Enhancing the Quality of Interactive Multimedia Services by Proactive Monitoring and Failure Prediction

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

Mohamed Hefeeda
Qatar Computing Research Institute
Hamad Bin Khalifa University
Doha, Qatar

ABSTRACT

Online multimedia communication services, such as Skype and Google Hangout, are used by millions of users every day. Although these services provide acceptable quality on average, users occasionally suffer from reduced audio quality, dropped video streams, and even failed sessions. To mitigate some of these problems, service providers closely monitor the performance of different parts of the system. However, most current techniques for monitoring and managing the quality of service (QoS) of online multimedia communication services are reactive and lack the ability to adapt to dynamic changes in real time. We propose a novel *proactive* approach for continuously monitoring the health of large-scale multimedia communication services, and dynamically managing and improving the quality of the multimedia sessions. The proposed approach, called Proactive QoS Manager, has novel light-weight methods for estimating the capacity of different components of the system and for using this capacity estimation in allocating resources to multimedia sessions in *real time*. We implement the proposed approach in one of the largest online multimedia communication services in the world and evaluate its performance on more than 100 million audio, video, and conferencing sessions. Our empirical results show that substantial quality improvements can be achieved using our proactive approach, without changing the production code of the service or imposing significant overheads. For example, in our experiments, the Proactive QoS Manager reduced the number of failed sessions by up to 25% and improved the quality (in terms of the Mean Opinion Score (MOS)) of the succeeded sessions by up to 12%. These improvements are achieved for the well-engineered and highly-provisioned online service examined in this paper; we expect higher gains for other similar services.

Categories and Subject Descriptors

H.4.3 [**Information Systems**]: Information Systems Applications: Communications Applications

MM'15, October 26–30, 2015, Brisbane, Australia.
© 2015 ACM. ISBN 978-1-4503-3459-4/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2733373.2806275.

Keywords

Online multimedia communications; quality of service; failure prediction; dynamic monitoring

1. INTRODUCTION

Interactive multimedia communication services such as Skype, Viber, Whatsapp, and Google Hangout offer a wide variety of services including calling, media sharing, and conferencing services. There are over a billion customers who use online communication and media sharing services everyday [21]. These social and multimedia communication services compete in the features they provide, as well as the quality of their services. They have Service Level Agreements (SLAs) covering various aspects such as call quality, media quality, content sharing latency, reliability, response times, and up-times. Failure to adhere to SLAs results not only in low customer satisfaction, but also comes with a heavy price tag due to fines and loss of business. For example, the estimated cost of the annual downtime of IT systems in North America is about $26.5 billion [2].

A typical interactive multimedia communication service includes client applications and an online cloud infrastructure deployed in multiple data centers around the world, and it uses the Internet as the backbone for communications. When a user calls another user, a client application initiates the session and invokes an online service endpoint in one of the data centers. This end point manages the session by invoking many other sub-services, referred to as components, including identity verification, call management, media adaptation, session routing, experimentation, and advertising. Instances of these components are created and provisioned on different data centers. The perceived quality of session is directly impacted by the performance of these components. Therefore, if we are able to monitor these various components in real-time to assess their current performance and predict any failures before they happen, we can significantly improve the quality observed by users, by routing the sessions and multimedia content through the components and data centers that currently have the highest performance and reliability.

We define a failure as an observed deviation of a component from its expected behavior. For example, some users may be having a video conference call using Skype, sharing a presentation, and exchanging messages. Users expect the audio and video streams and the shared presentation to work properly without failures such as intermittent audio, video glitches, and lost slides. Failure management, a term we use to refer to all aspects of dealing with failures, plays a key role in the reliability of online multimedia communication services. It includes service monitoring, failure prediction and detection, root-cause analysis, all the way to failure handling and prevention. We focus in this paper on end-to-end (e2e) real-time service health monitoring, failure prediction, and making real-time decisions about component selection and routing paths to enhance the reliability and quality of interactive multimedia communication services.

Current approaches to health monitoring and failure prediction in multimedia communication services, and online services in general, are mostly reactive [17, 11]. For example, Monitoring in Hystrix of Netflix takes place in real-time but failures are noted after they happen. Also, the creation of a failure predictor is complex and time consuming and thus not suitable for real-time management [25, 12]. Thus, failure prediction generation takes place before the run-time lifecycle of the online service. Also, the resulting predictor is built for certain configurations and working conditions like the number and performance of audio de-jitters and video transcoders. If these configurations change, the predictor may no longer be accurate. For example, the predictor could be designed for a video encoder that is able to encode N videos per minute and it fails beyond that. If the administrator adds more resources to the encoder so that it can handle $2 \times N$ videos per minute, the predictor would likely continue to predict failure if the number of videos per minute approaches N, not $2 \times N$. In this paper, we call these predictors **static predictors**, because they do not adapt to changes in the service functionality or the provisioned resources.

We propose a novel approach to monitoring the health and improving the quality of interactive multimedia communication services **in real-time**. Real-time refers to the runtime lifecycle of the service, where the service is in production and being used by real customers. We use *synthetic transactions* to monitor the service by exercising it like real customers, generate current data about it, and then use this *fresh* data to create and maintain up-to-date failure predictors for the service. The results of the synthetic transactions and the failure predictors are used to make decisions, in real-time, on which services and which paths to route audio and video sessions to.

We implement and evaluate the effectiveness of the proposed approach in a large interactive multimedia communication service in Microsoft[13]. The service handles around 2.5 million transactions per second at peak time. We run our experiments for 5 days in a test cluster that gets 1% of the traffic in a data center, and we process more than 100 million audio, video, and conferencing sessions. Our results show that our proactive approach not only substantially improves the quality of interactive multimedia communication services, but it also saves network and computing resources. This allows the service to admit more sessions and/or allow current sessions to further improve their quality by adding more streams.

