Hoecker and Kunze *Journal of Cloud Computing: Advances, Systems and Applications* 2013, **2**:14
http://www.journalofcloudcomputing.com/content/2/1/14

Journal of Cloud Computing
a SpringerOpen Journal

RESEARCH Open Access

An on-demand scaling stereoscopic 3D video streaming service in the cloud

Maximilian Hoecker* and Marcel Kunze

Abstract

We describe a web service providing a complete stereoscopic 3D video multi-stream cloud application to serve a potentially very large number of clients over the Internet. The system architecture consists of a stream provider that leverages highly scalable and reliable cloud computing and storage services, with automatic load balancing capability for live and content streaming. By use of a suiting flash media plugin the content is displayed on a wide variety of 3D capable devices like for example 3D workstations or smart TV sets. Videos are made available by an on-line stream provider for live broadcasting or by cloud storage services. Compared to conventional 3D video streaming over satellite channels there are considerable savings in cost as well as a wider range of applicability and functional improvements. Possible areas of application are medical surgery, live concerts, and sports events.

Introduction

The conventional method of distributing high definition stereoscopic 3D live video streams is the transmission via satellite links. While the provisioning of the necessary bandwidth via satellite is technically feasible, live transmission is very costly: An hour of satellite channel costs approximately 700$ and the rental of a transmission vehicle adds up to 2,000$ per day. Furthermore, secure satellite transmission usually works point-to-point only especially when stream encryption is used. It would thus be interesting to develop methods to mass distribute 3D live content over cheaper Internet broadcasting channels. However, the Internet up to now has played a minor role in this context due to the high bandwidth demand of high definition 3D video streams. New compression methods like H.264 and encoders as well as improvements in the available bandwidth, however, have changed the situation: While professional streams in medical applications with quadruple HD may require up to 40 MBit/s uncompressed, the typical bandwidth consumption of a compressed stereoscopic 3D video stream in 720p quality is 2 MBit/s, ranging up to 5 MBit/s for 1024p.

The cloud application presented in this paper discusses the live broadcasting of stereoscopic 3D video over the Internet by use of automatically scaling media clusters and user interface portals based on commodity cloud services.

*Correspondence: maximilian.hoecker@uni-heidelberg.de
Karlsruhe Institute of Technologie (KIT), Karlsruhe, Germany

The paper is organized as follows: First we describe the application context with its specific requirements. In the next chapter we discuss the architectural aspects of an on-demand, scaling live video broadcasting environment. Then we present performance studies and the paper concludes with a summary and outlook.

Stereoscopic 3D video streaming

Stereoscopic 3D video presents each eye of the observer with a slightly different perspective to enable a realistic 3D experience. A corresponding video streaming setup thus has to transport two pictures synchronously, one for each eye. Various techniques have been employed to provide stereoscopic video such as anaglyph 3D, polarization 3D, active shutter, and side-by-side that are already embedded in actual 3D TV sets. A comprehensive streaming application should thus not be limited to a particular type of stereoscopic format in order to reach a large audience. There are tools and encoders on the market that allow to handle a wide spectrum of formats such as the Invistra multi converter [1] and Adobe Flash plugin [2] that have been chosen to support the research described in this paper.

Download vs. streaming

The transmission of video and audio data over the Internet has become very prominent. File sharing services are usually offering files to download in a web browser or they

leverage specific file sharing protocols like BitTorrent [3]. However, the download of files has its problems with fault tolerance and it is difficult to steer the bandwidth consumption. If a media file is downloaded, the client keeps on transferring data in the background while the content is being consumed. If a client navigates forward, the already transferred portions of the file have to be skipped and it might happen that there is a waiting time until the client may continue at the new location. Usually the "header" information of a media file is located at the end of the file.

An example for a standard download method is Apple's PodCast. A PodCast file has to be downloaded completely before it is able to be played because the header information is at the end of a file. The technique of progressive download transfers a header at the start of the file in addition to the content that contains information to navigate in the media. This makes it possible to play the file while it is still downloading. Various companies like Adobe, Apple, and Microsoft offer proprietary protocols to support HTTP based applications with progressive download. The corresponding implementations mainly differ in the supported CODECs.

Streaming on the other hand is a technology that transfers a data stream from a service to a client with quality control. Server and client exchange state information and quality metrics as well as meta data describing the stream by means of a streaming protocol.

Protocols

A streaming protocol contains two layers: A transport stream and a control stream. The transport stream carries the media data and it may be based on both transfer protocols, UDP and TCP. The control stream takes care of the Quality of Service (QoS) and its implementation depends on the specific transfer protocol in use.

A prominent implementation of a streaming protocol is the Real Time Protocol (RTP) that has been specified in RFC 3550 [4]. It is based on UDP and uses the Real Time Control Protocol (RTCP) for QoS. The Real Time Messaging Protocol (RTMP) has been published by Adobe in 2009 [5]. It uses TCP and guarantees the correct order of the datagrams in the stream. On the server side RTMP implements command messages, data messages, shared object messages to embed e.g. Flash events, media messages, and aggregate messages to combine various messages gaining efficiency. On the client and server side RTMP offers user control messages to notify about events like stream begin or end. There are several variants of RTMP that allow to tunnel through firewalls (RTMP-T), to encrypt the media streams (RTMP-E, RTMP-S), and a combination of both (RTMP-TE) [6]. Examples for RTMP streaming clients are the VLC player [7] and the browser based FlowPlayer [8].

