Supporting Cloud IaaS Users in Detecting Performance-based Violation for Streaming Applications

Esha Barlaskar, Kiril Dichev, Peter Kilpatrick, Ivor Spence and Dimitrios S. Nikolopoulos
School of Electronics, Electrical Engineering and Computer Science
Queen's University Belfast, UK
Email: {ebarlaskar01, K.Dichev, p.kilpatrick, I.Spence, d.nikolopoulos}@qub.ac.uk

***Abstract*—Cloud infrastructure-as-a-service (IaaS) provides on-demand resources at low cost. IaaS users anticipate stable performance for their applications so that the applications' Quality of Service (QoS) constraints are satisfied. However, they generally experience performance variability in their deployed applications, primarily due to multi-tenancy. Such variability may lead to QoS violations for the applications. Existing cloud infrastructure solutions offered by different cloud providers (CPs) do not have facilities to detect such QoS violations. Hence, cloud users need to take responsibility for monitoring the performance of their applications in order to detect application-specific QoS violations. To support cloud users on this, researchers have proposed a few user-centric cloud monitoring frameworks. However, these frameworks do not detect application-specific QoS violations. In this paper, we propose a novel algorithm for detecting QoS violation for media streaming applications. At each detection instance, the algorithm compares the cumulative value of the expected streamed data against the cumulative value of the measured streamed data. Based on this comparison, the algorithm may raise QoS violation alarms. We evaluate the algorithm by deploying a media streaming application in a lab-based cloud set-up. Experimental results demonstrate that the proposed algorithm can reliably detect QoS violation for streaming applications deployed in the cloud.**

***Index Terms*—Cloud Computing, Cloud Monitoring, QoS Violation Detection, Media Streaming Application**

I. Introduction

A number of cloud infrastructure as a service (IaaS) providers such as Amazon Elastic Compute Cloud (EC2), Google Compute Engine (GCE), Microsoft Azure, IBM Bluemix, Rackspace, etc., offer computing resources on a pay-as-you-go basis at low cost. In addition, they provide elasticity through which cloud users can scale up or down the resources to match their performance requirements. These features have promoted the growth of IaaS. Gartner[1] has reported that IaaS has grown more than 40% in revenue every year since 2011 and they expect a growth of more than 25% every year through 2019. Despite this growth, often there is a conflict between the resource-centric interface exposed by cloud providers (CPs) and what cloud users actually require. Specifically, IaaS users anticipate stable performance for their applications so that the applications' QoS constraints/requirements are satisfied. But, cloud applications generally exhibit performance variability at the post-deployment phase. Work such as [1], [2] experimentally demonstrates the existence of performance variability in cloud applications. Such variability may lead to scenarios where the applications' expected performance is not fully satisfied causing QoS violation. Performance variability in cloud applications may arise for a number of reasons at different levels: cloud infrastructure (scheduled or unscheduled downtime), network (routing problems, loss of intermittent packets to and from the cloud central network, and problems with ISPs), and multi-tenancy (over-provisioning of physical resources). Amongst these, multi-tenancy is assumed to be the most dominant cause of performance variability and subsequent QoS violation for cloud applications [3]. The multi-tenancy problem occurs when CPs try to optimise cloud resource utilisation by allocating maximum number of virtual machines (VMs) on minimum number of physical machines (PMs). This is mainly because cloud resources, such as network and storage, are shared amongst all the VMs hosted in a single PM [4], [5]. Moreover, while undertaking the optimisation tasks, CPs' resource schedulers do not consider the characteristics of the applications running inside the VMs in terms of their performance requirements. The multi-tenancy problem may persist even when cloud users select a specialised dedicated cluster of PMs, as the central network of the cloud is shared amongst many such clusters. Co-located VMs (hosted in the same PM), which run similar applications, may compete for the same shared resources [2]. Examples of applications which may suffer from multi-tenancy are latency-sensitive applications like web services or bandwidth-sensitive applications like media streaming.

Existing cloud infrastructure solutions offered by different CPs such as Amazon EC2, Google Compute Engine (GCE), Microsoft Azure, etc., do not have facilities to detect the applications' QoS violations. Therefore, cloud users need to take responsibility for monitoring the performance of their deployed applications to detect the QoS violations. This motivates the need for user-centric cloud monitoring frameworks. Recently, cloud computing researchers in [6], [7], [8], [9] have proposed such frameworks. However, these frameworks do not detect cloud applications' QoS violations, i.e. application-specific QoS violations.

In this paper, we propose a solution for detecting

[1]http://www.gartner.com/newsroom/id/3354117

application-specific QoS violations through user-centric cloud monitoring. In order to demonstrate the viability of the proposed solution we consider the widely deployed media streaming application as a use case. We select media streaming as our use case due its growing popularity as reported by Global Internet Phenomena Report [10]. A recent report [11] estimates that streaming traffic will occupy up to 82% of the whole internet traffic by 2021. Specifically, we consider the media streaming services of small and medium-sized enterprises (SMEs) which are hosted in public clouds, rather than those hosted in private data centres, e.g. YouTube[2]. We also believe that media streaming service providers such as Netflix[3] who host their services in Amazon, may also benefit from using the proposed QoS violation detection approach. Moreover, media streaming applications are bandwidth-sensitive, and therefore, are expected to experience QoS violation due to the multi-tenancy problem caused by sharing of network resources.

To detect the QoS violations, we propose a novel detection algorithm (see Figure 1). At each detection instance, the algorithm compares the cumulative value of the expected streamed data against the cumulative value of the measured streamed data. Based on this comparison, the algorithm may raise QoS violation alarms. Measured streamed data are the network throughput collected from the VM that is running the streaming application, whereas the expected streamed data is calculated by using a model that considers application-level metrics (number and bit-rate of the active connections of the streaming application). The detection algorithm can be integrated into the user-centric cloud management framework, *MyMinder*. MyMinder was proposed by our previous work in [12] as a complete solution to cloud application management. In particular, on accurately identifying the QoS violations using the detection algorithm, *MyMinder* can support cloud users in taking dynamic decisions on whether to continue with the current CP or to migrate their services to another CP in order to achieve performance stability or to avoid future service failure. However, taking such dynamic decisions and the subsequent inter-cloud migrations are not within the scope of this paper, and interested readers can refer to [12].

To evaluate the performance of the proposed *Detection algorithm*, we performed experiments in a lab-based cloud set-up. The experiments considered the multi-tenancy problem as the cause for the QoS violation. Experimental results demonstrate that the proposed algorithm can reliably detect QoS violation for streaming applications deployed in the cloud.

