ADAPTIVE DOMAIN-SPECIFIC SERVICE MONITORING

A Thesis

by

Arda Ahmet Ünsal

Submitted to the
Graduate School of Sciences and Engineering
In Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in the
Department of Computer Science

Özyeğin University
2014

ADAPTIVE DOMAIN-SPECIFIC SERVICE MONITORING

Approved by:

ABSTRACT

We propose an adaptive and domain-specific service monitoring approach to detect partner service errors in a cost-effective manner. Hereby, we not only consider generic errors such as *file not found* or *connection timed out*, but also take domain-specific errors into account. The detection of each type of error entails a different monitoring cost in terms of the consumed resources. To reduce costs, we adapt the monitoring frequency for each service and for each type of error based on the measured error rates and a cost model. We introduce an industrial case study from the broadcasting and content-delivery domain for improving the user-perceived reliability of Smart TV systems. We demonstrate the effectiveness of our approach with real data collected to be relevant for a commercial TV portal application. We present empirical results regarding the trade-off between monitoring overhead and error detection accuracy. Our results show that each service is usually subject to various types of errors with different error rates and exploiting this variation can reduce monitoring costs by up to 30% with negligible compromise on the quality of monitoring.

ÖZETÇE

Bu çalışmamızda harici hizmet sağlayıcılardan kaynaklanan hataları tespit etmek üzere, maliyeti göz önünde bulunduran, uyarlanabilir ve hizmet tipine özel olarak çalışan bir izleme yaklaşımı öneriyoruz. Bu kapsamda, sadece dosya bulunamaması ve bağlantının zaman aşımına uğraması gibi hataları bulmakla kalmayıp, hizmet tipine özel hataları da tespit ediyoruz. Hizmet tipine özel bu hataların herbirinin tespiti için, kullandıkları kaynaklara göre farklı izleme maliyetleri oluşuyor. Bu maliyetleri düşürmek için hata oranlarına ve maliyetlere göre her hizmetin ve hata tipinin izleme sıklıklarını uyarlıyoruz. Akıllı televizyon kullanıcılarının yayın ve içerik alanındaki hizmet güvenilirliklerini arttırmak için endüstriyel bir vaka çalışması sunuyoruz. Yaklaşımımızın verimliliğini göstermek için akıllı TV uygulamaları tarafından erişilen hizmetlerden elde edilen gerçek veriler kullanıyoruz. İzleme maliyetleri ve hata belirleme doğruluğuna ilişkin deneysel sonuçlarımızı paylaşıyoruz. Sonuçlarımız gösteriyor ki, her hizmette farklı hata tiplerine farklı oranlarda rastlanıyor ve bu farklılıkları değerlendirerek izleme maliyetlerini %30 oranına kadar düşürebiliyoruz.

ACKNOWLEDGMENTS

This work is supported by Vestel Electronics and the Turkish Ministry of Science, Industry and Technology (00995.STZ.2011-2). The contents of this thesis reflect the ideas and positions of the authors and do not necessarily reflect the ideas or positions of Vestel Electronics and the Turkish Ministry of Science, Industry and Technology.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Service-oriented architecture (SOA) allows composing loosely-coupled services to build software; a typical SOA may utilize several third-party services. However, relying on external services comes with a price; if a service fails or has degraded quality, an error or an unsatisfactory quality can be observed by the users. To remedy this problem, a monitoring approach [1, 2, 3, 4, 5] can be utilized to tolerate [6] or avoid/mask [7] detected errors and to measure service quality. Existing approaches are mainly dedicated to monitoring basic quality factors such as availability, and they detect only common errors such as *file not found* or *connection timed out*. However, there also exist certain types of errors that are specific and highly relevant to particular application domains. For example, services that provide audio/video content over broadband connection might be subject to a variety of content-related errors such as wrong URLs, faulty feeds (e.g. unsupported formats and codecs), or undesired quality (e.g. low resolution). These problems may result in fatal errors, audio/video freezes, long buffering periods, synchronization errors, and poor customer satisfaction. Detecting each kind of content-reated problem entails a different monitoring cost in terms of the consumed computational resources. For instance, on one hand, a simple ping request is sufficient to check system availability. On the other hand, to detect a codec-related error in a video file, the file should be partially downloaded and the header of the video must be examined. Our work in this thesis is built on top of this observation: *different error types have different monitoring costs.* This variation of cost has not been considered by the service monitoring approaches proposed so far.

We motivate our work based on the architecture and use case of so called "Smart

TVs". Smart TVs enjoy the existence of broadband connection that has become available to TV systems. Various third-party services are used in Smart TVs, including video content providers, popular social media platforms and games. In particular, video-audio content is considered to be among the most important services for Smart TVs [8]. In this work, we investigated a Smart TV portal application developed by Vestek[1], a group company of Vestel, which is one of the largest TV manufacturers in Europe. The portal application is being utilized by Vestel as an online television service platform in Turkey. There are more than 200 third-party services in the portal, providing audio/video content, news, weather and finance information, games, social networking, etc. 70% of these services stream video content. The mostly-used applications are also video streaming applications like Youtube, BBCiPlayer, Netflix, and Turkish national channels. The portal has currently more than 150,000 connected TVs. This number increases by about 7,000 every week.

Smart TV market is very competitive; companies strive to provide richer content and more features to their customers by extremely strict deadlines. This pressure magnifies the importance of customer satisfaction. Because the Smart TV portal relies heavily on third-party providers, availability and quality of external services is vital to Smart TV systems. Vestek executes a test application daily to monitor the third-party services. The test application visits the given URLs, checks their availability, downloads and plays a portion of the audio/video content, and reports the findings so that broken links can be fixed and unsupported content types can be replaced. Some of the content providers frequently change their APIs and migrate/delete their contents without an effective notification mechanism. Therefore, it is common that the test application finds several errors — most typically missing content and video codec errors.

[1]`http://vestek.com.tr`

In this thesis, we propose an adaptive monitoring strategy. Adapting the frequency of monitoring is not a new idea; the novelty in our work is based on the observation that there are certain error types specific to the domain that require separate treatment. Hence the monitoring frequency is adapted based on not only the service availability [9], but also different types of errors relevant for the service. We expect cost-reduction benefits from this adaptation to be significant, because although third-party services usually have high availability rates, they have much lower scores when it comes to domain-specific problems. This is because an unsupported codec or a URL change, for instance, are types of errors that occur at the user-side, not at the provider-side. Hence, providers usually fix these problems only when reported by the users. From the customer's point of view, however, a codec error is just as disturbing as unavailability because what is observed in both cases is the same: a video playback error.

Contributions: In this work we make the following contributions.

- We propose domain-specific adaptation of the monitoring frequency based on the temporal history and the error rate for a particular partner service *and* error type.
- We formulate a cost model to measure the cost of monitoring. Our cost model is based on the price of paid resources consumed by the monitor in the cloud.
- We present an industrial case study from the broadcasting domain, where the utilization of third party Web services become predominant. We provide data set collected by using the Amazon Elastic Compute Cloud (EC2) [10] and Microsoft Azure [11] to monitor dozens of services from different locations for more than one month. We evaluate the effectiveness of adaptive domain-specific monitoring on this real-world data, using the cost model we derived. We also share our data set with the research community to enable further analysis.

