Reboot-based Recovery of Performance Anomalies in Adaptive Bitrate Video-Streaming Services

Carlos Augusto Cunha *, Luis Moura e Silva †
Centre for Informatics and Systems of University of Coimbra
* *ccunha@dei.uc.pt*, † *luis@dei.uc.pt*

***Abstract*—Performance anomalies represent one common type of failures in Internet servers. Overcoming these failures without introducing server downtimes is of the utmost importance in video-streaming services. These services have large user abandonment costs when failures occur after users watch a significant part of a video. Reboot is the most popular and effective technique for overcoming performance anomalies but it takes several minutes from start until the server is warmed-up again to run at its full capacity. During that period, the server is unavailable or provides limited capacity to process end-users' requests. This paper presents a recovery technique for performance anomalies in HTTP Streaming services, which relies on Container-based Virtualization to implement an efficient multi-phase server reboot technique that minimizes the service downtime. The recovery process includes analysis of variance of request-response times to delimit the server warm-up period, after which the server is running at its full capacity. Experimental results show that the Virtual Container recovery process completes in 72 seconds, which contrasts with the 434 seconds required for full operating system recovery. Both recovery types generate service downtimes imperceptible to end-users.**

I. Introduction

The complexity of modern software applications and the large frequency of updates make software systems vulnerable to failures. It is known that recovery from computer failures generates large costs in organizations, representing from 30 to 50 percent of the computer system's total cost of ownership [1]. Performance anomalies are one type of failure caused by *transient faults* [2] that occur often in software systems. They are caused by errors responsible for progressive degradation of system performance, which later manifests itself as a degradation of service quality. Software aging [3] is a typical manifestation of that phenomenon.

A system reboot is a popular maintenance task in Internet servers that has the purpose of reestablishing the correctness of the system's state and behavior. It is done periodically without awareness of the server behavior, or as a result of continuous monitoring and analysis of server behavior [4][5]. Notwithstanding the effectiveness of a system reboot in mitigating performance anomalies, it is an expensive technique that can be responsible for server downtimes of several minutes.

The most relevant previous work in recovery of performance anomalies includes the Recovery Oriented Computing (ROC) research area [6]. One important contribution in this area is the recursive restarting of fine-grained components in componentized applications to minimize the service downtime caused by reboots [7][8]. Virtualization techniques have been also explored to reduce the reboot-induced downtime, using snapshots [9] and virtual machine replication [10]. As well, jumping reboot steps has shown effective in reducing the reboot time [11]. However, most previous approaches require changes in the software or in the virtualization infrastructure, influencing negatively their adoption. Also, the applicability of previous work to video-streaming services is limited. The reduction of the server downtime generated by reboot operations is insufficient to avoid impacting the Quality of Experience (QoE) of video-streaming users. Not only the reboot downtime of several seconds can be unacceptable in most streaming services but also the server warm-up period should be accounted for as recovery time. The reason is that the warm-up phase spans over a large period of time, during which the server capacity is limited and, consequently, will likely force several requests to fail.

This paper extends our previous work addressing failure prediction in video-streaming servers [12] with a two-level server recovery technique. When performance anomalies are detected, the recovery process starts by rebooting the server application. Then, in case the problem persists, a full operating system reboot is executed with the assistance of another host. Any of the reboot granularities is followed by a server warm-up period.

The recovery approach focuses on proactive rather than reactive reboots — reboots are conducted during the normal server operation. Reboots are used as self-healing techniques [13] that provide each server with a self-repair mechanism. They exploit the characteristics of *Container-based Virtualization* and *Adaptive Bitrate (ABR)* streaming to handle requests during the recovery period without exposing server downtimes to end-users.

Container-based Virtualization is a technology that involves virtualization on top of the operating system. Thus, one single operating system can be shared by several Virtual Containers (VCs) that run the server applications. Due to the small size of VCs, it is fast to reboot and migrate them between hosts.

ABR technologies [14] are becoming default technologies for delivering videos over the Internet. They allow fragmentation of videos into small segments, each one requested independently. Hence, players can dynamically switch between segments with different video qualities, whenever the network and player conditions change — e.g., reduction of the available network bandwidth. The small size of video segments and the large frequency of player requests during video playback

create the opportunity to:

- Reboot VCs without the need of rescuing the state of the server and client-server connections;
- Perform progressive protocol-level redirection of requests to warm-up the secondary server during recovery;
- Exploit the variance of request-response delays (difference between the time the client transmits the request until it receives the response) for delimiting the server warm-up period.

The analysis of the VC infrastructure and the experimental evaluation of our approach revealed interesting findings:

- The VC infrastructure provides statistics that allow selection of hosts with available resources to assist recovery when a full operating system reboot is required;
- VCs restart the server application approximately 3 times faster than a restart of the server application process;
- The reboot-based recovery approach proposed has negligible server downtimes;
- Analysis of variance of request-response delays is effective in delimiting the server warm-up period.

The rest of this paper is structured as follows. Section II presents the related work. Section III defines the problem. Section IV presents the background of the main concepts related to our research. Section V explains our recovery approach. Section VI and Section VII reveal the experimental methodology and experimental results, respectively. Section VIII presents the conclusions.

II. Related Work

This paper addresses the problem of recovering HTTP Adaptive Bitrate video servers from performance anomalies. Reboot has been the most widely used technique for recovering servers from failures caused by performance anomalies at several granularity levels.

Recovery Oriented Computing (ROC) [6] is an important research area that has inspired several recovery techniques using the reduction of the Mean Time to Repair (MTTR) as a strategy for increasing software availability. Recursive Restartability [15][7] is a ROC recovery approach based on the design of systems to gracefully tolerate successive restarts at multiple granularity levels. It allows strong fault containment because dependencies between components are organized into a hierarchy of restartable components, in which nodes are highly failure-isolated. When one tree node (component) is restarted, the entire subtree rooted at that node (dependent components) is restarted with. Microrebooting [16] has also been explored in componentized applications to mitigate rebooting costs. This technique separates process recovery from data recovery in individual application components to reboot them at anytime.