The contributions of this work are as follows:

- Novel approach to proactively monitor the health and quality of interactive multimedia communication services.

- Light weight algorithm for building dynamic failure predictors in online multimedia communication services, and using these predictions to estimate the available capacity of different components of the service.

- Algorithm to provide service configuration and routing path recommendations in real-time for the sessions in multimedia communication services.

- Insights and experiences from our actual implementation and deployment of the proposed approach in a large online service.

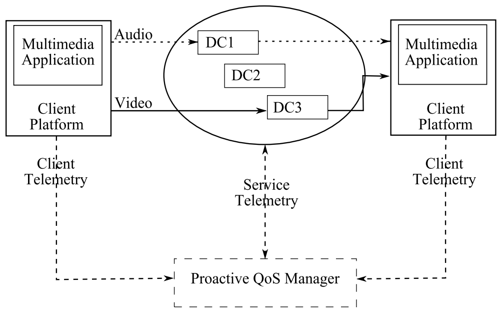

Figure 1: High-level architecture of multimedia communication services.

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

2. BACKGROUND AND RELATED WORK

In this section, we present a high level overview of interactive multimedia communication services, describe how audio and video sessions are created, the points of failure we try to address, and the current research methods that attempt to handle them.

2.1 Background

A high level diagram of an interactive multimedia communication service is depicted in Figure 1. When a user calls another user, the session is routed to the closest data center. The session may be composed of audio, image, and video streams. Each data center implements an instance of the multimedia communication service that is able to handle all streams in a session. Each data center also implements a service routing and selection component. If the data center is able to handle the session, it routes all aspects of the session to its internal components. If the data center is able to handle part of the session, e.g., the audio but not the video stream, it routes the video stream to another data center. Thus, a multimedia communication session may have all its streams going through the same path from the source to the destination, or may have some of the streams of the session go through different paths. An example is shown in Figure 1, where the audio from the source client on the left is processed by DC1, and the video is processed by DC3.

Figure 2 provides a more detailed look at the components in each data center as well as the components of the proposed approach, shown in the dotted rectangle. We describe the proposed approach in the next section. For now, we describe the various components in each data center, and how

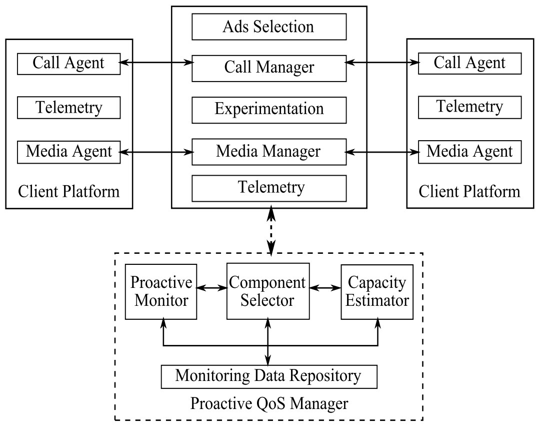

Figure 2: The components of the multimedia communication service and the Proactive QoS Manager.

a session is created between two client applications. The service has challenging requirements, and operates under strict SLAs. It receives multimedia communication requests from applications running on mobile devices, Web, and PCs. It processes each request, determines the source device, identifies ads and experimentation configurations based on app categories, device types, user information if available with user permission to use, and user location. It processes the media, and makes a call to the destination client app with the target media, ads, and experimental parameters. Each of the components in the multimedia communication service is comprised of many geo-located instances so that the client call is routed to the most suitable, geo-location and performance wise, data center. The service is composed mainly of Call Manager, Experimentation, Ads Selection, Media Manager, and Telemetry components. The Media Manager estimates the available compute and network resources to maximize the session quality. The Media Manager enables image, audio, and video content to be transported from one endpoint to another as part of a session using UDP or TCP. It implements audio and video encoding and decoding, and packet loss compensation. The Ads Selection component finds relevant ads for the session. The Experimentation component identifies sessions that are suitable for new experiments like new app functionality and/or new color schemes. The Telemetry component collects system performance indicators like available memory and compute resources, as well as functionality measurements like the quality of a video in a multimedia session and number of calls made per minute.

When a user makes a call from an application, the Call Agent tries to connect to the multimedia communication service. The Call Manager of the service handles the incoming session request. It breaks down the multimedia session into its constituents (i.e., audio, video, conferencing, ads, experiments) and tries to identify the best instances in the available data centers that are able to handle the multimedia session. Depending on the used transport protocol (UDP vs TCP) the packets of the session are routed to the selected components, and the connection is established between the two ends of the multimedia session. During the session, the selected components and routing paths do not change.

As can be noted from this high-level description, the quality of the multimedia calling experience has complex dependencies including the selection of components, the routing paths to get to the destination, and the scale and performance characteristics of the Media Manager. There is high demand on the Media Manager component, which may lead the Media Manager to reduce the quality of multimedia streams to ensure that communication sessions do not get dropped. So if we can, in real-time, continuously know the components that have the highest available resources, the highest performance characteristics, and the highest multimedia throughput quality as well as predict how these factors will change over the coming short period of time, we can make component selection and path routing decisions in real-time that will result in high quality multimedia communication sessions.

2.2 Related Work

Several works have tried to address the reduced quality stemming from contention in multimedia communication services. For example, 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. The work in [24] deals with the limitations caused by the platforms where the multimedia streaming takes place. Tasaka et al. [22] 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 to frame skipping depending on the nature of errors encountered in the video output. Ito et al. [5] study the tradeoff between quality and latency in interactive audio and video applications with the Internet being the medium of communications. They note that the temporal structure of audio-video communication is disturbed by the delay jitter of packets. They study the impact of de-jittering on latency, and the impact of improved latency on quality, which is how the temporal structure is preserved. Through the adoption of psychometric methods, they adjust the initial buffering to enhance latency and quality.

