On Designing a Cost-Aware Virtual CDN for the Federated Cloud

Dragos Ilie* and Vishnubhotla Venkata Krishna Sai Datta†
Dept. of Communication Systems
Blekinge Institute of Technology, Karlskrona, Sweden
*dragos.ilie@bth.se, †vvksdatta@gmail.com

***Abstract*—We have developed a prototype for a cost-aware, cloud-based content delivery network (CDN) suitable for a federated cloud scenario. The virtual CDN controller spawns and releases virtual caching proxies according to variations in user demand. A cost-based heuristic algorithm is used for selecting data centers where proxies are spawned. The functionality and performance of our virtual CDN prototype were evaluated in the XIFI federated OpenStack cloud. Initial results indicate that the virtual CDN can offer reliable and prompt service. Multimedia providers can use this virtual CDN solution to regulate expenses and have greater freedom in choosing the placement of virtual proxies as well as more flexibility in configuring the hardware resources available to the proxy (*e. g.,* CPU cores, memory and storage).**

***Index Terms*—cloud; CDN; OpenStack; proxy; virtualization.**

I. Introduction

Content Delivery Networks (CDNs) typically act as a shock absorbers during the contingency of traffic upsurge. During such times, immediate attention has to be laid on provisioning adequate resources. With the advent of cloud computing technologies, multimedia providers[1] (MPs) have scope for establishing CDN spanning multiple clouds. However, the main challenge while establishing such a CDN is implementing a cost efficient and dynamic mechanism which guarantees good service quality to users.

Although there are organizations offering traditional or cloud-based CDNs, it is usually expensive to use these type of services. MPs also face difficulties in estimating the traffic and user demand, as those CDN providers offer limited information regarding user behavior and request pattern [1]. We have developed a model which facilitates MPs to establish a self-managed *virtual CDN* by leveraging a OpenStack-based [2] federated cloud. According to this model, MPs can use services from several cloud providers to establish multiple points of presence and in this way create a distributed pool of resources. Virtual proxies, which are virtual machines (VMs) with associated storage, can be dynamically spawned and released on the basis of a heuristic algorithm. This heuristic algorithm estimates the number of proxies required to meet the user demand with the help of a mathematical model.

This work was partly funded by the European FP7 project "XIFI – eXperimental Infrastructures for the Future Internet" (Grant number 604590).

[1]Companies from the entertainment industry or regular users, such as independent artists, streaming movies and audio to a set of interested consumers.

Further, the performance of the virtual CDN was assessed over the federated XIFI cloud. XIFI is a project of the European Public-Private-Partnership on Future Internet (FI-PPP) programme [3]. The goal of XIFI is to provide and run a stable infrastructure for the large scale trials through federating OpenStack data centers over Europe into a massive test platform for Future Internet services. OpenStack is a open-source software platform for cloud orchestration.

The purpose of the paper is to investigate a mechanism for establishing a OpenStack-based CDN and examine the latencies and robustness involved in setting up virtual proxies. Our results indicate that virtual CDNs can offer reliable and prompt service. With virtual CDNs, MPs can regulate expenses and have greater level of flexibility for customizing the virtual proxies deployed at different locations.

II. Framework for Virtual CDN

In this section we present the components for the virtual CDN framework as well a number of assumptions and limitations related to it.

We focus on the scenario where a MP uses unicast to deliver video streams. Although multicast is more efficient, its deployment and management costs are significantly higher [4]. Due to the costs associated with running a multicast infrastructure in parallel with the unicast one, some ISPs refuse to offer multicast services.

The quality of streaming service depends on the routing distance between servers and users since latency and jitter tend to increase with distance. Routing distance can be expressed either in hop count or round-trip time (RTT). However, according to [11], RTT is considered to be the best choice for measuring the client perceived performance. It should be noted that the focus of this study is not on determining the quality of experience for end-users, but rather on investigating how efficiently can proxies be provisioned in a virtual environment.