Virtual machines (VMs) have been successfully explored for reduction of reboot downtimes. Phased-based reboot is presented in [9] as a technique for accelerating system reboots when recovering from kernel transient failures. It uses VM snapshots to restore the system to a restartable point, corresponding to a clean and consistent system state. Phase-based reboot uses the VM snapshot mechanism implemented by Xen, but enhanced to: (1) reduce the snapshot restoration time by avoiding saving unused memory pages; and (2) avoid inconsistency in files opened during the snapshot restoration process. Experimental results show that this technique reduces the downtime generated by operating system reboots considerably. However, a later study evidences that the phased-based reboot technique reduces significantly the response time of the system, adds CPU and I/O overheads at domU and also increases the memory consumed at dom0 [17]. A virtualization approach to reduce the reboot downtime caused by software upgrades is also presented in [10]. When an upgrade is necessary, an additional VM is started to install the updates and reboot the system. When the reboot process completes, the rebooted VM replaces the original VM. The approach requires changes in the VM infrastructure to control updates during the reboot process to working directories, on the reboot-dedicated VM, and to administrative directories, on the original VM.

Other approaches have been proposed to make the reboot process more efficient. One strategy is jumping stages during the reboot process. Kexec [11] patches the operating system kernel at runtime to perform a fast warm reboot by refreshing the OS internal state without reinitializing the hardware. By skipping the BIOS and the device check phases of the Linux boot process, the reboot time decreases significantly. Otherworld [18] addresses isolation of the kernel termination from the user-level applications. Thus, when a kernel failure occurs, only the OS kernel is restarted, keeping the user-level memory states of the processes. User-level processes are resumed after the OS kernel has been rebooted. This approach rescues application states but the reboot downtimes are not significantly lower than in typical virtualization approaches.

Several approaches have been proposed in previous work to reduce the server downtime generated by reboots. The most promising approaches are those based on virtualization, notwithstanding they require changes in the virtualization infrastructure and create overheads due to duplication of VMs and maintenance of snapshots. This paper explores the particularities of Container-based Virtualization to develop a method that tackles the limitations of previous approaches to: (1) recover HTTP video-streaming servers with inconsequential user-visible downtimes and small overheads; (2) adopt standard virtualization infrastructures; and (3) embrace the server warm-up time period as part of the recovery process.

III. Problem Statement

Despite some previous work having addressed the problem of repairing applications at runtime by changing them dynamically [19][20][21], none of them is generic, provides full recovery of applications and guarantees application correctness after recovery. By contrast, reboot-based techniques have shown a singular ability for overcoming transient failures.

Our work addresses the problem of minimizing the server downtime when performing reboot-based recovery. Accordingly, we define the following requirements for the reboot

approach:

- Client requests should be served during the rebooting process;
- The least disruptive reboot granularity (application or operating system) should be chosen to minimize the reboot time;
- The server should be appropriately warmed-up to run at its full capacity by the end of the recovery period.

Reboots should be performed with the appropriate granularity to avoid ineffective recovery actions or recovery costs larger than required. An operating system reboot is expensive and its cost is proportional to the server downtime caused by the execution of several activities: stopping all services, restarting the operating system, starting the services again and warming-up the server. However, reboots can be performed at a finer granularity than the operating system. Often, errors accrue at the server application level and, in many cases, the kernel goes back to a consistent and clean state simply by killing and revoking the resources of the faulting process [22]. Thus, whenever it is possible, only the server application should be rebooted, since it presents significantly smaller costs.

IV. Background

Our work uses two technologies that are described in this section: Container-based Virtualization an Adaptive Bitrate Streaming.

A. Container-based Virtualization

Typical Hypervisor Virtualization requires installation of one dedicated operating system instance for each Virtual Machine. By contrast, Container-based Virtualization uses one single operating system instance for all Virtual Containers running in the same host. VCs are user-space instances managed by a virtualization layer placed in the operating system kernel. Virtualization ensures isolation of processes running within VCs and fair scheduling on utilization of resources between VCs. Each VC holds its own set of processes, a private memory address space, logical file system and virtual network interface.

VCs have negligible overheads, since the virtualization layer is installed at the operating system level. The CPU overhead is less than 1% when compared to non-virtualized environments [23]. The efficiency of VCs is justified by the use of the standard system call interface provided by the operating system, without relying on emulation. For recovery purposes, VCs can be efficiently rebooted or migrated between hosts, since they have simple constructions that exclude operating system structures.

VCs can be used either as an alternative to typical VMs or can be combined with them. In the latter case, VCs can run inside VMs to isolate the performance of several applications and to support migration of applications between VMs with small performance penalties. This configuration is adequate for typical cloud infrastructures.

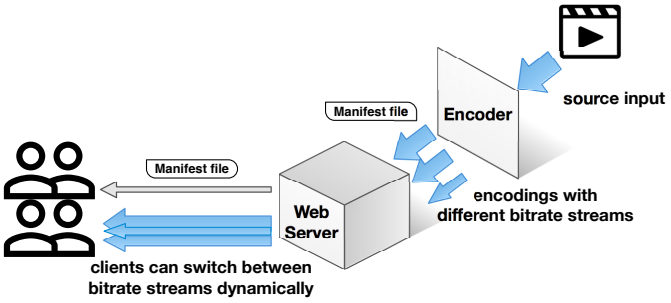

Fig. 1: Adaptive Bitrate streaming workflow.

B. Adaptive Bitrate Streaming

The simplicity of HTTP has come at the cost of lack of control over the content downloaded in video-streaming services. Consequently, the download process is inefficient — streaming users that abandon watching videos or seek another playback position in time can download significantly more content than that effectively played — and unadaptable to specific network conditions and client-side resources (e.g., the network bandwidth can be insufficient to maintain the player buffer with enough data for playback). These problems are tackled by Adaptive Bitrate streaming (ABR), which integrates the benefits of HTTP with the efficiency and adaptation features of dedicated video-streaming protocols.

ABR is a technique used to deliver video-streaming to end-users with different bandwidths and system resources (e.g., CPU and screen size). ABR videos are encoded at several different bitrate streams, each segmented into small multi-second segments. Therefore, streaming clients switch between different bitrate streams dynamically, according to their available local resources and network bandwidth (Fig. 1). Accordingly, before requesting the next video segment during playback, the player decides the corresponding bitrate with the support of a list of available bitrates defined by the *video's manifest file* provided by the server.

ABR has several implementations. MPEG-DASH [24] is the only ABR solution that is an international standard. Other commercial implementations are Microsoft Smooth Streaming, Adobe Dynamic Streaming and Apple HTTP Adaptive Streaming [14].