In summary, our contributions in this paper are two-fold:

1) We propose a novel detection algorithm, which can detect QoS violation for streaming applications in the form of insufficient data streamed to the clients. The algorithm raises a QoS violation alarm when the cumulative value of the measured streamed data is less than the expected streamed data.

[2]https://www.youtube.com/

[3]https://www.netflix.com/gb/

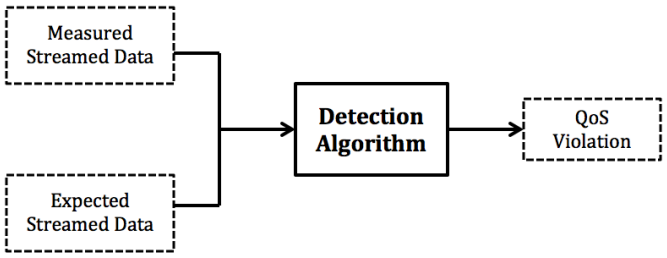

Fig. 1: Detection Algorithm

2) We evaluate the performance of the proposed QoS violation detection algorithm by deploying a media streaming application (representative of real-world media streaming applications running in the cloud), and by designing a benchmark for generating realistic media streaming client requests.

The remainder of the paper is organised as follows. Section II defines the both explicit and implicit QoS violation of user applications. Section III provides a detailed explanation of the proposed QoS violation detection approach. Experimental results are evaluated and discussed in Section IV. Section V discusses the related work in cloud monitoring, QoS violation detection, and management of media streaming applications in the cloud. Section VI concludes the paper and discusses the future work.

II. Explicit and implicit QoS violations of user applications

Explicit QoS Violation: CPs guarantee QoS to their users through SLAs which generally include resource availability in the form of number of processing cores, memory, storage, and in some cases network bandwidth. When CPs fail to maintain the guaranteed QoS (resource availability), explicit QoS violation can be considered. For example, in Amazon EC2 when a running instance has no external network connectivity, a QoS violation is signalled.

Implicit QoS Violation: Network bandwidth availability is not explicitly guaranteed in most of the cloud instances and only a few CPs specify guarantees of bandwidth available for a particular group of VM types, (e.g. Amazon EC2 EBS-optimised[4] VMs provide support for enhanced networking). If a cloud user fails to get the amount of network throughput that is required by the deployed application, then no QoS violation is considered. This is because the user cannot claim a violation for getting low throughput if the CP did not commit to it in the SLA. However, this can be considered as an implicit QoS violation which is specific to the deployed application. This motivates the need for a user-centric monitoring and QoS violation detection approach which can consider the QoS required by user applications, even if it is not part of the SLA. As we consider media streaming application as a use case in our work, we consider the following research questions to propose an application-specific QoS violation detection approach.

[4]https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSOptimized.html

1) How to define QoS violations which are specific to media streaming applications?
2) Which application-level and resource-level metrics to consider to detect such QoS violations?
3) How to integrate the application-level metrics with the resource-level metrics in order to propose a correct QoS violation detection algorithm for streaming applications?

In the following section we discuss how our approach addresses these research questions.

III. A QoS Violation Detection Approach for Streaming Applications

This section introduces the media streaming applications, defines the QoS violations specific to the media streaming applications, and discusses the proposed approach to detect such QoS violations.

A. Media Streaming Applications

Streaming video or audio incorporate the streaming of data from a server to a number of clients who can watch the video or listen to the audio stored in the server. In case of video streaming applications like YouTube, Netflix, etc., the server hosts a variety of videos to serve the clients who request the videos from different media playing devices such as smartphones, computers, tablets, etc. Specifically, the server hosts different types of videos which have different bit-rate (speed of the video transfer from server to the clients) in order to meet the clients' device characteristics (i.e. supported resolution, network bandwidth, etc.). For example, low bit-rate videos for smartphones and high bit-rate videos for computer systems. The streaming of the videos from the server to the clients is performed through various internet protocols which allow the streaming in real-time, e.g. real-time streaming protocol (RTSP). To play the streamed videos without any interruption or quality compromise, these protocols support pre-buffering by streaming the video content at a higher bit-rate.

B. QoS Violations Specific to Streaming Applications

For applications like media streaming, the QoS specific to the applications can be defined in terms of the quality of the videos that are streamed to their viewers or clients (from here on we will use the terms viewers and clients interchangeably). Violation of such QoS can be considered when viewers experience time delays or degraded video quality while watching the streaming videos. Time delays during streaming may occur due to incomplete transcoding of the videos by the streaming server processors or due to insufficient availability of the network bandwidth. Transcoding is the technique of converting the video streams into the format that matches viewers' device characteristics. Degradation of video quality may result from arrival of too many client requests at the streaming server. This is because all the media streaming servers have a saturation point (maximum number of acceptable client requests), beyond which they cannot serve the client requests with the expected quality and in some cases they may even fail to serve few of the requests.

Media streaming service providers who deploy their services in cloud data centres may resolve the time delays due to the transcoding task either by scaling up the number of VMs that are used for performing the transcoding, or by storing several pre-transcoded versions of the videos to serve different types of devices [13]. Also, they can maintain the quality of the streamed videos by limiting the number of client requests within the saturation point of the streaming server. In addition, media streaming service providers can have the guarantee of QoS in terms of the CPU performance from the CPs so that they can maintain the quality of the streamed videos. Hence, there are adequate solutions to avoid QoS violations caused either due to incomplete transcoding, or due to the arrival of too many client requests. However, there is no specific solution to avoid QoS violations occurring due to insufficient availability of the network bandwidth. This is mainly because CPs do not explicitly guarantee QoS in terms of the network performance. Therefore, we aim to detect QoS violations caused by insufficient availability of the network bandwidth so that the media streaming service providers can maintain the QoS of the applications by migrating them to other VMs which show less volatility in its network performance. Specifically, we aim to detect QoS violations in the form of insufficient amount of data streamed to the clients, which arise due to multi-tenancy (sharing of network resources amongst a number of VMs) adopted by the CPs. Such QoS violations may also arise due to factors outside the cloud network (e.g. in the viewers' network), but, in this paper, our assumption is that those factors are insignificant. In particular, we do not consider the potential side effects of severe network issues which may arise outside the cloud infrastructure.