Streaming services

Streaming servers are able to transport almost arbitrary video streams in almost any format. They offer capabilities such as live transmission, live transcoding into various formats, quality control, and digital rights management. Examples are the Adobe Flash Media Server [2] or the Wowza media server [9]. The Amazon Web Services (AWS) [10] offer paid instances of media servers that may be rented by the hour, thus implementing a dynamically scalable streaming service. We are utilizing these cloud based streaming services to construct our scalable media service based on RTMP.

Security

Broadcasting of video streams over the internet works on a global scale. Hence, a corresponding video streaming service should ensure the secure and reliable transmission of content. Depending on the application context, stream access control is needed e.g. in commercial portals selling video-on-demand. In addition the service may consume large amounts of bandwidth, especially when 3D videos in HD quality are distributed. A single stream may consume a bandwidth between 2 MBit/s and 40 MBit/s.

Several possibilities exist to transport live video data from a source to a streaming client: One could deploy a peripheral mesh network where each client is a stream distributor for many other clients, or a central streaming cluster multiplying streams locally.

The application presented in this paper is focussing on stereoscopic 3D live broadcasting in the public cloud combined with secure transmission of video as an aliased stream with optional encryption. A stream represented by an aliased name prevents replaying content on several clients. Stream aliasing is realized by masquerading real stream names with an alphanumeric hash derived from the stream name, current time, client IP-Address, and a customer specific secret key.

Architecture

Blueprint

Figure 1 shows an architectural blueprint of a system to transport stereoscopic 3D live content from a source to a potentially large number of viewers. The design philosophy is that all components may be instantiated as virtual machines in the public cloud. Video clients are requesting a live stream via a video streaming portal. The portal forwards the request to a streaming origin server accessing the content provided by a stream provider. The origin server delivers the stream to a dynamically scaling cluster of edge servers. The edge servers deliver the live stream to the requesting streaming clients. The whole system is managed and monitored by a management server (3Distribute). The management server is composed of

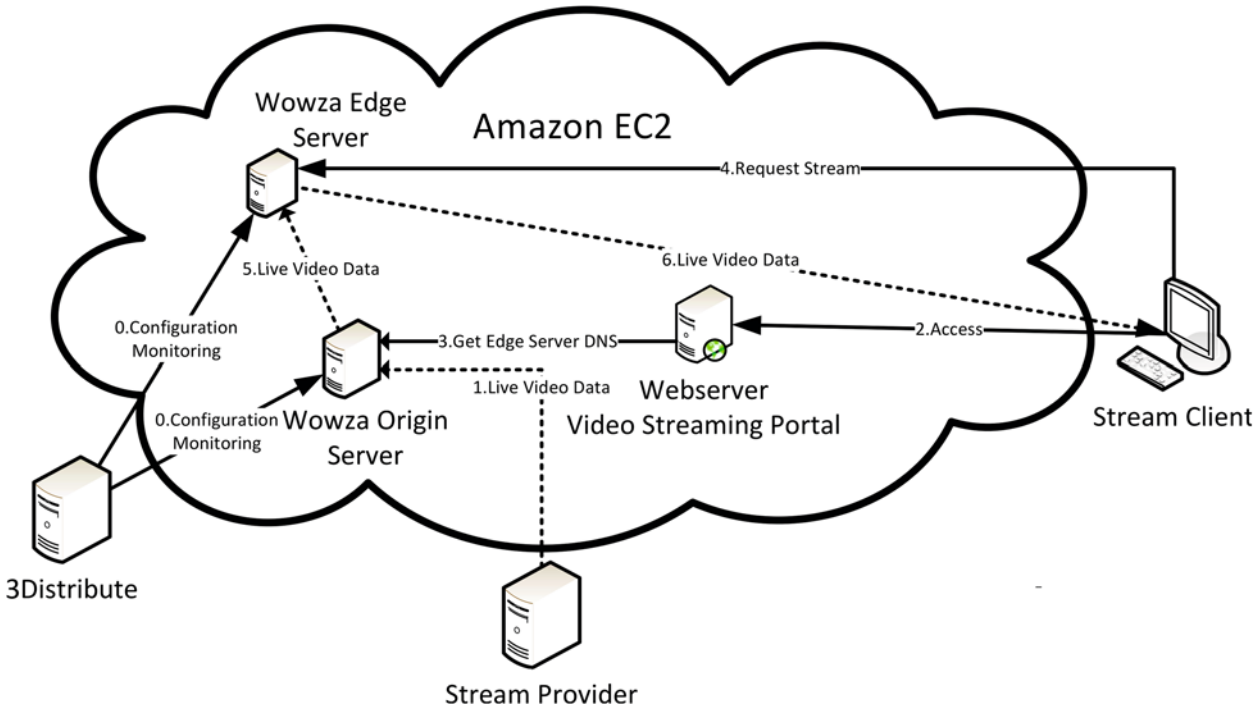

Figure 1 Scaling 3D video streaming architecture.

an application interface server (API-Server) and a user interface server (UI-Server).