V. Recovery Approach

Our recovery approach performs reboot of servers at two granularity levels: (1) Virtual Container and (2) Operating System. We design a two-phase reboot strategy (Fig. 2), which starts by rebooting the faulty VC. Then, if the faulty behavior persists after the reboot, a full operating system reboot is performed afterwards. This multi-phase reboot strategy aims to reduce recovery costs by avoiding full reboots when errors are confined to specific VCs. A VC reboot offers several advantages over a operating system reboot:

- It is less expensive, since it has smaller reboot delays;
- It reduces the server warm-up period significantly, since the kernel in-memory structures are isolated from the VC

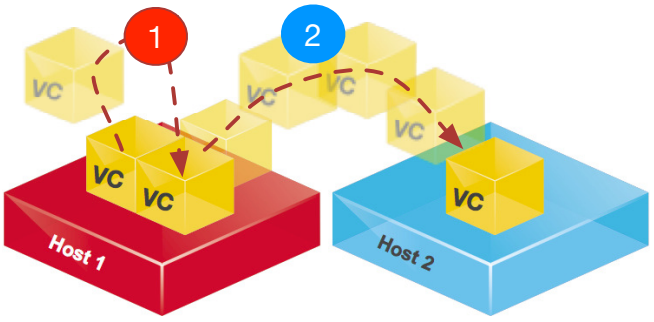

Fig. 2: Two-phase reboot process.

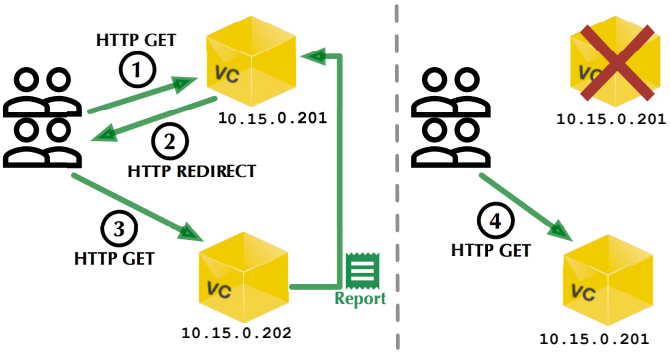

Fig. 3: Progressive migration of requests to warm-up the server running in the secondary VC.

reboot process;

- It can be performed without additional resources provided by the actual host or other hosts selected to assist the reboot process.

We adopt the OpenVZ [25] implementation of Container-based Virtualization in our recovery approach. OpenVZ is open-source, well documented and provides a rich set of features for handling and managing Virtual Containers.

A. VC Reboots

The VC reboot process restarts the VC along with all internal processes. Despite being more efficient than a operating system reboot, the efficiency of a VC reboot would be improved by replacing the VC by a replica of it, snapshotted after a previous reboot. We adopt this reboot strategy for recovery of faulty VCs. We refer to the faulty VC as *primary VC* and to the VC replica that will replace the *primary VC* as the *secondary VC*.

Our VC reboot process starts by creating the secondary VC in the primary VC's host, with its own IP address. Afterwards, the web server running in the primary VC redirects part of the requests to the secondary VC, using the HTTP REDIRECT method (Fig. 3). The number of requests redirected grows progressively to warm-up the server running in the secondary VC. Finally, the primary VC is destroyed and the secondary VC becomes the primary VC, by taking its IP address.

B. Operating System Reboot

When the server faulty behavior persists after reboot the VC, a full operating system reboot is started. This process is accompanied by the instantiation of a VC replica into another host to handle the server load during the rebooting process. We use the term *primary host* to refer to the faulty host and *secondary host* to refer to an alternative host that assists temporarily the recovery process of the *primary host*.

The operating system reboot process has two phases. In the first phase, a replica of the primary VC is created into another host with resources available for running the VC. Then, the requests are redirected progressively until the secondary VC is warmed-up and takes the IP of the primary VC, similarly to a VC reboot. Finally, a full operating system reboot is executed. The second phase initiates when the primary host finishes the operating system reboot process. Then, the entire process followed in the first phase is executed again to move the server back to the primary host.

C. Selection of the Secondary Host

Operating system reboots require the assistance of other hosts to receive the streaming server during the reboot period. Any host with available resources to handle the load of the faulty VC can be used to that end. Thus, the secondary host can be either one passive machine specifically dedicated to assist the recovery of other hosts or any of the active machines with available resources.

Container-based Virtualization infrastructures provide statistics about utilization of system resources for each VC. These statistics are useful to select one of the active hosts as the secondary host. Host selection is performed through comparison of resource utilization statistics provided for the primary VC with equivalent statistics provided for VCs running in other hosts. In other words, any host running VCs with idle resources sufficient to afford the consumption of resources by the primary VC can assist its recovery.

OpenVZ measures the utilization and capacity of resources for each VC using *beancounters*. *Beancounters* represent the units of utilization of resources and are presented as such:

```
vzctl exec 101 cat /proc/user_beancounters

uid  resource          held    maxheld    barrier      limit
101: kmemsize        803866    1246758    2457600    2621440
     lockedpages          0          0         32         32
     privvmpages       5611       7709      22528      24576
     shmpages            39        695       8192       8192

          [...]
```

The *held* column represents the current resource utilization, the *maxheld* the maximum utilization since the last VC reboot and the *barrier* and *limit* represents the capacity of the given resource — the distinction between the last two metrics is specific to each resource.

The OpenVZ statistics provided for each VC enable the selection of eligible secondary hosts based on resources not consumed by their VCs. One straightforward method for selection of the secondary host is to find a host where the *maxheld* value of all resources in the primary VC fits the

available resources of one VC running in the secondary host, as formulated in (1). Thus, a replica of the primary VC can be instantiated in the secondary host to assist the reboot of the primary host.

$$maxheld_{primary} < limit_{secondary} - maxheld_{secondary} \quad (1)$$

The host selection process depends on a data sharing mechanism that provides the faulty servers with resource utilization statistics of the other hosts. This mechanism can be implemented by a reporting service accessible to all hosts to report resource utilization statistics periodically. The design of this service is out of the scope of this paper.

D. Detection of Performance Anomalies

Performance anomalies are abnormal server conditions that can lead, sooner or later, to user-visible failures. Thus, one server experiencing performance anomalies may not lead necessarily to immediate degradation of service quality perceptible by end-users. As an example, one memory leak allocating more 50% of memory than normal may not impact the service quality in the short-term.

Detection of performance anomalies represents a complex process. It requires knowledge about the normal server behavior in order to recognize any deviation hinting a potential failure. In [12], we present SHStream, a self-healing framework for HTTP servers that implements prediction of performance failures by detecting anomalous server states. SHStream uses machine learning algorithms to create and evaluate models of the normal server behavior automatically and iteratively. This framework can be integrated in our work to perform detection of performance anomalies.