To study the impact of geographical distribution of multimedia services and distributed peers, Rainer and Timmerer [14] 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 improve synchronization, enhance quality, reduce jittering, and enhance latency.

A number of works have attempted to address video rendering problems 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. [20] propose a video encoder to select a set of key frames in the video sequence which uses a 3D image warping algorithm to interpolate non-key frames. This approach takes advantage of the run-time graphics rendering contexts to enhance the performance of video encoding.

The aforementioned approaches work on the limitation of the systems, and try to make the best of available resources to enhance the multimedia communication experiences. These schemes aim at controlling QoS, and managing latency and lack of quality. They come short of addressing the actual cause of the problem, which is knowing the state of the services, identifying the services and components that are not performing well, and/or will not perform well in the coming period, and trying to avoid contention and congestion before they happen. We propose a different approach to enhancing the quality in multimedia communication services. We proactively and dynamically monitor the multimedia services to get an insight into their state, and continuously know the best available components where there is minimal contention and congestion, best quality, and least jitter and delay.

3. PROACTIVE QOS MANAGER

We define three terms that we use in this paper: **Application**, **Service**, and **Platform**. Application refers to the software used by real customers to make calls. Service is the online backend system that applications invoke to accomplish the required functionality such as calling other parties and sharing media with them. A service may be comprised of one or more online services. To avoid confusion, we refer to these other services as components of the main service. Platform refers to the stack of software, hardware, and operating system that is used to run the applications and services. For example an Apple iPhone is a client platform, and Microsoft Azure is a service platform.

3.1 Overview

The proposed approach, denoted by Proactive QoS Manager in Figure 2, monitors the health and QoS of multimedia communication services and predicts failures in realtime. We use synthetic transactions to proactively and continuously test the services and platforms and generate current data about them. The data is used in its ephemeral state to: (1) provide situational awareness about the whole service, (2) correlate failures between applications, services, and platforms, and (3) build a failure prediction algorithm that predicts future failures. The data is used to keep the failure prediction system up to date with the changes of the services and platforms. These three steps result in the ability to make decisions about which components to use that result in the best possible multimedia session experiences.

Before getting into the details of the proposed approach, we describe the Testing in Production (TiP) concept, because our approach uses it. TiP entails software testing techniques that utilize real production environments, while mitigating risks to end users [16, 3, 23]. Online service providers such as Facebook, Google, Microsoft, and Yahoo use TiP to perform functional, stress and performance testing in real-time [23]. Synthetic transactions are used in TiP to generate testing loads, and they are marked with special moniker(s) so that they are distinguished from real transactions. Synthetic Transactions should not interfere with the service destructively, or result in an incorrect state of the service like product inventory reduction due to test purchases. Synthetic transactions in TiP do use the computing resources of the tested service. Service designers account for such an impact due to the importance of TiP; without it the service would be flying blind [16, 23]. We add our synthetic transactions for QoS management to the existing TiP transactions as described in the next section. No code is instrumented in the production code of the service to generate the data for our approach. This is an important advantage for our approach, because we do not impact the production code, and thus no extra testing is needed. Also, updates to the service monitoring and failure predictor do not require production code update or redeployment.

Figure 2 shows the high-level diagram of the multimedia communication service with the proposed approach in the dotted rectangle. The proposed approach consists of four components: (1) Proactive Monitor, (2) Capacity Estimator, (3) Component Selector, and (4) Monitoring Data Repository. The first three components are implemented for each service instance deployed in a single data center, and the Monitoring Data Repository is centralized to all instances; i.e., one instance for the whole system. The Proactive Monitor tests the individual components of the service as well as the whole service e2e. It collects the test results and makes them available to the Capacity Estimator and the Component Selector. The Capacity Estimator uses this data to implement a dynamic failure predictor and to keep it up to date to handle any changes. It predicts the capacity of individual components of the service and the loads at which the components will start to violate their SLAs. The Component Selector collects the current status of the service and its components from the Proactive Monitor, and the capacity limits that will result in SLA violations for the coming monitoring period from the Capacity Estimator. It then creates a **Service Capacity Plan** for the coming monitoring period. The plan contains the available components in the service, their current state, and their projected capacity before SLA violations start to happen. It pushes this plan to the Monitoring Data Repository which gets the service capacity plans from all service instances from all data centers. It combines these plans into one **Global Capacity Plan**, and makes it available to each service instance to pull through an API. The plans are updated regularly; the frequency of update depends on the service at hand, typically this would range from 30 seconds to a few minutes. The following subsections describe each of the elements of the proposed approach.

3.2 Proactive Monitor

The goal of the Proactive Monitor is to provide real-time insights into the state of the multimedia communication service. We define two concepts: **Local System** and **Scenario**. Local System refers to the stack of software, hardware, and operating system used to perform a specific functionality in the online service. Scenario refers to the collaboration of the set of local systems that are used to implement an e2e user scenario. As an example, assume a multimedia communication service that allows users to invite other users to watch a video together. The e2e scenario is the process of calling others and sharing videos with them. Assume the event of interest is video quality. The video quality SLA for the e2e transaction could be 4, i.e., good quality, using the Mean Opinion Score (MOS) [10]. The video sharing process meets its SLA if the video MOS quality is 4 or 5, and fails otherwise. Assume that the sharing service makes the following calls behind the scene: buffer and encode video, transmit video, receive video, decode video, and de-jitter video. Each of these calls represents a local system. The collaboration of these local systems to implement the video sharing process represents the e2e scenario.