Logical components

The core part of our system is the API-Server. Key functionalities are: Starting and configuring of virtual machines, as well as managing and monitoring of the media streaming cluster, including dynamic scaling of the cluster size. As shown in Figure 2 the API-Server consists of three layers of components, each of which abstracts the layer below: The very low layer consists of connectors to utilize IaaS services of various providers (Cloud connector) and grants access to virtual machine resources (Machine connector). A cluster and monitoring component abstracts this first layer and represents the basic functionality to manage a virtual cluster of machines. Finally, an application is able to access the system by use of the top level control component that constructs and steers instances of the cluster component. All components are defined by interfaces to provide the possibility to choose or change a specific implementation.

Cloud connector

The *cloud connector* component realizes transparent access to the proprietary API of an IaaS cloud service offering. Each individual implementation of this component has to provide the following functionalities:

- Launch and terminate virtual machines of a cloud service.
- Capture and collect performance metrics of virtual machines as a basis to perform service analysis.
- Associate reserved IP addresses with instances.

Machine connector

Controlling virtual machines with different protocols and connections is the task of the *machine connector* component. The machine connector takes care of the following tasks:

- Grant remote access to a machine in order to submit and execute remote procedures. Each task returns a result indicating if a command has been run successfully or not. Additionally the standard output and error messages of the remote command may be transferred.
- Provide a file transfer method to e.g. distribute configuration files.

The standard implementation of the machine connector is establishing connections via Secure Shell (SSH) and uses a Secure Copy (SCP) session for file transfers. It is possible to re-implement the connector with different protocols to transmit commands or local files to a remote machine. The portability of the component could be helpful to support different operating systems and media services.

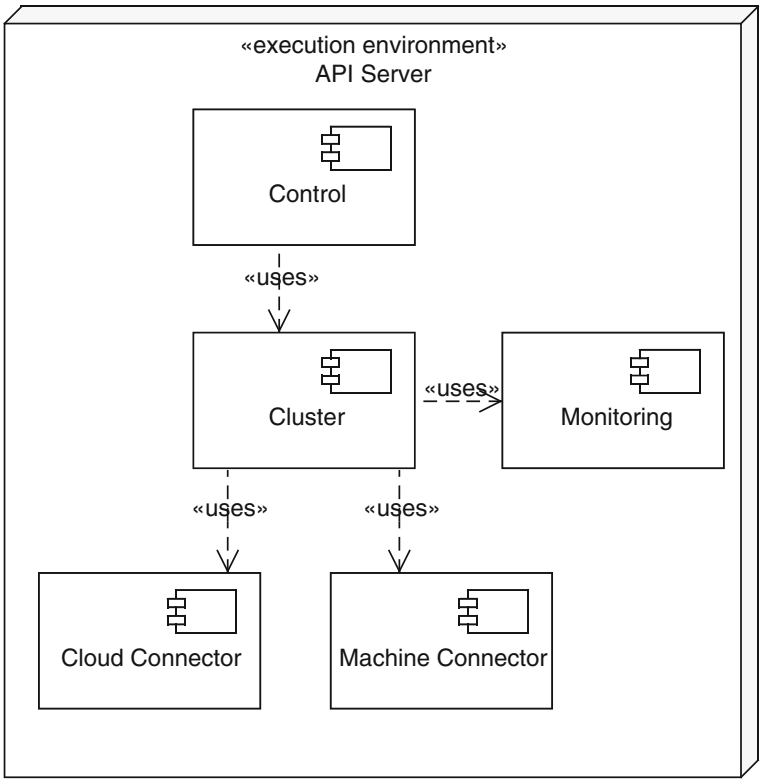

Figure 2 Logical components of 3Distribute.

Cluster component

The *cluster component* has been designed to construct a virtual cluster of virtual machines. It depends on all components located in the layer below and is organized as follows:

- A component called *scenario* is representing a data model providing information about virtual machine properties in a cloud environment. Instances of a cluster component scenario contain one ore more so-called *scenario parts*. Scenario parts represent a deployment information knowledge base for different independent parts of the cluster containing e.g. machine size and image information, key pair names, and files. Using scenario parts, different types of virtual machines in the cloud may be orchestrated in a cluster.
- In addition to the scenario part, a *machine configuration* represents another data model containing a set of ordered commands and local files to be uploaded and executed to configure a virtual machine dynamically during cluster startup or sizing operations.
- A central subcomponent is the *cluster machine* representing abstract access methods to interact with the virtual machines in a cloud. The main task of the cluster machine is to update internal state relevant for cluster sizing and to control the connected virtual machines in the cloud using the connector components.
- Finally, the *clustering component* manages all subcomponents described above. It is creating and controlling all cluster machines and maps scenario parts and machine configurations accordingly.

Analysis component

The *analysis component* allows to monitor and control the cluster in a regular fashion. To provide this functionality, the component is organized as follows:

- The *cluster machine analyzer* analyzes the state of single cluster machines using data of metrics generated by a cloud connector. A report is generated including a qualitative measurement of the current streaming load of a cluster machine.
- Based on this report the *cluster analyzer* component prepares another report providing qualitative and quantitative information about the streaming load of a video streaming cluster as a whole.
- The *actions catalogue* component is able to determine the action needed to handle a specific load condition taking into account both, cluster and cluster machine reports.