E. Detection of Service Failures

We perform server-side failure detection to assume the use of standard video players. However, client-side factors like the amount of data buffered by players would compensate the increased server-side request processing delays. Therefore, we are unable to observe, on the server-side, the impact of server performance degradation on the user experience. Yet, as long as the main concern of our work is to ensure that the server is providing the service correctly, we evaluate the service quality provided by the server instead of measuring the quality of user experience.

One healthy server should ensure that each video segment is transmitted to the player before the previous segment has completed its playback. Accordingly, we consider that the player requests each video segment V_{i+1} at latest when the previous video segment V_i starts playback. Thus, it is a fair assumption to consider that the server is failing to provide the service correctly when the *gap* defined as in (2) is negative.

$$gap = 10 - (R_{rec}(V_{i+1}) - R_{send}(V_{i+1})) \quad (2)$$

$R_{rec}(V_{i+1})$ represents the reception time of the next video segment and $R_{send}(V_{i+1})$ the transmission time of the request of the same video segment. We assume video segments with duration of 10 seconds and that the player issues the request for a specific video segment when it starts the playback of the previous one (worst scenario). As a deduction, the player will receive the data for the requested segment after its playback time when the gap is negative.

Video segments with time lengths of 10 seconds are typical in ABR services. Smaller segment sizes have a disadvantage because of the: (1) high number of I-frames, demanding more bits in the overall bitstream [26]; and (2) small Groups of Pictures (GOP), providing a lower encoding performance and quality [27]. On the other hand, video segments larger than 10 seconds increase the adaptation time unnecessarily. However, our approach can be used with video segments of other sizes.

F. Delimitation of the Server Warm-up Period

The server warm-up period should be respected to avoid failures caused by the transition from the primary VC to the secondary VC. We analyze the request-response delays to determine when the secondary server is ready to replace the primary server. Figure 4 relates the running standard deviation [28] of request-response delays with the number of video requests handled by the server after the reboot. These values are the result of experimental tests ran using different workload levels and video configurations. It is noticeable a pattern of large standard deviations of the request-response delay during the server warm-up period. The variability of these values is dictated by the large delays of some request-responses.

Request-responses with large delays can be responsible for user-visible failures during the server warm-up period. Fig. 5 shows the gap between the length of each video segment streamed (10 seconds) and its download delay, calculated as in (2). It exposes several requests with negative gaps during the server warm-up period, representing potential failures experienced by streaming users.

It is noticeable that the variability of request-response delays stabilizes after several requests have been handled by the server. We use that characteristic to determine the end of the server warm-up period. Accordingly, the server warm-up period is delimited through analysis of variance of request-response delays.

We use the Kruskal-Wallis method to evaluate if groups of n samples of request-response delays belong to the same statistical distribution. It it is an efficient non-parametric method (does not assume that the data are normally distributed) used to test whether several groups of samples originate from the same distribution. The test statistic is given by (3), being n_i the number of observations of group i, r_{ij} the rank of the observation j from group i, $\bar{r}_i$ and $\bar{r}$ calculated as in (4) and N the total number of observations.

$$K = (N-1)\frac{\sum_{i=1}^{g} n_i(\bar{r}_i - \bar{r})^2}{\sum_{i=1}^{g}\sum_{j=1}^{n_i}(r_{ij} - \bar{r})^2} \quad (3)$$

$$\bar{r}_i = \frac{\sum_{j=1}^{n_i} r_{ij}}{n_i} \qquad \bar{r} = \frac{1}{2}(N+1) \quad (4)$$

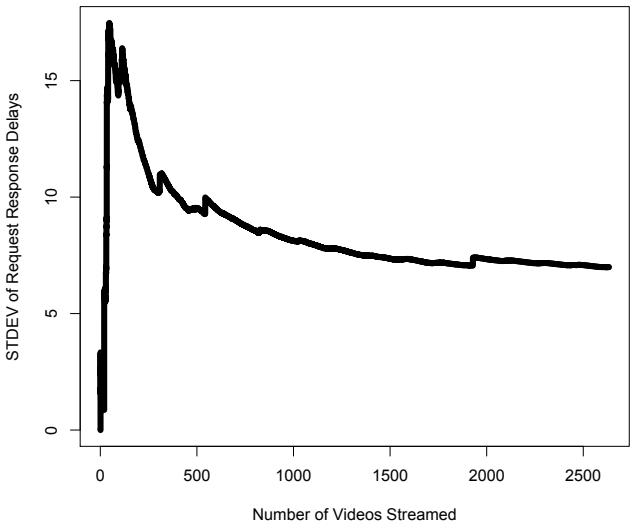

Fig. 4: Relation between the standard deviation of request-response delays and the number of complete videos streamed after the reboot.

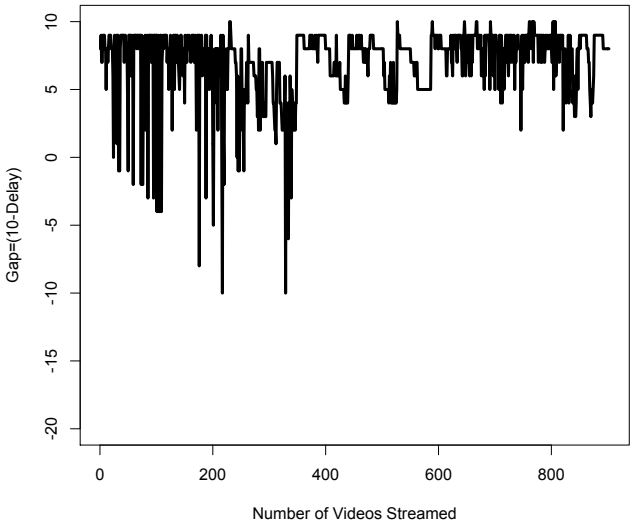

Fig. 5: Difference between the length of video segments (10 seconds) and each request-response delay, after the reboot.

The evaluation of our server warm-up approach depends on the observation that the Kruskal-Wallis test rejects the null hypothesis that statistical distributions of groups of request-response delays are similar during and after the server warm-up period.

