Dynamic Input Anomaly Detection in Interactive Multimedia Services

Mohammed Shatnawi
Microsoft, Redmond, WA, and
Simon Fraser University
Burnaby, BC, Canada

Mohamed Hefeeda
Simon Fraser University
Burnaby, BC, Canada

ABSTRACT

Multimedia services like Skype, WhatsApp, and Google Hangouts have strict Service Level Agreements (SLAs). These services attempt to address the root causes of SLA violations through techniques such as detecting anomalies in the inputs of the services. The key problem with current anomaly detection and handling techniques is that they can't adapt to service changes in real-time. In current techniques, historic data from prior runs of the service are used to identify anomalies in the service inputs like number of concurrent users, and system states like CPU utilization. These techniques do not evaluate the current impact of anomalies on the service. Thus, they may raise alerts and take corrective measures even if the detected anomalies do not cause SLA violations. Alerts are expensive to handle from a system and engineering support perspectives, and should be raised only if necessary. We propose a dynamic approach for handling service input and system state anomalies in multimedia services in real-time, by evaluating the impact of anomalies, independently and associatively, on the service outputs. Our proposed approach alerts and takes corrective measures like capacity allocations if the detected anomalies result in SLA violations. We implement our approach in a large-scale operational multimedia service, and show that it increases anomaly detection accuracy by 31%, reduces anomaly alerting false positives by 71%, false negatives by 69%, and enhances media sharing quality by 14%.

CCS CONCEPTS

• **Information systems → Multimedia information systems**;

KEYWORDS

Multimedia communication services, Multimedia service anomaly detection, Multimedia service reliability

ACM Reference Format:
Mohammed Shatnawi and Mohamed Hefeeda. 2018. Dynamic Input Anomaly Detection in Interactive Multimedia Services. In *MMSys'18: 9th ACM Multimedia Systems Conference, June 12–15, 2018, Amsterdam, Netherlands.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3204949.3204954

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
MMSys'18, June 12–15, 2018, Amsterdam, Netherlands
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5192-8/18/06…$15.00
https://doi.org/10.1145/3204949.3204954

1 INTRODUCTION

Interactive multimedia services such as Skype and Google Hangouts have strict Service Level Agreements (SLAs) governing various aspects of their service like multimedia quality and response time. For example, the quality of multimedia sessions, measured by Mean Opinion Score (MOS), may have an SLA of MOS 4 or 5, good/excellent quality. Multimedia sessions with MOS 1 through 3, poor quality, are violating the SLA with customers. The cost of such violations is high including loss of customers to competitors [20], so these violations should be reduced as much as possible.

In a typical use case of a multimedia service, a user calls another using a client application. The application initiates a session and invokes online services in the cloud. The services are deployed in data centers around the world. Each service may in turn call other sub-services or components for encoding, rendering, dejitter, and ad serving. A user perceived quality of the multimedia session is directly impacted by the performance of these components. If we understand the anomalies in the service input, working conditions, and system states of these components, and *their impact on the service outputs*, we can manage the anomalies, improve the reliability and performance of multimedia services, and reduce SLA violations. Working conditions refer to the activities running on the service like number of active processes and number of open sockets. System states refer to the service internal states like CPU and memory utilization. An anomaly in this context refers to values of the service inputs, component working conditions, and system states that deviate from the expected values [1]. For example, a display ad in a video call is expected to last a few seconds before it is registered as a billable impression. If ad impressions are swapping at rates of more than one ad per second, for example, then that's an anomaly that indicates ad fraud, and warrants alerting and taking corrective measures like blocking the ad source. On the other hand, customers generally share a single video in a multimedia session. Sharing two videos in a session is an anomaly, but can potentially happen if the customers want to compare two videos side by side. So this type of anomalies may not warrant an alert, unless it is not supported and cause failures.

State of the art anomaly detectors follow a data-driven and mathematical models to define an anomaly [1, 5]. They analyze large amounts of historic data that represent the service inputs and its working conditions, such as number of users and their active video sessions, and the number of active processes and their CPU utilization. Using mathematical modeling, current anomaly detectors find the expected values for these properties as well as the outliers or anomalies [1, 12]. In the current approaches for handling anomalies, service implementers generally raise alerts and request corrective measures when anomalies are encountered, without considering the actual impact of anomalies on the service [19, 20]. Alerts are expensive from a system and engineering support perspectives, and should be raised only if necessary [24].

There are multiple key problems with this common theme in anomaly detection and handling approaches, especially for interactive multimedia services. First, the use of historic data to determine the impact of anomalies on the service, raise alerts, and request corrective measures. In the case of multimedia services, the data comes from logs of prior runs of the service [11]. That data may no longer reflect the current conditions of the service accurately, as new services are continuously added to data centers, and storage and compute provisions are updated through collaboration with other data centers. For example, if a multimedia service expects no more than 100 participants in an online meeting, then having 110 participants is considered an anomaly that raises alerts, even if the service has enough current capacity to handle the extra participants; so why raise alerts and take the cost hit if that is not needed.

Another problem with state of the art anomaly detection and handling techniques is that they are generally designed to monitor and alert on specific metrics of the system independently; for example, number of processes or CPU utilization [20]. However, it is rarely the case that a single metric can be correlated to multimedia SLA violations. Usually, the output of the multimedia service, like multimedia quality, reduces below an acceptable value under a few conditions together, like number of customers, number of concurrent video sessions, and CPU utilization. So it is important to consider the *association of metrics* that cause SLA violations.

Lastly, the cost of pre-processing previous logs to prepare them for anomaly detection and impact analysis is substantial. The majority of multimedia transactions in production service logs have the expected service inputs and working conditions, and succeed without causing SLA violations [4, 22, 25]. Thus, the SLA violation rate is low, and so is the recall in the data. Recall in this context refers to the percentage of data that is relevant and usable in the analysis of anomalies. Having relevant and current data with high recall is more important to the success of anomaly detection than advanced and deep algorithms [5]. Because of that, hundreds of gigabytes of log data over months are needed to find enough relevant data for anomaly detection and analysis for multimedia services [22]. Using historic data in generating analytical systems like data mining and predictive modeling works well in environments that do not change often; such as transportation systems like airplanes and ships. On the other hand, multimedia services lack such stability over time at many levels including service hardware provisions. The ever-changing landscape of multimedia services, coupled with requirements such as continuous up-times, make the use of historic data about the service challenging and ineffective [20].