Our results show that each service is indeed subject to various types of errors with different error rates. We exploit this variation in the broadcasting domain and show that monitoring costs can be reduced by up to 30% by compromising error detection accuracy negligibly.

Here, we focus on the Smart TV domain and take codec-checking as a domain-specific monitoring action. However, the approach we present is not limited to this domain, nor tied particularly to codec-checks. The adaptation approach we propose is applicable to any domain where various error types are experienced and each error incurs a different cost.

This thesis is organized as follows. In Chapter II, we introduce Vestel Smart TV Portal and explain data collection process and features of the collected data set. In Chapter III, we introduce a set of analytical metrics and related mathematical analysis with an experimental evaluation. An industrial case study for improving the user-perceived reliability of Smart TV systems is given in Chapter IV. In Chapter V, related previous work is summarized. Finally, in Chapter VI we discuss possible extensions of this work for the future and provide the conclusions.

CHAPTER II

EXPERIMENTAL SETUP AND DATA COLLECTION

Two times, for periods of five weeks, we monitored a set of third party services used by Vestel's Smart TV portal to collect real-world data regarding errors. We then applied various monitoring approaches to these data as offline processes. We compared the approaches according to the cost savings they offer, and the compromise they make on the quality of monitoring. In this section we explain the experimental setup we used and provide statistical information.

2.1 Vestel Smart TV Portal

There exist around 80,000 daily connections to the Vestel Smart TV portal from 25,000 different TV systems. These connections are related with various types of services, of which approximately 52% are based on image and video content; 15% are life-style and social networking applications; 9% provide text-based information. Services that are dedicated to sports, music, and games constitute 3%, 3%, and 2% of the whole set of services, respectively. The remaining 16% include miscellaneous services. 75% of all the services are free, whereas the rest of the services are paid.

2.2 Data Collection Process

We identified the 6 mostly used service providers that provide content for free on the Vestel Smart TV portal. Half of these service providers are associated with nation-wide TV channels in Turkey, and they stream video. The other half provide short videos and text-based content.

We developed a data collection application (DCA) to monitor the selected services and create our data set for offline processing. We first ran DCA on three different machines deployed to Amazon's Elastic Compute Cloud (EC2) [10] for five weeks. These instances were located in the USA, Ireland, and Japan. For the first data collection period, we wanted to collect data from geographically far away locations because each DCA has its own view of the network. We wanted to see whether the results from different locations are consistent with each other. Each instance on Amazon EC2 and Microsoft Azure ran DCA individually and independent from the others. They queried each service with a period of about 40 minutes. Each DCA had its own database where the results are stored. Later, we deployed DCA to Microsoft Azure [11] and ran it for another five weeks. This time we have deployed only one instance, which was located in Ireland.

For text-based services, DCA checks the availability over HTTP. If the service returns HTTP 200 (OK), the response time is logged into the database. In case of an error, the error stack trace along with the error code is stored. The video services return a page in JSON or XML format where the video links are included. DCA parses the contents, obtains video URLs, and puts these URLs into the list of URLs to be checked. A video service potentially returns a different list of videos each time it is queried (e.g., the video links returned for the category of "cats" are likely to be updated frequently). Hence, the set of videos monitored in each period may have differences with the previous one.

For each video link, DCA first checks the video's codec type, which is included in the first 1024 Kbytes of the video request response. If no proper codec is found in this header, an error message is logged for the corresponding service. If a proper codec is found, DCA attempts to play the frst three seconds of the video[1] using the Windows Media Player API. If the video player successfully plays the video, DCA logs the successful response in the database along with the video duration, file size, resolution

[1]Even if the file header is fine, the content can be inconsistent with the header. Such cases can be revealed by actually playing the video.

and bit rate information. If any problem is encountered during video replay, the error message raised from the player is logged in the database.

2.3 Collected Data Set

Both the DCA instances on Amazon EC2 and the one deployed to Microsoft Azure ran for five weeks. We observed that the data collected from Amazon EC2 at different geographical locations were consistent with each other. This was confirmed by the cosine similarity measures of error rates between data sets collected from each pair of locations: Japan-Ireland (0.99), Japan-USA (0.98) and Ireland-USA (0.97). But the DCA instance on Microsoft Azure ran on different time interval so error rates are different from Amazon EC2 error rates. Therefore, we used the results from one of the DCA instances on Amazon EC2 and DCA instance on Microsoft Azure. We selected the DCA instance deployed in Ireland for Amazon EC2 since it is the closest geographical location to Turkey. The data we collected are publicly available at `http://srl.ozyegin.edu.tr/projects/fathoms/`.

Amazon EC2 collected data revealed that in total 132,532 requests were made to 51 different services of the selected 6 service providers. Among these requests, 8127 requests were subject to "HTTP 404 not found" error and 9079 requests were subject to a "codec error". Microsoft Azure collected data revealed that in total 113,966 requests were made to 45 different services of the selected 6 service providers. Among these requests, 169 requests were subject to "HTTP 404 not found" error and 11926 requests were subject to a "codec error".

CHAPTER III

ADAPTIVE DOMAIN-SPECIFIC MONITORING

The aim of monitoring a third-party service is to detect when it raises errors and notify the client so that the client may omit using the service or may be directed to an alternative service, and hence avoid the error. A monitor that notifies the clients as soon as a service state change occurs is considered to be high quality. To achieve high quality, monitoring should be done very frequently. However, frequent monitoring puts a high load on the monitoring server. To reduce the associated costs, frequency should be kept as low as possible. This raises a trade-off between the quality and cost of monitoring.

To answer the question of how frequent monitoring should be done, we take a domain-specific, adaptive approach. In Section 2.2 we explained how a video codec error checking is different than checking a text-based service. The associated costs also differ significantly as the former requires downloading a piece of the video and playing it. We adapt the frequency of monitoring by taking into account the history of the occurrence of particular errors for a particular service. If a service has been relatively healthy for a certain error check, following the temporal locality principle, we decrease the corresponding frequency of monitoring in anticipation that the service will continue to be in good status regarding the same error type. When considering the history of a service, we put more value on the recent past than the older history, and make this adjustable via a parameter.

In the following we first present the model we used to calculate the costs incurred by monitoring, followed by the parameters we used for adaptation.

3.1 Cost Model

The goal of our work is to reduce the cost of monitoring. Text-based services consume very little of the network bandwidth, and require almost no computation. Therefore, their cost is negligible when compared to video-based services. Checking a video service consumes resources in two dimensions: (1) part of the video is downloaded, using the network connection, (2) the downloaded video is played, consuming CPU time. Hence, the cost of a video service check, $\mathcal{C}_{video}$, is

$$\mathcal{C}_{video} = (Size \times \mathcal{C}_{net}) + (Duration \times \mathcal{C}_{cpu})$$

where

$Size$ is the size of downloaded piece of the video

$\mathcal{C}_{net}$ is the cost of network usage per unit size

$Duration$ is the duration of the video

$\mathcal{C}_{cpu}$ is the cost of using the CPU per unit time

In our case, the *Duration* parameter is fixed as 3 seconds (recall that we only play the first 3 seconds of the video). The size of a video is on the average 705 Kbytes for 3 seconds of video content, and the file header is 1024 bytes, adding up to 706 Kbytes in total. The parameters $\mathcal{C}_{net}$ and $\mathcal{C}_{cpu}$ depend on the cloud provider and the allocated instances. For instance, $\mathcal{C}_{cpu}$ is currently around \$0.15 per hour, based on the pricing of Amazon [10], Microsoft Azure [11] and Google Cloud [12]. If a service has a charge, it should also be included in the formula; in our case all the services are free, therefore we ignore this issue.