C. Moving Average Based QoS Violation Detection

To propose a QoS violation detection algorithm, we carried out an experiment on the normal behaviour of a media streaming application using the popularly used moving average based approach to detect violation. During the experiment, we set up a media streaming server inside a VM hosted in a physical machine in our lab-based cloud testbed (Section IV details the experimental set-up). Client requests for the streaming server were sent from another machine (outside the cloud testbed). Specifically, during the experimental period, 100 simultaneous client requests were sent for a 10-minute video with bit-rate of 790 Kbps. Figure 2 (top) presents the timeseries graphs for the instantaneous and moving average values of the network throughput collected from the VM that is running the media streaming server. The window size for the moving average is considered to be 60. As is evident from the graphs the fluctuating network behaviour of media streaming causes the instantaneous values of the throughput (represented by the black line) to frequently drop below the expected throughput (ET) values (represented by the blue line) resulting in false positives. ET is calculated accruing to the Formula 1 which considers the application-level metrics (number and bit-rate

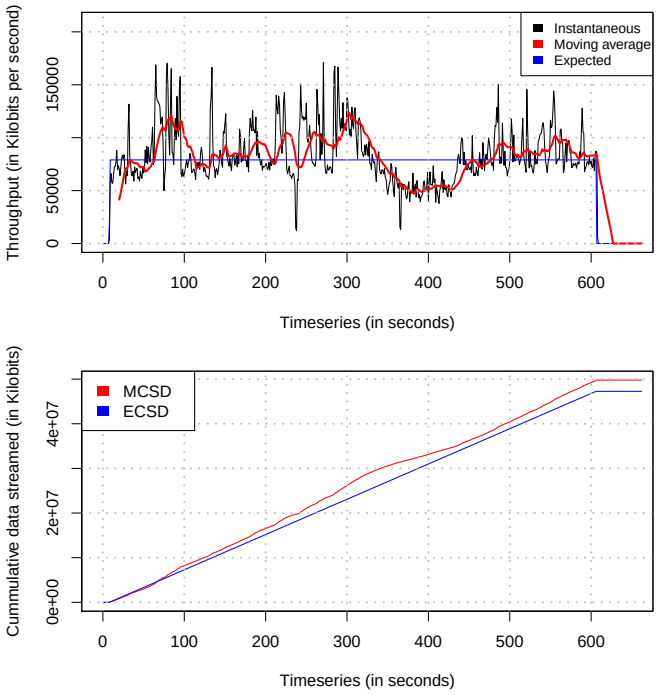

Fig. 2: Network throughput (top) and cumulative data streamed (bottom) by the streaming server

of the active connections of the media streaming application). The formula is based on the fact that, for each type of requested video, the media streaming server is expected to provide the bit-rate that the video type requires for maintaining the quality of the streamed videos. In our experiment we used RTSP as the internet protocol, which supports pre-buffering by streaming the video content at a higher bit-rate. This explains the reason for network throughput being less than the ET at some points of time, in which we assume that the streaming server is sending the video content at a lower bit-rate. In Figure 2 we can observe that considering moving average values (represented by the red line) reduces the false positives to a large extent, but, there still remains a few false positives.

$$ET(t) = \sum_{i \in video\_types} n_i(t) \times bitrate_i \quad (1)$$

where $ET(t)$ = expected throughput at time t
n_i = no. of active connections of video type i
$bitrate_i$ = bit-rate of connections of video type i

D. Cumulative Value Based QoS Violation Detection

Based on the experimental findings from the previous sub-section, we propose a cumulative value based approach to detect the QoS violation which does not raise false positives. The approach uses a novel detection algorithm which firstly calculates the cumulative data that is expected to be streamed (Expected Cumulative Streamed Data - ECSD) at every point of time. Secondly, at every detection point, the algorithm compares the cumulative data that is actually streamed (Measured Cumulative Streamed Data - MCSD) with the ECSD. If MCSD $<$ ECSD, the algorithm raises a QoS violation alarm indicating that the streaming of the videos are delayed or the quality of the streamed videos are compromised. MCSD is the cumulative value of the network throughput (resource-level metric), whereas ECSD is calculated by the ECSD calculation model which uses Formula 3. The formula considers the application-level metrics (duration of active sessions and their bit-rate). In the formula, we consider three cases for the duration of active sessions (see Formula 2): (i) for all active sessions, i.e. if the client i is alive and streaming at the current time t; (ii) for the sessions which are not started, i.e. if the client i has not started streaming at the current time t; and (iii) for the sessions which are completed, i.e. if the client i has finished the streaming.

$$d_i(t) = \begin{cases} 0 & t < s_i \\ t - s_i & s_i \leq t \leq f_i \\ f_i - s_i & f_i < t \end{cases} \quad (2)$$

Where d_i = duration of session i
t = current time
s_i = start time of session i
f_i = end time of session i

$$ECSD(t) = \sum_{i \in all\_sessions} d_i(t) \times bitrate_i \quad (3)$$

Figure 2 (bottom) presents the timeseries graphs of the MCSD and ECSD values which are obtained from the same normal behaviour experiment that is explained in the previous sub-section. From the graphs we can observe that the the MCSD values are always higher than the ECSD values except for a few occasions in the beginning of the timeseries, which may be seen as false positives To reduce such false positives, the algorithm further considers a violation detection window that sets the number of consecutive violations which need to appear before raising the QoS violation issue to the cloud user. Important to note that, at the end of the streaming the MCSD value is higher than the ECSD value. This is due to the protocol overhead which includes headers and metadata.

Overall, the novelty of our approach lies in the consideration of dynamic values of the cumulative streamed data to perform the QoS violation check. This is important because of the dynamic nature of the media streaming requests in terms of the variations in the number of active client sessions and the bit-rate of the streamed videos. The accuracy of detection may be compromised along with high false positives if we consider static QoS violation detection approach, where the threshold for QoS violation is statically decided based on only the resource-level metrics such as the network throughput.