We propose a dynamic approach to the analysis of multimedia service anomalies. We use synthetic transactions, explained in Section 3, to generate fresh and small, yet highly relevant data about the current state of the service in near real-time. We employ machine learning techniques to correlate the ranges of anomalous service inputs and its working conditions with the service SLA violations. The anomalies themselves are found using current state of the art techniques. Our proposed approach *identifies the impact of these anomalies, independently and associatively*, on the service and its ability to adhere to SLAs, and recommends whether to raise alerts and invoke corrective measures. We consider service inputs and working conditions anomalies worth alerting on *if and only if they impact the output of the service negatively and result in SLA violations.*

We implemented the proposed approach in one of Microsoft's Skype data services in the application and services group, which handles millions of multimedia sessions per second. We show that the proposed approach is able to reduce the number of false positives in anomaly alerts by about 71%, reduce false negatives by about 69%, enhance the accuracy of anomaly detection by about 31%, and enhance the media sharing quality by about 14%. The recall in the data generated by the synthetic transactions is 100%. In contrast, the recall in the production logs is less than 2%. In addition, we show that we can update the anomaly detector in near real-time in about 7 minutes. On the other hand, building a detector model using current anomaly detection techniques for the same service by using production logs requires about 7 weeks of production log data that takes several hours of pre-processing before the data is usable.

The contributions of this work are (1) new approach for dynamic anomaly detection in multimedia services in real-time, (2) machine learning method to correlate the multimedia service inputs, working conditions, and system states with its outputs and their SLAs, and (3) actual implementation and evaluation of the proposed approach in a real multimedia communication service.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 presents the proposed approach. Section 4 describes our implementation and evaluation. Section 5 concludes the paper.

2 RELATED WORK

Multimedia services are subject to conditions that impact their SLAs like data center faults and anomalies in the service inputs and working conditions. The complexity and number of components the service depends on exacerbate the impact of anomalies in any component. Current approaches to anomaly detection range from mathematical and data driven machine learning approaches to system-based methodologies. Chandola et al. [1] present a survey of the available anomaly detection techniques and their applications.

Anomaly detection based on machine learning techniques use either historic data about the system at hand, or rule-based approaches. The output of the current anomaly detection techniques used in online multimedia services is in the form of a set of static boundary conditions on the service inputs and its system states [19]. Outliers in such models are considered anomalies even if they do not result in any SLA violations or failures like dropped sessions. The key issue with almost all machine learning approaches is their dependence on large amounts of data to create, train, and test new models [1, 5]. The data preparation time is too high for real-time changes and updates [20]. In addition, these approaches rely on service production logs to find the ranges of service inputs and system working conditions to identify the outlier boundaries and their impact on the service. Production Logs are complex and hard to mine [11, 25]. Data in logs may not be sufficient for mining, analysis, and anomaly detection models [8, 22]. Pre-processing the logs to prepare them for anomaly detection models is hard and expensive [26]. Leners et al. [7] use service informers to improve the availability of distributed services; these are built using system messages found in production logs from prior runs of the service, not from the current service in real-time. The resulting anomaly detection models created using logs from prior runs of services may not accurately represent the current service [16, 18].

Even with online system-monitoring-based approaches, like Hystrix of Netflix [2], the monitoring is still reactive, as the anomalies, faults, and failures need to happen and customers endure them before they are controlled. Anomaly can manifest in many forms, including in the process of synchronization of audio and video sessions. To study the impact of geographical distribution of multimedia services and distributed peers, Rainer and Timmerer [13] suggest a self-organized distributed synchronization method for multimedia content. They adapt IDMS MPEG-DASH to synchronize multimedia playback among the geographically distributed peers. They introduce session management to MPEG-DASH and propose a new distributed control scheme that negotiates a reference for the playback time-stamp among participating peers in the multimedia session. The goal is to avoid synchronization and latency anomalies, enhance quality, and reduce jittering. Trajkovska et al. [24] propose an algorithm to join P2P and cloud computing to enhance the Quality of Service (QoS) of multimedia streaming systems. They investigate cloud APIs with built-in functions that allow the automatic computation of QoS. This enables negotiating QoS parameters such as bandwidth, jitter and latency, and avoid wrong characterization of state anomalies.

Many efforts attempted to study the impact of anomalies in video rendering in real-time. Li et al. [9] propose a new rendering technique, LiveRender, that addresses the problems of bandwidth optimization techniques like inter-frame compression and caching. They address problems of latency and quality by introducing compression in graphics streaming. Shi et al. [21] propose a video encoder to select a set of key frames in the video sequence. It uses a 3D image warping algorithm to interpolate non-key frames that otherwise can increase anomalies detected in the frames without true impact on the video session. This approach takes advantage of the run-time graphics rendering contexts to enhance the performance of video encoding. Tasaka et al. [23] study the feasibility of switching between error concealment and frame skipping to enhance the Quality of Experiences (QoE). This approach utilizes a tradeoff between the spatial and temporal quality that is caused by error concealment and frame skipping. The algorithm they suggest switches between error concealment and frame skipping depending on the nature of errors encountered in the video output; this technique also avoids characterizing unimportant frames as anomalies.

Shatnawi and Hefeeda studied system-based approaches for real-time service failure prediction [20], and service capacity estimation [19]. They address the problem of getting real-time data representing the current service by using synthetic transactions. They build predictive models in real-time to predict multimedia session failures like dropping a call. Their approach does not identify anomalies in service inputs and working conditions that may result in SLA violations.

Our approach is different in multiple aspects. We focus on anomalies in service inputs and system states that result in SLA violations. We find SLA violations as they happen under synthetic transactions, not through failure prediction which can be inaccurate and result in over and/or under-stating the anomalies in the system. We use machine learning combined with multidimensional analysis to correlate the service anomalous inputs and working conditions with the service outputs and SLA violations, and find the individual and associative impacts of these parameters on the output of the service. We change the reaction to anomalies based on their current impact on the service, not their historic impact. We monitor the accuracy of the anomaly analysis model and regenerate it in near real-time if it drops below acceptable accuracy.

3 DYNAMIC ANOMALY ANALYSIS

3.1 Overview

Multimedia services have complex inter-dependencies between their systems, like call managers, encoders, de-jitters, renderers, decoders, and storage systems [9, 13]. The range of issues that impact the quality of a multimedia session include computation capacity, network bandwidth, as well as the varying customer load on the system. So it is almost impossible to correlate an anomaly of one aspect of the system like CPU utilization, memory consumption, or user count with degradation of the multimedia service, like poor media quality measured by MOS [21, 23]. Also, the anomalies' impact on the service is not consistent throughout the day and the lifecycle of the service; for example, an anomaly of higher number of videos shared per session may result in SLA violations during peak hours, but may have no impact at all during slow traffic. So it is not optimal to have the same reaction to anomalies like raising alerts all the time, as done in the current anomaly detectors, because alerts are expensive, and their impact is not constant.