Under these assumptions, the total *cost of monitoring*, denoted as $\mathcal{C}$, is

$$\mathcal{C} = (\text{\# of videos checked}) \times \mathcal{C}_{video}$$

Hence, $\mathcal{C}$ is directly proportional to the number of video checks performed.

Undetected client-side errors affect customer satisfaction and thus indirectly incur costs (e.g., by influencing the customers' perception of the brand). Because measuring this effect is outside the scope of our study, we do not include customer satisfaction in our cost model; instead, we define the *quality of monitoring*, denoted $\mathcal{Q}$, as

$$\mathcal{Q} = \#\text{ of detected errors}$$

because the more number of errors monitoring detects, the better the quality of monitoring is. The quality gets compromised as more errors are left undetected and as such, the error detection accuracy is degraded.

In our evaluation of adaptation, we present the reduction of total cost along with the change in the quality of monitoring.

3.2 Adaptation of Monitoring Frequency

We adapt the frequency of monitoring a service against a particular error type based on the history of occurrence of that error type for that service. To refer to the past, time is divided into enumerated periods (e.g., day 1, day 2, etc.). To keep the discussion straightforward and without loss of generality, we limit ourselves to two types of errors, availability and codec, with the following counts:

V_i is the total number of video checks during the time period i.

$\mathcal{E}_i^{avail}$ is the number of availability errors during the time period i.

$\mathcal{E}_i^{codec}$ is the number of codec errors during the time period i.

Note that an availability check is a prerequisite to a codec check: if a video is unavailable, no codec validation can be made. So, the codec error rate at time period i, denoted as $\hat{\mathcal{E}}_i^{codec}$, is defined as

$$\hat{\mathcal{E}}_i^{codec} = \frac{\mathcal{E}_i^{codec}}{V_i - \mathcal{E}_i^{avail}}$$

Based on these, the *accumulated error rate (AER)* at the end of the time period n, denoted as AER_n, is

$$AER_n = \begin{cases} \hat{\mathcal{E}}_0^{codec} & \text{if } n = 0 \\ \alpha * AER_{n-1} + (1-\alpha) * \hat{\mathcal{E}}_n^{codec} & \text{if } (n > 0) \end{cases}$$

where α is a coefficient $(0 \leq \alpha \leq 1)$ that allows us adjust the weight of the calculated past AER values on calculating the current one. If α is 0, calculation of AER does not depend on the past AER values but is completely determined by the error rate measured in the latest time period. As α gets closer to 1, previously calculated AER values have more influence on the future. Also note that according to this formulation, a relatively older error rate has less influence on the current value than a more recent error rate. This means, the effect of a measured error rate gradually diminishes as time goes by.

At the end of each time period, AER is calculated according to the formula above. Then, the monitoring frequency is adjusted based on this AER. The new frequency is used during the next time period. Frequencies are set using a *frequency pattern.* A frequency pattern is a circular bit-value sequence read from left to right where each bit value denotes whether to skip the corresponding test. For instance, the bit pattern 1110 means that for every four checks, the last codec check shall be skipped, resulting in 25% reduction compared to the original number of codec checks. Availability checks are always performed, regardless of the adopted pattern.

Frequency mappings with regard to AER values are given in Table 1. The table is interpreted as follows. For instance, if frequency scheme F8 is in effect, frequency pattern 1000 is used when AER is less than or equal to 0.1%; pattern 100 is used when AER is larger than 0.1% but less than or equal to 0.2%, and so on. For AER values that are larger than 0.5%, the full frequency pattern is used. Frequency schemes have a varying level of conservatism. On one hand, F0 is very conservative; it uses

Scheme	Accumulated error rate cutoff values (%)					
F0	-	-	-	-	0	100
F1	-	-	-	0	0.001	100
F2	-	-	0	0.001	0.002	100
F3	-	0	0.001	0.002	0.003	100
F4	0	0.001	0.002	0.003	0.004	100
F5	0.001	0.002	0.003	0.004	0.005	100
F6	0.01	0.02	0.03	0.04	0.05	100
F7	0.05	0.1	0.15	0.2	0.25	100
F8	0.1	0.2	0.3	0.4	0.5	100
F9	1	2	3	4	5	100
Frequency patterns	1000	100	10	110	1110	1

Table 1: Adaptation schemes for the monitoring frequency based on accumulated error rates.

frequency pattern 1110 (and hence reduces corresponding frequency by 25%) only for extremely reliable services where $AER = 0$. On the other hand, F9 is the most aggressive/optimistic approach; it reduces the frequency of monitoring for any service that has an AER value of 5% or less. In the following section, we evaluate how these frequency schemes compare in terms of cost savings and quality of monitoring.

CHAPTER IV

EVALUATION

We evaluate the effectiveness of frequency adaptation by simulating an adaptive monitor according to the original data collected during our five-week testing periods (see Section 2.2). Recall that the data contain responses of services to requests sent in periods of approximately 40 minutes. We call a single 40-minute period a *test batch*. Based on the frequency pattern associated with a service, the simulator may skip monitoring the service in a particular test batch. If the pattern requires the service to be monitored, the simulator reads the response from the collected data instead of sending an HTTP request to the service. This way, our simulator behaves like a second monitor that would have been monitoring requests at exactly the same time as the actual monitor. The only difference is that some subset of the test batches for certain services would have been skipped. During the simulation, for each service, we calculate AER values at the end of each day. The current error rate, $\hat{\mathcal{E}}_i^{codec}$, is calculated over the last three days. In the following, we present our results for the two data sets we collected; the one collected by the monitor deployed on Amazon EC2, and the data set that was collected later by another monitor deployed on Microsoft Azure. In the rest of this chapter, we will refer to these data sets as EC2 and MA, respectively.

Figure 1 and Figure 2 show the ratio of skipped codec checks to the number of checks in the original monitor for EC2 and MA, respectively. Recall that the more codec checks we skip, the more we can save on the cost of monitoring; therefore, larger numbers mean more savings. It is not surprising to see that conservative schemes provide less savings (as little as ∼15% skipped checks in F0 for EC2 and

