this scenario are depicted (in italic) in the fourth column of data in Table 3(1). These results tell us that if our requirement is that *"the proportion of response times that are greater than 2 seconds is 0.05"* and the required violation size to be quickly detected is 0.05, then the approach (with statistical confidence and power level of about 99.75%) requires about 109 samples to detect a violation of size 0.05, about 70 samples to detect a violation of size 0.08, and about 10 samples to detect a violation of size 0.45. However, it gives a false alarm at about every 1188 samples (i.e. where the violation size = 0).

For more investigation, all the results for the requirement *"the proportion of response times that are greater than 2 seconds is 0.05"* with respect to different violation sizes and decision interval (H) values are depicted in Table 3. From this Table we can conclude the following:

1. Given the other parameters are fixed, an increasing value of the decision interval decreases the false alarm rate; however, at the same time it increases the required number of samples to detect a violation in the requirements. A similar result is concluded for the required violation size to be detected.

2. The larger the size of occurred violation, the smaller the number of samples required to detect this violation with the other parameters are fixed.

3. These results imply that there is a trade-off between decreasing the false alarm rate and decreasing the number of samples required to detect violations in the QoS requirements. In other words, increasing the statistical confidence and power levels will increase the number of samples and make the detection late. As a result, it is very important to choose reasonable values (not very high) for the chart confidence and power to enable the approach to provide better performance. Additionally, this confirms the common understanding based on [20, 34, 15].

We have obtained similar results by applying this approach to the other response time requirements, and consequently we can generalize that these parameters make the approach extensible for various preferences and requirements of software system engineers. They need only to decide a priori the maximum required false alarm rate and violation size required to be detected. Deciding the maximum required false alarm rate is very important as high false alarm rate may lead to perform unnecessary adaptations which cause severe shortcomings such as follow-up failures or increased costs. Currently, our approach uses default values for these two parameters as initial values and they can be changed according to software engineers' preferences. The default false alarm rate is a value that corresponds to statistical confidence and power levels of 99%, and the violation size equals to the required proportion to be monitored, i.e. $P_1 - P_0 = P_0$. However, it is worth mentioning that the optimal design of the proposed approach is the subject of our

(1) Required violation size $(P_1 - P_0)$ to be detected = 0.05										
$\delta P_0^* \ddots H$	1	2	3	4	5	6	7	8	9	10
0.00**	60.87	181.99	473.22	*1187.67*	2736.28	5889.48	12767.20	30567.20	38211.75	47037.67
0.05	23.60	45.86	76.38	*109.20*	142.57	179.11	214.39	246.88	279.47	320.33
0.08	18.22	33.19	51.19	*70.31*	89.97	109.33	129.89	147.43	168.02	187.88
0.10	14.34	24.68	36.85	*49.76*	61.98	75.06	87.74	100.38	113.32	126.06
0.15	10.33	16.73	23.95	*31.72*	39.52	47.23	55.07	62.84	70.58	78.30
0.25	6.72	10.34	14.30	*18.65*	23.18	27.58	32.01	36.40	40.80	45.18
0.35	4.99	7.54	10.20	*13.08*	16.13	19.22	22.39	25.38	28.42	31.40
0.45	3.99	5.99	8.00	*10.10*	12.34	14.69	17.15	19.57	21.92	24.18
0.70	2.66	3.99	5.33	*6.66*	7.99	9.34	10.73	12.18	13.78	15.46
0.95	2.00	3.00	4.00	*5.00*	6.00	7.00	8.00	9.00	10.00	11.00