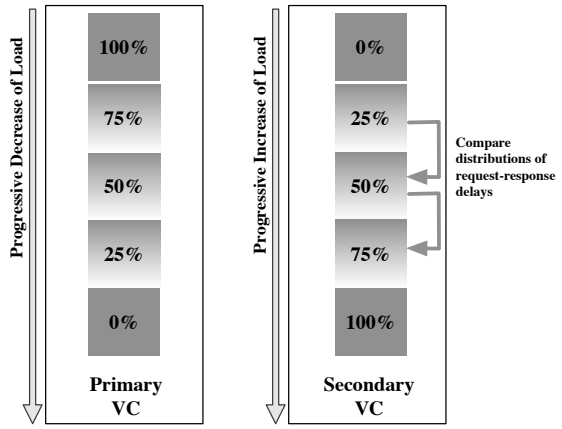

Fig. 6: Distribution of load between the primary VC and secondary VC and comparison between the statistical distributions of request-response delays of different server load levels.

G. Server Warm-up Process

During recovery, the primary server redirects the load progressively to warm-up the secondary server. At each warm-up stage, the secondary server receives more $L\%$ of the primary server's requests (randomly selected) and the resulting load level is only increased again after stabilization of the request-response delays. This incremental process intends to avoid transient failures during the server warm-up period and to warm-up the secondary server with a realistic workload taken from the primary server.

Fig. 6 shows the distribution of the load between the primary and secondary servers along time, during the warm-up period of the secondary server. The primary server controls the warm-up process by redirecting requests and deciding when to increase the number of redirected requests. That decision is based on the analysis of server request-response delays gathered from the logs of the secondary server.

The Kruskal-Wallis test is used to compare each group of request-response delays gathered during the server warm-up for the current server load, with a group of request-response delays belonging to a lower server load level already validated by the warm-up process. When the Kruskal-Wallis test does not reject the null hypothesis that the distributions of both groups of request-response delays are similar, the server is considered warmed-up for that load level. Then, the server load is increased in the secondary server and the same process repeats.

When the primary server is redirecting requests corresponding to $25\%(N/4)$ of its load to the secondary server, the request-response delays of the lower server load level are unavailable for comparison. During that period, the Kruskal-Wallis test is applied to successive groups of request- response delays of the secondary server for that load level. We use this technique based on the hypothesis that the distribution of groups of successive request-response delays are different during the warm-up phase.

The server warm-up finishes when more than $3N/4$ of the

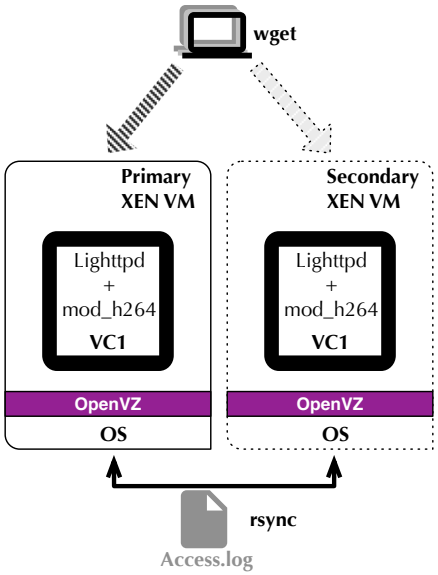

Fig. 7: Experimental testbed.

requests are being redirected to the secondary server or when 100% of the primary server's load is being redirected to the secondary server (when the number of requests received by the primary server is less than $3N/4$ of its maximum capacity).

We store the *access log* of the secondary server in a folder shared between both VCs, to provide data for analysis during the reboot of VCs (Fig. 3). For operating system reboots, the content of the *access log* of the secondary server is synchronized between hosts using *rsync* [29], at each t seconds.

VI. Experimental Work

This section presents the experimental design followed for evaluation of our recovery approach.

A. Testbed

The experimental testbed is composed by four XEN virtual machines connected by a 1Gbps Ethernet Network (Fig. 7). Two of these virtual machines are used for workload generation using a Bash script that issues requests using the *wget* tool [30]. The other virtual machines contain the VCs that run the web servers. Each of these machines has 2 CPUs @ 2.00GHz, 4GB Memory and runs the Linux CentOS version $2.6.32 - 358.6.1.el6.x86\_64$. The two server machines have installed the:

- *OpenVZ* infrastructure for Container-based Virtualization;
- *Lighttpd web server 1.4.30*. This web server have been used by Youtube for several years to deliver videos [31];
- *H264 Streaming Module* (mod_h264_streaming version 2.2.7) for Lighttpd. This module performs bandwidth control and supports Adaptive Bitrate streaming by handling requests for specific video segments.

B. Workloads

Typical video-streaming workloads are characterized by the *number of connections*, *video encoding bitrates*, *duration of videos*, *inter-arrival request times*, and *user abandonment of videos in the middle of playback*.

The workload adopted for experimental evaluation has 50 H.264 videos, encoded with bitrates of approximately 300 Kbps, 900 Kbps and 2 Mbps, providing a bitrate amplitude similar to that encountered in typical workloads [32][33]. The number of connections varies sinusoidally between 0 and n, being n the server capacity.

According to [34][32], approximately 90% of Internet videos have a duration between 10 seconds and 16 minutes. We adopt a similar configuration using videos with durations ranging from 10 seconds to 1000 seconds.

Request inter-arrival times have been modeled by a Poisson process [35] for several years for web workloads in general [36][37] and specifically for video-streaming workloads [38][39][40]. Similarly, our workload represents the request inter-arrival times by a Poisson distribution.

Workload studies providing user abandonment statistics are limited in number. In the Youtube service, approximately one quarter of the transactions run until the end [41]. However, to represent user abandonment of visualization of videos, we use the statistics provided by a recent study [33]. Accordingly, we force randomly 40% of the videos to terminate at 10% of playback time, plus 20% of videos at 20% of playback time and finally, plus 30% at 50% of playback time.

C. Evaluation Metrics

Our recovery approach is evaluated for its efficacy and efficiency. Thus, the experimental work should provide the:

- Total recovering time;
- Time required to instantiate the secondary VC in the same or in another host;
- Number of failed requests served by the secondary host. This value represents the efficacy of the: (1) recovery approach in avoiding failures caused by the recovery process; and (2) analysis of variance of request-response times in determining the server warm-up period;
- Time required by the secondary VC to take the IP of the primary VC. This metric represents the service downtime.