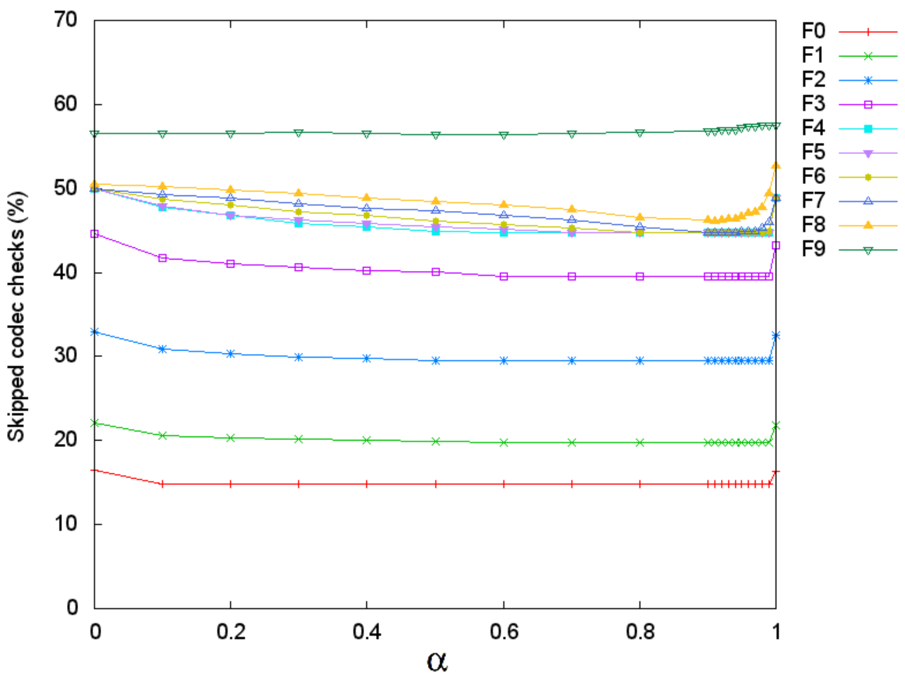

Figure 1: The change in the ratio of skipped codec checks to the number of checks in the original monitor for Amazon EC2. Recall that the number of codec checks is directly proportional to the cost of monitoring; hence, this graph illustrates cost savings.

$\sim$10% skipped checks in F0 for MA), whereas significant savings can be obtained when the scheme is more liberal (57% omitted checks in F9 for EC2 and 60% omitted checks in F9 for MA). Also notice that savings gradually decrease as we increase α in MA, that is, as we decrease the role of current error rate and put more weight in older history when determining the new frequency pattern. On the other hand, cost savings remain the same for most of the F values in EC2. This is due to the relatively stable error rates observed for EC2. For both figures, we can see a sudden increase for $\alpha = 1$. This is because of the fact that the monitoring is performed based on only the first observed error rate when $\alpha = 1$. The monitoring frequency is not adapted according to the error rates that are observed thereafter.

Figure 3 and Figure 4 shows the ratio of undetected codec errors to the number of codec errors in the original monitor for EC2 and MA, respectively. Recall that the fewer errors we miss, the higher the quality of monitoring. Therefore, smaller numbers

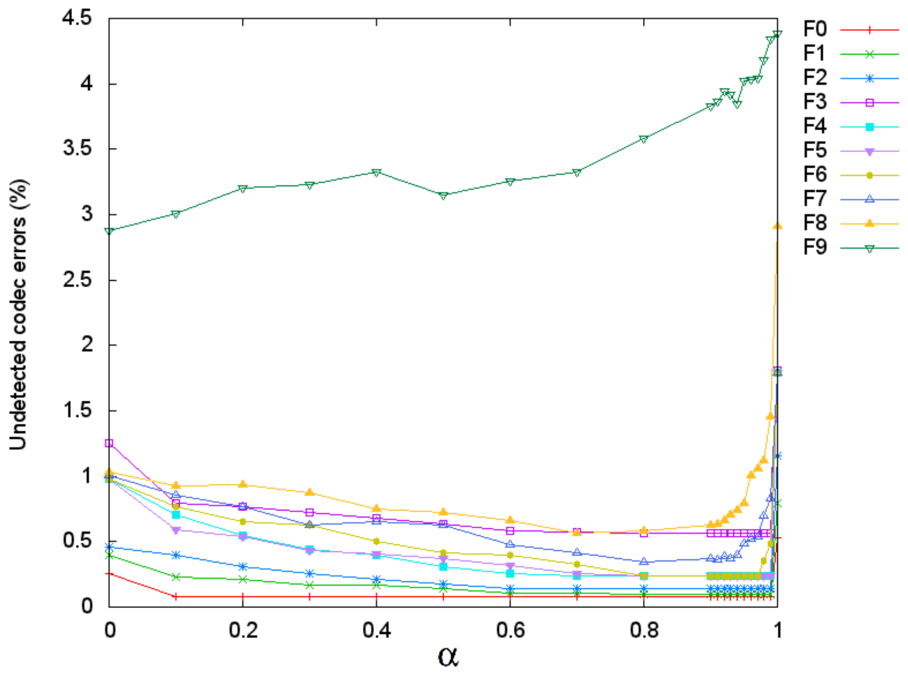

Figure 2: The change in the ratio of skipped codec checks to the number of checks in the original monitor for Microsoft Azure.

mean better quality. Not surprisingly, conservative schemes miss fewer errors; at the extreme, F0 misses no errors for MA when the α value is between 0.1 and 0.99 and misses as little as $\sim$1% for EC2. On the other hand, in our most optimistic scheme F9, up to 4.4% of the codec errors go unnoticed for EC2 and up to 30% of the codec errors go unnoticed for MA.

Finally we consider the combination of cost savings and quality. Ideally, one would like to cut costs as much as possible while keeping the quality high. The two are competing factors; to reduce costs, we need to decrease the frequency, which results in worse quality by failing to detect errors. To be able to find an optimum case, we define the following function to give an *effectiveness score*, denoted as $\mathcal{F}$, to a monitoring configuration.

$$\mathcal{F} = (\text{rate of skipped checks}) - \beta \times (\text{rate of undetected errors}) \qquad (1)$$

In this formulation, the effectiveness depends on how much weight, via the β

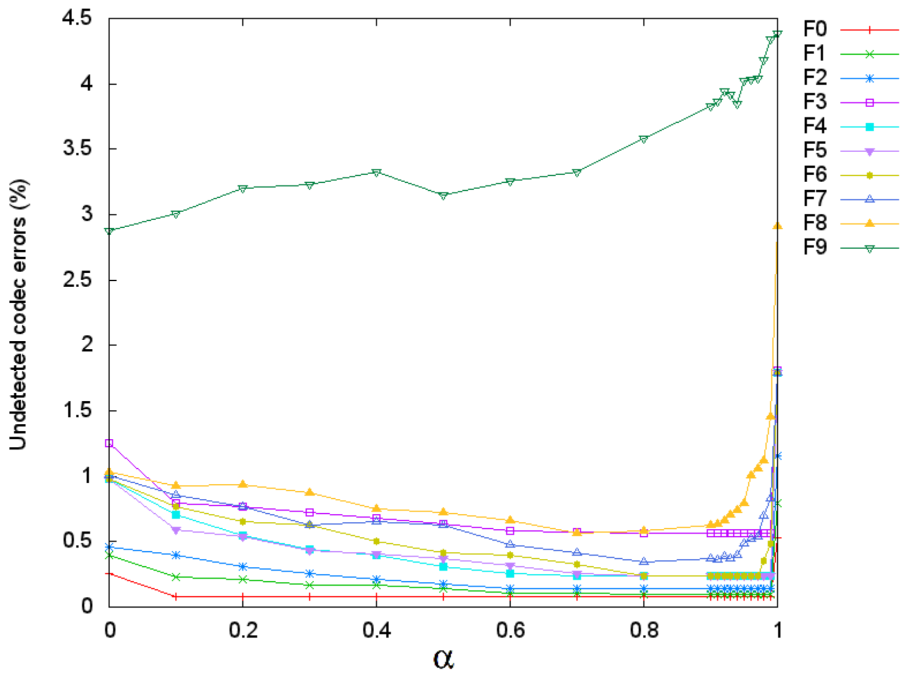

Figure 3: Ratio of undetected codec errors to the number of codec errors in the original monitor for Amazon EC2. Recall that the number of undetected codec errors is inversely proportional to the quality of monitoring.

parameter, is given to the undetected errors as opposed to skipped checks. If the calculated score is negative, we conclude that the corresponding configuration is not feasible because the quality of the monitor has been compromised beyond the acceptable limits by failing to detect errors.

