Application of a Self-Healing Video-Streaming Architecture to RTSP Servers

Carlos A S Cunha
Centre for Informatics and Systems
University of Coimbra
Portugal
ccunha@dei.uc.pt

Luis Moura e Silva
Centre for Informatics and Systems
University of Coimbra
Portugal
ccunha@dei.uc.pt

***Abstract*— Streaming media is now one of the killer applications on the Internet. Availability in streaming services is a critical concern, as consumer expectations are drawn around decades of traditional TV experience. Server performance has particular importance in streaming, as its sensitiveness to delays makes it vulnerable to performance anomalies. Current work on server-level performance analysis fails to cope with performance failures not explained by the workload. We propose a self-healing architecture for streaming servers sustained by a biological metaphor of heart that explores proactive server recovery by anticipating performance failures through detection of *arrhythmias* (transmission delays of streaming content) and session probing. We evaluated the approach in RTSP streaming through experimental work in several resource exhaustion scenarios. Results have shown that our approach is able to predict and localize service failures several seconds before their occurrence for most failure scenarios.**

Keywords- self-healing; video-streaming; dependability

I. Introduction

Video streaming is a popular class of Internet services gaining an increasing acceptance in the last years [1][2][3][4]. Streaming users put high expectations on quality, as they are traditional TV consumers, a class of people that soak up decades of TV quality and availability patterns. For that reason, handling the fail-stutter failure model [5] at network and server levels is mandatory because performance anomalies may impact considerably streaming services due to their sensitiveness to delay jitter.

Research on graceful service degradation techniques to overcome network anomalies in streaming is extensive. Adaptive streaming delivery techniques (e.g., HTTP Adaptive streaming [6]) dominate this area by allowing dynamic switching of streaming quality levels to adapt to network service degradations. At server level, performance degradation has been studied to predict resource exhaustion [7][8][9], a typical cause of performance failures [10][11][12]. These works address capacity determination in streaming systems, but only captures failure states explained by the workload.

The ability to handle resource variability and system faults is a software requisite with increasing importance. Actual virtualization infrastructures potentiate dynamic allocation of resources to optimize their utilization. Cloud infrastructures in particular will have an important role in reducing hardware costs [13] due to the streaming traffic profile: it is resource intensive and varies significantly by the time-of-day, day-of-week and in the long-term [14]. Thus, companies can reduce costs significantly by allocating resources on-demand on third-party clouds for particular time periods.

Considering the streaming characteristics and the dynamicity of infrastructures, we devise in this paper a streaming architecture for monitoring, localization and repair of failures with the following requirements: *strict recovery delays* (below client buffering capacity), *infrastructure agnosticism* (no assumption about the system nominal capacity) and *application-level awareness*. To implement these requirements, we explore the use of *self-healing*, a software property that enables a system to perceive that it is not operating correctly and with or without limited human intervention, make the necessary adjustments to restore itself to normalcy [15][16]. Our contributions unveil the ability of anticipating failure scenarios and identify the resources exhausted during such periods in RTSP (Real-time Streaming Protocol) streaming.

This paper is structured as follows. Section II presents the related work. Section III describes our self-healing architecture. Section IV explains the application of our architecture to RTSP streaming. Section V presents results of the experimental work. Section VI discusses results. Section VII concludes.

II. Related Work

Previous work on fault-tolerance of streaming servers is classified at *external*, *architectural* and *server* levels.

Proxies are external components that create a level of indirection between clients and server for independent monitoring and failover. Jeon et al [20] proposed a proxy-based mechanism for monitoring and failover of streaming servers. Monitoring is performed through analysis of RTP packet inter-arrival times, using dynamic thresholds.

Architectural configurations can support fault-tolerant streaming servers. Layered coding was explored for graceful degradation during server failures. Nakatogawa et al [21] propose a decentralized video-streaming fault recovery architecture. Video is encoded into data layers with different quality placed in separated nodes. A desirable quality is obtained through combination of several layers. The Yima media server [22] is implemented over a scalable real-time streaming architecture. If a server fails, the system reassigns their sessions to another server without disruption in data delivery. Maharana et al [23] present RSerPool fault tolerant architecture for video on demand. Nodes providing a given service are grouped into a pool and registered into a name server. Name servers monitor the health of their nodes using heartbeats. Clients assist failure recovery using a local copy of the server's session state to support its resumption during failover.