A separate thread triggers the analysis of all clusters, including all necessary actions determined by the catalogue.

One possible action is resizing a cluster: If a cluster overload condition occurs, the number of virtual machines is dynamically increased by an upsizing process. On the other hand, if idle machines exist a downsizing process takes care to decrease the number of virtual machines by sending a termination or stop signal to underutilized machines. As a boundary condition the API server ensures the cost effectiveness of the operation as the resource usage accounting model may vary for different IaaS cloud providers. For example, the Amazon Web Services (AWS) compute service EC2 [11] charges any started hour of instance usage independent of the actual utilization profile (wallclock time). Thus it would be interesting to stop machines right before the end of the hour in order to take full advantage of an already billed resource. Another accounting model applied by Google [12] or Profitbricks [13] continuously charges the actually consumed resources (CPU time). The current implementation is utilizing AWS services and instances are therefore terminated on an hourly basis.

Control component

This component builds an abstract layer of all components below and provides functionalities to control, launch and terminate clusters. First of all the control component performs validity and integrity checks of parameters supplied to instantiate a cluster component instance. After the instantiation the cluster is passed to the analysis component taking over responsibility of cluster operations. The current implementation of the web interface uses an RMI Interface to connect to the API server.

User interface

The user interface consists of an administrative console and a web front end to play and watch stereoscopic content. The administrative console allows to configure and manage the complete environment and perform detailed monitoring of the system. The web front-end has to support a wide range of stereoscopic 3D formats, such as anaglyph 3D, polarization 3D, active shutter, and side-by-side. For this purpose a web browser plugin has been developed based on Adobe Flash technology [2]. A Microsoft Silverlight [14] implementation exists in addition to support NVIDIA 3D vision shutter glasses for desktop PCs [15]. The plugin supports to play a wide range of formats via a popup selection bar and is also used in the enduser interface of the commercial product. In principal, the control component may be embedded into arbitrary web pages in order to support customer specific portals with stereoscopic 3D content delivery.

Automated load balancing and scaling

Live streaming events like e.g. a medical congress or the transmission of a concerto may attract an unpredictable number of requesting streams. One of our targets during the design phase was to create a system scaling in every instance of time taking into account parameters like load and cost. A new streaming request is always redirected to the video streaming server having the highest load and the capacity to feed further clients. In case of a potential overload condition the next less loaded server is chosen. This procedure maximizes the usage efficiency of a video streaming server and minimizes the cost since servers are only kept online in case of a need.

We developed a load balancing Plugin for the Wowza Media Server to cover these requirements based on the heartbeat architecture pattern. An architecture including a heartbeat pattern consists of a master server and multiple slave nodes. Our setup implements the origin server as master node and edge servers as slave nodes sending heartbeats. The master node collects all load information (heartbeats) of each streaming server in the cluster by use of Remote Procedure Calls (RPC) via an encrypted protocol. The number of connections on a server as well as the outgoing network traffic is transmitted in a message. When a streaming request of a user is submitted as shown in Figure 1 (step 2) the IP-Address of an edge server is requested (step 3), selected and delivered by the load balancer plugin.

It is a challenge to scale the streaming capacity up and down in near-realtime. For EC2 the AWS CloudWatch service provides a time-aggregated view of the system performance at intervals of every one or five minutes. The delivered aggregation dataset holds a small collection of statistical metrics like average, minimum and maximum value of each system state. However, the CloudWatch service may take to long if QoS needs to be guaranteed.

In order to avoid this kind of delays and to minimize the time to scale, we implemented a faster scaling method in the API Server calling a specific entry of the load balancing plugin each 10 seconds. The encrypted call transmits collective cluster information consisting of all edge server heartbeats including a timestamp. A history of the last minute of heartbeats is kept in memory to determine the actual progression of load. Depending on the load history the necessary measures like cluster resizing are initiated by the API Server.

Measurement of server loads and capacity

An important role plays the load balancing in dependence of resource analysis and monitoring. We define a metric called `streaming capacity` which indicates

the current workload. The metric is implemented as a percentage value, where 100% describes the best condition (no load) and 0% represents the least capacity left (cluster is fully loaded), respectively.

For the determination of the streaming capacity the application uses monitoring data that are provided and elevated by the cloud connector interface.

The main metric to calculate the streaming capacity is called `performance`. The performance is computed on the basis of the performance measurements of each scenario part. The *actions catalogue* component offers specific actions for the analysis report of each scenario part. The computation proceeds according to the following procedure:

One scenario part determines the streaming capacity at a certain point in time depending on the available streaming capacity of the individual streaming cluster machines of the scenario part. The application additionally discretizes the numerical metric into a nominal metric with the possible values *capable* and *overloaded*. This nominal value is determined by comparison to a threshold. If the actual streaming capacity of the cluster falls below the threshold, the nominal value is set to *overloaded*.

Figure 3 shows a sample graph of progressing streaming capacity with a threshold of 23%. At time step 24 the streaming capacity of the cluster falls below the threshold. The cluster is, per definition, overloaded at this point in time.