We propose a novel approach, called Dynamic Anomaly Analysis (DAA), to analyze, in real-time, the anomaly impact of the service input and working conditions that are found in the trailing 30 days, on the multimedia service. We study the anomalies independently and associatively, and classify their impact into three categories (1) Impactful, (2) Borderline, and (3) Non-impactful. Impactful anomalies result in SLA violations. These warrant alerting and managing as we describe later. Borderline are anomalies that do not result in SLA violations, but impact the measure of interest in the service. DAA monitors borderline anomalies closely without raising alerts, until they start to cause SLA violations. Non-impactful anomalies do not have any negative impact on the measures of interest. For example, if the expected shared videos in a multimedia session is one video per session, then sharing two videos side by side is an anomaly. However, if sharing two videos side by side doesn't impact any measure of interest like media session quality, then this anomaly is considered non-impactful, and DAA takes no actions on these anomalies.

As shown in Figure 1, DAA is a Testing in Production (TiP) service to improve the reliability of multimedia services. If DAA or

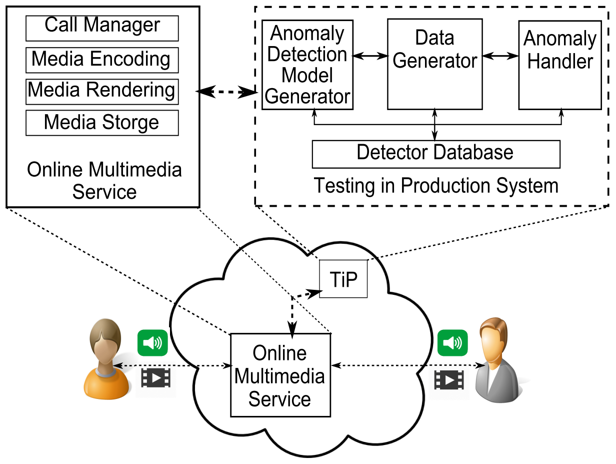
Call Manager
Media Encoding
Media Rendering
Media Storge
Online Multimedia Service
Anomaly Detection Model Generator
Data Generator
Anomaly Handler
Detector Database
Testing in Production System
TiP
Online Multimedia Service

Figure 1: High-level architecture of multimedia communication services.

TiP goes down, the service continues to function properly. TiP is common practice in modern online services, especially in interactive multimedia services [19], as without it the service is flying blind. DAA consists of three main components: **Data Generator**, **Anomaly Detection Model Generator**, and **Anomaly Handler**. The Data Generator creates synthetic transactions that replay actual customer transactions that had anomalies in their inputs from the past 30 days, rolling window, of the service deployment. The anomalous values, i.e., the outlier values of service inputs like number of users and number of video sessions, are the values that are found from the trailing 30 day window as encountered by the service. These values are used in the synthetic transactions made by the data generator. By running these synthetic transactions with anomalous inputs and loads, the Data Generator collects the current reaction of the service under the current working conditions like CPU utilization, memory consumption, and number of processes, as well as the service outputs like number of active multimedia connections and their quality measured in MOS.

The Anomaly Detection Model Generator uses the measurements from the Data Generator to establish correlations between the anomalous service inputs, service working conditions, and system states with the service output. The service output is represented by a metric of interest like MOS for shared media quality, service response time, or communication lag time. The correlations are built using current data from the system. We leverage concepts from the *Association Rule Machine Learning* algorithm that are able to determine the level of independence of each input and working condition and their individual impact on the output of the service, as well as the associative impact of multiple inputs and working conditions. The *Anomaly Handler* uses the model and correlations generated by the Model Generator to identify the groups of anomalous inputs and working conditions that cause SLA violations, classify the anomalies based on their impact, and raise alerts on Impactful anomalies that result in SLA violations.

The following sub-sections detail the functionality of the Data Generator, Anomaly Detection Model Generator, and Anomaly Handler.

3.2 Data Generator

The Data Generator is comprised of two components. The *Synthetic Transaction Provider (STP)* and the service *Item and Feature Evaluator (IFE)*. The STP makes Testing in Production (TiP) API calls to each component of the service, and uses the anomalous service input values that were found in the service in the trailing 30 days. The service is assumed to have basic anomaly detection, as described in [1], and logs these anomalous values in a database that is accessible by the STP. The STP uses these anomalous service input values in the synthetic loads it generates; for example the number of users, number of sessions created every minute, and the number of videos shared. The STP generates service calls to each component, like the encoder, decoder, dejitter, renderer, and storage components, and passes them the anomalous test loads, and collects their outputs into the Detector Database, shown in Figure 1. The service output represents the current service reaction to the anomalies in the service inputs. The data in the Detector Database is real time data, that is generated currently from the system; the lifecycle of this data is in the order of minutes to help monitor the service components. All components of DAA have access to the Detector Database.

In addition to the component synthetic transactions, the STP makes *scenario calls* that represent a true customer e2e transaction. For example, the STP emulates a video call of certain length between two test nodes representing two customers, and shares an actual video between them. Such a scenario call exercises the multimedia service components in a way that mimics real user behavior, using anomalous inputs. The STP collects the results and outputs of these tests into the Detector Database. Test loads to mimic user behavior are generally acceptable to replicate the characteristics and metadata of user transactions. However, if the service at hand requires an exact replica of the user behavior and loads, then parts of previous production logs representing that behavior can be replayed as described in [14, 17]. The STP also makes *operating system calls* to collect system information such as CPU utilization, memory consumption, and process counts. The STP runs the component and scenario tests with progressively larger loads to generate actual SLA violations in the tested service and its components. The combination of regular and anomalous inputs, working conditions, and service outputs under low, medium, and high loads are collected into the Detector Database.

Before we describe the Item and Feature Evaluator (IFE) component, we present a few concepts in real-time Dimensional Modeling [6] and Associative Rule based machine learning techniques [10] as they relate to multimedia services:

- **Feature:** is an individual measurable property of a phenomenon or transaction. For example, MOS representing the quality of a multimedia session, is a Feature. Users of DAA provide the Features of interest, like MOS, as a configuration value of DAA.
- **Item:** is a property representing a transaction attribute; for example number of users, number of processes, CPU utilization, and memory utilization are all transaction *Items*. Items map to dimensions in Dimensional Modeling. *Itemset* is the group of Items in a transaction, and maps to Groups in Dimensional Modeling. This allows the study of the root-cause

analysis between the Items/dimensions and Features/measures, i.e., which Item values caused which Feature values, as enabled by Dimensional Modeling [6].