(2) Required violation size $(P_1 - P_0)$ to be detected = 0.075										
$\delta P_0 \ddots H$	1	2	3	4	5	6	7	8	9	10
0.00	64.94	221.07	786.57	2672.81	9855.55	60011.67	180063.00	180064.00	185324.15	189271.31
0.05	24.63	51.97	91.92	139.79	191.78	243.06	300.44	362.53	411.80	469.63
0.08	18.48	35.16	56.75	80.18	103.33	127.40	150.39	174.55	198.06	222.64
0.10	14.56	25.81	39.91	54.80	69.08	83.95	98.61	113.24	127.67	142.45
0.15	10.47	17.28	25.32	33.83	42.55	51.20	59.57	68.23	76.61	85.01
0.25	6.70	10.40	14.46	18.99	23.59	28.24	32.84	37.51	41.99	46.57
0.35	4.99	7.58	10.31	13.29	16.45	19.72	22.90	26.04	29.18	32.40
0.45	4.01	6.04	8.08	10.24	12.59	15.05	17.57	20.08	22.41	24.76
0.70	2.67	4.01	5.34	6.67	8.02	9.39	10.83	12.38	14.06	15.75
0.95	2.00	3.00	4.00	5.00	6.00	7.00	8.00	9.00	10.00	11.00

(3) Required violation size $(P_1 - P_0)$ to be detected = 0.10										
$\delta P_0 \ddots H$	1	2	3	4	5	6	7	8	9	10
0.00	74.19	296.51	1234.18	4905.31	17396.91	26686.80	27528.20	28421.58	29751.91	30785.23
0.05	26.53	59.49	112.48	179.42	252.63	341.57	440.14	541.17	628.56	735.62
0.08	19.43	39.13	65.23	93.61	121.63	153.00	182.08	213.92	244.88	274.64
0.10	15.29	28.38	44.23	61.59	78.09	95.34	112.80	130.03	146.76	164.85
0.15	10.75	18.37	27.30	36.48	45.60	54.88	64.03	73.36	82.00	91.72
0.25	6.83	10.74	15.26	20.13	24.90	29.77	34.51	39.32	44.09	48.87
0.35	5.04	7.69	10.63	13.87	17.21	20.52	23.78	26.96	30.25	33.45
0.45	4.01	6.05	8.19	10.53	13.05	15.61	18.09	20.52	22.91	25.39
0.70	2.67	4.00	5.34	6.68	8.07	9.58	11.21	12.95	14.59	16.03
0.95	2.00	3.00	4.00	5.00	6.00	7.00	8.00	9.00	10.00	12.00

(4) Required violation size $(P_1 - P_0)$ to be detected = 0.15										
$\delta P_0 \ddots H$	1	2	3	4	5	6	7	8	9	10
0.00	75.48	353.64	1568.22	6698.72	27758.43	145364.00	151218.12	159851.52	165578.56	171582.45
0.05	27.78	70.37	154.52	281.10	465.03	710.50	1037.37	1444.22	1935.81	2585.88
0.08	20.33	44.71	83.49	131.01	184.50	245.46	309.68	365.37	438.64	511.03
0.10	15.92	32.22	53.96	78.27	103.81	129.02	156.26	180.92	207.17	233.84
0.15	10.96	19.45	29.94	41.33	52.44	63.72	74.86	85.98	96.82	107.88
0.25	6.88	11.08	16.03	21.36	26.65	31.96	37.17	42.57	47.87	53.25
0.35	5.04	7.80	10.93	14.34	17.91	21.45	24.90	28.30	31.80	35.24
0.45	3.99	6.05	8.26	10.74	13.38	16.01	18.58	21.12	23.68	26.25
0.70	2.67	4.00	5.34	6.70	8.15	9.76	11.53	13.23	14.72	16.14
0.95	2.00	3.00	4.00	5.00	6.00	7.00	8.00	9.00	11.00	12.00

* refers to occurred violation size

** refers to ARL_0 values

Table 2: ARL performance of AuDeQav approach

on-going research, as we need to develop an optimization-based method that helps in providing the optimal values of the approach parameters.

RQ2: Real advantages of using the proposed approach over a threshold-based method:

To emphasize the real advantages of using our proposed approach to detect QoS violations over a simple threshold-based method, we have applied the same experimental setup (in RQ1), and then we used our approach and the threshold-based method to detect violations.