Users from a particular location experience similar RTTs to a common server. In order to effectively observe user behavior and handle requests from a location, users can be grouped into a cluster [5]. We have defined clusters on the basis of Classless Inter-Domain Routing (CIDR) address blocks. Each cluster is associated with multiple CIDR blocks and thus, requests from different topological locations can be distinguished. Some organizations offer geolocation databases, which provide information about geographic location of users

based on their IP addresses. The databases can be used to build clusters based on geographical location. We have not used geolocation for the virtual CDN, but mention it here as a potential improvement.

The VMs belonging to a virtual CDN are configured as caching reverse proxies. These are surrogate servers placed inside or in close proximity to a cluster. Their mission is to distribute content on the behalf of the origin server where the MP stores the content. We call such a proxy as *virtual caching proxy*. Since we consider a federated cloud, the virtual caching proxies can run on hosts located at different sites that may be managed by different organizations.

Collaboration among the surrogate servers of a CDN can be implemented through push- or pull-based mechanisms [6]. When a push-based mechanism is used, the origin server is responsible for pushing updated content to surrogate servers. This reduces latency because the content is readily available on a surrogate server when a user tries access it. On the other hand, if the updated content is of low interest to users, the network capacity used for pushing content to surrogate servers is practically wasted. When pull-based mechanisms are used, the user is directed to the closest surrogate server. The surrogate server retrieves the content for the origin server if it does not have it already. If a cooperative pull-based mechanism is used, the surrogate server will attempt to obtain the content from peer surrogate servers in order to offload the origin server. Pull-based mechanisms are considered to be cost effective as they minimize storage and bandwidth costs. However, the cooperative pull-based mechanism might make the CDN system complex and unpredictable, especially in case of cache misses [7]. Instead, we used a non-cooperative pull-based mechanism for building a centralized virtual CDN.

CDNs deliver content to users by selecting proxy servers that can provide reasonable performance. The selection doesn't always choose the server with shortest response time or one that is topologically closest to users [8] (*e. g.*, the selection may take cost into consideration). For catering reliable services, we assigned each user cluster to at least two sites. Thus, in case of service outage or overload at one site, users can safely access services from proxies deployed on the secondary site.

We developed an algorithm that manages caching proxies based on the demand from user clusters. The demand from a cluster can be interpreted as the number of unique users accessing the services of MP. Rise in demand indicates increase in the number of users accessing the services. The number of connections handled by a proxy depends on the user demand and fluctuates over time.

Cloud providers levy charges for real time usage of network, computation and storage resources. To simplify management, users are offered hardware templates (referred to as *flavors* in OpenStack) that define the number of virtual CPUs (vCPUs), the size of RAM memory and the amount of storage allocated in the cloud. Cloud providers offer flavors at specific prices and bills tenants for network utilization based on bandwidth usage. These resources have to be managed effectively to minimize the overall CDN expenditures. In our framework, we take into account that prices of cloud resources vary among providers and locations.