- **Transaction:** is the activity of interest, like a multimedia session.
- **Support:** is the frequency an Item is seen in the multimedia session under study. Example: if an Item is shown 8 times in 10 transactions, the Support of that Item is 80%.
- **Confidence:** is the frequency that a deduction is found to be true in the multimedia sessions of study. Example: if a deduction like MOS is below 4 every time CPU utilization is above 77% is found to be true in 900 out of 10,000 transactions, the Confidence in such a deduction is 90%. This data is found in the Detector Database, and Confidence computation is a matter of counting the transactions and their content. Users of DAA provide their required Confidence level as a configuration value of DAA.
- **Lift:** is the ratio between the Support of two Items in the set to the Support of both Items in the set if they were independent. Users of DAA provide their required Lift as a configuration value of DAA. The Lift for Item X and Item Y is given by: *(Support of Union of X and Y) / (Support(X) * Support(Y))*
- **Conviction:** is the ratio of the expected frequency that Item X (e.g., CPU Utilization) happens without Item Y (e.g., Memory Utilization) and causes the multimedia session Feature of interest to happen (e.g., MOS = 5). It is given by: *conv(X, Y) = ((1 - Support(Y)) / (1 - Confidence(X, Y)))*

After the data is generated by the STP, the IFE finds the Feature(s) in each multimedia session in the database. The IFE then finds the Items; these are the remaining multimedia session attributes excluding the Feature(s). It then computes the Support, Confidence, and Lift of each Item in the Itemset of each multimedia session. If the computed values meet the configuration values for Confidence and Lift, data generation is complete. If the computed Confidence and Lift for any Item are lower than the configured values, the IFE requests more tests from the STP to provide more correlation data that can achieve the required Confidence and Lift. The new STP tests use service input load values that were not used in the previous tests, i.e., it extends the range of used inputs and their load values to ensure that the newly generated data can generate new correlations. The previous inputs and loads are stored in the Detector Database, and are updated after each test. The DAA Data Generation algorithm is summarized in Procedure 1.

Procedure 1 Data Generation Algorithm

```
ANOMALY DETECTION DATA GENERATION
1:  function GENERATEANOMALYDETECTIONDATA
2:      while (Confidence < ConfidenceConfiguration and Lift < LiftConfiguration do
3:          STPGenerateData();
4:          for each Multimedia Session; do
5:              for each Items do
6:                  Compute Support, Confidence, and Lift;
7:              end for
8:          end for
9:      end while
10:     Collect Items, Features, Support, Confidence, Lift;
11:     Populate CollectorDatabase with Anomaly Detection Data;
12: end function
SYNTHETIC TEST PROVIDER
1:  function STPGENERATEDATA
2:      for each Low Service Loads to SLA-Violation-Causing Loads do
3:          Run component level tests;
4:          Run scenario level tests;
5:          Run system level tests;
6:          Capture test inputs, system states, component outputs and Store in DetectorDatabase;
7:      end for
8:  end function
```

3.3 Anomaly Detection Model Generator

The multimedia session Features and Items and their correlations as computed by the metrics of Support, Confidence, and Lift are determined by the DAA Data Generator, as described in the previous subsection. The DAA Anomaly Detection Model Generator evaluates each Item in the multimedia session Itemset and their impact on the Features, and computes the Conviction associated with each Item like number of users and their dependent Items like CPU Utilization and Memory Utilization. The Conviction value between two items determines how independent they are from each other. For example, a Conviction between two Items like number of users and CPU utilization of 1.15 means that the correlation between them and the quality of multimedia session is 85% more accurate than the correlation between each of them alone with the quality of service. In other words, building an anomaly detector that would fire alerts based on values of one of these Items alone without association with the other has an 85% chance of raising a false positive alert. Users of DAA may configure it to consider items independently if the convection between them is above 1.85 for example.

The Model Generator produces a set of tables of associate Items and their ranges that correlate to a given Feature value like media quality MOS = 1, 2, 3, 4, and 5. Table 1 contains a sample correlation between the upper bound of number of users and CPU utilization, given a Conviction configuration of 1.85 that results in MOS Feature values of 3, 4, and 5. Table 1 in practice is a high cardinality table, and can be Normalized into multiple tables; each representing one Feature, or even one Feature value. Anomalous inputs that cause SLA violations, like MOS 3 or less, are considered impactful and worth raising alerts. Anomalous inputs that result in acceptable measurements of interest, like MOS 4, are considered borderline anomalies that require monitoring but not alerting. Anomalous inputs that do not impact the measurement of interest, like MOS 5, are ignored. The Anomaly Detection Model Generator algorithm is summarized in Procedure 2.

3.4 Anomaly Handler

State of the art anomaly detectors are configured to raise alerts when anomalies are encountered [1]. This may result in high number of alerts, with high cost, as we show in the evaluation section later. In DAA, the Anomaly Handler module identifies the anomalies that warrant alerts, and fire alerts when anomalies in

Table 1: Anomaly detection model for the multimedia service.

User-CPU Conviction	User Upper Bound	CPU Upper Bound	Feature: Video Quality (MOS)
1.85	110	73%	3
1.85	123	61%	4
1.85	128	57%	5

Procedure 2 Model Generation Algorithm

```
ANOMALY DETECTION MODEL GENERATION
 1: function GenerateAnomalyDetectionModel
 2:     for each Feature in DetectorDatabase do
 3:         Group Transactions by Feature Value;
 4:     end for
 5:     for each Feature in DetectorDatabase do
 6:         for each Transaction per Feature Value do
 7:             Group each Transaction Item into Transaction Itemset;
 8:         end for
 9:         for each Item in Transaction Itemset do
10:             Compute Conviction;
11:         end for
12:     end for
13:     Generate Anomaly Detection Model (Table 1);
14:     Publish Model to DetectorDatabase;
15:     Publish Conviction of each Item in Transaction Itemset;
16: end function
ANOMALY HANDLER ALGORITHM
 1: function RaiseAlerts
 2:     Define Anomaly-Based Alert Levels (1 to $N$);
 3:     Create a Mapping between Feature measurement and Alert Level;
 4:     for each Real-Transaction Anomalies that cause SLA violations do
 5:         Find the Alert Level Mapping to the Feature measurement
 6:         Raise the appropriate Alert Level
 7:     end for
 8: end function
```

the real service inputs and working conditions, i.e., in the Itemset, result in undesired Feature state like MOS 3 or below. DAA users configure the Features to define the measurements of interest and the measurement thresholds at which to fire alerts. Users of DAA also configure Lift and Confidence of DAA to define the associative dependence between the service inputs and working conditions, and when to treat them practically independently, and the required level of confidence in the data before firing alerts. By configuring DAA's Features, Lift, and Confidence, users of DAA decide which measurements they want to consider for firing alerts, and the thresholds they consider to be harmful. Anomalies that do not result in harm, like SLA violations, are ignored. This is summarized in the Anomaly Handler algorithm in Procedure 2.