In particular, the results of applying the threshold based method and our approach (with $P_1 = 0.10$ and $H = 3$, 4, 5, and 6) for the requirement that *"the proportion of response times that are greater than 2 seconds is 0.05"* are depicted in Table 4. Initially, it is clear that the threshold based method does not require any parameters other than the QoS requirement (P_0 value); in contrast, our approach requires two other parameters P_1 and H to be specified. Thus the approach can cater for different preferences and requirements. To compare the performance in terms of the false alarm rate and the required samples to detect a violation, we compute a metric δARL that measures the change in ARL values of our approach comparing to threshold based method. This metric can be computed as:

$$\delta ARL = \frac{(ARL_{AuDeQAV} - ARL_{Threshold})}{ARL_{Threshold}} \tag{10}$$

where $ARL_{AuDeQAV}$ and $ARL_{Threshold}$ are ARL values of our proposed approach and threshold based method, respectively. This metric indicates that how many times the proposed approach duplicates the ARL values of the threshold based method. Based on this metric, if our approach duplicates ARL_0 more than its duplication for ARL_1 values, it implies its performance is better and gives smaller false alarm rate; and otherwise the threshold based method is better.

From Table 4, we can see that the performance of threshold based method is slightly better than our approach's with $H = 3$; however, our approach's performance is better with all other values of H. This result confirms that specifying the value of decision interval is very important in determining the proposed approach performance. Additionally, we can notice that the larger H value, the larger duplication of ARL_0 than ARL_1.

Generally, based on our experiments we can conclude the real advantages of our approach over the threshold based methods as follows:

1. First, our approach can estimate the current status of the running system through the capability estimation phase, and in some cases this can help the software system engineers to refine and update the QoS requirements. In contrast, the threshold based method only monitors the requirements by only comparing observed QoS values to predefined thresholds.

2. Second, in our approach statistical confidence and power levels can be specified a priori to represent the level of certainty in the taken decisions. On the other hand, threshold based method does not provide any confidence or certainty to generalize to the running system over time.

3. Third, as a way to enable a trade-off between the criticality of the QoS requirements and the cost-effectiveness of triggering adaptations, our approach provides the ability for the software system engineers to a priori specify the size of the required violation that is to be monitored and detected. Using a threshold based method does not provide this ability and gives all the violation sizes the same importance to be detected.

5.2 Evaluating Applicability

To further demonstrate the applicability of the proposed approach, we have selected five real-world web services' response time datasets. These response time datasets are collected by Cavallo et al. [5] by invoking the web services every hour for about four months. The description and response time requirements of these web services is reported in Table 5.

AuDeQAV approach can be applied at runtime to estimate these five web services' performance capability and detect violations in their requirements as follows:

1. The AuDeQAV uses the first 30 observations of the datasets to estimate the web services capability. The results are presented in the five left-most side columns of data of Table 6, and we can say initially that the response time requirements are fulfilled.

2. To continuously ensure that the performance of these web services is at the required level, the AuDeQAV uses the response time requirements and runs CUSUM chart (with $H = 3$ to give statistical confidence and power levels of about 99%) to monitor the web services response times and detect violations in their requirements. The results are depicted in the two right-most side columns of Table 6. To illustrate, the sequentially computed proportions of the response time values that are greater than 5.5 seconds for WS_4 and 7.5 seconds for WS_5 and their corresponding CUSUM statistics are depicted in Figures 3(1-4). Where the upper Figures 3(1) and 3(2) illustrate all the computed sequential proportions which provide the whole view of the behavior of WS_4 and WS_5 during about four months. The inner plots in Figures 3(3) and 3(4) visualize the approach behavior until the violation is detected. On the other hand, the outer plots in Figures 3(3) and 3(4) explain how the approach behaves over time to monitor WS_4 and WS_5, and they verify that this behavior is consistent with the corresponding sequential proportions.

3. After detecting changes in the response times of these web services, AuDeQAV gives warning signals to the management layer which in turn may trigger some adaptations, e.g. increasing the allocated resources for the web service application if the service provider is doing the monitoring or selecting a new service if the client is doing the monitoring.