Performance analysis in video servers at system level has been explored for server capacity determination. Vin et al [8] proposed an admission control algorithm based on analysis of variation in the access times of media blocks from disk. Covell et al. [7] propose a resource modeling approach to predict failures caused by saturation of streaming servers. Cherkasova et al. work [9] address capacity determination of media servers for utility-aware streaming media services, by creating a model to calculate the cost of each session for each streaming type.

III. Paradigm and Approach

The cardiovascular system is a blood streaming system controlled by the heart. Blood streaming is pumped by heart at a given pace through electrical stimulus regulated through heart receptors. Abnormal paces are referred as *arrhythmias* and are manifested by slower (i.e., *bradycardia*), faster (i.e., *tachycardia*) and irregular pace frequencies. Effects of abnormal blood flows due to *arrhythmias* are propagated through the cardiovascular system, causing malfunction of other organs.

Video-streaming systems have similarities with the cardiovascular system. Streaming systems have their own heart (i.e., the server) that precisely pumps frame blocks at a given pace to clients. Server activity is regulated by VCR-like protocols (e.g., the popular RTSP) through channels functionally equivalent to heart receptors. During faulty states, abnormal paces occur when the server struggles to timely send the frames precisely scheduled in advance, resulting in a condition equivalent to *heart arrhythmias*.

We propose a self-healing architecture for streaming servers that has roots on the heart metaphor. The server has *sensors* that measure anomalous states, which will feed the analysis process developed by a *pacemaker* equivalent component. After detection of anomalies the pacemaker is responsible to restore the system to normalcy by injecting *repair stimulus* that returns the server to normalcy.

A. Self-healing Architecture

The self-healing architecture is shown in Fig. 1. It is comprised by four main components: ***performance analyzer***, ***repair planner***, ***sensors*** and ***effectors***.

The performance analyzer takes: (1) server performance parameters collected by *sensors* to detect performance degradation; and (2) service parameters collected from synthetic sessions established with the server. It detects performance degradation and pinpoints the resources responsible for these anomalous states. The *repair planner* takes notifications issued by the performance analyzer, decides the best repair action to be executed (e.g., session migration [17][18] and protocol-level redirection of sessions [19]) and plans the repair to be executed by *effectors*.

Sensors are responsible for collecting performance parameters. Two types of sensors are devised server-side: (1) ***pace sensors***; and (2) ***resource sensors***. *Pace sensors* capture delays on accomplishment of scheduling of video content. The unit of scheduled work depends on server implementations and streaming protocols. In RTSP streaming the packet is the unit of work to be scheduled and transmitted. Other video standards require coarse-grain performance metrics at server: HTTP Streaming and HTTP Adaptive streaming [24]. They fragment video objects into slices that are selectively requested by clients and downloaded similarly to other web objects.

B. First-level Monitoring: Server Performance Degradation

Performance analysis resumes to the search of anomalous time windows, defined as those having a mean transmission delay higher than α, defined as follows.

$$mean\ delay = \frac{1}{n}\sum_{i=1}^{n} delay_i \qquad (1)$$

Being n the number of segments transmitted since the last computation of (1), $delay_i$ the accumulated transmission delays of video-segments in the same period and α adjusted to the server implementation (Section IV).

C. Second-level Monitoring: Service Failures

Detection of service failures uncovers unpredicted failures and assesses the effectiveness of repair actions after being executed.

1) Data Gathering Method - We use synthetic sessions established periodically by the monitor to assess service quality to provide: (1) ***isolation from network and client interference*** (sessions served over a network providing stable delay and jitter parameters and single client configuration; (2) ***Manageable data volume*** (performance metrics collected only for synthetic sessions); and (3) ***Time constrains*** (the number, length and encoding bitrate are carefully specified).