Each scenario part is separately analyzed and reported. Possible actions are extending or shrinking the cluster size in a specific scenario or leaving the cluster size untouched in case of optimal load conditions. In order to create a report for a scenario part, the *cluster analyzer* component computes the streaming capacity according to the following procedure:

First of all, a value named `derivation` is computed to indicate the difference of the status quo to the minimum required streaming capacity.

$$derivation = threshold - performance_{scenariopart}. \quad (1)$$

Another value called `streaming capacity reserve` is defined by $reserve = -1 * derivation$ and defines a metric for a reserve or lack of streaming capacity of a complete cluster.

In a second step the total media server streaming capacity of a scenario part cluster is compared to the threshold value with the possible conditions in equation

$$\sum_{i=1}^{n}(performance_i) <> threshold$$

If the sum of all available streaming capacity is smaller than the threshold, the derivation value is rounded to the next higher number. This rounded value defines the number of additional media servers to be started in addition to the actual number of cluster machines.

If the streaming capacity of a cluster complies to the threshold, the cluster size remains unchanged. If the threshold is defined low enough to accept a large number of streams on one streaming server in a stable streaming environment even if the stream bandwidth changes, this reflects the most cost effective point of operation.

The third situation indicates that the cluster is underutilized. In this case the `reserve` value of the cluster corresponds to the number of media servers to be removed from the corresponding streaming cluster.

Evaluation

Test setup

All studies have been performed with public cloud services offered by the AWS [10] and Wowza Media Server

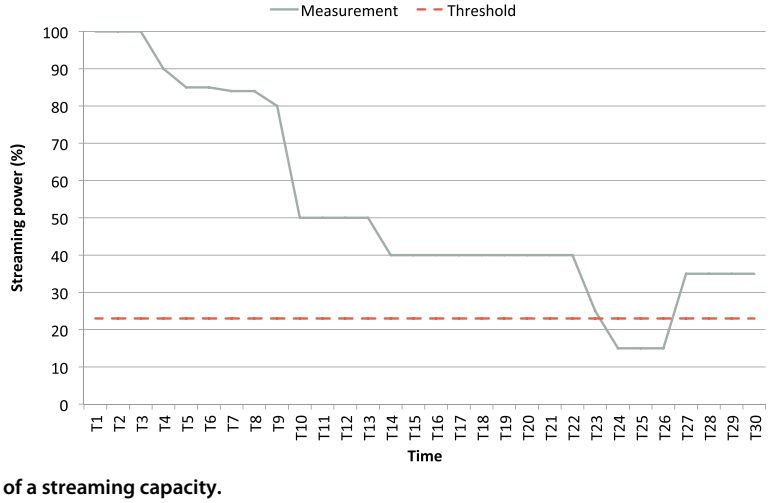

Figure 3 Sample progress of a streaming capacity.

[9]. For scalability and quality measurements we have used Flazr[16], a small command line tool to simulate a defined set of parallel video streaming consumers. In addition to this basic feature, advanced options like saving or consumption of encrypted video streams are also provided by this tool.

All testing procedures were subject to a well-defined topological setup:

- A set of 3Distribute API-Server and UI-Server located in a private context.
- One stream publisher that streams stereographic 3D content to a public iaaS based on AWS EC2.
- Virtual machines hosted in the AWS EC2 availability zone "eu-west-1", implementing a streaming service provided by a streaming cluster (Origin server and edge servers).
- Another cluster of virtual machines hosted in the AWS EC2 availability zone "us-west-1", simulating users consuming the content provided by the streaming service.

The deployment of the streaming cluster is the first step performed by the setup procedure. 3Distribute initializes and configures a video streaming cluster in the European availability zone, initially composed of a single origin server and one edge server. In order to put a load on the video streaming cluster a consumer virtual machine is instantiated in the availability zone "us-west-1".

After the streaming cluster has been deployed and configured, the stream publisher is manually enabled to launch the transmission of stereoscopic 3D live video content to the origin server of the streaming cluster. The consumption of streams is additionally started on the virtual machine in the other availability zone using Flazr. The number of streams to be consumed is initially calculated regarding the bandwidth of the stream and the network output capacity of the edge server.

Various quality attributes have been investigated and evaluated during the testing that are described in the following subsections.

Scalability

The current implementation of 3Distribute has limitations regarding the number of media servers. Due to the layered origin/edge server setup and the maximum networking bandwidth of a virtual machine in the cloud, a streaming cluster is able to provide a maximum of

$$n_{max} = \frac{origin_{max} * edge_{max}}{streambandwidth^2}$$

streams. The factors $origin_{max}$ and $edge_{max}$ denote the maximum possible outgoing networking bandwidth of the virtual machines available to the origin or edge servers. Cluster Compute Instances (CCI) of the AWS EC2 service provide a 10 GBit ethernet connection. Thus, with CCI it would be possible to achieve a maximum of 25 million independent output streams in presence of a 2 MBit input stream. This limit could be expanded by introduction of a meshed cluster architecture. In a meshed cluster the edge servers would have the possibility to connect to a further layer of edge servers, implementing a tree-like streaming network to transmit a potentially unlimited number of streams.

The number of streams in a meshed setup is calculated by $n_{total} = n_{OriginServer} * n^h_{EdgeServer}$. The exponent h corresponds to the height of the spanned tree. As a boundary condition it is required that each height of a leaf in the tree is identical.