4. If an adaptation is triggered, the web services' capability would need to be recalculated as in Step 1 above while the approach continues monitoring their experienced response time values.

It is worth mentioning that in our above example external web services are being invoked by a client, so little remedial action available to the client other than selecting another service. However, the AuDeQAV approach could also be used by service providers as part of a feedback loop to control provisioning levels for services in response to detected violations wrt their QoS requirements.

5.3 Threats to Validity

The threats to *internal validity* of the conducted experiments include the way we collect and measure QoS values

δP_0^*	Threshold Method	AuDeQAV ($H = 3$)	δARL	AuDeQAV ($H = 4$)	δARL	AuDeQAV ($H = 5$)	δARL	AuDeQAV ($H = 6$)	δARL
0.00**	107.05	473.22	3.42	1187.67	10.09	2736.28	24.56	5889.48	54.02
0.05	10.89	76.38	6.01	109.20	9.03	142.57	12.09	179.11	15.44
0.08	7.56	51.19	5.77	70.31	8.30	89.97	10.90	109.33	13.46
0.10	5.98	36.85	5.16	49.76	7.32	61.98	9.36	75.06	11.54
0.15	4.24	23.95	4.65	31.72	6.49	39.52	8.33	47.23	10.15
0.25	2.59	14.30	4.51	18.65	6.19	23.18	7.94	27.58	9.63
0.35	1.93	10.20	4.28	13.08	5.78	16.13	7.36	19.22	8.96
0.45	1.51	8.00	4.31	10.10	5.71	12.34	7.20	14.69	8.76
0.70	1.08	5.33	3.94	6.66	5.17	7.99	6.41	9.34	7.65
0.95	1.08	4.00	2.71	5.00	3.63	6.00	4.56	7.00	5.49

* refers to occurred violation size
** refers to ARL_0 values

Table 3: ARL performance of AuDeQav approach comparing to threshold based method

WSid	WS Name	Description	URL	rt.Req(s) [%]*
WS_1	XML Daily Fact	Returns a daily fact with an emphasis on XML Web Services and the use of XML within the Microsoft .NET Framework	`http://www.xmlme.com/WSDailyXml.asmx`	2.0 [95%]
WS_2	GetJoke	Outputs a random joke	`http://www.interpressfact.net/webservices/getJoke.asmx`	3.5 [95%]
WS_3	BLiquidity	Provides information on liquidity in a banking system	`http://webservices.lb.lt/BLiquidity/BLiquidity.asmx`	5.0 [90%]
WS_4	Fast Weather	Reports weather info for a given city	`http://ws2.serviceobjects.net/fw/FastWeather.asmx`	5.5 [90%]
WS_5	Currency Converter	Performs a currency conversion using the current quotation	`http://www.webservicex.com/CurrencyConvertor.asmx`	7.5 [90%]

* refers to response time requirement in seconds (s) with probability [%]

Table 4: Characteristics and response time requirements of the real web services

	Capability Estimation (in ms)					Violation Detection	
WSid	Mean	Std. Dev.	Lower 95% limit	Upper 95% limit	ReqFul (%)	Sample Number	Value (in ms)
WS_1	1512.5	169.4	1449.3	1575.8	96.67	305	3948
WS_2	2326.0	616.0	2096.0	2557.0	96.67	50	3798
WS_3	2495.0	2197.0	1674.0	3315.0	90.00	142	5743
WS_4	3861.0	4291.0	2259.0	5464.0	90.00	172	6166
WS_5	4676.0	1112.0	4253.0	5098.0	93.33	295	7715

Table 5: Real web services' capability estimates and their requirements' violation detection

and the range of scenarios that are simulated. To reduce the impact of these threats we have simulated a significant number of scenarios of the Patient Assistance (PA) system with a large variety of violation sizes and measured response time as a performance metric. This is to simulate various scenarios in reality and evaluate the corresponding accuracy and performance of the proposed approach.