We define three levels of monitoring: (1) local system level, (2) scenario level, and (3) platform level. The Proactive Monitor implements the first level by regularly performing Local System synthetic transaction Tests (LSTs), to continuously monitor the health of each local system. For example, this would be calling the video buffer and encode function and passing it a video, and noting its output video and the time it took to perform the encoding. The Proactive Monitor implements the scenario level of monitoring by performing an e2e Scenario Synthetic Transaction Tests (SSTs) that implement the functionality provided to customers, i.e., sharing a video between two clients. The SST test calls all the local systems that collaborate to perform the functionality, and notes the e2e output of the scenario. These two sets of tests, the LSTs and SSTs, are used to monitor the individual local systems and the e2e scenarios. The Proactive Monitor also implements Platform Synthetic Transaction Tests (PSTs) to collect performance indicators like CPU utilization, memory consumption, and process count to correlate with the observed failures. We do not implement application synthetic transactions, i.e., test calls to the applications, as applications are generally installed on user devices and are mostly on metered networks. We use the default client telemetry implemented on these devices that provide information about the application usage. We combine this information with the service and platform synthetic transaction information to monitor the service e2e, and to generate failure prediction systems.

The Proactive Monitor makes LST and SST calls using test loads, which are similar to real loads. The real user behavior, loads and call distributions, is found from logs of previous deployments of the service, or from field/market studies about the service. For example, it would be known, based on previous production logs and/or estimated from market research, that the average user of a multimedia communication service shares a video of 60 seconds with 2-5 friends at a time, and that the service gets around 1,000 concurrent users at peak times. The SSTs made by the Proactive Monitor start with these loads, and progressively add more loads until the failure causing loads are found. If the SLA of video quality is, say, a minimum MOS value of 4, then any video quality of MOS value less than 4 is considered a failure. The test loads are executed on the current service, thus they generate information that represent the current state of the service and its performance characteristics. The synthetic transactions are run for a short period of time, e.g., 5 seconds every couple of minutes. Thus, they do not increase the load on the service considerably.

The LST, SST, and PST calls are made at equidistant time series; once every M seconds. The value of M is configurable, depending on the service, and it varies during run time, depending on the state of the service. If the service is churning and failures are happening more, then M is reduced to get a better pulse of the service. The event, i.e., video quality, is captured after each of these tests, and its value is compared to the result of the prediction. If the accuracy of the predictor starts to drop, the tests are done at a higher rate to generate data to build a new predictor.

The set of tests performed every M seconds need not be exactly like those of customer behavior. However, if the service at hand requires an exact replica of the user behavior, then parts of previous production logs representing real user behavior can be replayed as suggested by [15, 7, 18].

3.3 Capacity Estimator

The Capacity Estimator tries to find the maximum capacity that the service and its components can handle before SLA violations start to happen. It does so by predicting the capacity at which failure starts to happen. It implements a dynamic failure predictor that is able to cope with the service and component changes in real-time. The failure predictor is an independent module that runs in the same environment where the online multimedia service runs. It is part of the TiP system. It has access to the same resources as the production service, but is not part of the online service code. Updates and re-deployments of the predictor code can be done anytime without interfering with the service production system.

Failure predictors are designed to predict the outcome of an event [4, 6]. In multimedia communication applications and services, events represent a variety of measurable aspects of the service, such as the call quality, the response time of the service, and the availability of the service. In this paper, failure prediction means predicting when the outcome of an event does not meet its SLA. For example, if the event of interest is multimedia quality, then failure prediction means predicting the cases where the quality of the shared media drops below the MOS value specified in the SLA. We need a dynamic failure predictor that can be created and updated in real-time. We implement one of the latest failure predictors called Real-Time Dynamic Failure Prediction (RTDFP) algorithm [19]; other predictors can be used in our approach as well. We choose RTDFP because it can be created and maintained in real-time in a short amount of time, and has high prediction accuracy. We customize RTDFP for our Proactive QoS Management approach. The main steps of the Capacity Estimator are summarized as follows:

- Step 1: Use the Proactive Monitor to exercise the service and generate current data about it.
- Step 2: Collect, from the Proactive Monitor, the synthetic transactions and applications' inputs, service and platform outputs, e2e scenario outputs, such as media quality and response times, and events of interest. This data is collected into in-memory constructs with a predefined dimensional model (explained below).
- Step 3: Build a real-time predictor, from the data collected from the applications, services, and platforms and the events of interest.
- Step 4: Estimate the available capacity based on the outcomes from the failure predictor.

The data collected in Step 2 includes the current load and state of the service like the number of videos to process, and the used resources like media processors and call de-jitters. Service and platform related data such as memory utilization, number of processes, and CPU utilization are also collected. We describe in the evaluation section how we capture such data in our experiments. The data is collected and hosted in memory in an array with a dimensional model schema [8]. Dimensional modeling refers to a set of techniques used in data analysis to provide insights into the cause-effect relationships between entities. Data is organized into two sets, a set of measured or monitored data, called **facts**, and a set of parameters, called **dimensions** that define and control the facts. For this paper, the fact is the event of interest that is to be predicted, such as media quality. The dimensions are the other sets of data that impact the event, such as:

- The **LST** that is called: this allows us to know which test is run.
- The **time** that LST is called: the time stamp allows us to relate the cause and effect in the service calls, i.e., at a specific time there were that many function calls and tasks, which caused the observed response time.
- The **number of tasks**: the number of media processing jobs in the service.
- The **component**: the specific component, like Ads Selection or Experimentation, in the service instance.
- The **data center**: where the components are deployed; this could include more details such as the deployment ID.

Table 1: Dimensional Model for the online Multimedia Service.

Dimensions					Fact
Dim1 LST	Dim2 Tasks	Dim3 Data center	Dim4 Component	Dim5 LST time	Fact1 Video Quality (MOS)
1	53	3	5	23	4
2	53	3	2	17	4
3	53	3	3	31	3
4	53	3	1	19	5

Table 1 represents a service QoS dimensional model. Note that we use surrogate keys [8] to represent non-measured and non-numeric values. A surrogate key is a unique identifier of an entity; it is an integer and it is not derived. This makes a table with smaller footprint in memory, as only integers are used, and enhances the performance of operations done on it. The map between the surrogate keys and the actual values is stored in a local table that has the actual dimension values and the corresponding surrogate keys. For example, the service instance in the US western data center maps to "Dim3 Service" surrogate key value 3.