Figures 5, 7, and 9 show the effectiveness score of monitoring for EC2, when β is set to 10, 30, and 50, respectively. Figures 6 and 8 show the effectiveness score of monitoring for MA when β is set to 10, and 30, respectively. We did not plot the effectiveness score of monitoring for MA when β is set to 50. This is due to the high error rates observed for MA, which lead to negative effectiveness scores for high β values. Even when β is set to 30, the effectiveness scores for many schemes turn out to be less than 0 (See Figure 8).

As it can be observed in the presented figures, more liberal schemes lose ranking as the quality of monitoring is given more weight. In Figure 7, for instance, F9 is

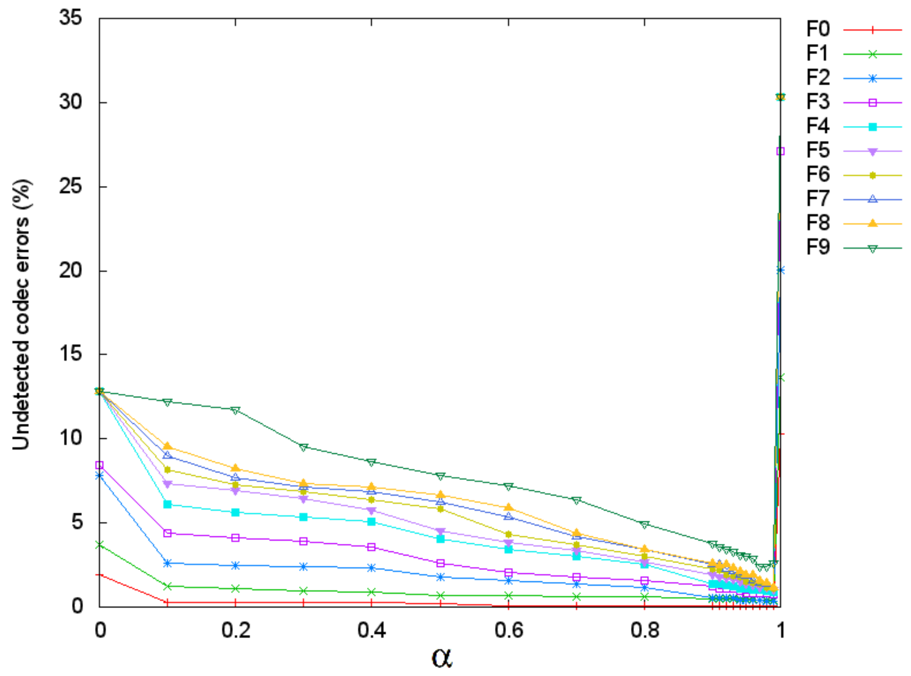

Figure 4: Ratio of undetected codec errors to the number of codec errors in the original monitor for Microsoft Azure.

not even in the window of positive scores, hence it is not an acceptable choice; in Figure 6, part of all F values except F0 are below the 0-line. For all effectiveness score graphs, α is between 0.9 and 1 is the most effective configuration. In this case, when $\beta = 50$ for Amazon EC2, F5 scheme for $\alpha = 0.9$, the cost of monitoring can be reduced by a significant amount of 34% by compromising the error detection accuracy by 0.2%. Even when F1 scheme is adopted for $\alpha = 0.9$, the monitoring cost is reduced by more than 10%, while the ratio of undetected errors is 0.04%. Hence, significant cost savings can be made by compromising the monitoring quality (error detection accuracy) negligibly.

We can measure the amount of cost reduction based on current prices of cloud services and the cost model that was described in the previous chapter. Cost of CPU is currently around \$0.15 per hour, based on the pricing of Amazon [10], Microsoft Azure [11] and Google Cloud [12]. Only Google Cloud has network usage cost; so, we ignore this cost. In Vestel portal, we have approximately 100 video services. Each one

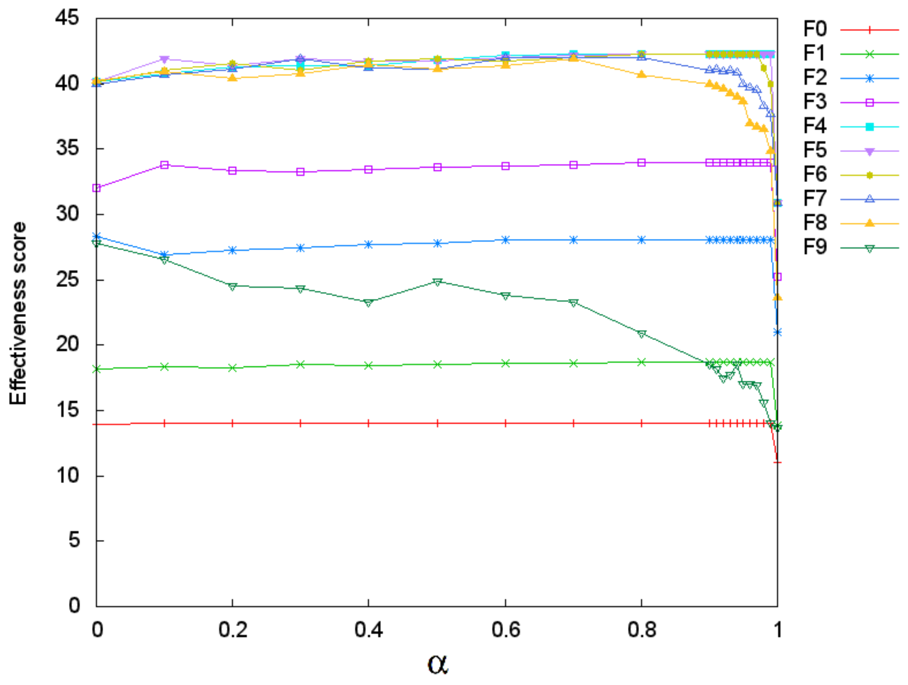

Figure 5: Effectiveness scores, calculated according to Equation 1 for Amazon EC2 when $\beta = 10$.

of them has more than 100 video links. According to our model the cost reduction would be \$1000.8 to \$32.4 every month, and \$12900.6 to \$38800.8 every year.

Figures 10 and 11 show error rate variations of EC2 and MA. Variations in MA are more unsteady with regard to EC2, especially for service numbered 3, 5, 9, and 10. Most of the service error rates observed in EC2 are sinusoidal. This is why 0.9 turns out to be the best choice for α when we consider the effectiveness scores for MA (See Figures 6 and 8). Because if the changes become more unsteady, we have to decrease the importance of last observed error rate to achieve the best effectiveness score. If the changes are steady like Amazon data, there is no significant changes between the α values.