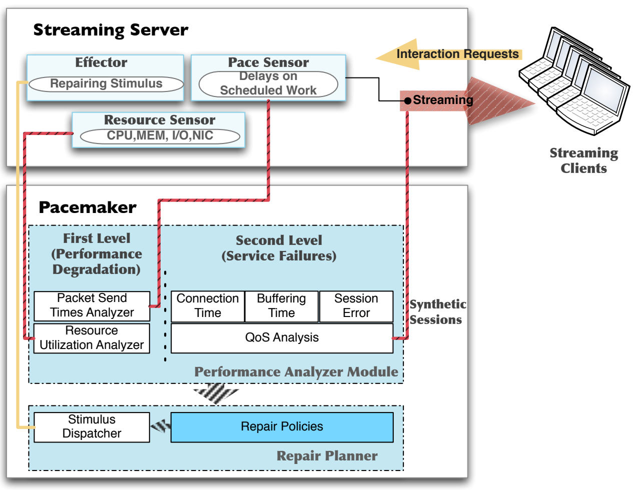

Figure 1. Self-healing Architecture for Video-Streaming

2) Definition of Failure - We adopt the Keynote StreamQ [25], a leading industry standard metric to measure quality of video-streams. To assess degradation of service quality we capture both session errors and QoS degradation metrics (Table I).

D. Localization of Degradation

We calculate the utilization of each resource using the median resource utilization within the time window marked as anomalous, as in (2).

$$D_i = median\left(x_j\right), \quad i - w < j \le i \tag{2}$$

The D_i is the median resource utilization during the time window of size w that precedes the time i. The x_j is the resource utilization read for the resource analyzed at time j.

E. Repair Planner

The *repair planner* schedules execution of proactive and repair actions to bring the server to a non-degraded state. Mapping between events and actions are defined in the form of rules-e.g.

"Performance degradation caused by memory exhaustion leads to failover of the server instance."

Notifications issued by the performance analyzer carry the anomaly profile describing its type (e.g., server performance slowdown or QoS failure), predicted localization (e.g., memory, I/O) and other relevant parameters.

IV. RTSP PERFORMANCE ANALYSIS

Performance analysis is the most complex part of our architecture. In this section we present its application to RTSP streaming [19] using the Darwin Streaming Server [26], a popular streaming server.

A. Representativeness of Synthetic Sessions

Representativeness of synthetic sessions is assessed through comparison of our *default configuration* (2 sessions of 300Kbps issued each 10 seconds) with each configuration presented in Table II. The alternative configurations explore the impact of the *number of sessions*, *bitrate* and *popularity* of videos on the representativeness of synthetic sessions. Results have shown the same values for all configurations, validating the assumption of representativeness of the *default configuration*.

V. EXPERIMENTAL WORK

The experimental methodology developed to evaluate the performance analyzer component is described as follows: (1) **generate monitoring stimulus**; (2) **stress the server and log parameters**; and (3) **statistical analysis**.

TABLE I. QOS METRICS ANALYZED FOR SYNTHETIC SESSIONS

Parameter	Description
Connection Time	Time spent since the client sends a session establishment request until it receives the first media packet.
Buffering Time	Total time spent by the client doing buffering (before playing) and rebuffering (when the client is forced to stop due to absence of frames received).
Session Error	Session failure caused by a unresponsive server or error.

TABLE II. WORKLOAD COMBINATION USED TO ASSESS THE REPRESENTATIVENESS OF SESSIONS. EACH ALTERNATIVE SCENARIO IS EXECUTED IN PARALLEL WITH OUR DEFAULT CONFIGURATION (I.E., 2 SESSIONS OF 300KBPS).

Configuration		**Alternative Scenarios (Number of Parallel Sessions)**		
Popularity	***Encoding Bitrate***	***1. Number***	***2. Bit Rate***	***3. Popularity***
Popular	300k	2, 4, 6		
	1000k		1	
Unpopular	300k			1
	1000k		1	1

Our failure model encloses server overloading explained by workload and exhaustion of individual resources, namely, *CPU*, *Memory* and *I/O*. We induce individual resource exhaustion using the Stress tool [27]. The experimental tests are repeated 10 times for statistical significance.

A. Testbed Configuration

Our testbed encloses three machines connected by a 100Mbps Ethernet Network: the *server*, the *workload generator* and the *synthetic sessions generator*. All machines are configured with an Intel(R) Pentium(R) D CPU 3.00GHz, 1GB RAM, running a Linux 2.6.18-92.1.22.el5 Kernel.