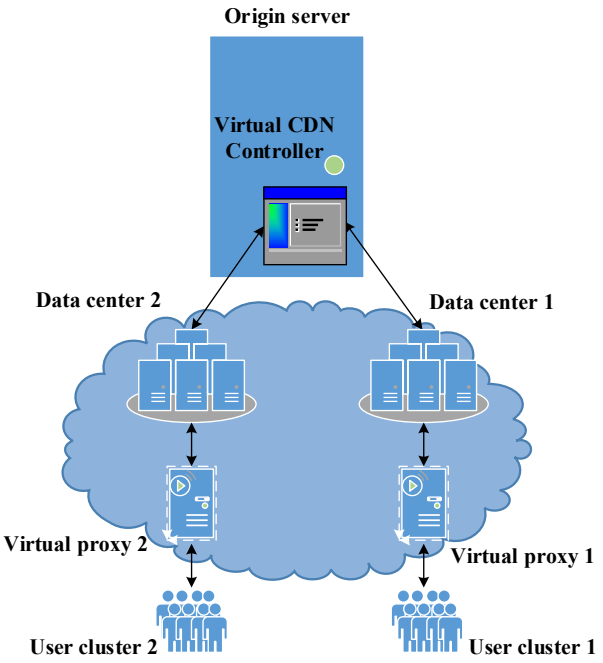

Fig. 1. Virtual CDN framework

We use a *virtual CDN controller* to coordinate proxy deployments. The main objective of this controller is to regulate the virtual caching proxies of CDN according to the demand from users. The controller is responsible for spawning virtual caching proxies when there is rise in demand. It further facilitates minimizing CDN operational costs by shutting down the virtual proxies in case of fall in demand.

The controller functions on the basis of a heuristic algorithm that decides the placement of caching proxies based on demand from each user cluster. The algorithm first estimates the number of virtual proxies required to meet the demand and then decides the placement location of these proxies with the help of a cost function. Further, the virtual CDN controller maintains a database which stores information about cloud resources, costs of various resources at each location, nodes associated with each cluster, measured RTT to proxies and clusters as well as other useful information. Figure 1 presents on overview of the virtual CDN framework. In our implementation, the controller manages cloud resources at various sites with the OpenStack HTTP-based API.

III. The Cost Function

The design of a CDN should be a compromise between performance and costs. Cloud providers offer different flavors at specific prices and charge tenants mainly for outgoing traffic from virtual servers. We have considered the flavor cost and network costs in developing the cost function. Other important parameters that affect the overall expenses of a CDN [10] and that are used by the cost function are presented below. All used parameters are listed in Table I.

The *origin server upload cost* α is the cost associated with origin server's outgoing traffic while serving content to

user clusters and proxies. MPs pay α dollars[2] per GB while transferring videos to proxies and users. Network charges associated with incoming requests from users to proxies are not considered as the video traffic from servers to users dominates other traffic [10].

Outgoing traffic from virtual proxy i, referred to as *virtual proxy upload cost*, is denoted by β_i. Cloud providers charge β_i dollars per GB when a virtual proxy streams videos to users.

The *opening cost* Ω_i for virtual proxy i is the cost associated with a particular flavor associated with the proxy VM. This is the cost paid by MP for data storage, CPU and memory resources. Virtual proxy opening cost is expressed in dollars per month.

Parameters μ_{O,P_i}, μ_{P_i,U_n}, and μ_{O,U_n} represent values for the RTT between origin server and virtual proxy i, between virtual proxy i and user n, and between origin server and user n, respectively. Section V explains how these values were estimated.

The cost of streaming a video of size S from the origin server to N users of a cluster is denoted by δ_O. This cost is the aggregate of the cost for each user n weighted by μ_{O,U_n}, the RTT between origin server and user n.

$$\delta_O = \sum_{n=1}^{N} \left(\alpha * \mu_{O,U_n} * S \right) \tag{1}$$

The cost of streaming a video of size S from the virtual proxy i to N users of a cluster is represented by δ_{P_i}. Similarly to δ_O, this cost is the aggregate of the cost for each individual user weighted by μ_{P,U_n}, the RTT between virtual proxy i and user n.

$$\delta_{P_i} = \sum_{n=1}^{N} \left(\beta * \mu_{P_i,U_n} * S \right) \tag{2}$$

We use the parameter ρ_{O,P_i} to denote the cost of transferring a video of size S from the origin server to a virtual proxy i (*i. e.,* replication cost). As before, this cost is weighted by μ_{O,P_i}, the RTT between origin server and virtual proxy i.

$$\rho_{O,P_i} = \left(\alpha * \mu_{O,P_i} * S \right) \tag{3}$$

The total cost, ϵ_i, of starting the new virtual proxy i consists of the virtual proxy opening cost, replication cost and streaming costs.