The collected data is used for two purposes. The first purpose is to feed the Monitoring Data Repository to build reports and dashboards that represent the current state of the service such as how many users are making multimedia calls in a given data center and the average quality of shared media over time. This provides situational awareness about the service on an ongoing basis. The other purpose that we use this data for is to generate a real-time failure predictor, and maintain its accuracy over time (Step 3).

A predictor is a classification system [4, 6] that defines and monitors boundaries of working conditions that result in an event success or failure. The event could be the MOS quality. The working conditions are the independent and dependent variables that control the outcome of the event. We use the load of each component as the independent variable, and the CPU utilization as the dependent variable; because it is dependent on the load of each component, yet it plays a key role in controlling the event outcome. We implement a light-weight predictor based on RTDFP that uses the load and the CPU utilization of each component as a logical expression model [19]. The logical expression model is a set of conditions defining the safe component working-conditions that result in the success of the event. We define the following model for the MOS value to be successful:

- MOS Quality Condition for Success for Component N:
  - Independent Variable for Component $N <$ $IndependentVariableMaxValue$ found by the Proactive Monitor
  - Dependent Variable For Component $N <$ $DependentVariableMaxValue$ found by the Proactive Monitor

The Capacity Estimator (Step 4) gets the maximum values for the independent and dependent variables that are found by the Proactive Monitor LST and SST tests, builds the Service Capacity Plan for the coming monitoring period, and passes it to the Component Selector. In other words, the Capacity Estimator utilizes the RTDFP failure prediction to build the component capacity for the coming monitoring period that will not result in SLA violations.

3.4 Component Selector

The Component Selector combines the current data about the service from the Proactive Monitor with the projected capacity that the service and its components can handle from the Capacity Estimator. Table 2 shows an example of this combined data that represents the Service Capacity Plan for the coming monitoring period. The Component Selector pushes Table 2 to the Monitoring Data Repository to be combined with similar data about the components of the rest of the data centers. The Component Selector pulls the Global Capacity Plan from the Monitoring Data Repository into a new instance of Table 2.

The Service and Global Capacity Plans are computed at the beginning of each monitoring period. Each Component Selector reads these plans from the Monitoring Data Repository, and makes them available for its local service instance in its data center. The local service reads the Global Capacity Plan from the Component Selector into an in-memory array to make access to it fast and suitable for routing every session in real-time. When a client invokes the local service, the local service finds the most suitable component to handle the session from its in-memory copy of Table 2, and routes the session to it. Two factors are used in determining the most suitable component: (1) geo-graphical location; the Data Center dimension in Table 2 is used to find the closest component that can be used, so locally first then the same region, and (2) the component that has the highest available capacity; the Projected Extra Media Jobs fact in Table 2 provides this information. Once a session is committed to a component, it remains there until completed. In other words, session routing to components using

Procedure 1 Component Selector Algorithm

```
COMPONENT SELECTION
    function GetCurrentStateOfServices
    Select Service, Component, ProcessingJobCount, AvgMOS
        from Table1
        orderdescendingby AvgMOS
        groupby Service and Component;
    end function

    function GenerateServiceCapacityPlan
    GetCurrentStateOfServices();
    Use RTDFP to get Predicted JobCount until failure;

    MergeJoin Ordered list with Predicted JobCount until failure
            orderdescendingby Service and Component;

    Select Service, Component, ProcessingJogCount,
        AvgMOS, JobCountUntilFailure
        orderdescendingby AvgMOS and JobCountUntilFailure;

    Populate Service Capacity Plan instance of Table 2;
    end function

    function GetGlobalCapacityPlan
    Push Service Capacity Plan to Monitoring Data Repository;
    Wait until Monitoring Data Repository builds Global Capacity Plan;
    Pull Global Capacity Plan from Monitoring Data Repository;
    Push Global Capacity Plan to local service Configurations component;
    end function
```

the Global Capacity Plan affects new sessions only. This prevents session oscillation between components.

As shown in Procedure 1, the Component Selector implements three functions: GetCurrentStateOfServices(), GenerateServiceCapacityPlan(), and GetGlobalCapacityPlan(). GetCurrentStateOfServices() gets the current loads and status of each component in the system from the Proactive Monitor. This function returns an instance of Table 1. GenerateServiceCapacityPlan() uses the failure predictor to predict the loads that will result in SLA violations. These loads are used to determine the available capacities in the components for the next monitoring period. The available capacities are combined with the instance of Table 1 to produce an instance of Table 2, which is the Service Capacity Plan. GetGlobalCapacityPlan() pushes the Service Capacity Plan to the Monitoring Data Repository which gets all such plans from all Component Selectors. The Monitoring Data Repository combines all these instances into the Global Capacity Plan. GetGlobalCapacityPlan() pulls the Global Capacity Plan from the Monitoring Data Repository and pushes it to the service configurations so that the service can use it in routing sessions.

Table 2: Service Capacity Plan.

Data Center	Component	Current Processing Jobs	Current Avg MOS at Service	Projected Extra Media Jobs
4	5	36	5	28
2	6	41	5	23
3	2	27	5	19
7	4	17	5	17

3.5 Monitoring Data Repository

The main function of the Monitoring Data Repository is to combine the local Service Capacity Plans into one Global Capacity Plan that represents the whole system, and to make it available to all components to pull as needed. This is an important functionality that prevents excessive communication between the Component Selectors of each data center instance of the Proactive QoS Manager. With the Monitoring Data Repository, every Component Selector computes the Service Capacity Plan, and communicates twice with the Monitoring Data Repository. The first is to send its Service Capacity Plan, and the second is to get the Global Capacity Plan. The Monitoring Data Repository gets all Capacity Plans, merges them, and generates one Global Capacity Plan that represents the current usage of all components as well as their projected capacities. The Monitoring Data Repository is implemented as a single instance. It is part of the TiP system, so it does not have the high availability requirements that production systems have. In case it is down, each Component Selector continues to use its own Service Capacity Plan until the Global Capacity Plan is built. The production service uses the suggested Proactive QoS Monitor as a heuristic agent to enhance its component selection and routing functionality. The production system is designed to survive and function well, even if the whole TiP system is down.