On the other hand, *external validity* is threatened if obtained results cannot be generalized. For more realistic scenarios, we have applied the approach for the response time datasets of five real-world web services belonging to different domains. However, further applications to other web services are desirable. Additionally, we focus only on response time as a performance metric, the generalizations to other QoS attributes should be considered in future studies.

6. CONCLUSION AND FUTURE WORK

In this paper we have proposed a statistical approach AuDeQAV for runtime automated detection of violations in the QoS attributes requirements. This approach basically has two main tasks. First, it uses the collected QoS data to estimate the capability of the running software system in terms of descriptive statistics, i.e. mean, standard devia-

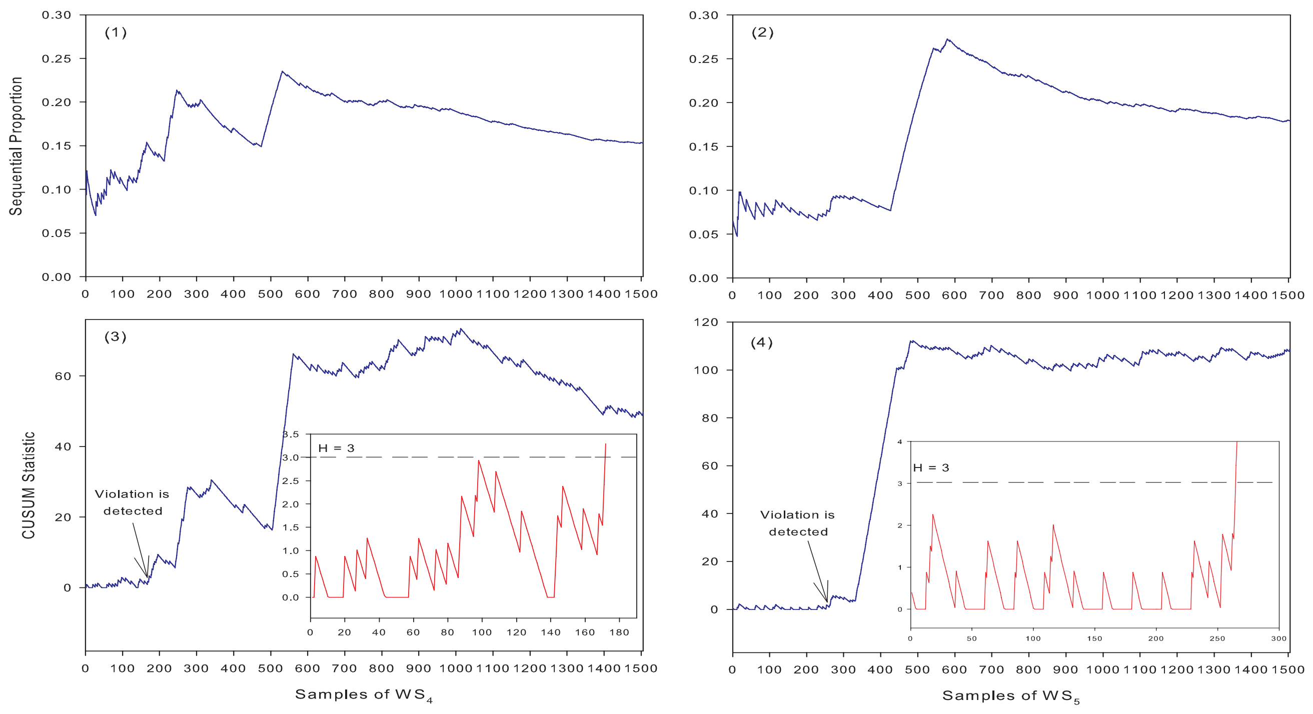

Figure 3: Applying AuDeQAV to monitor response time of WS_4 and WS_5

tion, and confidence interval of QoS attributes, in order to provide the current status of the system. Second, it uses the given QoS requirements to build the CUSUM chart model to monitor QoS attributes and directly detect violations wrt their requirements before undesired consequences occur.