Figure 3 depicts the overall framework of the proposed QoS violation detection. The framework consists of data collection from the VM, where a media streaming server is deployed, and data processing which is performed in a remote detection system. The *Data Collector* module collects the network

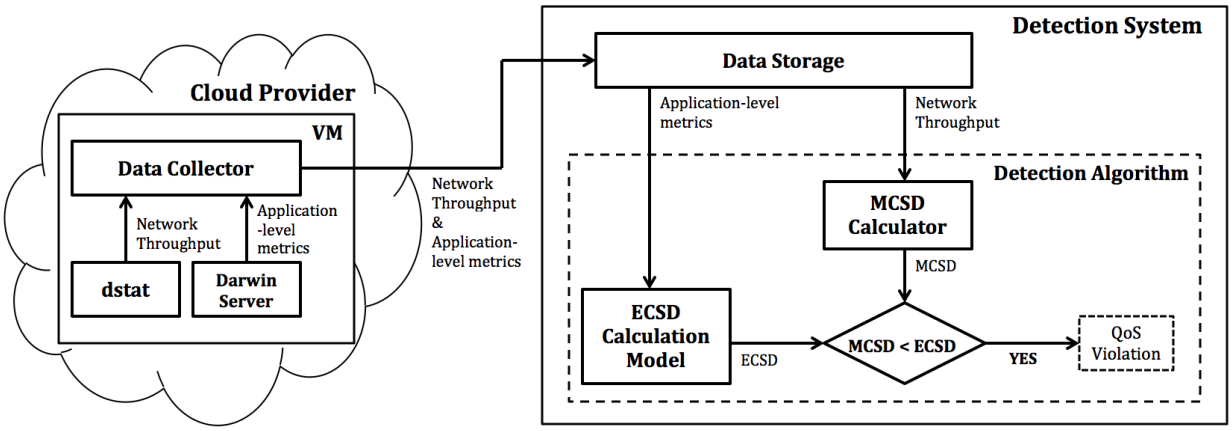

Fig. 3: QoS Violation Detection Framework

throughput generated by "dstat"[5] (a tool for generating resource statistics) inside the VM. The *Data Collector* further collects the application-level metrics (number and bit-rate of the active connections of the media streaming application). The frequency of collecting these metrics is 1 second. The *Data Collector* combines these metrics by their data collection timestamps and saves them in a CSV file. This file is then sent to the detection system via HTTP. The detection system processes the data (saved in the *Data Storage*) sent by the *Data Collector* module in order to detect the QoS violation. Specifically, the detection system uses the *Detection Algorithm* which takes the input data points from the *Data Storage* and identifies the QoS violation by processing these data points. The detection system is integrated with a Spark[6] framework to enable faster processing of the data and to achieve scalability while processing data collected from large number of monitored VMs. The pseudocode in Algorithm 1 presents the steps of the *Detection algorithm*: pseudocode lines 2 and 3 show how the MCSD and ECSD are calculated, respectively; and pseudocode lines 4-12 illustrate the condition for raising the QoS violation alarms. Importantly, the MCSD and ECSD values are always reset (pseudocode lines 11) once the algorithm raises a QoS violation. Although we decided the window size for the violation detection based on our experiments, this may require further tuning for the same or different use cases.

Algorithm 1 QoS Violation Detection Algorithm

input: NT - Network throughput
input: $bitrate$ - Bit-rate of requested videos
input: VWS - Violation detection window size = 5
output: $ViolationAlarm$ - True/False

1: **for each** time t **do**
2: $MCSD(t) \leftarrow \sum_{i \in all\_clients} NT_i$
3: $ECSD(t) \leftarrow \sum_{i \in all\_clients} d_i \times bitrate_i$
4: **if** $MCSD(t) < ECSD(t)$ **then**
5: $ViolationCount \leftarrow ViolationCount + 1$
6: **else**
7: $ViolationCount \leftarrow 0$
8: **end if**
9: **if** ViolationCount = VWS **then**
10: **return** True
11: **end if**
12: **end for**
13: **return** False

IV. Experimental Evaluation

We evaluate the proposed QoS violation detection algorithm by deploying a Darwin media streaming server[7] in a lab-based cloud setup; we then monitor the streaming data it provides across all clients, and employ the violation detection algorithm. We intentionally choose the lab-based cloud setup over public clouds, which allows us to control and evaluate the correctness of the proposed detection mechanism, since the server-client configuration is non-trivial. This setup does not restrict the applicability of the proposed algorithm, which can be translated to applications deployed in public clouds. The metrics required by the algorithm, such as *dstat* and Darwin measurements, do not require privileges beyond the admin privileges on the deployed VM; therefore, they can be used in public clouds. This section presents our evaluation of the proposed QoS violation detection, in particular its reliability and accuracy.

A. Experimental Setup

Testbed: We deploy the Darwin server in our lab-based cloud testbed that is set up on Machine-I via a kvm hypervisor. The cloud testbed hosts three VMs: VM1, VM2, and VM3. The VMs are accessible from outside the testbed via a bridged network that is connected to a 1GbE network switch. Figure 4 depicts the overall setup of the cloud testbed – the server, the client machines, and the monitoring machine. Table I presents the configuration of the machines. Client requests for the Darwin server are sent from three different client

[5]http://dag.wiee.rs/home-made/dstat/
[6]http://spark.apache.org
[7]https://macosforge.github.io/dss/

TABLE I: Experimental Machine Configuration

	Machine-I, Machine-II	Machine-III	Machine-IV	Machine-V
Processor	Single Intel Xeon E5-2650v4 @2.2Ghz (12 cores)	4x Intel(R) Xeon(R) CPU E5-2609 v2 @2.50GHz (4 cores each)	96x Intel(R) Xeon(R) CPU E7-4860 v2 @2.60GHz (12 cores each)	Dual Intel E5-2620v4 @2.1Ghz (8 cores each)
RAM	64GB DDR4	16GB	256GB	32GB DDR4
Disk Capacity	1TB SATA HDD	500GB	2x 500GB	1TB SATA HDD
OS	CentOS 7.4.1708	CentOS 7.2.1511	CentOS 7.1.1503	CentOS 7.2.1511