4. EVALUATION

We implement the proposed approach in an operational online multimedia communication service offered by Microsoft [13] and we present results from more than 100 million video and audio sessions.

4.1 Implementation and Setup

We implement LSTs for four local systems: Call Manager, Experimentation, Ads Selection, and Media Manager. The geo-distributed multimedia communication service processes about 2.5 million requests per second at the peak. It is deployed in 8 data centers in 3 continents. LSTs are API calls to each of these components, with loads as explained in the Proactive Monitor section. The setup we implement is based on the model shown in Figure 2. We use a small 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. We implement a Proactive Monitor that makes LST, SST, and PST calls to exercise each component and we capture the inputs to the LSTs and the outputs of the Local systems. Similarly, we record the output of SSTs and PSTs. An SST is composed of the LSTs that represent the scenario; so a session with audio and video is composed of Call Manager, Experimentation, Ads Selection, and Media Manager LSTs with parameters that specify the audio and video characteristics like length and encoding. We measure the SST response time and quality of the multimedia session using a proprietary automated MOS algorithm. PSTs 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 component every 30 seconds.

The Proactive Monitor makes the calls to each of the local systems and controls the various aspects of the multimedia communication request, including media type, media size, location, and automated user information. The Proactive Monitor also simulates multimedia sessions made by multiple clients, by making simultaneous calls with different user agent information. It controls the load in two ways: the number of media sessions made by each simulated client in a given time, and the number of simultaneous media sessions representing multiple client calls. The data is collected into in-memory arrays with a schema similar to Table 1. The synthetic transactions are designed to run 100 concurrent LSTs every minute. The accuracy of the predictor is measured by comparing its predictions with the actual results and monitored event value from running LSTs and SSTs. Each LST set of tests takes about 5 seconds to complete. The resulting data from the Proactive Monitor synthetic transactions are collected by the Capacity Estimator and Component Selector. The Capacity Estimator makes capacity estimates for the coming monitoring period of one minute. The Component Selector builds the Service Capacity Plan every minute, and communicates with the Monitoring Data Repository to get the updated Global Capacity Plan.

4.2 Performance Metrics

We study the quality of media during the multimedia session and the number of shared media streams, i.e., video, audio, and image. We use the following metrics in our experiments:

- **Media Quality**: the quality of media (audio, video, and image) that is shared between clients. We use MOS for this metric. We use a proprietary *automated* MOS algorithm. Other automated MOS test software is commercially available, e.g., [1]. These automated MOS algorithms enable quality measurements in test environments, where sessions are generated programmatically between test clients.

- **Number of Failures**: the number of sessions that failed to meet the required MOS quality as specified by the service SLAs.

- **Increase in Service Capacity**: the number of additional sessions that the service can handle. This value is found by the Capacity Estimator and using the Proactive Monitor tests to verify its accuracy. We measure the service capacity, by finding the loads that the service can handle until failure, with and without the proposed approach.

- **CPU Utilization**: this reflects how much of the total available resources are freed up due to better resource utilization and load balancing. We get this as part of running the PSTs, through an operating system API call every 30 seconds. This is extracted for each component.

- **Overhead**: We measure the CPU utilization of the service with and without the TiP system, to find the overhead of TiP.

4.3 Results

The service we test our approach in has a high track record for meeting its strict SLAs; on average and over a period of almost three years, the service is able to meet its multimedia communication quality SLA more than 99.9% of the time. The service strives to do better, as a 0.1% failure rate is

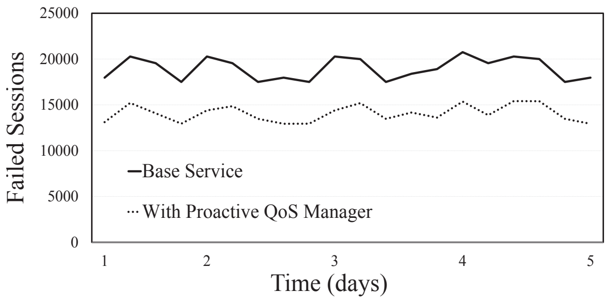

Figure 3: Failure reduction with proposed approach.

still high and costs a lot of money for a service that manages 2.5 million transactions per second at peak. That's 25,000 transactions per second at peak that potentially did not meet SLA. To provide an idea of the monetary impact alone of such a failure rate, note that the ads monetization for such a service runs at a rate higher than \$10 per a thousand ad impressions. That is, advertisers pay \$10 for every 1,000 ads shown to customers. So that's an opportunity loss of minimally \$250 *per second*, let alone the negative impact of the lower customer satisfaction, which can lead to losing customers to competitors.

Our test cluster gets about 3,000 transactions per second at peak. We ran the experiment for 5 days, and we divide the test time into two alternating parts: (1) the base or reference part, where we run the traffic over the service without the proposed solution for one hour, and (2) the updated service or optimized part, where we run the traffic over the service with the proposed solution for another hour. We aggregate the results of our tests at a granularity of 6 hours; i.e., each point in the graphs we show in this section represents a 6-hour aggregation, unless otherwise mentioned. In each 6-hour period, 3 hours have the results for the base service, and the other 3 hours have the results for the optimized service with our Proactive QoS monitoring approach.

We processed more than 100 million sessions over the 5-day period. For each period of 6 hours, the average number of multimedia sessions we get is about 5 million, half of them with our approach and the other half without it.