Users of DAA may consider a multi-level alerting system based on the Feature values they are monitoring. For example, if MOS is the Feature of interest, users of DAA may define 3 levels of alerts like Critical for MOS value 1, High Severity for MOS value 2, Medium Severity for MOS value 3. Such sub-classification of the Impactful anomalies is left to the users of DAA to design and implement as they deem fit, which can reduce the cost of handling alerts considerably.

3.5 Remarks and Practical Considerations

As explained in the Data Generator section, the computation of the ARL concepts of Confidence, Lift, and Conviction use the value of Support. The computation of Support is quite expensive, and many algorithms like Apriori, Eclat, and FP-growth have been designed to make its computation efficient [1, 5]. State of the art anomaly detection models read massive amounts of data from logs with very low recall, less than 2%, generate connected data graphs, and leverage those algorithms to perform either breadth first search (Apriori), depth first search (Eclat), multi-pass search algorithms (FP-growth) to count the Support for a given feature. These are good optimizations that can make the Support computation tractable. In our proposed approach, we do not need to use any of those search algorithms, because we generate a small amount of data, in the order of mega bytes, as opposed to peta bytes in the logs. The recall in the proposed approach is 100%. The resulting data from the proposed approach is hosted in a dimensional model that lends itself naturally to counting and grouping. This is an important practical advantage of our proposed approach, that drops the machine learning data preparation time from the order of hours or days to a few minutes as shown in the evaluation section. We believe this work is novel because it is the first to combine: (1) synthetic transactions to generate data with high recall in short time, order of seconds, (2) dimensional modeling to identify features and dimensional schema impacting those features, and (3) association rule learning to create an accurate and dynamic anomaly detector, which can be updated in the order of minutes, as opposed to weeks for existing algorithms.

DAA combines machine learning techniques from the Associative Rules Learning (ARL) algorithm and Dimensional Modeling concepts like Features and Groups. The choice of ARL over other algorithms is made to leverage its ability to compute the interdependence of parameters contributing to a transaction. Other algorithms are able to find correlations equally well, but lack the ability to find the association between the parameters in the transaction [5, 10]. We combine these techniques with near real-time Dimensional Modeling by defining the transaction output as a monitored Feature, and the Items and Itemsets as Dimensions. This approach and the resulting model are novel and of practical value, when used to correlate the ranges of service inputs and working conditions, with the monitored value of the multimedia service output like video session quality.

Synthetic transactions in TiP do utilize the service resources, and this impacts the service. Service designers account for such an impact due to the importance of TiP [3, 15]. We utilize TiP principles and infrastructures as the platform for DAA. No production code is instrumented to generate the data. Data is generated through TiP synthetic transactions. The Data Generator is executed regularly as part of the TiP system. It collects component, scenario, and system information that are used for generic TiP purposes. The Anomaly Detection Model Generation, on the other hand, is the functionality that takes place on demand, when the anomaly detection accuracy drops below the acceptable value.

4 EVALUATION

We present the results of running DAA for one month, and exercising more than 400 million multimedia sessions.

4.1 Implementation and Setup

We implement synthetic transactions for four multimedia components: Call Manager, Media Encoder, Media Renderer, and Media Storage, as shown in Figure 1. The considered geo-distributed multimedia communication service processes over 3 million requests per second at peak time. It is deployed in 8 data centers in 3 continents. We use a test cluster of 10 servers in the data center, which gets about 1% of the data center traffic to run our experiments. Each server is a quad-core Intel Xeon server with 12 GB RAM. The STP makes component, scenario, and system calls, and we capture the service components inputs and their outputs. Similarly, we record the outputs of sharing a video scenario. The TiP test cluster we used received about 400 million transactions over the four weeks of the experiment, with about 3,000 transactions per second at peak. We find the results of the proposed DAA approach from the TiP system, as we capture the test inputs, system states, and the scenario outputs. We find the results of the current anomaly detector of the production service from the system logs, that show the inputs that were considered an anomaly and the resulting output based on that. The production service implements a detector based on Neural Network machine learning. The 1% traffic in the test cluster is split equally between the servers implementing the proposed and current approaches. The traffic split is done by user to ensure continuity of activities received by each system; so 50% of the user base is sent to each system. The results are found for each group and compared.

The STP makes the calls to each of the service components and controls the various aspects of the multimedia request like media type and media size. We measure the quality of the multimedia session using an automated MOS measurement algorithm. Automated MOS measurement algorithms built using actual prior customer assignments that can detect white noise, echo, and other problems are common in test environments that require real-time assessment of media quality [19]. System calls to get system states are implemented in an infinite loop that reads CPU utilization, memory utilization, and number of processes from the performance monitoring APIs of the operating system of each server every 30 seconds. The STP makes simultaneous calls with different user agent information, and controls the load in two ways: (1) number of media sessions made by each client in a given time, and (2) number of simultaneous media sessions representing multiple client calls. The data is collected and stored in the Detector Database with a schema similar to Table 1.

DAA can be used as a standalone anomaly detection system. It finds the anomalies and logs them into the Detector Database. However, if the service at hand prefers to find its own anomalies and leverage DAA for analyzing the impact of these anomalies, the service needs to log the anomalies it finds into the Detector Database, so that DAA can use these anomalous values in its synthetic transactions. In the case of our experiments, we used the anomaly values found by the production service, and pulled them into the Detector Database.