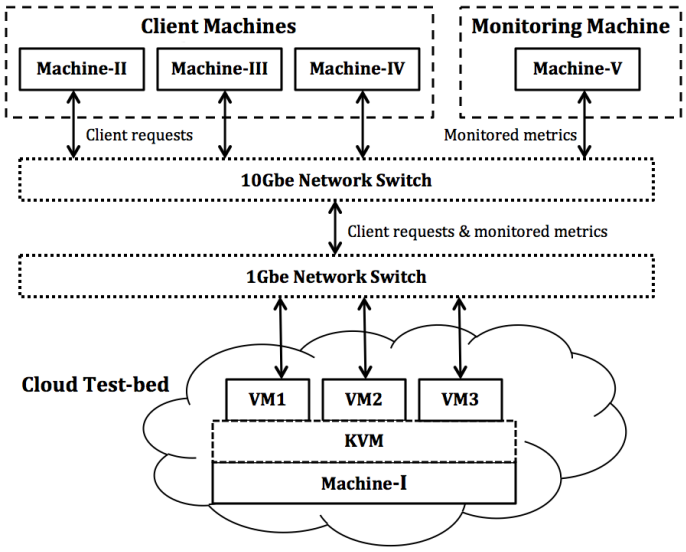

Fig. 4: Experimental Setup

machines: Machine-II, Machine-III, and Machine-IV, which are connected to a 10GbE network switch. All the client machines have sufficient compute, I/O, and network capacity, so performance loss is only propagated through multi-tenancy on the server side. This is part of our assumption that outside the cloud infrastructure, no significant bottlenecks exist. The QoS violation detection system is set up on a remote monitoring machine, Machine V. This monitoring node monitors the application-level and the resource-level metrics from the cloud testbed in order to analyse them and to provide general QoS violation detection capabilities, which are not tied to the physical server or the service it provides. This setup of a violation detection system outside the cloud testbed eliminates performance degradation due to coscheduling a service, such as a streaming service and a monitoring service, on the same physical node. In general, the monitoring and detection node may require intensive CPU resources in its own right, while running more advanced QoS violation detection algorithms.

B. Darwin Server

We use Darwin server release version 5.3.0. We introduce minor modifications in order to enable the proposed ECSD model of this work (see Equation 2).

C. Client Requests for Darwin Server

We generate the client requests for Darwin server using a Poisson distribution[8], so that the client requests closely follow the pattern of the real-world media steaming client requests. For experimenting with different types of streaming requests, clients may request:

- sample videos of duration ranging from 1 to 10 minutes
- for each of these, clients may request a bitrate of low (102kbps), medium (290kbps), or high (790kbps).

The underlying media streaming protocol is RTSP which is taken from the CloudSuite benchmark [9].

D. Detection of QoS Violation

In this section we demonstrate how the proposed violation detection algorithm can detect QoS violations. Specifically, we present the results from two experiments: *Experiment-I* and *Experiment-II*. The experiments evaluate the proposed algorithm on different streaming scenarios: (a) without any co-located service hosted on the physical machine running the streaming service, and (b) with co-located service hosted on the physical machine running the streaming service. In both experiments, the Darwin streaming server is hosted in VM1 of the cloud testbed (Machine-I).

Experiment-I: In this experiment 200 clients request high bitrate videos with 10-minute duration from a client machine (Machine-II), with exactly one new client request arriving each second (within the first 200 seconds). This setup emulates the scenario where Darwin server is streaming "popular" videos. Figure 5 presents the timeseries graphs of the network throughput (collected by *dstat*) and the cumulative data streamed (calculated by MCSD calculator and ECSD calculation model) by the Darwin server while there is no co-located service hosted on the cloud testbed. From the throughput graph we can observe the fluctuating network behaviour of the streaming, which as discussed earlier in Section III, cannot justify a moving average based QoS violation detection approach. However, the cumulative data streamed graph clearly shows that MCSD values are always higher than the ECSD values, i.e. the cumulative values do not fulfil the condition ($MCSD < ECSD$) set by the detection algorithm for QoS violation. Hence, the algorithm is performing correctly without raising any false positives. Figure 6 presents the timeseries graphs of the network throughput and the cumulative data streamed by the Darwin server while a co-located Darwin

[8]https://www.umass.edu/wsp/resources/poisson/
[9]http://cloudsuite.ch

streaming server is hosted in VM2 of the cloud testbed. The co-located server receives 800 streaming requests for high bitrate videos with 10-minute duration from a different client machine (Machine-III). While serving the client requests for two different streaming servers, the available network bandwidth of the cloud testbed gets overwhelmed. Hence, the cloud testbed starts suffering from the multi-tenancy effect. This is evident from the throughput graph (Figure 6) which shows that the Darwin server (running in VM1) is streaming less data compared to the previous scenario (see throughput graph in Figure 5) where there is no co-located service hosted on the cloud testbed. This behaviour indicates that the QoS for the Darwin streaming service (running in VM1) is violated due to the multi-tenancy effect in the cloud testbed. Our algorithm correctly detects this QoS violation as we can observe from the cumulative data streamed graph (Figure 6) that the MCSD values are always lower than the ECSD values, fulfilling the condition ($MCSD < ECSD$) set by the algorithm.

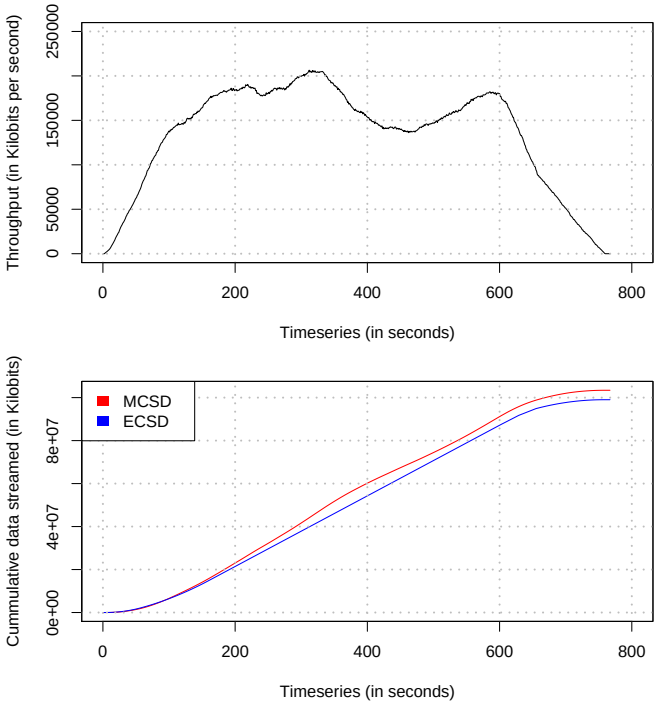

Fig. 5: Streaming Hi10 videos without co-located service

Experiment-II: In this experiment 400 clients request mixed bitrate (high: 50 clients, medium: 150 clients , and low: 200 clients) videos with 3-10 minute duration from all the three client machines (Machine-II, III, IV). This setup emulates the scenario where Darwin server is streaming mixed type of videos. Figure 7 presents the timeseries graph of the cumulative data streamed by the Darwin server while there is no co-located service hosted on the cloud testbed. From the graph we can observe the behaviour similar to *Experiment-I*, i.e. $MCSD < ECSD$ for all the detection points and hence, there is no false positives. However, while performing a series of experiments with different combinations of the bit-