$$\epsilon_i = \left(\Omega + \rho_{O,P_i} + \delta_{P_i} \right) \tag{4}$$

IV. Heuristic Algorithm for Virtual CDN

The virtual CDN controller uses a heuristic algorithm to makes all decisions related to caching proxies. The goal of this algorithm is to spawn or turn off caching proxies in response to varying user demand in a cluster. These caching proxies are instantiated at datacentres which are in close proximity to users. We express user demand as percentage of the total cluster population. Statistics of CDN traffic presented in [12] show that over the time span of 9 days, the CDN under study received requests from approximately 1 million unique IP addresses. Based on these statistics, we defined the maximum number of users in a cluster to be 1 million. Thus, 1% increment in demand corresponds to an increase by 10,000 users. A 100% demand from a user cluster corresponds to 1 million users requesting a video.

[2]We use dollar as currency unit since the cloud providers considered here list their prices in USD.

TABLE I
Parameters of cost function

Parameter	Description
α	Cost of uploading from origin server (USD/ GB)
β_i	Cost of uploading from virtual proxy i (USD/ GB)
Ω_i	Opening cost for virtual proxy i (USD/month)
μ_{O,P_i}	RTT between origin server and virtual proxy i (s)
μ_{P_i,U_n}	RTT between virtual proxy i and user n (s)
$\overline{\mu_{P_i,U}}$	The set consisting of all μ_{P_i,U_n} parameters for proxy i
μ_{O,U_n}	RTT between origin server and user n (s)
$\overline{\mu_{O,U}}$	The set consisting of all μ_{O,U_n} parameters
S	Size of video (GB)
N	Number of users being served
δ_O	Cost of streaming from origin server (USD)
δ_{P_i}	Cost of streaming from virtual proxy i (USD)
$\rho_{O,P}$	Cost of replicating content to virtual proxy i (USD)
ϵ_i	Total cost of streaming from virtual proxy i (USD)

To explain our heuristic algorithm, we consider a scenario with a user cluster associated with two sites (*i. e.,* two data centers). Algorithm 1 is executed whenever there is a change in user demand in the cluster. We let P_1 denote a virtual proxy at site one, and similarly we use P_2 to denote a virtual proxy at site 2. Given a increase in the user demand, the algorithm decides if either P_1 or P_2 is instantiated to handle the new demand, or, if it is cheaper (according to the cost function presented in Section III) to serve the demand from the origin server. On the other hand, if the demand decreases, the algorithm determines if it can adapt to the new demand by turning of one or more virtual proxies. Turning off proxies can save operational costs.

The algorithm requires the cost function mentioned above as well as the current user demand d. The user demand is used in step 1 to derive the current number of users in the cluster. Step 2–6 determine the maximum number of users that can be served with the current number of proxies. The procedure for estimating this number is not shown in Algorithm 1, but the fundamental details are explained below. Step 7 calculates the change in the number of users. The case of an increase in the number of users is handled in step 8–25, while the reminder of the algorithm handles the case of a decrease in the number of users.

The virtual caching proxies are VMs running NGINX, a well-known web server and reverse proxy [13]. NGINX can run multiple worker processes, each capable of processing a large number of connections. In general, it is recommended to keep the number of worker processes equal to the number of CPU cores [13] and therefore the virtual proxies are configured to behave this way. Since virtual proxies are VMs, we are in fact configuring the number of vCPUs instead of real CPU cores. In addition, we have configured NGINX to allow each