4.2 Performance Metrics

We study the quality of media sharing during the multimedia session and the number of shared video and audio streams. We compare against the state of the art anomaly detection implemented in the online service using Neural Network machine learning algorithm. The Current anomaly detection used in the service is a Replicator Neural Network detector. It has the classic three phases of: (1) input layer, (2) 5 hidden/internal staircase-like activation layers, and (3) linear output layer. Having 5 internal layers technically classifies it as a deep learning algorithm. The training cycle of the anomaly detector is implemented with backpropagation (i.e. backward pass) for error reconstruction. Here, the error between the actual outputs and the presumed/target outputs is computed, and back-propagated to the hidden layers to update the weights matrix of the neural network neurons. As expected, the training process is lengthy and requires high recall in the training data, which is only guaranteed by large volumes of data from previous runs of the service. The Current anomaly detector also implements a preprocessing step, before the replicator neural network, which utilizes a Holt-Winters algorithm for smoothing the time series data representing the service inputs. This step favors fresh data and caters for the seasonality in the data. The following are the metrics we use to assess the performance of DAA:

- **False Positives (FP):** the number of sessions that are considered to have anomaly in their inputs and working conditions, yet did not result in SLA violations.
- **False Negatives (FN):** the number of sessions that are not considered to have anomaly in their inputs and working conditions, yet resulted in SLA violations.
- **True Positives (TP):** the number of sessions that are considered to have anomaly in their inputs and working conditions, and actually resulted in SLA violations.
- **True Negatives (TN):** the number of sessions that are not considered to have anomaly in their inputs and working conditions, and did not result in SLA violations.
- **Accuracy:** the ratio of (true positives + true negatives) to the sum of (false positives, false negatives, true positives, and true negatives).
- **Recall:** in the context of data retrieval from the source, recall refers to the percentage of data that is usable in anomaly detection analysis. In the context of detection analysis, recall is the ratio of (true positives) to the sum of (true positives and false negatives). This the *true positive rate*, or *sensitivity* of the model.
- **Precision:** in the context of anomaly detection analysis, Precision is the ratio of (true positives) to the sum of (true positives and false positives). This is the *Positive Predictive Value* (*PPV*) of the model.

- **Time to Detect Model Changes:** the time it took the Anomaly Detection Analysis module to detect that the model is no longer accurately representing the current system.
- **Time to Update Anomaly Detection Model:** the time it takes to create a new Dynamic Anomaly Analysis (DAA) model after changes in the system.
- **Number of Failures:** the number of sessions that failed to meet the MOS quality SLA due to inaccurate anomaly handling.
- **Media Quality:** the quality of media, audio and video, that is shared between clients; it is measured in MOS.
- **Overhead:** the CPU utilization of the production service with and without the TiP system.

4.3 Results

We measure each of the performance metrics described above for the state of the art approach in anomaly detection implemented in the service, we refer to it as Current, and for DAA, and compare the results. First, we summarize the findings of the false positives/negatives, true positives/negatives, recall, precision, and accuracy for the four weeks of the experiment in Table 2. The current system and its static way of reacting to anomalies result in huge waste, in the form or false positives and negatives, and the accuracy suffers accordingly. On the other hand, DAA finds the impact of the anomalies in near real-time through synthetic transactions and only alerts if the anomalies result in SLA violations. This reduces false positives and negatives, and enhances the accuracy.

The following figures have only one week of the results, with hourly aggregations of data, to make them clearer. We observed a cyclical pattern daily and weekly, so there are no lost insights by the omission of the remaining three weeks of experiment data from the graphs. We show the detailed graphs for CPU utilization and number of users in the system and analyze the impact of DAA on the performance metrics defined earlier. The proposed DAA approach outperforms the current anomaly detector in all metrics.

Accuracy: In addition to Table 2, we detail the accuracy of DAA versus the current detector for users service input and CPU system state, as they are the most impactful on the output of the service. Figures 2 and 3 summarize the enhancements to CPU and user anomaly detection accuracy. We show the details of accuracy as it provides insights into all the remaining metrics; FP, FN, TP, and TN. Using a static boundary for CPU utilization of 50%, which is what the production service had, results in hundreds of CPU violations per hour that end up being raised as false positive alerts. Using DAA, the accuracy of anomaly detection based on real-time monitoring went up from about 59% to 89%. Using DAA, the CPU boundary of safe functionality varied between 40% CPU utilization to 73% before anomalies result in SLA violations. So to assume that we can raise or lower the static boundary, or even use a deterministic cyclical model like sinusoidal to enhance the accuracy, or false positives/negatives, is not true. It needs to be based on data from the current system. Likewise, we see that the accuracy of user anomaly detection using DAA went up from 63% on average to about 90%. The memory consumption anomaly detection accuracy went up from 58% to about 91%, and the number of sessions anomaly detection accuracy went up from 57% to about 89%.

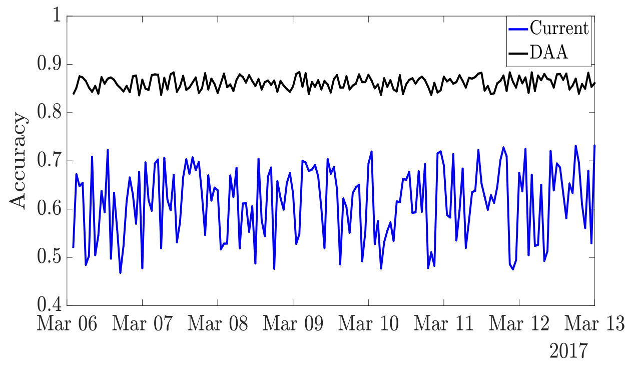

Figure 2: Anomaly detection accuracy for CPU.

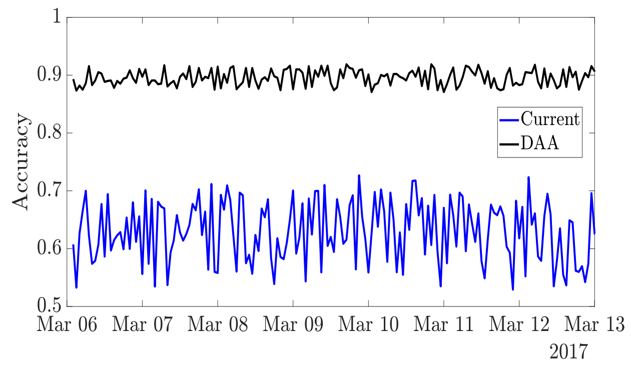

Figure 3: Anomaly detection accuracy for user.

Recall: We compare the findings of data generation time, pre-processing time, and data recall for DAA and the Current detector in Table 3. There is significant time saving in the generation of the detector analysis model. Finding and capturing anomalies is an ongoing process throughout the lifecycle of the service. The savings are in the analysis done on the system and its reaction to anomalies. Using current approaches, data about the system needs to be logged for real transactions and then processed and analyzed. Whereas using DAA, we generate small, highly relevant, near real-time data about the current system, not previous deployments of the service. The recall in the original production data, before preparing it for detector model generation was about 2%. This is due to the fact that production logs have a multitude of data describing all aspects of the service like user sign-in/sign-out, authentication, payments, application usage, and other data. It took 22 hours of processing and preparation for the log data to be usable in anomaly detection. DAA has a recall of 100% due to using synthetic transactions that test the required components for the media quality, and the data is usable directly without any pre-processing. **Recall**: We compare the findings of data generation time, pre-processing time, and data recall for DAA and the Current detector in Table 3. There is significant time saving in the generation of the detector analysis model. Finding and capturing anomalies is an ongoing process throughout the lifecycle of the service.