The proposed approach was applied to experimental and real-world QoS datasets, especially response times; and the results demonstrate its accuracy in estimating the running software system capability and in monitoring and detecting QoS violations. Comparing the proposed approach with the threshold based method, we established the main advantages of our approach are: (1) Estimating the running system capability; (2) Providing statistical confidence and power levels which represent the level of certainty in the decisions taken; (3) Specifying the size of violation that is required to be quickly detected. Also, the results illustrate how the proposed approach might be applied to real-world software systems. The AuDeQAV approach therefore can be integrated into the existing software system management frameworks to support and advance dynamic runtime adaptation procedures.

We are planning to generalize this approach to add more QoS attributes, such as reliability and availability. The overhead of the proposed approach also needs to be carefully evaluated, especially for time-critical software systems. Similarly, to reduce the consumed memory a sliding window approach for the observed QoS attributes values needs to be investigated.

7. REFERENCES

[1] A. Amin, A. Colman, and L. Grunske. Using Automated Control Charts for the Runtime Evaluation of QoS Attributes. In *Proceedings of the 13ht IEEE International High Assurance Systems Engineering Symposium*. IEEE Computer Society, 2011.

[2] R. Asadollahi, M. Salehie, and L. Tahvildari. Starmx: A framework for developing self-managing java-based systems. In *ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 58–67. IEEE Computer Society, 2009.

[3] R. Calinescu, L. Grunske, M. Z. Kwiatkowska, R. Mirandola, and G. Tamburrelli. Dynamic qos management and optimization in service-based systems. *IEEE Trans. Software Eng.*, 37(3):387–409, 2011.

[4] R. Calinescu and M. Kwiatkowska. Using quantitative analysis to implement autonomic IT systems. In *Proceedings of the 31st International Conference on Software Engineering*, pages 100–110. IEEE Computer Society, 2009.

[5] B. Cavallo, M. D. Penta, and G. Canfora. An empirical comparison of methods to support QoS-aware service selection. In *Proceedings of the 2nd International Workshop on Principles of Engineering Service-Oriented Systems*, pages 64–70. ACM, 2010.

[6] B. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, et al. Software engineering for self-adaptive systems: A research roadmap. *Software Engineering for Self-Adaptive Systems*, pages 1–26, 2009.

[7] S.-W. Cheng, A.-C. Huang, D. Garlan, B. Schmerl, and P. Steenkiste. An architecture for coordinating

multiple self-management systems. In *Proceedings of the 4th Working IEEE/IFIP Conference on Software Architecture*, pages 243–252. IEEE, 2004.

[8] S.-W. Cheng, A.-C. Huang, D. Garlan, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. In *Proceedings of the First International Conference on Autonomic Computing*, pages 276–277. IEEE Computer Society, 2004.

[9] A. cheng Huang and P. Steenkiste. Building self-adapting services using service-specific knowledge. In *Proceedings of IEEE High Performance Distributed Computing*, pages 34–43. IEEE, 2005.

[10] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54, 2004.

[11] R. D. Gibbons. Use of combined shewhart-cusum control charts for ground water monitoring applications. *Ground water*, 37(5):682–691, 1999.

[12] L. Grunske. An effective sequential statistical test for probabilistic monitoring. *Information and Software Technology*, 53(3):190 – 199, 2011.

[13] L. Grunske and P. Zhang. Monitoring probabilistic properties. In H. van Vliet and V. Issarny, editors, *Proceedings of the 7th joint meeting of the European Software Engineering Conference and the International Symposium on Foundations of Software Engineering*, pages 183–192. ACM, 2009.

[14] B. Halima, M. Guennoun, K. Drira, and M. Jmaiel. Providing Predictive Self-Healing for Web Services: A QoS Monitoring and Analysis-based Approach. *Journal of Information Assurance and Security*, 3(3):175–184, 2008.

[15] D. Hawkins and D. Olwell. *Cumulative sum charts and charting for quality improvement*. Springer Verlag, 1998.