Number of Failures: We measure the number of failed sessions with and without our approach, and we plot the results in Figure 3. As the figure shows, the average number of failed sessions is about 19,000 failures without our approach. Using the proposed approach to monitor, estimate capacity, predict failures, and route to components with higher capacity and better performance, the failed sessions count drops to an average of 14,000. Therefore, the proposed approach manages to drop the failed sessions by about 26%, on average. The average is fairly smooth because it is measured across a large period of 3 hours. The gain from our approach is much higher during peak times, where resources are constrained.

Media Quality: We measure the media quality of the succeeded sessions. Our results show that not only did the failed session count drop by about 26%, but the quality of the sessions that meet SLA have seen an improvement as well. Figure 4 shows that, on average, around 12% of the sessions that used to meet SLA with MOS 4, are now meeting their SLA with MOS 5, Excellent Quality. We study

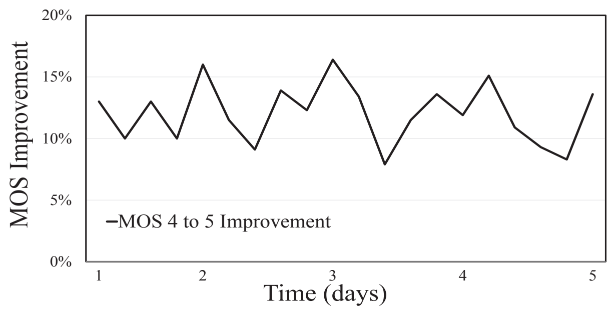

Figure 4: MOS increase from 4 to 5 over 5 days.

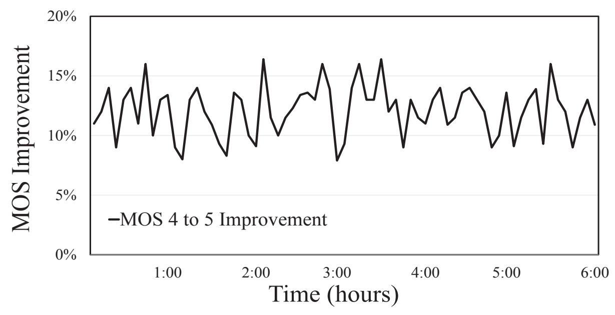

Figure 5: MOS increase from 4 to 5 over 6 hours.

the MOS improvement from 4 to 5 in more details over 12 hours, while alternating between using the service with and without our approach every hour. We average the MOS measurement every 5 minutes. We see an improvement of about the same average, around 12%, as we see in the 5 days experiment with 6 hour aggregation; please refer to Figure 5. This shows the stability and predictability of the proposed approach over different time ranges and aggregation periods.

Increase in Service Capacity: The average available service capacity has also seen an improvement. This is the number of extra multimedia sessions the service can handle. Because of the enhanced component selection, the service is seeing less bottlenecks and delays, and so is able to handle more multimedia sessions. Figure 6 shows that the service has an average of about 17% session capacity increase over the 5 day experiment, and a maximum of up to 21% increase can be achieved.

CPU Utilization: The service CPU utilization has seen a drop of an average of 10% over the 5 day experiment, as shown in Figure 7. This is closely related to the increase in service capacity. Because of the optimized component selection, the components are observing better load distribution and so less congestion. The demand on their CPUs has dropped by about 10%, which allows the components to handle more sessions, by about 17%, with the same resources.

Overhead: we measure the overhead of the TiP system on the production system to determine the cost of the proposed approach. We measure the service CPU utilization with and without TiP. We find that the average service CPU impact caused by the whole TiP system is around 3% over 5 days, as seen in Figure 8. This includes the proposed Proactive QoS Manager approach as well as all other test-

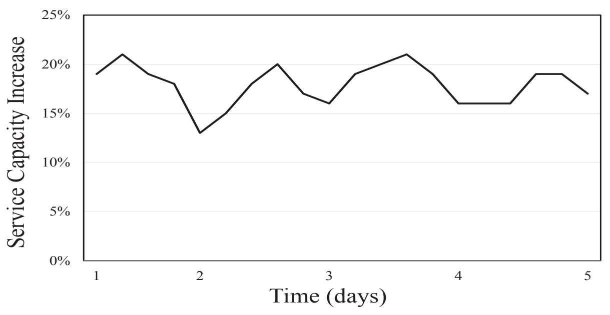

Figure 6: Increase in the available capacity.

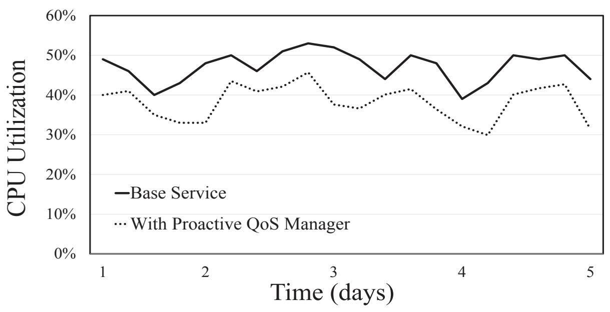

Figure 7: Reduction of CPU utilization.

ing in production functionality. All computations of our approach are done on the TiP system, not the production system. The improved multimedia quality, reduced failures, enhanced service session capacity, and reduced CPU usage are gains made by the proposed approach that make the overall investment in the TiP system more than justified.

5. CONCLUSIONS AND FUTURE WORK

Current approaches to the management of the health and quality of service in online multimedia communication services are mostly static and can not keep up with the frequent changes that happen during the real-time lifecycle of the services. Monitoring the services and waiting for real sessions to fail in order to gain insights into the state of the service is a not an efficient solution. Online services have dynamic situations that require them to change often, and the cost of failures is higher. We presented a dynamic approach to monitoring the health and quality of multimedia communication services. We use *synthetic transactions* to monitor the service by exercising it like real customers, generate current data about it, and then use this *fresh* data to create and maintain up-to-date capacity predictions of different components of the multimedia communication service. We then take component selection and routing path actions in real-time to enhance the quality of multimedia sessions. On average, the proposed approach increased the overall media sharing quality by 12%, decreased the percentage of failures by 25%, reduced the CPU usage by 10%, and increased the session capacity in the service by 17%.