Table 2: Summary of the results over the entire 4-week period.

Anomaly Detector	FP Rate	FN Rate	TP Rate	TN Rate	Recall	Precision	Accuracy
DAA	10.8	11.5	91.0	93.5	88.8	89.4	89.2
Current	35.2	36.6	50.6	53.4	58.1	58.9	59.2

The savings are in the analysis done on the system and its reaction to anomalies. Using current approaches, data about the system needs to be logged for real transactions and then processed and analyzed. Whereas using DAA, we generate small, highly relevant, near real-time data about the current system, not previous deployments of the service. The recall in the original production data, before preparing it for detector model generation was about 2%. This is due to the fact that production logs have a multitude of data describing all aspects of the service like user sign-in/sign-out, authentication, payments, application usage, and other data. It took 22 hours of processing and preparation for the log data to be usable in anomaly detection. DAA has a recall of 100% due to using synthetic transactions that test the required components for the media quality, and the data is usable directly without any pre-processing.

Time to Detect Model Changes: After system changes, like adding new compute resources, it takes DAA at most one minute to detect that the anomaly analysis is no longer accurate for the current system. DAA's Anomaly Detector issues calls to the STP to run more tests in such cases to verify its findings, before it attempts to create new analysis models. We chose three different sets of component, system, and scenario tests to verify. Each run for 5 seconds every 30 seconds. If the model is no longer accurate, we generate a new model.

Time to Update Model: It takes about 7-10 seconds to analyze the results of component inputs, system states, and scenario outputs. If the results are close (variance less than 5% in accuracy) from the three tests described earlier, we use the model built from the last set of tests. If the results are not close, DAA assumes the system is still not stable, DAA continues testing the system until it reaches a steady state. On average, it takes about 5-7 minutes to update the model. Updating the model usually happens around business day boundaries, and if resources from other data centers are added. During the time of changes, DAA short-circuits itself and lets the production service use its default production anomaly detection service, to avoid introducing TiP-based failures into the production service. When DAA is updated, it requests activation from the production service to perform its anomaly analysis functionality.

Number of Failures: Figure 4 shows the number of session failures, quality below MOS 4, caused by inaccurate anomaly reaction with and without DAA. On average, the failures without DAA where about 0.082%. This translates to tens of media sessions failing every hour (about 63 in 1% of the service traffic); that is thousands of sessions dropping from MOS 4 or 5, good/excellent quality, to below MOS 4, poor quality, in the data center every hour. Using DAA, SLA violations dropped to about 0.051%. Thus, using DAA results in thousands of customers every hour improving their MOS from poor to good/excellent quality.

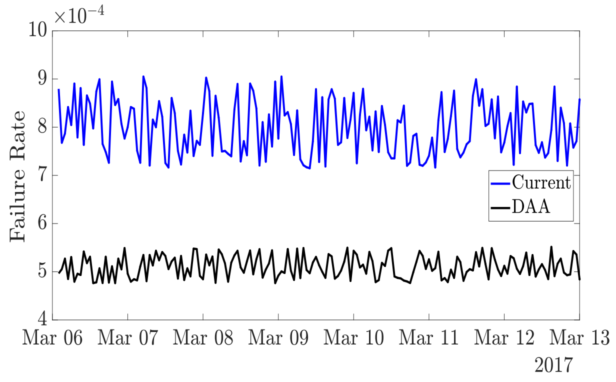

Figure 4: Session failures.

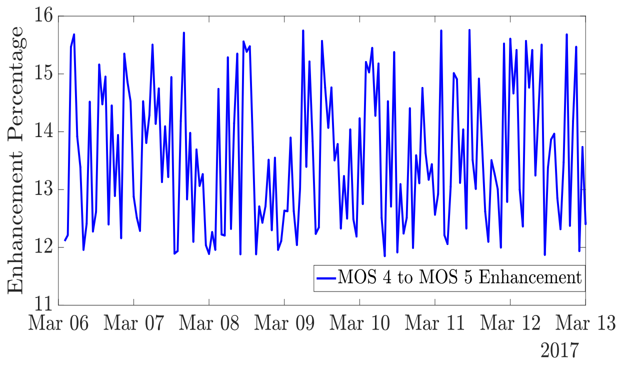

Figure 5: Media quality.

Media Quality: Figure 5 shows the impact of DAA on media quality enhancement of successful sessions. About 14% of media sessions have seen an increase from quality of MOS 4, good quality, to MOS 5, excellent quality. DAA reduced the false negative rate from 36.6% to 11.5%. These are inputs that were not supposed to cause SLA violations in the service outputs, but ended up reducing media quality. DAA detected and marked these inputs as anomalies, overriding the Current detector, so the highest safe range of user count per server in a given time window was changed in real-time. This resulted in different routing scheme of new users to other servers. Otherwise, these users would have been added to the wrong server, overloading it, and resulting in compute resource contention and so media quality drop.

Overhead: The overhead of the TiP system is measured by the online multimedia service, continuously. The service measured the hourly average production service CPU utilization with and without the *whole* TiP sytem, which includes DAA. The average service

Table 3: Data generation and processing times.

Anomaly Detector	Generation Time	Processing Time	Data Recall
DAA	5-7 Mins	2-4 Mins	100%
Current	7 Weeks	22 Hrs	2%

CPU impact caused by the whole TiP system is around 2.8%. The improved multimedia quality, reduced SLA violations, and reduction in false positives and negatives using DAA make the overall investment in the TiP system well justified.

5 CONCLUSIONS

Current approaches for anomaly detection, analysis, and handling are static and cannot keep up with the frequent changes that happen during the lifecycle of online services. Our experimental results collected from a large-scale multimedia system show the current approaches result in large waste in the system resources due to the high percentages of false positives and negatives. Current approaches generate many unwarranted alerts that have high maintenance and support cost. This results in poor confidence in the anomaly detection and the alerts they generate. To overcome these problems, we introduced a new approach that generates current data about the service in real-time, and uses that data to analyze the impact of anomalies on the service. If the inputs and system states do not result in SLA violations, they are not considered anomalies worth alerting on. Through implementation in a production system and running experiments for 4 weeks, we showed that using the proposed approach reduces the amount of false positives in anomaly detection alerts by about 71%, reduces false negatives by about 69%, enhances the accuracy of anomaly detection by about 31%, and enhances the media sharing quality by about 14%.