Despite not impacting the service directly, the total recovering time has a direct impact on the risk of failure. Typically, performance anomalies will lead to failure conditions experienced by streaming users and increase their severity in shorter or longer periods. Consequently, the occurrence of hard failures during the recovery process would avoid completing the warm-up phase in the secondary VC. The reason is that the secondary VC is forced to take the IP of the primary VC and handle its entire load. So, to minimize the risk of hard failures, the recovery time should be minimized as well.

The number of failed requests is equivalent to the number of requests with negative gaps, defined as in (2). These requests represent potential interruptions of video playback, seeing that

the gap represents the time distance between the reception of one video segment and its playback time, in the worst-case scenario. Therefore, to avoid failures, the downtime created by the recovery process should be smaller than the gap of each video segment experienced without recovery. However, since the gap varies for each request, we consider enough to have service downtimes significantly smaller than the time length of segments (10 seconds) to reduce the chance of being larger than the gaps. The service downtime is the time required by the secondary VC to take the IP of the primary VC — and vice-versa in the case of operating system reboots — considering that the other recovery-related operations are performed while the primary server is running.

VII. Experimental Results

This section presents the experimental results obtained for the evaluation of our recovery approach.

A. Comparison between Reboot Techniques

We studied three techniques to reboot the server application: (1) terminate the respective process; (2) restart the respective VC; and (3) replace the respective VC by a fresh VC in the same host. We run the tests 50 times using the workload presented in Section VI-B to have statistical significance.

Table I presents the time required for executing each reboot technique. It shows that the technique (3) is the most efficient — it restarts the server 50% of times in less than 1 second. This observation explains the selection of this technique to restart the server application in our recovery approach.

	Percentiles in Seconds		
Technique	**5th**	**50th**	**95th**
Restart the video-streaming application process	2.4	3.1	3.7
Restart the *virtual container*	10.2	11.5	12.3
Start a fresh replica of the *virtual container*	0.8	0.9	1.4

TABLE I: Downtime generated by reboot techniques

B. Server Warm-up Time

Fig. 8 and Fig. 9 show the p-values of the Kruskal-Wallis test using groups of 20 samples, segmented by the proportion of requests redirected to the secondary server. The primary server increases the number of redirections by $N/4$ when the p-value does not rejects the null hypothesis that the distributions of groups of request-response delays belonging to the current server load and to the server load immediately below it (already validated by the warm-up process) have similar statistical distributions. Assuming a significance level of 95%, the null hypothesis is rejected for p-values lower than 0.05. Accordingly, the load of the secondary server is increased when the p-value increases above 0.05.

Fig. 8 shows that the server warm-up time is approximately 178 seconds for a operating system reboot. This value contrasts with the 70 seconds of server warm-up time (Fig. 9) required for a VC reboot. The smaller warm-up times observed for VC reboots are expected, since the VC state are renewed in the same host, preserving the kernel structures and caches.

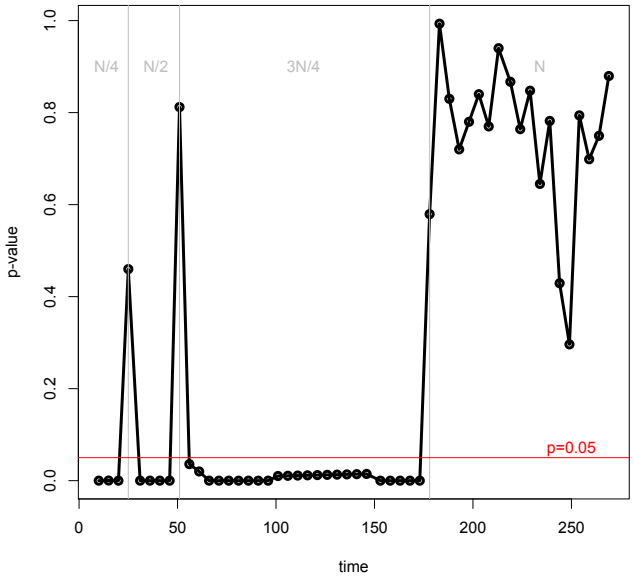

Fig. 8: p-values of the Kruskal-Wallis test for the operating system reboot. N represents the server capacity.

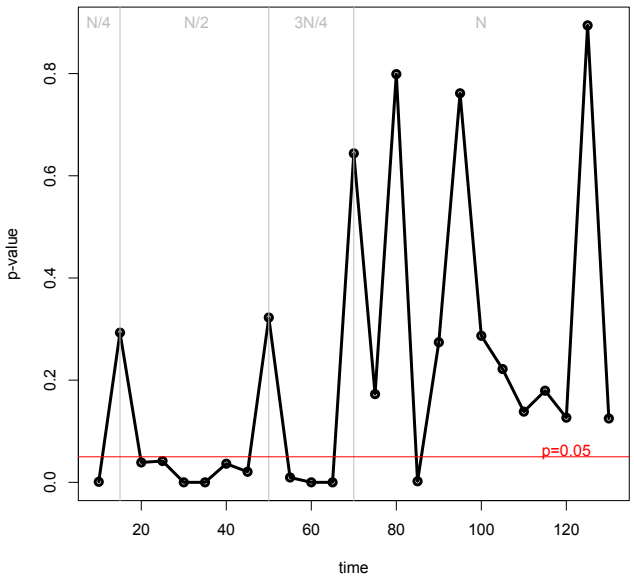

Fig. 9: p-values of the Kruskal-Wallis test for the VC reboot. N represents the server capacity.

All requests handled during the operating system and VC reboot activities have positive gap values. That means that all requests were processed without failures.

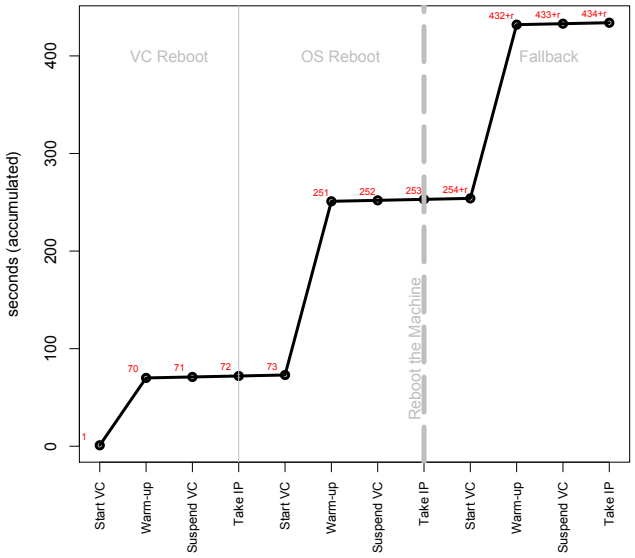

Fig. 10: Cumulative time required to recover the server.