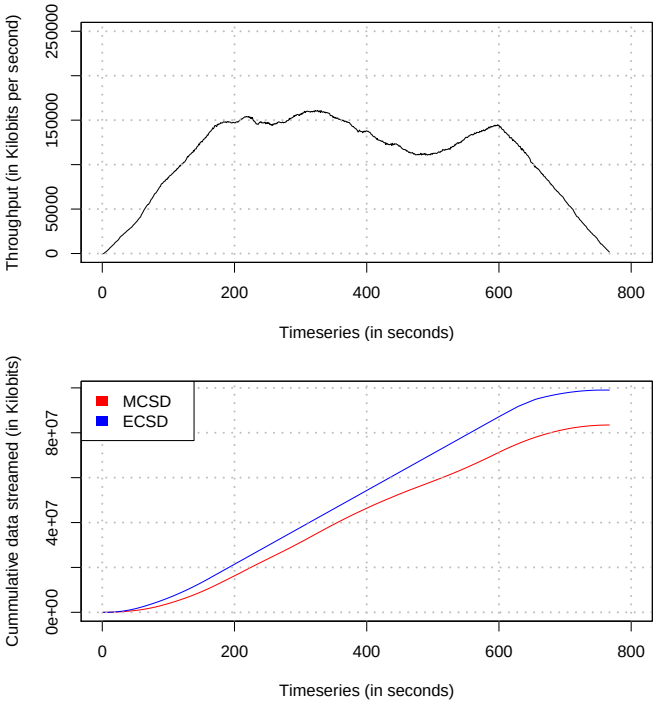

Fig. 6: Streaming Hi10 videos with co-located service

rate and the number of requests we observe that in some occasions, the MCSD values drop slightly below the ECSD values at the end of the streaming. An example can be seen in Figure 8. However, we do not expect false positives due to such behaviour, because the proposed detection algorithm considers a violation detection window, specifically to deal with such situations. Figure 9 presents the timeseries graph of the cumulative data streamed by the Darwin server while two co-located Darwin streaming servers are hosted in VM2 and VM3 of the cloud testbed. The co-located servers receive in total $600(VM1)+600(VM2) = 1200$ streaming requests for high bitrate videos with 10-minute duration from two different client machines (Machine-III and Machine-IV). Similar to *Experiment-I*, our algorithm correctly detects the QoS violation due to the multi-tenancy effect that is arising in this scenario. We can observe this from the graph presented in Figure 9.

E. Observations and Limitations

We found during our experimental evaluation that starting an excessive number of client requests per Darwin server instance does not lead to all requests being served. In particular:

- a configuration file restricts the number of active sessions per Darwin server, so that they do not exceed the saturation point
- when deploying one Darwin server per VM, all server instances across different VMs start limiting the number of active sessions. Usually, once a certain threshold of active sessions is reached, client requests do not get an active data connection.

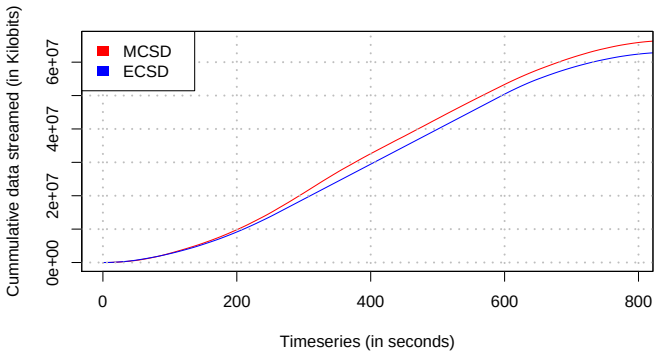

Fig. 7: Streaming mixed videos without co-located service

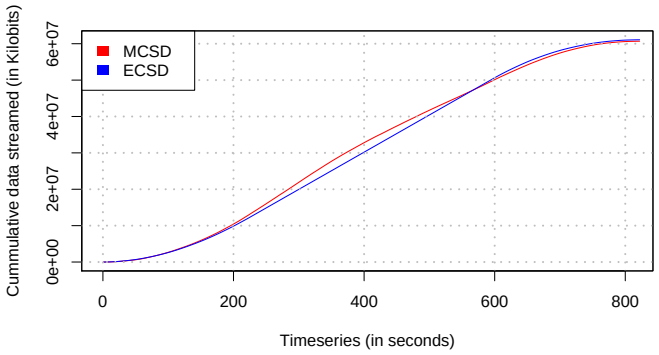

Fig. 8: Streaming mixed videos without co-located service

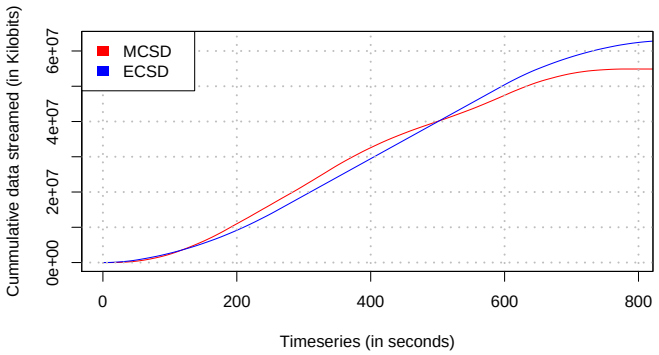

Fig. 9: Streaming mixed videos with co-located service

The implications of these observations are that we are unable to set up experiments with arbitrarily large number of client requests per server instance.

V. Related Work

We discuss various cloud monitoring frameworks and the management of media streaming applications in the following sub-sections.

A. Cloud Monitoring for QoS Violation Detection

Cloud IaaS users may exhibit QoS violations (both explicit and implicit) for a number of reasons which can be traced by monitoring the behaviour of the applications or the underlying cloud infrastructure at runtime. Accurate detection of QoS violations require monitoring some specific performance metrics from the cloud, which in some cases may be restricted to only the CPs. Thus, we can perform QoS violation detection based on user-centric and CP-centric monitoring.

CP-centric monitoring: In case of IaaS, CPs monitor the performance metrics from their cloud infrastructure (e.g. VM or physical machines hosting the VMs) in order to detect performance bottlenecks of the applications hosted in the cloud. The performance bottlenecks or anomalies in the cloud may arise due to the failure of the data centre software/hardware or due to security attacks such as the distributed denial of service (DDoS) attack. Recently, researchers like [14], [15], [16], [17] have implemented statistical and machine learning based techniques to detect performance anomalies in the cloud, specifically in IaaS. Authors in [18], [19] have proposed a monitoring system for the CPs, which can detect violations for all the services (IaaS, PaaS, SaaS) in a cloud data centre. For example, they can monitor the request rates of a web application deployed in the cloud to detect if the request rates exceed the threshold value, i.e. the capacity limit of the VMs provisioned to the user. To detect the violations, they propose a window-based state monitoring approach which is resilient to noises and outliers and performs the monitoring with low communication cost and high scalability.