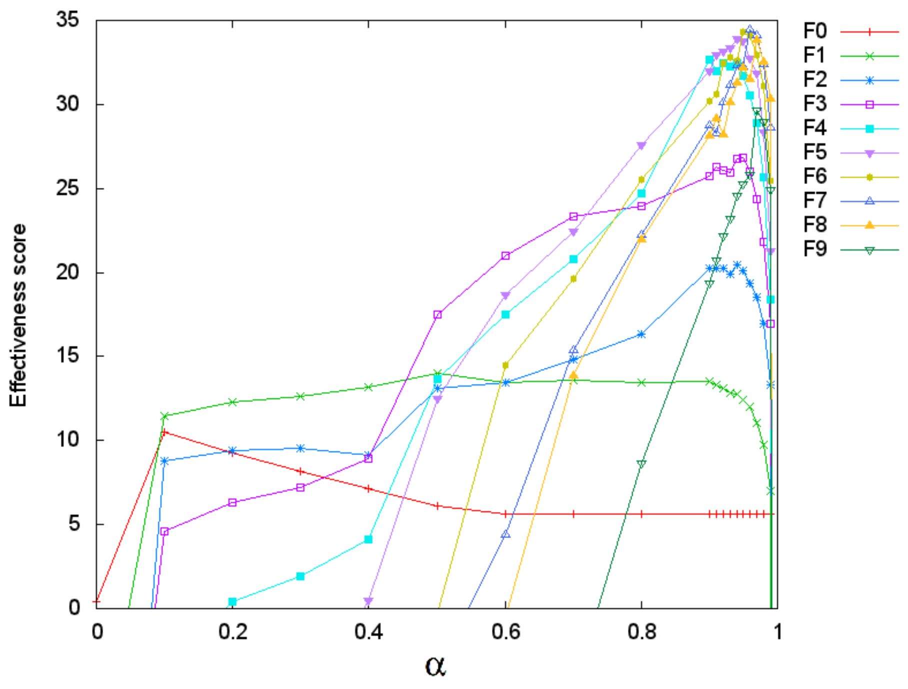

Figure 6: Effectiveness scores, calculated according to Equation 1 for Microsoft Azure when $\beta = 10$.

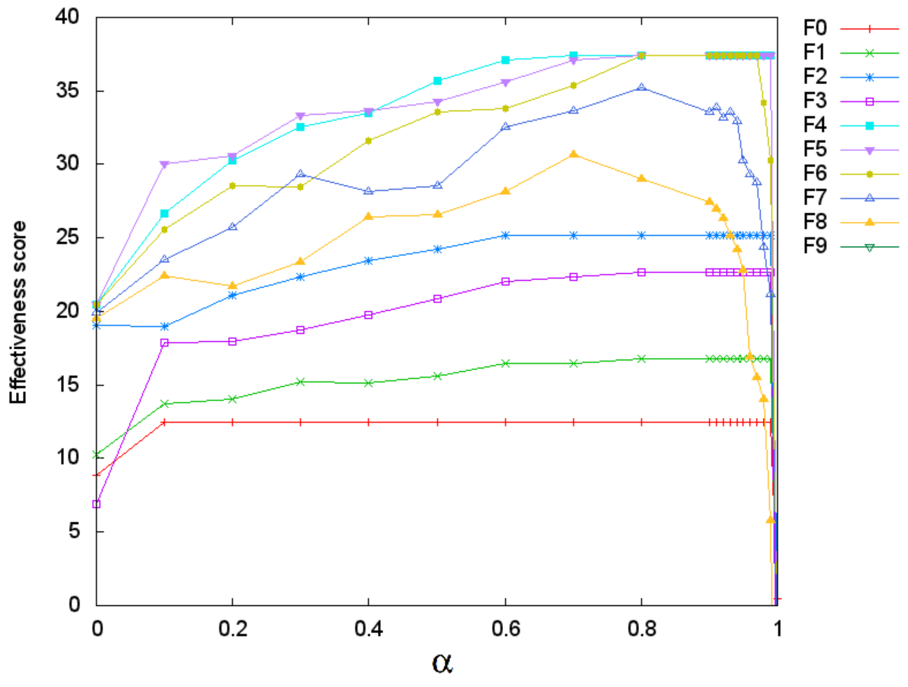

Figure 7: Effectiveness scores, calculated according to Equation 1 for Amazon EC2 when $\beta = 30$.

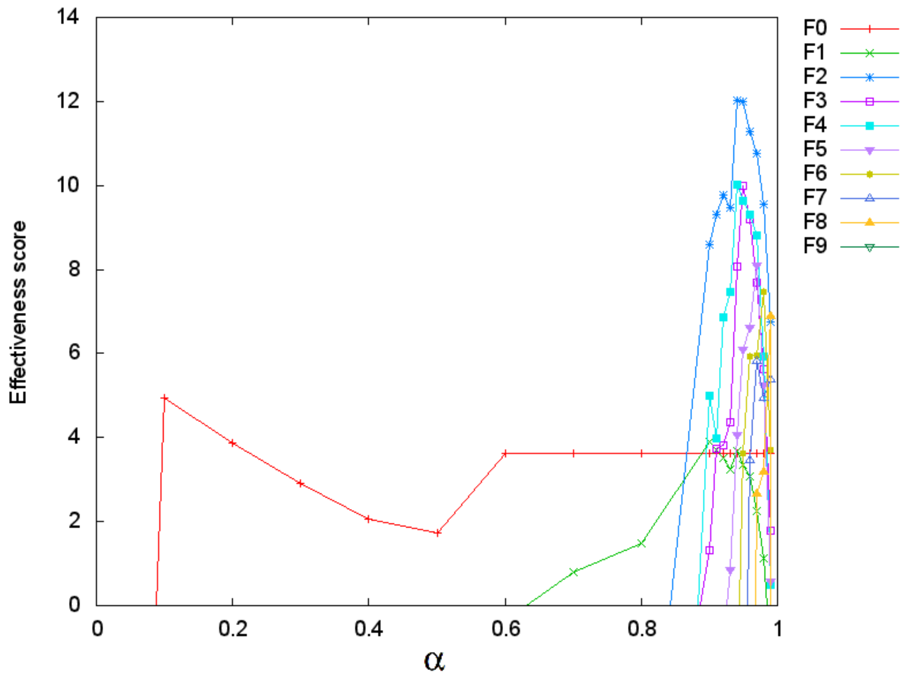

Figure 8: Effectiveness scores, calculated according to Equation 1 for Microsoft Azure when $\beta = 30$.