C. Recovery Time and Downtime

Fig. 10 presents the time required to execute each recovery step. It is required 72 seconds to recover the VC and, if the failure condition persists, 253 seconds to continue the service into the secondary host after rebooting the VC. The entire reboot lifecycle is completed after 434 seconds, plus the time required to perform a operating system reboot in the primary host. The service downtime, represented by the time required by the secondary VC to take the IP of the primary VC, is less than 1 second (rounded to 1 second).

D. Discussion of Results

Experimental results have shown that our reboot-based approach for recovery of performance anomalies in video servers can be executed with user-visible downtime smaller than 1 second. The downtime represents the time required by the secondary VC to take the IP of the primary VC. Downtimes of that order are negligible, considering that, in the worse scenario, the request has to be downloaded in less than 10 seconds to avoid playback interruptions (one order of magnitude higher than the expected downtime). Similarly, the delay added by the redirection process during the warm-up period — forcing the player to issue each request twice — is not significant. It is known that the round-trip time of one typical request in the Internet is in the order of milliseconds.

Our recovery approach assumes that performance anomalies are not the cause of service failures before and during the recovery process. That means that during 72 seconds for VC recovery and 434 seconds for operating system recovery (in the worst-case scenario, when the server is running at full capacity), the primary server should be able to handle the requests not redirected to the secondary server. This is an important assumption because most of the time spent by the recovery process is spent warming-up the server. Otherwise, replacing the primary VC by the secondary VC without warming-up the server, when the primary server is experiencing severe failures, can impact less the service quality than maintaining the primary server active during the warm-up period. On the other hand, by redirecting part of the primary server's load to the secondary server during the warm-up phase, the service quality provided by the primary server could return to normality — e.g., the utilization of an exhausted resource can decrease below its limit. For the aforementioned reasons, the server warm-up process should be tried before replacing the primary VC by the secondary VC. Then, if failures still occur during the warm-up phase, the secondary VC replaces the primary VC immediately.

VIII. Conclusion

This paper addresses the problem of recovering HTTP streaming servers from performance anomalies. Recovery is performed by means of rebooting the server and, if necessary, the operating system. The main challenge of this process involves implementing the recovery process with negligible service downtimes to avoid the large costs attached to abandonment of video-streaming users.

We use Container-based Virtualization technology to: (1) instantiate the rebooted server replica that will replace the faulty server, when an operating system reboot is not necessary; and (2) ensure service continuity when an operating system reboot is required, by transferring the server temporarily to another host with enough resources. We also use analysis of variance of request-response times to determine when the server is properly warmed-up after a reboot.

The experimental evaluation of our approach has shown that through analysis of variance of request-response times, it is possible to delimit the server warm-up period during the recovery cycle. It also showed that our approach requires 72 seconds to recover the server by rebooting the VC, which contrasts with the 434 seconds required for operating system recovery. Most of that recovery time is spent warming-up the server. During the recovery period the server continues serving requests, until being replaced by the rebooted replica with negligible service downtimes (less than 1 second).

Despite our approach has been tested with ABR streaming services, we believe it can be applied to other HTTP-based services. However, the characteristics of these services may present several challenges. For example, the server downtime and redirection of connections can impact the service quality in interactive applications. As well, the size of each request can challenge the use of analysis of variance of response times for delimiting the server warm-up period.

As future work, we plan to design and evaluate a service that shares information about resource beancounters to support the selection of secondary hosts during the execution of operating system recovery actions.

Acknowledgment

This work was partially supported by FCT-Portugal under grant SFRH/BD/35784/2007 and CISUC (Centre for Informatics and Sys-

tems of University of Coimbra).

REFERENCES

[1] D. A. Patterson, "Recovery oriented computing: A new research agenda for a new century." in *HPCA*, 2002, p. 247.

[2] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *Dependable and Secure Computing, IEEE Transactions on*, vol. 1, no. 1, pp. 11–33, Jan 2004.

[3] M. Grottke, R. Matias, and K. Trivedi, "The fundamentals of software aging," in *Software Reliability Engineering Workshops, 2008. ISSRE Wksp 2008. IEEE International Conference on*, Nov 2008, pp. 1–6.

[4] L. Li, K. Vaidyanathan, and K. Trivedi, "An approach for estimation of software aging in a web server," in *Empirical Software Engineering. Proceedings of the International Symposium on*, 2002, pp. 91 – 100.

[5] M. Grottke, L. Li, K. Vaidyanathan, and K. Trivedi, "Analysis of software aging in a web server," *Reliability, IEEE Transactions on*, vol. 55, no. 3, pp. 411 –420, sept. 2006.

[6] D. Patterson, A. Brown, P. Broadwell, G. Candea, M. Chen, J. Cutler, P. Enriquez, A. Fox, E. Kiciman, M. Merzbacher, D. Oppenheimer, N. Sastry, W. Tetzlaff, J. Traupman, and N. Treuhaft, "Recovery oriented computing (roc): Motivation, definition, techniques,," Berkeley, CA, USA, Tech. Rep., 2002.

[7] G. Candea and A. Fox, "Recursive restartability: turning the reboot sledgehammer into a scalpel," *Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on*, pp. 125–130, May 2001.

[8] G. Candea, E. Kiciman, S. Kawamoto, and A. Fox, "Autonomous recovery in componentized internet applications," *Cluster Computing*, vol. 9, no. 2, pp. 175–190, 2006. [Online]. Available: http://dx.doi.org/10.1007/s10586-006-7562-4

[9] K. Yamakita, H. Yamada, and K. Kono, "Phase-based reboot: Reusing operating system execution phases for cheap reboot-based recovery," in *Dependable Systems Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*, June 2011, pp. 169–180.

[10] H. Yamada and K. Kono, "Traveling forward in time to newer operating systems using shadowreboot," in *Proceedings of the 9th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, ser. VEE '13. New York, NY, USA: ACM, 2013, pp. 121–130. [Online]. Available: http://doi.acm.org/10.1145/2451512.2451536

[11] V. Goyal, E. W. Biederman, H. Nellitheertha *et al.*, "Kdump, a kexec based kernel crash dumping mechanism," in *Proceedings of the Linux Symposium, Ottawa*, 2005.