However, meshed video streaming clusters have another limitation: If a video streaming cluster is located in just a single availability zone of a cloud, it is possible to saturate the maximum available networking bandwidth of the provider. Whereas it is possible to support 25 Million streams with a tree height of one, a streaming cluster could theoretically already transmit $125*10^9$ streams with a height of two, resulting in a networking output bandwidth of 250 PBit per second. Such a load could only be handled, if the cluster would be operated in a multi cloud environment or across various availability zones.

In order to determine the streaming capacity of a cluster it is necessary to measure the streaming capacity of a single streaming server in the cloud with the test setup described in subsection "Test setup".

If the AWS EC2 service is used, it is relevant to understand which instance type should be deployed to provide the service for origin and edge servers. A 2.4 Mbit video encoded in full HD format has been selected as an input to study the streaming behavior of specific AWS EC2 instance types. The I/O classifications of AWS EC2 instance types can be reviewed in [17]. Table 1 shows the maximum number of stable streams and the corresponding maximum measured networking bandwidth per instance type. From a provisioning point of view, all instance types in the table have a GBit networking interface. It is an interesting observation that only the large instance types in reality reach GBit bandwidth; the small instance types seem to be throttled by the provider.

As described above, Flazr was used to generate a load on the streaming cluster. Each virtual machine running Flazr

Table 1 Average maximum number of streams and bandwidth provided by AWS instance types

AWS Instance Type	I/O	n_{max}	$\sum$ Bandwidth
m1.small	Moderate	110	ca. 308 Mbit/s
m1.large	High	300	ca. 830 Mbit/s
c1.xlarge	High	330	ca. 918 Mbit/s

was configured to pull up to a hundred streams from an edge server. With a binary search approach, the number of maximum streams was identified. As a start condition an initial maximum number of 500 streams was defined. To check if a streaming server is capable to provide the number of streams, a single stream has been watched visually as a control sample. If this control sample didn't show any deterioration of audio or video, the number of streams was set acceptable to be provided by the streaming server. During an initial measurement the test video consumed a bandwidth of 2.78 Mbit on average.

We designed and developed a special tool to measure the total bandwidth using the Wowza Media Systems API able to monitor bandwidth according to a one second precision. The tool takes measurements each five seconds yielding 12 measurements per minute. In addition, the median and average values are calculated and checked against the measurements of the AWS CloudWatch API to have a plausibility test. We observed a deviation of 5.4% between both measurements that may be explained by the measurement method of the CloudWatch API. CloudWatch measures in a one minute interval with an undefined number of values. The measurements contain values for all possible I/O performance and Wowza image combinations. Currently there is no possibility to deploy a Wowza image on a virtual machine having a "very high" or "low" I/O Performance.

Figure 4 shows the results and demonstrates, that the potential network traffic and the number of possible streams vary by almost a factor of three. The measurements illustrate that the utilized bandwidth is increasing linearly with the number of streams provided. If the server is not able to provide additional streams any more, the bandwidth for each stream (and for all streams summarized) would be decreasing. This behavior may be explained by the scheduling of multiple virtual machines on a single hypervisor of a host in a cloud, resulting in a specific profile for each I/O type to schedule the resources.

Figure 5 illustrates the limit of possible stable streams for a specific instance type. An origin server using a virtual machine of instance type "m1.small" is capable of providing 110 streams. If more streams are requested on the same server, the bandwidth per stream is decreasing until artifacts, audio or video glitches are observed. It became evident that artifacts occurred once the bandwidth of a video stream fell below 2.7 MBit. In Figure 5 the graph shows an analogue behavior for instance type "c1.xlarge" and "m1.large". Both figures indicate a variation by a factor three in streaming bandwidth per stream between small and large instance types. Since the allocation of virtual machines with different instance types on a physical host is not fixed, all measurements have been repeated multiple times using a freshly deployed cluster per measurement.

These measurements are specific for AWS EC2 and the conclusions may not be generally applicable to other cloud providers and environments. However, we can anticipate that different hypervisors or virtualization technologies of other cloud providers may deliver different results but of similar characteristics.

Availability

Cloud systems should be fault tolerant as a service should be available at all times. An evaluation regarding availability of the 3Distribute software and video streaming clusters is performed in this subsection. The consequences of a component failure will be discussed for customers, monitoring processes, or for the stream providers. The term *failure* in this context refers to the termination of virtual machines in the cloud or to the crash of a software component.

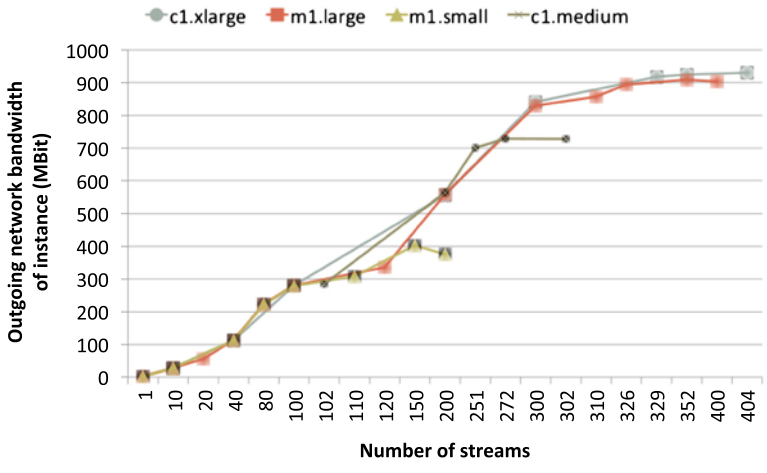

Figure 4 Bandwidth gained for different instance types.