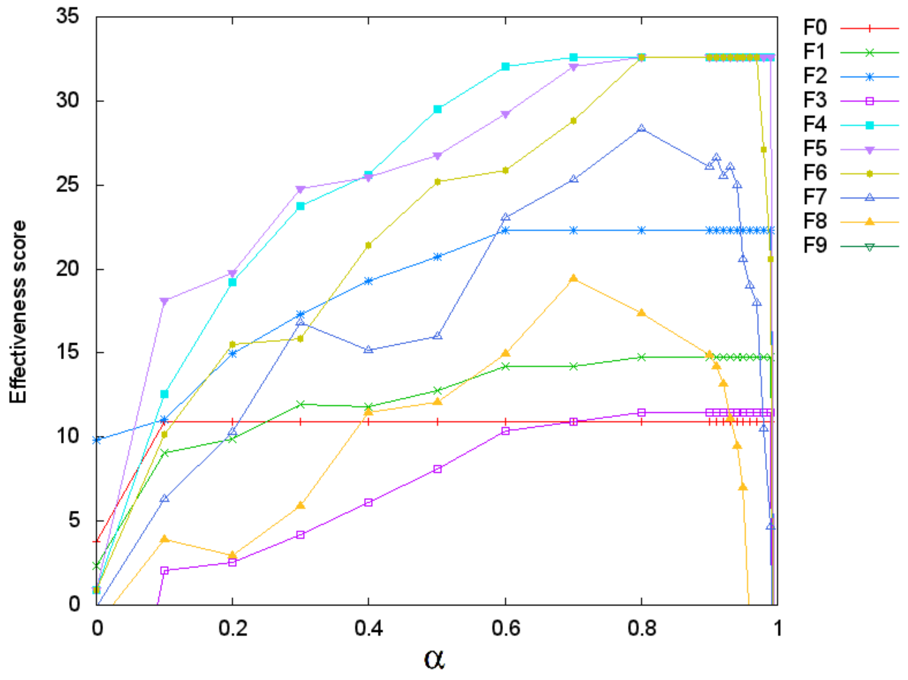

Figure 9: Effectiveness scores, calculated according to Equation 1 for Amazon EC2 when $\beta = 50$.

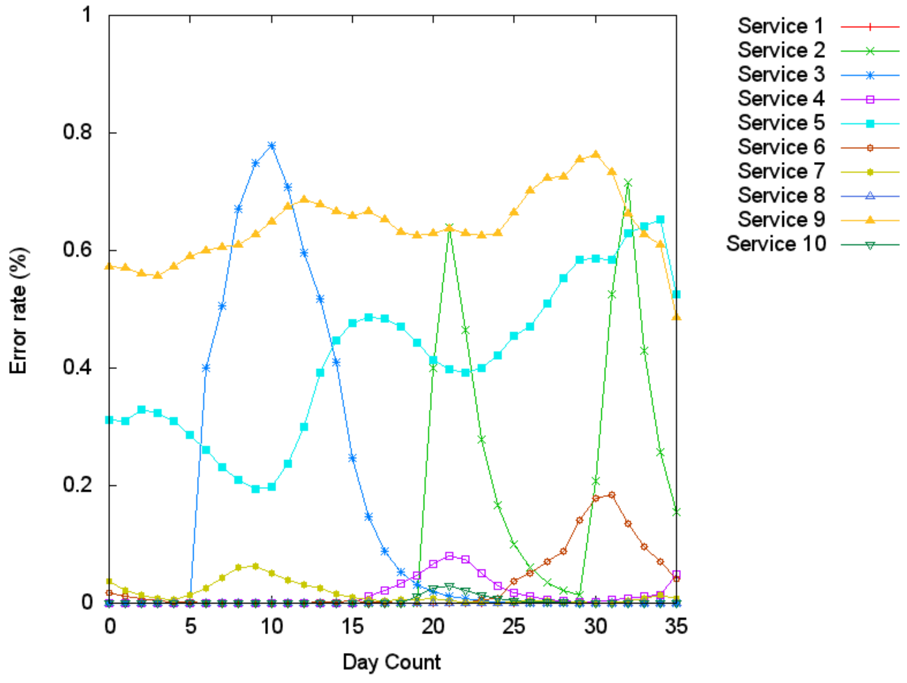

Figure 10: Error rates of services for Amazon EC2.

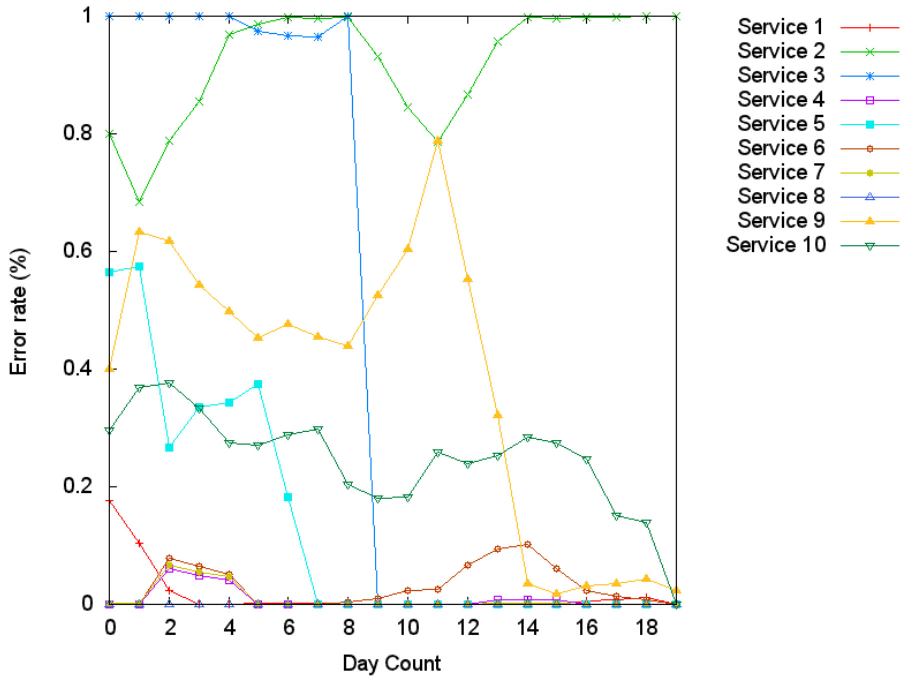

Figure 11: Error rates of services for Microsoft Azure.

CHAPTER V

RELATED WORK

There have been many service monitoring approaches [1, 2, 3, 4, 5] proposed in the literature to tolerate [2] or avoid/mask [9] detected errors in external services. Techniques and tools have been introduced to automatically generate online monitors based on Service Level Agreement (SLA) specifications [13]. These approaches mainly adopt reactive monitoring. Hence, an adaptation can occur only after observing a failure. Online testing of Web services [14] was introduced for facilitating pro-active adaptation. This approach employs functional testing where test cases are generated and executed based on a functional specification [15]. In general, service monitoring approaches proposed so far rely on such standard specifications (or SLAs) and they consider only common quality attributes such as reliability, throughput and latency. However, standard specifications fall short to express domain-specific errors (e.g., codec-related errors while using a video content delivery service) to detect them and to facilitate runtime adaptation with respect to these error types. We have previously studied adaptive service monitoring for cost-effectiveness [9] but the scope of the study was only a single monitor that considers a single quality attribute (availability) regarding services.

There have been also other approaches that adopt adaptive monitoring; however, the majority of these [16, 17, 18, 19, 20] are concerned with the monitoring of hardware resources such as memory, disk and CPU. Other adaptive approaches [21] mainly focus on general properties of web services such as the availability and response time. There are a few studies, where domain-specific cases are considered. For instance, adaptive monitoring was discussed for dynamic data streams [22]. In this domain, each user has a varying interest in each type of information. The approach exploits this fact and adapts the monitoring mechanism for each user. Another approach for monitoring streaming data [23] was proposed for providing adaptivity based on changes in the content of data. Hereby, they propose an algorithm to detect changes in data. The monitoring frequency is adapted based on the detected changes. A similar approach was proposed for adaptive process monitoring [24] as well.