[16] G. Lodi, F. Panzieri, D. Rossi, and E. Turrini. SLA-driven clustering of QoS-aware application servers. *IEEE Transactions on Software Engineering*, pages 186–197, 2007.

[17] A. Luceno and A. Cofino. The random intrinsic fast initial response of two-sided cusum charts. *An Official Journal of the Spanish Society of Statistics and Operations Research*, 15(2):505–524, 2006.

[18] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar. Comprehensive QoS monitoring of Web services and event-based SLA violation detection. In *Proceedings of the 4th International Workshop on Middleware for Service Oriented Computing*, pages 1–6. ACM, 2009.

[19] R. Mirandola and P. Potena. A qos-based framework for the adaptation of service-based systems. *Scalable Computing: Practice and Experience*, 12(1), 2011.

[20] D. C. Montgomery. *Introduction to statistical quality control*. Wiley-India, 2007.

[21] D. C. Montgomery and G. C. Runger. *Applied statistics and probability for engineers*. New York; John Wiley, 2003.

[22] E. Page. Continuous inspection schemes. *Biometrika*, 41(1-2):100–115, 1954.

[23] F. Raimondi, J. Skene, and W. Emmerich. Efficient online monitoring of web-service SLAs. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 170–180, New York, NY, USA, 2008. ACM Press.

[24] M. Reynolds and Z. Stoumbos. A general approach to modeling cusum charts for a proportion. *IIE Transactions*, 32(6):515–535, 2000.

[25] M. Reynolds Jr and Z. Stoumbos. A cusum chart for monitoring a proportion when inspecting continuously. *Journal of Quality Technology*, 31(1):87–108, 1999.

[26] R. Rouvoy, P. Barone, Y. Ding, F. Eliassen, S. Hallsteinsen, J. Lorenzo, A. Mamelli, and U. Scholz. Music: Middleware support for self-adaptation in ubiquitous and service-oriented environments. *Software Engineering for Self-Adaptive Systems*, pages 164–182, 2009.

[27] D. Rud, A. Schmietendorf, and R. Dumke. Resource metrics for service-oriented infrastructures. *Proc. SEMSOA 2007*, pages 90–98, 2007.

[28] M. Salehie and L. Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4(2):1–42, 2009.

[29] F. Schulz. Towards Measuring the Degree of Fulfillment of Service Level Agreements. In *Third International Conference on Information and Computing*, pages 273–276. IEEE, 2010.

[30] W. A. Shewhart. Economic control of quality of manufactured product. *New York: Van Nostrand*, 1931.

[31] J. Siljee, I. Bosloper, J. Nijhuis, and D. Hammer. Dysoa: making service systems self-adaptive. In *ICSOC*, pages 255–268. Springer, 2005.

[32] J. Simmonds, Y. Gan, M. Chechik, S. Nejati, B. O'Farrell, E. Litani, and J. Waterhouse. Runtime monitoring of web service conversations. *IEEE T. Services Computing*, 2(3):223–244, 2009.

[33] J. Skene, A. Skene, J. Crampton, and W. Emmerich. The monitorability of service-level agreements for application-service provision. In V. Cortellessa, S. Uchitel, and D. Yankelevich, editors, *Proceedings of the 6th International Workshop on Software and Performance*, pages 3–14. ACM, 2007.

[34] Z. G. Stoumbos, M. R. Reynolds, T. P. Ryan, and W. H.Woodall. The state of statistical process control as we proceed into the 21st century. *Journal of the American Statistical Association*, 95(451):992–998, 2000.

[35] P. Thongtra and F. A. Aagesen. An autonomic framework for service configuration. In *Proceedings of the Sixth International Multi-Conference on Computing in the Global Information Technology*, pages 116–124, 2011.

[36] P. Zhang, B. Li, and L. Grunske. Timed property sequence chart. *Journal of Systems and Software*, 83(3):371–390, 2010.

[37] P. Zhang, W. Li, D. Wan, and L. Grunske. Monitoring of probabilistic timed property sequence charts. *Softw, Pract. Exper*, 41(7):841–866, 2011.