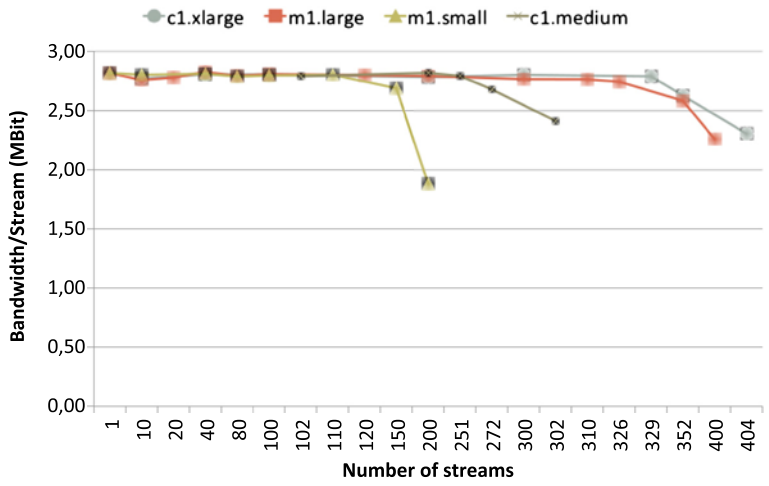

Figure 5 Maximum number of streams for different instance types.

A crash of the complete 3Distribute management system would result in a situation where the streaming cluster is not monitored and managed anymore. While operation continues in this case, scaling up or down in size would not be possible. This could cause two problems over time: Either the cost could be too high due to over provisioning, or the quality of the video stream could degrade due to a lack of video streaming server capacity. This scenario would result in a potential graceful degradation of the system.

However, failures of origin servers in a hierarchical origin/edge server setup is a yet unsolved open issue. If the origin part of a cluster is missing, no edge server would get any video data and no client could receive a video stream anymore. One possible preventive method to solve this issue is the setup of two or more spare origin servers. Each origin server would be receiving the same stream from the stream provider independently. Using a playlist, a multi-origin setup could be manually realized in all edge servers. A special configuration would deploy a second or third video streaming cluster just consisting of one origin server in hot standby mode. The switching between the origin servers may be done automatically by the Wowza streaming server itself. If one stream is ending (because of a failure), the next origin server is requested to jump in and deliver the stream. During the failover procedure a user would observe a small interrupt or artifact caused by the RTMP handshake between edge and origin server. This time period of switching is comparable to the initial handshake during a client connect initialization. In the clients view, the switchover is done transparently since the stream delivery from edge server to a final client is not interrupted.

Failures of edge servers would affect just the viewers consuming streams from the corresponding location. The streaming client would simply reconnect requesting a new edge server from the origin server. On the client side the player would react on an edge server fault automatically.

Related work

A substantial number of providers already offer video streaming services in the Internet, but almost any of these host 2D video applications only. The most prominent video service, YouTube [18], only works in the area of static content and has recently established the possibility to upload 3D videos in modest quality. KUK Filmproduktion is producing stereoscopic 3D live concert events, but only via satellite and into movie theaters [19]. There are some companies experimenting with 3D HD video live transmission of endoscopic medical surgery [20]. In the area of media oriented protocols there are developments in the direction of predictable reliability and predictable delivery (PRPD) to minimize the amount of required bandwidth [21]. In this context there are research efforts in the area of adaptive coding to optimize the coding parameters such that they are producing the minimal amount of redundancy. Another field of interest are Peer-to-Peer (P2P) solutions for live streaming video. AngelCast [22] is a system extending a P2P network with the resources of a cloud provider. AngelCast networks are capable to streami videos live and may be extended by cloud resources using a controller. The Elastic Stream Cloud (ESC) [23] represents a concept to connect multiple smaller private clouds for video streaming. ESC does not make use of specific cloud resources but provides the delivery model that has been implemented as a decentralized SaaS, forcing streaming providers to run a proprietary web OS with self defined protocols.

The authors in [24] describe an application called MediaWise Cloud Content Orchestrator (MCCO) for the

delivery of static (recorded) content. MCCO deploys a multi-cloud content delivery network (CDN) using public cloud resources and consists of a specific three tier architecture controlled by widgets in the front-end tier.

The CDN architecture typically deploys a load balanced setup with distributed cache servers spread geographically around the world in order to optimize latency, bandwidth, and cost of networking [25,26], leading to data replication into various centers and a static tier hierarchy.

A live streaming application as described in this paper, however, demands dynamic adaptation of networking and streaming capacity. In order to achieve this, content is distributed with small partial buffers (0-3 seconds) in the media servers across the deployed tree topology, growing in height and width as clients connect.