[12] C. Cunha and L. Moura e Silva, "Shstream: Self-healing framework for http video-streaming," in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, May 2013, pp. 514–521.

[13] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[14] T. Stockhammer, "Dynamic adaptive streaming over http –: standards and design principles," in *Proceedings of the second annual ACM conference on Multimedia systems*, ser. MMSys '11. New York, NY, USA: ACM, 2011, pp. 133–144.

[15] G. Candea and A. Fox, "Designing for high availability and measurability," in *Proceedings of the 1st Workshop on Evaluating and Architecting System Dependability*, 2001.

[16] G. Candea, S. Kawamoto, Y. Fujiki, G. Friedman, and A. Fox, "Microreboot-a technique for cheap recovery." in *OSDI*, vol. 4, 2004, pp. 31–44.

[17] A. Bovenzi, J. Alonso, H. Yamada, S. Russo, and K. Trivedi, "Towards fast os rejuvenation: An experimental evaluation of fast os reboot techniques," in *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*, Nov 2013, pp. 61–70.

[18] A. Depoutovitch and M. Stumm, "Otherworld: Giving applications a chance to survive os kernel crashes," in *Proceedings of the 5th European Conference on Computer Systems*, ser. EuroSys '10. New York, NY, USA: ACM, 2010, pp. 181–194.

[19] M. Fuad, D. Deb, and M. Oudshoorn, "Adding self-healing capabilities into legacy object oriented application," in *Autonomic and Autonomous Systems, 2006. ICAS '06. 2006 International Conference on*, July 2006, pp. 51–51.

[20] A. Carzaniga, A. Gorla, and M. Pezz, "Healing web applications through automatic workarounds," *International Journal on Software Tools for Technology Transfer*, vol. 10, no. 6, pp. 493–502, 2008. [Online]. Available: http://dx.doi.org/10.1007/s10009-008-0088-8

[21] G. Portokalidis and A. Keromytis, "Reassure: A self-contained mechanism for healing software using rescue points," in *Advances in Information and Computer Security*, ser. Lecture Notes in Computer Science, T. Iwata and M. Nishigaki, Eds. Springer Berlin Heidelberg, 2011, vol. 7038, pp. 16–32.

[22] T. Yoshimura, H. Yamada, and K. Kono, "Can linux be rejuvenated without reboots?" in *Software Aging and Rejuvenation (WoSAR), 2011 IEEE Third International Workshop on*, Nov 2011, pp. 50–55.

[23] P. Padala, X. Zhu, Z. Wang, S. Singhal, K. Shin *et al.*, "Performance evaluation of virtualization technologies for server consolidation," *HP Laboratories Technical Report*, 2007.

[24] I. Sodagar, "The mpeg-dash standard for multimedia streaming over the internet," *Multimedia, IEEE*, vol. 18, no. 4, pp. 62 –67, april 2011.

[25] (2012, Apr.) Openvz. http://wiki.openvz.org/Main_Page.

[26] S. Lederer, C. Müller, and C. Timmerer, "Dynamic adaptive streaming over http dataset," in *Proceedings of the 3rd Multimedia Systems Conference*, ser. MMSys '12. New York, NY, USA: ACM, 2012, pp. 89–94. [Online]. Available: http://doi.acm.org/10.1145/2155555.2155570

[27] E. Schonfeld, "Netflix now the largest single source of internet traffic in north america," 2011, (Accessed: 2015-03-05). [Online]. Available: http://techcrunch.com/2011/05/17/netflix-largest-internet-traffic/

[28] D. E. Knuth, *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997.

[29] "rsync," (Accessed: 2014-09-30). [Online]. Available: http://rsync.samba.org/

[30] Gnu wget. (Accessed: 2014-09-30). [Online]. Available: https://www.gnu.org/s/wget/

[31] C. Do Cuong, "Seattle conference on scalability: Youtube scalability," *Video, June*, 2007.

[32] M. Zink, K. Suh, Y. Gu, and J. Kurose, "Characteristics of youtube network traffic at a campus network – measurements, models, and implications," *Computer Networks*, vol. 53, no. 4, pp. 501 – 514, 2009, content Distribution Infrastructures for Community Networks.

[33] A. Finamore, M. Mellia, M. M. Munafò, R. Torres, and S. G. Rao, "Youtube everywhere: Impact of device and infrastructure synergies on user experience," in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '11. New York, NY, USA: ACM, 2011, pp. 345–360. [Online]. Available: http://doi.acm.org/10.1145/2068816.2068849

[34] P. Ameigeiras, J. J. Ramos-Munoz, J. Navarro-Ortiz, and J. Lopez-Soler, "Analysis and modeling of youtube traffic," *Transactions on Emerging Telecommunications Technologies*, vol. 23, no. 4, pp. 360–377, 2012.

[35] C. Forbes, M. Evans, N. Hastings, and B. Peacock, *Statistical distributions*. John Wiley & Sons, 2011.

[36] M. F. Arlitt and C. L. Williamson, "Internet web servers: Workload characterization and performance implications," *IEEE/ACM Trans. Netw.*, vol. 5, no. 5, pp. 631–645, Oct. 1997.

[37] H. Gupta, A. Mahanti, and V. Ribeiro, "Revisiting coexistence of poissonity and self-similarity in internet traffic," in *Modeling, Analysis Simulation of Computer and Telecommunication Systems, 2009. MASCOTS '09. IEEE International Symposium on*, Sept 2009, pp. 1–10.

[38] X. Kang, H. Zhang, G. Jiang, H. Chen, X. Meng, and K. Yoshihira, "Measurement, modeling, and analysis of internet video sharing site workload: A case study," in *Web Services, 2008. ICWS'08. IEEE International Conference on*. IEEE, 2008, pp. 278–285.

[39] T. Mori, R. Kawahara, H. Hasegawa, and S. Shimogawa, "Characterizing traffic flows originating from large-scale video sharing services," in *Traffic Monitoring and Analysis*, ser. Lecture Notes in Computer Science, F. Ricciato, M. Mellia, and E. Biersack, Eds. Springer Berlin Heidelberg, 2010, vol. 6003, pp. 17–31.

[40] V. Adhikari, S. Jain, Y. Chen, and Z.-L. Zhang, "Vivisecting youtube: An active measurement study," in *INFOCOM, 2012 Proceedings IEEE*, March 2012, pp. 2521–2525.

[41] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "Youtube traffic characterization: A view from the edge," in *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '07. New York, NY, USA: ACM, 2007, pp. 15–28.