ACKNOWLEDGMENTS

This work is supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada, and by the Qatar National Research Fund (grant [NPRP8-519-1-108]).

REFERENCES

[1] V. Chandola, A. Bnerjee, and V. Kumar. 2009. Anomaly detection: A survey. In *Journal of ACM Computing Surveys (CSUR'09)*. New York, NY.
[2] Ben Christensen. 2012. *Introducing Hystrix for Resilience Engineering.* http://techblog.netflix.com/2012/11/hystrix.html. and https://github.com/Netflix/Hystrix.
[3] E. Elliot. 2012. Testing in Production A to Z - TiP Methodologies, Techniques, and Examples. In *Proc. of Software Test Professionals (STP'12)*. New Orleans, LA.
[4] M. Kantarczic. 2011. *Data Mining Concepts, Models, Methods and Algorithms* (2nd ed.). Wiley and IEEE Press.
[5] John D. Kelleher, Brian Mac Namee, and Aoife D'Arcy. 2015. *Fundamentals of Machine Learning for Predictive Data Analytics.* MIT Press.
[6] R. Kimball and M. Ross. 2013. *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling* (3rd ed.). Wiley Computer Publishing.
[7] J. B. Leners, T. Gupta, M. K. Aguilera, and M. Walfish. 2013. Improving availability in distributed systems with failure informers. In *Proc. of USENIX conference on Networked Systems Design and Implementation (NSDI'13)*. Lombard, IL, 427–442.
[8] Z. Li, M. Zhang, Z. Zhu, Y. Chen, A. Greenberg, and Y. Wang. 2010. WebProphet: Automating Performance Prediction for Web Services. In *Proc. of USENIX Symp. on Networked Systems Design and Implementation (NSDI'10)*. San Jose, CA, 143–158.
[9] L. Lin, X. Liao, G. Tan, H. Jin, X. Yang, W. Zhang, and B. Li. 2014. LiveRender: A Cloud Gaming System Based on Compressed Graphics Streaming. In *Proc. of ACM International Conference on Multimedia (MM'14)*. Orlando, Florida, 347 – 356.
[10] Kevin P. Murphy. 2012. *Machine Learning: A Probabilistic Perspective (Adaptive Computation and Machine Learning series)*. MIT Press.
[11] K. Nagaraj, C. Killian, and J. Neville. 2012. Structured Comparative Analysis of Systems Logs to Diagnose Performance Problems. In *Proc. of USENIX Conference on Networked Systems Design and Implementation (NSDI'12)*. San Jose, CA, 353–366.
[12] F. Provost and T. Fawcett. 2013. *Data Science for Business: What you need to know about data mining*. O'Reilly Media Inc.
[13] B. Rainer and C. Timmerer. 2014. Self-Organized Inter-Destination Multimedia Synchronization For Adaptive Media Streaming. In *Proc. of ACM International Conference on Multimedia (MM'14)*. Orlando, FL, 327–336.
[14] P. Reynolds, C. Killian, J. L. Wiener, J. C. Mogul, M. A. Shah, and A. Vahdat. 2006. Pip: Detecting the Unexpected In Distributed Systems. In *Proc. of Symp. on Networked Systems Design and Implementation (NSDI'06)*. San Jose, CA, 115–128.
[15] L. Riungu-Kalliosaari, O. Taipale, and K. Smolander. 2012. *Testing in the Cloud: Exploring the Practice*. IEEE Software Magazine.
[16] F. Salfner, M. Lenk, and M. Malek. 2010. A Survey of Online Failure Prediction Methods. In *Proc. of ACM Computing Surveys, Vol. 42, No. 3, Article 10.*
[17] F. Salfner and S. Tschirpke. 2008. Error Log Processing for Accurate Failure Prediction. In *Proc. of USENIX Workshop on Analysis of System Logs (WASL'08)*. San Diego, CA.
[18] R. R. Sambasivan, A. X. Zheng, M. D. Rosa, E. Krevat, S. Whitman, M. Stroucken, M. Wang, L. Xu, and G. R. Ganger. 2011. Diagnosing Performance Changes by Comparing Request Flows. In *Proc. of USENIX Symposium on Networked Systems Design and Implementation (NSDI'11)*. Boston, MA, 43–56.
[19] M. Shatnawi and M. Hefeeda. 2015. Enhancing the Quality of Interactive Multimedia Services by Proactive Monitoring and Failure Prediction. In *Proc. of 23rd ACM international conference on Multimedia (MM'15)*. Brisbane, Australia, 521–530.
[20] M. Shatnawi and M. Hefeeda. 2015. Real-time Failure Prediction in Online Services. In *Proc. of IEEE INFOCOM'15*. Hong Kong.
[21] S. Shi, C. Hsu, K. Nahrstedt, and R. Campbell. 2011. Using graphics rendering contexts to enhance the real-time video coding for mobile cloud gaming. In *Proc. of ACM International Conference on Multimedia (MM'11)*. Scottsdale, AZ, 103–112.
[22] M. E. Snyder, R. Sundaram, and M. Thakur. 2009. Preprocessing DNS Log Data for Effective Data Mining. In *Proc. of IEEE International Conference on Communications (ICC'09)*. Dresden, Germany, 1366–1370.
[23] S. Tasaka, H. Yoshimi, A. Hirashima, and T. Nunome. 2008. The effectiveness of a QoE-based video output scheme for audio-video ip transmission. In *Proc. of ACM International Conference on Multimedia (MM'08)*. Vancouver, BC, Canada, 259–268.
[24] I. Trajkovska, J. Rodriguez, and A. Velasco. 2010. A novel P2P and cloud computing hybrid architecture for multimedia streaming with QoS cost functions. In *Proc. of International Conference on Multimedia (MM'10)*. Firenze, Italy, 1227–1230.
[25] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan. 2009. Detecting Large-Scale System Problems by Mining Console Logs. In *Proc. of ACM Symp. on Operating Systems Principles (SOSP'09)*. Big Sky, MT, 117–132.
[26] Z. Zheng, Z. Lan, B. H. Park, and A. Geist. 2009. System Log Pre-Processing to Improve Failure Prediction. In *Proc. of IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'09)*. Estoril, Lisbon, Portugal, 572–577.