Monitoring services are a major architectural component of any cloud application to support scaling and QoS [27]. As the cloud monitoring systems offered by providers very often bear technical limitations application architects tend to work out their own individual setup. Hence, we defined a specifically well suited QoS metric for our application called streaming capacity as described in section "Measurement of server loads and capacity".

Conclusions and future work

Cloud services have a huge potential to support the broadcasting of stereoscopic 3D live events over the Internet. The rapidly growing IP-TV market with its 3D enabled smart TV sets could benefit a lot by corresponding service offerings. The architecture presented in this paper has been implemented to support Trivido [28], a 3D live video portal run by the startup Invistra [1]. The client side of the application is currently based on proprietary Flash and Silverlight enabled endpoints. The transition to HTML5 is on the roadmap and would offer a broader support of devices by the corresponding 3D TV application stores. As there is a growing uptake of LTE broadband internet connection, 3D video streaming could soon be a promising option on mobile phones and tablets as well. Further areas of interest could be 3D live chats in social networks, as well as technical training and lecturing. In order to further optimize the efficiency of the setup, another direction of development is to intelligently leverage affordable infrastructure services from the growing cloud spot market.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
All the listed authors made substantive intellectual contributions to the research and manuscript. Specific details are as follows: MH designed the architecture and implemented solution. He evaluated the work and drafted the paper. MK is 3D-Live project lead. He was responsible for the overall technical approach and architecture and edited the paper. Both authors read and approved the final manuscript.

Acknowledgements
Part of the work presented in this paper has been funded by the KIT innovation management.

Received: 28 June 2013 Accepted: 13 September 2013
Published: 25 September 2013

References

1. Invistra GmbH (2012) Invistra website. http://www.invistra.de/
2. Adobe Systems Inc (2012) Adobe flash media server. http://www.adobe.com/products/flash-media-server-family.html
3. BitTorrent Forum (2012) BitTorrent. http://www.bittorrent.org/
4. Schulzrinne E (2003) RTP: a transport protocol for real-time applications. http://tools.ietf.org/html/rfc3550
5. Parmar H, Thornburgh M (2009) Real time messaging protocol chunk stream. http://www.adobe.com/devnet/rtmp.html
6. Parmar H, Thornburgh M (2012) RTMPE description. http://www.adobe.com/devnet/rtmp.html
7. VideoLAN Organisation (2012) Video LAN website. http://www.videolan.org/
8. FlowPlayer Ltd (2012) FlowPlayer website. http://flowplayer.org/
9. Wowza Media Systems (2012) Wowza media systems home page. http://www.wowza.com/
10. Amazon Web Services (2012) Amazon web services homepage. http://aws.amazon.com
11. Webservices A (2012) Amazon EC2 home page. http://aws.amazon.com/de/ec2/
12. Google Inc (2012) Google App engine billing. https://developers.google.com/appengine/kb/billing
13. Profitbricks (2012) Profitbricks. https://www.profitbricks.com/
14. Microsoft Inc (2012) Microsoft silverlight. http://www.microsoft.com/silverlight/
15. Nvidia Inc (2012) NVIDIA 3D vision. http://www.nvidia.com/object/3d-vision-main.html
16. The Flazr Project (2012) The Flazr project website. http://flazr.com/
17. Webservices A (2012) Amazon EC2 instance types. http://aws.amazon.com/de/ec2/instance-types/
18. YouTube Inc (2011) YouTube. http://www.youtube.com/
19. KUK Filmproduktion (2012) KUK Homepage. http://www.kuk-film.de/en/stereoskopie.php
20. TV-Studios Leonberg (2012) Audiovisuelle medien. http://www.tvstudios.de/filmproduktionstechniken/3d-produktionen.html
21. Miroll GH (2011) Service compatible efficient 3D-HDTV Delivery. In: 14th ITG Conference on Electronic Media Technology
22. Sweha R, Ishakian V, Bestavros A (2012) AngelCast: Cloud-based peer-assisted live streaming using optimized multi-tree construction. In: Proceedings of the 3rd Multimedia Systems Conference. ACM, New York, pp 191–202
23. Feng J, Wen P, Liu J, Li H (2010) Elastic stream cloud (ESC): A stream-oriented cloud computing platform for Rich Internet Application In: High Performance Computing and Simulation (HPCS), 2010 International Conference on, pp 203–208. doi:10.1109/HPCS.2010.5547135
24. Ranjan R, Mitra K, Georgakopoulos D (2013) MediaWise cloud content orchestrator. J Internet Serv Appl 4: 1–14
25. Vakali A, Pallis G (2003) Content delivery networks: status and trends. Internet Comput, IEEE 7(6): 68–74
26. Mulerikkal J, Khalil I (2007) An architecture for distributed content delivery network In: Networks, 2007. ICON 2007. 15th IEEE International Conference on, pp 359–364
27. Alhamazani K, Ranjan R, Rabhi F, Wang L, Mitra K (2012) Cloud monitoring for optimizing the QoS of hosted applications. In: Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on, pp 765–770
28. Invistra GmbH (2012) Trivido Website. http://www.trivido.com/

doi:10.1186/2192-113X-2-14
Cite this article as: Hoecker and Kunze: **An on-demand scaling stereoscopic 3D video streaming service in the cloud.** *Journal of Cloud Computing: Advances, Systems and Applications* 2013 **2**:14.