A potential future work is to apply the proactive QoS management approach to other multimedia services, such as on-demand and live streaming systems.

6. REFERENCES

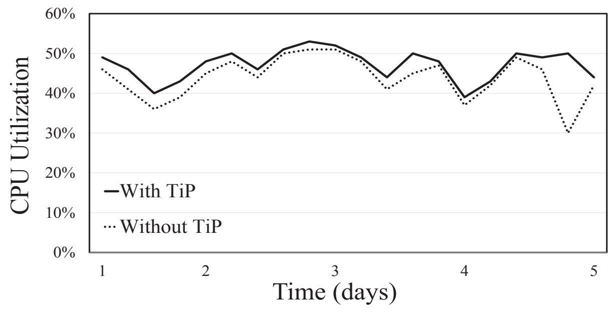

Figure 8: TiP impact on CPU utilization.

[1] Automated mean opinion score. http://voip.about.com/od/voipbasics/a/MOS.htm.

[2] IT Channel. http://www.itchannelplanet.com/.

[3] E. Elliot. Testing in production A to Z - tip methodologies, techniques, and examples. In *Proc. of Software Test Professionals (STP'12)*, New Orleans, LA, March 2012.

[4] J. Han, M. Kamber, and J. Pei. *Data Mining Concepts and Techniques*. Morgan Kaufmann, 3rd edition, 2011.

[5] Y. Ito, S. Tasaka, and Y. Fukuta. Psychometric analysis of the effect of buffering control on user-level QoS in an interactive audio-visual application. In *Proc. of ACM Multimedia Workshop on Next-generation Residential Broadband Challenges (NRBC'04)*, New York, NY, October 2004.

[6] M. Kantarczic. *Data Mining Concepts, Models, Methods and Algorithms*. Wiley and IEEE Press, 2nd edition, 2011.

[7] C. Killian, J. W. Anderson, R. Jhala, and A. Vahdat. Life, death, and the critical transition: Detecting liveness bugs in systems code. In *Proc. of USENIX Symp. on Networked Systems Design and Implementation (NSDI'07)*, pages 243–256, Cambridge, MA, April 2007.

[8] R. Kimball and M. Ross. *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*. Wiley Computer Publishing, 3rd edition, 2013.

[9] L. Lin, X. Liao, G. Tan, H. Jin, X. Yang, W. Zhang, and B. Li. Liverender: A cloud gaming system based on compressed graphics streaming. In *Proc. of ACM International Conference on Multimedia (MM'14)*, pages 347 – 356, Orlando, Florida, November 2014.

[10] Mean opinion score. https://technet.microsoft.com/en-us/library/bb894481(v=office.12).aspx.

[11] Real-time monitoring. http://techblog.netflix.com/2012/11/hystrix.html. and https://github.com/Netflix/Hystrix.

[12] K. Nagaraj, C. Killian, and J. Neville. Structured comparative analysis of systems logs to diagnose performance problems. In *Proc. of USENIX Conference on Networked Systems Design and Implementation (NSDI'12)*, pages 353–366, San Jose, CA, April 2012.

[13] Microsoft application and services group. http://www.microsoft.com.

[14] B. Rainer and C. Timmerer. Self-organized inter-destination multimedia synchronization for adaptive media streaming. In *Proc. of ACM International Conference on Multimedia (MM'14)*, pages 327–336, Orlando, FL, November 2014.

[15] P. Reynolds, C. Killian, J. L. Wiener, J. C. Mogul, M. A. Shah, and A. Vahdat. Pip: Detecting the unexpected in distributed systems. In *Proc. of Symp. on Networked Systems Design and Implementation (NSDI'06)*, pages 115–128, San Jose, CA, May 2006.

[16] L. Riungu-Kalliosaari, O. Taipale, and K. Smolander. *Testing in the Cloud: Exploring the Practice*. IEEE Software Magazine, 2012.

[17] F. Salfner, M. Lenk, and M. Malek. A survey of online failure prediction methods. In *Proc. of ACM Computing Surveys, Vol. 42, No. 3, Article 10*, March 2010.

[18] F. Salfner and S. Tschirpke. Error log processing for accurate failure prediction. In *Proc. of USENIX Workshop on Analysis of System Logs (WASL'08)*, San Diego, CA, December 2008.

[19] M. Shatnawi and M. Hefeeda. Real-time failure prediction in online services. In *Proc. of IEEE INFOCOM'15*, Hong Kong, April 2015.

[20] S. Shi, C. Hsu, K. Nahrstedt, and R. Campbell. 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)*, pages 103–112, Scottsdale, AZ, November 2011.

[21] Social networking statistics. http://www.statisticbrain.com/social-networking-statistics.

[22] S. Tasaka, H. Yoshimi, A. Hirashima, and T. Nunome. The effectiveness of a qoe-based video output scheme for audio-video ip transmission. In *Proc. of ACM International Conference on Multimedia (MM'08)*, pages 259–268, Vancouver, BC, Canada, October 2008.

[23] Testing in production. http://blogs.msdn.com/b/seliot/archive/2011/06/07/testing-in-production-tip-it-really-happens-and-that-s-a-good-thing.aspx.

[24] I. Trajkovska, J. Rodriguez, and A. Velasco. A novel p2p and cloud computing hybrid architecture for multimedia streaming with QoS cost functions. In *Proc. of International Conference on Multimedia (MM'10)*, pages 1227–1230, Firenze, Italy, October 2010.

[25] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan. Detecting large-scale system problems by mining console logs. In *Proc. of ACM Symp. on Operating Systems Principles (SOSP'09)*, pages 117–132, Big Sky, MT, October 2009.