User-centric monitoring: Public CPs like Amazon EC2, Rackspace, Microsoft Azure, etc. provide monitoring frameworks for their users so that they can monitor and analyse the performance of the deployed applications. Specifically, these frameworks monitor resource utilisation metrics (e.g. CPU utilisation, data transfer, disk usage, etc.) from the VMs that are assigned to the users as well as the metrics and log files related to the deployed applications. Using these monitoring frameworks cloud users can even set alarms based on fixed threshold in order to receive notifications on important performance variation or to take automated actions such as autoscaling. However, these frameworks do not provide in-depth analysis of the cloud application behaviour in terms of detecting QoS violations.

Researchers in [7], [8], [20], [21], [22], [23], [24], [25] have proposed various user-centric monitoring, benchmarking and prediction techniques which can monitor and analyse the performance metrics such as CPU utilisation, network bandwidth, throughput, latency etc. Researchers in [26] propose a cloud management framework named Bazaar which provides a job-centric interface for IaaS users, specifically for supporting deployment of data analytics applications. In Bazaar, tenants can feed their high-level goals for their applications, based on which Bazaar can predict the resources required to achieve those goals. Importantly, Bazaar claims to choose the best combination of resources for the users. The main aim of Bazaar is to bridge the gap between the cloud providers and their users. All these research work can help cloud users in the pre-deployment phase in terms of selecting the most suitable cloud IaaS. However, for detecting QoS violations specific to the deployed applications, cloud users need monitoring in the

post-deployment phase.

The work in [6] propose user-centric cloud monitoring system in the post-deployment phase. Specifically, they use an early warning indicator system to detect possible future violation in the performance metrics or in the performance of the provisioned VM. They also provide support for cloud users in managing the QoS of their deployed services based on the comparison between the observed and the predicted QoS. They only consider VM-level metrics (CPU, memory, and disk) to detect the QoS violations, which may not be sufficient to detect application-specific QoS violations. There are works like [9], [27] which aim to provide monitoring as a service (MaaS) in order to support cloud users in achieving their requirements such as scalability, fault tolerance, self-managing, and application-level monitoring. Their goal is to provide a cloud management system which brings simplicity for managing the deployed applications and they do not focus on monitoring the performance for violation detection. Our work is different from these as we monitor and analyse the application-level metrics as well as the VM-level metrics in order to detect QoS violations which are specific to the deployed applications.

B. Managing Media Streaming Applications in the Cloud

Media streaming applications are often deployed in the cloud in order to avail the benefit of IaaS such as scalability, flexibility, etc. Management of streaming applications (deployed using cloud IaaS) with respect to maintaining the QoS for their viewers has recently received attention from the cloud researchers [28], [29], [30], [31]. Most recently the work in [13] have proposed an on-demand video stream transcoding for streaming applications deployed in the cloud in order to maximise viewers' satisfaction. They consider QoS demand of the viewers in terms of minimising startup delay and meeting presentation deadline. They introduce a cost efficient VM provisioning which can allow dynamic reconfiguring of the cluster of heterogeneous VMs in order to maintain viewers' QoS. Similar to them, we also aim to maintain the QoS for the streaming application viewers, but, in our case we detect the QoS violation occurring due to multi-tenancy, which can be resolved by migrating the streaming application to another VM which has better network performance.

VI. Conclusion and Future Work

Despite the proliferation of cloud computing services, cloud users are not always satisfied with the services that they receive. In general, IaaS users experience performance variability in their deployed applications for a number of reasons; amongst which multi-tenancy is assumed to be the dominant reason. Multi-tenancy results from a phenomenon where CPs try to optimise cloud resource utilisation by allocating maximum number of VMs on minimum number of physical machines. Existing cloud infrastructure solutions offered by different CPs do not have facilities to detect QoS violations for cloud applications. Also, the user-centric cloud monitoring frameworks proposed by various researchers do not detect such application-specific QoS violations. Therefore, we propose a novel algorithm for detecting QoS violation for cloud applications, specifically for media streaming applications. The algorithm considers the QoS violation in the form of insufficient data streamed to the media streaming clients. We evaluate the algorithm by deploying a Darwin media streaming service in a lab-based cloud test-bed. Experimental results demonstrate that the proposed algorithm can reliably detect QoS violation with few or no false positives. Hence, we can adopt the algorithm in the user-centric cloud management framework, *MyMinder*, which is proposed by our previous work in [12]. The algorithm will serve in the re-active detection module in *MyMinder*.

In future, we aim to deliver services for the other modules in *MyMinder*: (i) pro-active QoS violation detection algorithm, so that we can predict future QoS violations to avoid service failure; (ii) decision making algorithm to decide on which CP to migrate the deployed application in case QoS violation is detected; and (iii) prototype for inter-cloud migration that can enable migrating the deployed application to a better performing CP.

References

[1] N. Kratzke and P.-C. Quint, "About automatic benchmarking of iaas cloud service providers for a world of container clusters," *Journal of Cloud Computing Research*, vol. 1, no. 1, pp. 16–34, 2015.

[2] P. Leitner and J. Cito, "Patterns in the chaos:a study of performance variation and predictability in public iaas clouds," *ACM Trans. Internet Technol.*, vol. 16, no. 3, pp. 15:1–15:23, Apr. 2016. [Online]. Available: http://doi.acm.org/10.1145/2885497

[3] T. Lorido-Botran, S. Huerta, L. Toms, J. Tordsson, and B. Sanz, "An unsupervised approach to online noisy-neighbor detection in cloud data centers," *Expert Syst. Appl.*, vol. 89, no. C, pp. 188–204, Dec. 2017. [Online]. Available: https://doi.org/10.1016/j.eswa.2017.07.038

[4] Z. Li, L. O'Brien, H. Zhang, and R. Cai, "On a catalogue of metrics for evaluating commercial cloud services," in *2012 ACM/IEEE 13th International Conference on Grid Computing*, Sept 2012, pp. 164–173.

[5] Z. Li, L. OBrien, and H. Zhang, "Ceem: A practical methodology for cloud services evaluation," in *2013 IEEE Ninth World Congress on Services*, June 2013, pp. 44–51.