Domain-specific quality attributes have been taken into account in a recent study [25] for service selection. However, the proposed service selection approach considers service monitoring to be out-of-scope and the selection of services is performed based on monitoring results assumed to be available. A toolset and ontology have been previously proposed [26] to express and monitor custom quality attributes regarding Web services. The toolset enables the specification of custom quality metrics but these metrics are defined in terms of only a standard set of service properties and measurements including, for instance, price, delay, throughput, the number of packets lost and availability. The approach does not support the incorporation of custom domain-specific service properties or errors. Similarly, previously proposed customizable service selection policies [27] rely on reactive monitoring of common service properties such as service cost (price), bandwidth and availability.

CHAPTER VI

CONCLUSION

We introduced a novel domain-specific service monitoring approach. We instantiated our approach for detecting errors specific to the services in the broadcasting and content-delivery domain. We developed a cost model for calculating the monitoring overhead in terms of the consumed resources in the cloud. The monitoring frequency for each type of error is dynamically adapted based on this cost model and the measured error rates. We prepared an extensive data set by monitoring services used in a commercial Smart TV from a set of monitors deployed in the cloud. We observed more than 30% reduction in monitoring costs without compromising the error detection accuracy significantly.

Our approach can be applied to other application domains as well. In the future, we plan to develop a plug-in architecture to provide a generic framework that can be extended with custom monitor implementations for different domains. The execution of these monitors will be managed by the framework based on a configurable cost model.

References

[1] J. Simmonds, G. Yuan, M. Chechik, S. Nejati, B. O'Farrell, E. Litani, and J. Waterhouse, "Runtime monitoring of web service conversations," *IEEE Transactions on Services Computing*, vol. 2, no. 3, pp. 223 –244, 2009.

[2] Z. Zheng and M. Lyu, "An adaptive QoS aware fault tolerance strategy for web services," *Journal of Empirical Software Engineering*, vol. 15, no. 4, pp. 323–345, 2010.

[3] W. Robinson and S. Purao, "Monitoring service systems from a language-action perspective," *IEEE Transactions on Services Computing*, vol. 4, no. 1, pp. 17 –30, 2011.

[4] Y. Wei and M. Blake, "An agent-based services framework with adaptive monitoring in cloud environments," in *Proceedings of the 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, (Toulouse, France), pp. 4–9, 2012.

[5] G. Aceto, A. Botta, W. de Donato, and A. Pescap, "Cloud monitoring: A survey," *Computer Networks*, vol. 57, no. 9, pp. 2093 – 2115, 2013.

[6] A. Liu, Q. Li, L. Huang, and M. Xiao, "FACTS: A framework for fault-tolerant composition of transactional web services," *IEEE Transactions on Services Computing*, vol. 3, no. 1, pp. 46–59, 2010.

[7] K. Gulcu, H. Sozer, and B. Aktemur, "FAS: Introducing a service for avoiding faults in composite services," in *Proceedings of the 4th International Workshop on Software Engineering for Resilient Systems*, (Pisa, Italy), pp. 106–120, 2012.

[8] T. Lo, "Trends in the Smart TV industry," 2012. Technical Report. accessed in May 2014.

[9] K. Gulcu, H. Sozer, B. Aktemur, and A. Ercan, "Fault masking as a service," *Software: Practice and Experience*, 2014. In Press. DOI:10.1002/spe.2255.

[10] Amazon.com, "Elastic Compute Cloud (EC2)." accessed in May 2014.

[11] Microsoft, "Windows Azure." accessed in May 2014.

[12] Google, "Google Cloud." accessed in May 2014.

[13] 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*, pp. 170–180, 2008.

[14] A. Metzger, O. Sammodi, K. Pohl, and M. Rzepka, "Towards pro-active adaptation with confidence: augmenting service monitoring with online testing," in *Proceedings of the Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pp. 20– 28, 2010.

[15] X. Bai, W. Dong, W. Tsai, and Y. Chen, "WSDL-based automatic test case generation for web services testing," in *Proceedings of the IEEE International Workshop on Service-Oriented Systems*, pp. 215–220, 2005.

[16] J. Alcaraz Calero and J. Gutierrez Aguado, "Monpaas: An adaptive monitoring platform as a service for cloud computing infrastructures and services," *IEEE Transactions on Services Computing*, 2014. To appear.

[17] K. Clark, M. Warnier, and F. Brazier, "Self-adaptive service monitoring," in *Adaptive and Intelligent Systems* (A. Bouchachia, ed.), vol. 6943 of *Lecture Notes in Computer Science*, pp. 119–130, Springer Berlin Heidelberg, 2011.

[18] M. N. Deepak Jeswani, R. K. Ghosh, "Adaptive monitoring: A hybrid approach for monitoring using probing," in *International Conference on High Performance Computing (HiPC)*, 2010.

[19] D. Jeswani, M. Natu, and R. Ghosh, "Adaptive monitoring: A framework to adapt passive monitoring using probing," in *Proceedings of the 8th international conference and workshop on systems virtualiztion management*, pp. 350–356, 2012.

[20] S. Kwon and J. Choi, "An agent-based adaptive monitoring system," in *Agent Computing and Multi-Agent Systems* (Z.-Z. Shi and R. Sadananda, eds.), vol. 4088 of *Lecture Notes in Computer Science*, pp. 672–677, Springer Berlin Heidelberg, 2006.

[21] K. Clark, M. Warnier, and F. M. T. Brazier, "Self-adaptive service monitoring," in *Proceedings of the Second International Conference on Adaptive and Intelligent Systems*, pp. 119–130, 2011.

[22] B. L. Duc, P. Collet, J. Malenfant, and N. Rivierre, "A QoI-aware Framework for Adaptive Monitoring," in *2nd International Conference on Adaptive and Self-adaptive Systems and Applications*, pp. 133–141, IEEE, 2010.

[23] V. Puttagunta and K. Kalpakis, "Adaptive methods for activity monitoring of streaming data," in *Proceedigns of the 11th International Conference on Machine Learning and Applications*, pp. 197–203, 2002.

[24] W. Li, H. Yue, S. Valle-Cervantes, and S. Qin, "Recursive {PCA} for adaptive process monitoring," *Journal of Process Control*, vol. 10, no. 5, pp. 471 – 486, 2000.

[25] O. Moser, F. Rosenberg, and S. Dustdar, “Domain-specific service selection for composite services,” *IEEE Transactions on Software Engineering*, vol. 38, no. 4, pp. 828–843, 2012.

[26] M. Tian, A. Gramm, H. Ritter, J. Schiller, and M. Reichert, “Efficient selection and monitoring of qos-aware web services with the ws-qos framework,” in *WI '04 Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence*, pp. 152–158, 2004.

[27] V. J. Bart Verheecke, Maria Agustina Cibran, *Aspect-Oriented Programming for Dynamic Web Service Monitoring and Selection.* Springer Berlin Heidelberg, 2004.