B. Overloading Detection

Server overloading occurs when the server is brought to its limit for each workload type - characterized by its *popularity*, *bit rate* and *number of connections* [7][9]. We selected representative encoding bit-rates used for streaming of video on the Internet [28]. Streaming the same video object in all sessions generates popular workloads. The unpopular workload counterpart requires a different video object for each session to force reading the video from the disk. A ramp-up workload with an inter-session creation time of 2 seconds drives data gathering of anticipation times of application-level metrics over service failures.

1) Overloading by Popular Workloads – Popular workloads are network-bound in our server configuration. Application-level performance degradation is observed for 300Kbps videos 14.5 seconds (median) before its manifestation as a service failure (Table III). That degradation coincides with exhaustion of the network interface resource. Popular 1000Kbps videos were not captured as packet send delays. A deep analysis of this metric atributted this phenomenon to negative delays (i.e., packets are sent in advance), justified by an adaptive behavior activated by the server for this workload type.

2) Overloading by Unpopular Workloads – Unpopular workloads are disk-bound. The server performance starts struggling when the disk I/O and CPU reach their limit. In the 1000Kbps configuration, the bootleneck is the CPU, accompanied later by the memory. The median anticipation times for 300 Kbps and 1000 Kbps are 40 seconds and 15 seconds, respectively.

C. Resource exhaustion not explained by workload

The Stress tool recreates failure scenarios where performance failures have origin in exhaustion of individual resources, not caused by excess of server nominal capacity.

TABLE III. ANTICIPATION TIMES

Fault Type	**Exhausted Resources**	**Anticipation Times**		
		Min	***Median***	***Max***
Overload 300 Kbps Popular	Network	9	14.5	30
Overload 1000 Kbps Popular	Network	(*)	(*)	(*)
Overload 300 Kbps Unpopular	Disk I/O, CPU, Memory	20	40	60
Overload 1000Kbps Unpopular	CPU, Memory	10	15	90
CPU Stress	CPU, Memory	56	60	65
I/O Stress	CPU, Memory	8	10	12
Memory Stress	CPU,, Memory	18	20	21

(*) Unpredicted as the server sends packets in advance

The server is subjected to a workload of 50% of its nominal capacity before start stressing resources. We injected three fault types: **(1) CPU Stress** by spinning on *sqrt()*; **(2) I/O Stress** by spinning on *sync()*; **(3) Memory Stress** by spinning on *malloc()*. The medians calculated for anticipation times of predictions are 60 seconds, 10 seconds and 20 seconds for CPU, I/O and memory, respectively (Table III).

VI. CONCLUSIONS

This paper proposes a self-healing architecture for video-streaming servers and validates its application to RTSP streaming. Experimental results validated the application of the *arrhythmia* concept to RTSP streaming. Several lessons can be taken from the analysis of results: (1) application-level performance analysis anticipates service failures dozens of seconds before the client experiences service failures, except in overloading conditions for high bitrate popular videos; and (2) identification of resources responsible for the failures is blurred by exhaustion of other dependent resources. This observation is especially aberrant in resource exhaustion scenarios where the server load is below its nominal capacity. Thus, performance analysis and resource localization are unattainable through simple analysis of individual resources.

As future work, we plan to diagnose performance failures by breaking down failures caused by transient faults [29] from workload-related failures. This separation is important for recovery, as the former can be resolved through allocation of new resources, but the latter is usually corrected automatically or through rejuvenation techniques.

ACKNOWLEDGMENT

This work was partially supported by FCT-Portugal under grant SFRH/BD/35784/2007 and CISUC (Centre for Informatics and Systems of University of Coimbra).

REFERENCES

[1] "IGI Forecast 2010," highbeam.com.

[2] H. Schulze and K. Mochalski, "IPOQUE Internet Study 2008/2009," ipoque.com.

[3] "Allot Global Mobile Broadband Traffic Report 2009," allot.com.

[4] J. Careless, "The State of Streaming Media and Entertainment 2011," streamingmedia.com.

[5] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau, "Fail-stutter fault tolerance," in Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on, 2001, p. 33-38.