[6] Z. ur Rehman, O. K. Hussain, F. K. Hussain, E. Chang, and T. Dillon, "User-side qos forecasting and management of cloud services," *World Wide Web*, vol. 18, no. 6, pp. 1677–1716, Nov 2015. [Online]. Available: https://doi.org/10.1007/s11280-014-0319-8

[7] J. Scheuner, P. Leitner, J. Cito, and H. C. Gall, "Cloud workbench - infrastructure-as-code based cloud benchmarking," *CoRR*, vol. abs/1408.4565, 2014. [Online]. Available: http://arxiv.org/abs/1408.4565

[8] I. Silva-Lepe, R. Subramanian, I. Rouvellou, T. Mikalsen, J. Diament, and A. Iyengar, *SOAlive Service Catalog: A Simplified Approach to Describing, Discovering and Composing Situational Enterprise Services*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 422–437. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-89652-4_32

[9] A. Ciuffoletti, "Application level interface for a cloud monitoring service," *Computer Standards and Interfaces*, vol. 46, pp. 15 – 22, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0920548916000027

[10] G. I. P. Report, "Global internet phenomena report 2016," https://www.sandvine.com/hubfs/downloads/archive/2016-global-internet-phenomena-report-latin-america-and-north-america.pdf, 2016, [Online; accessed 28-February-2018].

[11] C. V. N. Index, "Forecast and methodology, 2016-2021," https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html, 2018, [Online; accessed 2-March-2018].

[12] E. Barlaskar, P. Kilpatrick, I. Spence, and D. S. Nikolopoulos, "Myminder: A user-centric decision making framework for intercloud migration," in *Proceedings of the 7th International Conference on Cloud Computing and Services Science*, 2017, pp. 588–595.

[13] X. Li, M. A. Salehi, M. Bayoumi, N. F. Tzeng, and R. Buyya, "Cost-efficient and robust on-demand video transcoding using heterogeneous cloud services," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 3, pp. 556–571, March 2018.

[14] S. Barbhuiya, Z. Papazachos, P. Kilpatrick, and D. S. Nikolopoulos, "A lightweight tool for anomaly detection in cloud data centres," in *Proceedings of the 5th International Conference on Cloud Computing and Services Science*, 2015, pp. 343–351.

[15] N. Pandeeswari and G. Kumar, "Anomaly detection system in cloud environment using fuzzy clustering based ann," *Mobile Networks and Applications*, vol. 21, no. 3, pp. 494–505, Jun 2016. [Online]. Available: https://doi.org/10.1007/s11036-015-0644-x

[16] A. Gulenko, M. Wallschlger, F. Schmidt, O. Kao, and F. Liu, "Evaluating machine learning algorithms for anomaly detection in clouds," in *2016 IEEE International Conference on Big Data (Big Data)*, Dec 2016, pp. 2716–2721.

[17] T. Salman, D. Bhamare, A. Erbad, R. Jain, and M. Samaka, "Machine learning for anomaly detection and categorization in multi-cloud environments," in *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*, June 2017, pp. 97–103.

[18] S. Meng, T. Wang, and L. Liu, "Monitoring continuous state violation in datacenters: Exploring the time dimension," in *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, March 2010, pp. 968–979.

[19] S. Meng and L. Liu, "Enhanced monitoring-as-a-service for effective cloud management," *IEEE Transactions on Computers*, vol. 62, no. 9, pp. 1705–1720, Sept 2013.

[20] A. Li, X. Yang, S. Kandula, and M. Zhang, "Cloudcmp: Comparing public cloud providers," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10. New York, NY, USA: ACM, 2010, pp. 1–14. [Online]. Available: http://doi.acm.org/10.1145/1879141.1879143

[21] M. Brock and A. Goscinski, "Toward ease of discovery, selection and use of clusters within a cloud," in *2010 IEEE 3rd International Conference on Cloud Computing*, July 2010, pp. 289–296.

[22] S.-M. Han, M. M. Hassan, C.-W. Yoon, and E.-N. Huh, "Efficient service recommendation system for cloud computing market," in *Proceedings of the 2Nd International Conference on Interaction Sciences: Information Technology, Culture and Human*, ser. ICIS '09. New York, NY, USA: ACM, 2009, pp. 839–845. [Online]. Available: http://doi.acm.org/10.1145/1655925.1656078

[23] S. Silas, E. B. Rajsingh, and K. Ezra, "Efficient service selection middleware using electre methodology for cloud environments," *Information Technology Journal*, vol. 11, no. 7, p. 868, 2012.

[24] Z. U. Rehman, O. K. Hussain, and F. K. Hussain, "Parallel cloud service selection and ranking based on qos history," *Int. J. Parallel Program.*, vol. 42, no. 5, pp. 820–852, Oct. 2014. [Online]. Available: http://dx.doi.org/10.1007/s10766-013-0276-3

[25] Z. u. Rehman, O. K. Hussain, and F. K. Hussain, "Multi-criteria iaas service selection based on qos history," in *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*, March 2013, pp. 1129–1135.

[26] V. Jalaparti, H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Bridging the tenant-provider gap in cloud services," in *Proceedings of the Third ACM Symposium on Cloud Computing*, ser. SoCC '12. New York, NY, USA: ACM, 2012, pp. 10:1–10:14. [Online]. Available: http://doi.acm.org/10.1145/2391229.2391239

[27] M. Smit, B. Simmons, and M. Litoiu, "Distributed, application-level monitoring for heterogeneous clouds using stream processing," *Future Generation Computer Systems*, vol. 29, no. 8, pp. 2103 – 2114, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X1300023X

[28] F. Jokhio, A. Ashraf, S. Lafond, and J. Lilius, "A computation and storage trade-off strategy for cost-efficient video transcoding in the cloud," in *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, Sept 2013, pp. 365–372.

[29] F. Jokhio, A. Ashraf, S. Lafond, I. Porres, and J. Lilius, "Prediction-based dynamic resource allocation for video transcoding in cloud computing," in *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, Feb 2013, pp. 254–261.

[30] H. Zhao, Q. Zheng, W. Zhang, B. Du, and Y. Chen, "A version-aware computation and storage trade-off strategy for multi-version vod systems in the cloud," in *2015 IEEE Symposium on Computers and Communication (ISCC)*, July 2015, pp. 943–948.

[31] S. Lin, X. Zhang, Q. Yu, H. Qi, and S. Ma, "Parallelizing video transcoding with load balancing on cloud computing," in *2013 IEEE International Symposium on Circuits and Systems (ISCAS2013)*, May 2013, pp. 2864–2867.