[6] T. Stockhammer, "Dynamic adaptive streaming over HTTP --: standards and design principles," in Proceedings of the second annual ACM conference on Multimedia systems, New York, NY, USA, 2011, p. 133-144.

[7] M. Covell et al., "Calibration and prediction of streaming-server performance," HP Labs Tecnical Report HPL-2004-206, 2004.

[8] H. Vin, P. Goyal, and A. Goyal, "A statistical admission control algorithm for multimedia servers," in MULTIMEDIA '94: Proceedings of the second ACM international conference on Multimedia, New York, NY, USA, 1994, p. 33-40.

[9] L. Cherkasova and L. Staley, "Building a Performance Model of Streaming Media Applications in Utility Data Center Environment," in CCGRID '03: Proceedings of the 3st International Symposium on Cluster Computing and the Grid, Washington, DC, USA, 2003, p. 52.

[10] D. Oppenheimer, A. Ganapathi, and D. A. Patterson, "Why do internet services fail, and what can be done about it?," in Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems - Volume 4, Berkeley, CA, USA, 2003, p. 1-1.

[11] S. Pertet and P. Narasimhan, "Causes of Failure in Web Applications," Technical Report PDL-CMU-05-109, Carnegie Mellon University, 2005.

[12] J. Gray, "Why Do Computers Stop and What Can Be Done About It?," in Symposium on Reliability in Distributed Software and Database Systems, 1986, p. 3-12.

[13] P. Csathy, "Industry Perspectives: Enterprise Video Encoding---The Power and Promise of the Cloud in 2011," streamingmedia.com.

[14] 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, New York, NY, USA, 2007, p. 15-28.

[15] Y. Brun et al., "Engineering Self-Adaptive Systems through Feedback Loops," in Software Engineering for Self-Adaptive Systems, vol. 5525, B. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, Eds. Springer Berlin / Heidelberg, 2009, p. 48-70.

[16] P. Koopman, "Elements of the Self-Healing System Problem Space," WADS Workshop on Software Architectures for Dependable Systems, 2003.

[17] F. Sultan, K. Srinivasan, D. Iyer, and L. Iftode, "Migratory TCP: Connection Migration for Service Continuity in the Internet," in Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02), Washington, DC, USA, 2002, p. 469-469.

[18] A. C. Snoeren, D. G. Andersen, and H. Balakrishnan, "Fine-grained failover using connection migration," in Proceedings of the 3rd conference on USENIX Symposium on Internet Technologies and Systems - Volume 3, Berkeley, CA, USA, 2001, p. 19-19.

[19] H. Schulzrinne, A. Rao, and R. Lanphier, "Real Time Streaming Protocol (RTSP)," 1998.

[20] S. Jeon, J. Lee, H. Cha, and R. Ha, "Proxy-based failure detection in multimedia streaming environments," Int. J. Commun. Syst., vol. 20, no. 2, p. 131-145, 2007.

[21] Y. Nakatogawa, Y. Jiang, M. Kanda, K. Mori, R. Takanuki, and Y. Kuba, "Autonomous Fault Recovery Technology for Achieving Fault-Tolerance in Video on Demand System," in Proceedings of the Eighth IEEE International Symposium on Multimedia, Washington, DC, USA, 2006, p. 113-120.

[22] C. Shahabi, R. Zimmermann, K. Fu, and S. D. Yao, "Yima: A Second-Generation Continuous Media Server," Computer, vol. 35, no. 6, p. 56-64, 2002.

[23] A. Maharana and G. N. Rathna, "Fault-tolerant Video on Demand in RSerPool Architecture," Advanced Computing and Communications, 2006. ADCOM 2006. International Conference on, p. 534-539, Dec. 2006.

[24] T. Stockhammer, "Dynamic adaptive streaming over HTTP --: standards and design principles," in Proceedings of the second annual ACM conference on Multimedia systems, New York, NY, USA, 2011, p. 133-144.

[25] "Keynote Streaming Perspective StreamQ," keynote.com.

[26] "Darwin Streaming Server," dss.macosforge.org.

[27] "Stress Project," weather.ou.edu.

[28] 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, New York, NY, USA, 2007, p. 15-28.

[29] A. Avizienis, J. 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, p. 11-33, jan.-march. 2004.