Algorithm 1 Proxy spawning and removal

Require: Cost function (CF), user demand d

1: $U \leftarrow$ number of current users derived from d
2: **if** proxies exist **then**
3: $M \leftarrow$ maximum users handled by existing proxies
4: **else**
5: $M \leftarrow 0$ ▷ no users can be handled without proxies
6: **end if**
7: $N \leftarrow (U - M)$ ▷ number of arriving or departing users
8: **if** $N > 0$ **then**
9: Estimate number of new proxies based on N
10: **for** each new proxy **do**
11: **if** resources exist **then**
12: Compute δ_O
13: **for** each site j **do**
14: $\epsilon_j \leftarrow \text{CF}(S, \alpha, \beta_j, \Omega_j, \overline{\mu_{O,U}}, \overline{\mu_{P_j,U}}, \mu_{O,P_j})$
15: **end for**
16: **if** $\min \epsilon_j \leq \delta_O$ **then**
17: Spawn new proxy at site with $\min \epsilon_j$
18: Update database and redirect users
19: **else**
20: Stream from origin server
21: **end if**
22: **else if** no resources **then**
23: Stream from origin server
24: **end if**
25: **end for**
26: **else if** $N < 0$ **then**
27: $I \leftarrow$ Proxies required to handle current demand
28: $J \leftarrow$ Current number of proxies
29: **if** $I < J$ **then**
30: Shutdown $\theta * (J - I)$ proxies and update database
31: **end if**
32: **end if**

worker process to maintain up to 1024 worker connections. With these settings, a virtual proxy with 2 vCPUs streaming HD video with an average bitrate of 5 Mbps will effectively saturate a 10 Gbps link, as long as the vCPUs and storage can keep up. This is an extreme example, but serves to illustrate that 1024 worker connections is a reasonable upper bound for most environments.

Because we consider the case of unicast video streams, our algorithm assumes that each user from the cluster opens one connection to the virtual proxy. When N new users are added to a cluster, we divide their number by 1024 (*i. e.*, the maximum number of worker connections) and round up to the closest integer. The integer is the number of worker process required to handle the new demand, which is equivalent to the number of vCPUs required. The algorithm looks for a VM flavor large enough to satisfy the requirement. If no such flavor is found, then multiple proxies must be used.

A new proxy is spawned when the cost of streaming from the origin server, δ_O, exceeds the total cost ϵ of streaming from the proxy (step 17). The proxy is spawned at the site with the minimum ϵ cost. After the proxy is created, the database managed by virtual CDN is updated with configuration details, location, IP address and maximum number of users served by that caching proxy. The controller node configures the origin server to redirect all the requests from user cluster to newly created proxies. This is accomplished by associating the new proxy's IP address with the CIDR address blocks belonging to the cluster.

Whenever there is a decrease in the percentage of demand, some proxies are shut down based on a hysteresis mechanism. Hysteresis switching prevents instant shutdown of virtual proxies and is useful to handle a sharp fall in demand. This shutdown is regulated by the hysteresis gain factor, θ which ranges from 0 to 1 (step 30). When θ is very small, the hysteresis equation is very conservative while shutting down proxies and when θ is large, the proxies are shutdown instantly.

MPs can choose appropriate factor θ by estimating demand variations over historic data [14]. Shutdown signals can be issued to multiple proxies using OpenStack API commands. The caching proxies are configured to serve all the existing active connections before shutting down. This is achieved by issuing graceful shutdown signal to NGINX as soon as the virtual caching proxy receives a shutdown signal.

V. Testbed

We setup a testbed to examine the effectiveness and measure the performance of virtual CDNs while delivering multimedia content. XIFI sites located in Budapest, Spain[3], Stockholm and Volos are considered for the testbed. The origin server is located in Budapest and remaining three locations are remote sites where virtual caching proxies are spawned. The sites support spawning small, medium and large flavor VMs and allocate adequate network capacity for transferring a video file. The small flavor consists of 1 vCPU, 2 GB RAM and 20 GB storage and the medium and large flavor offer two times and four times, respectively, more resources. One user cluster is assumed at every location.

The origin server in Budapest is a VM running NGINX on top of Ubuntu Linux 14.04 LTS. NGINX is used for HTTP-based video streaming. The VM is reachable via a public IP address assigned to it. A single video file is stored on this VM. The file is 200 MB in size and contains a super HD video of 5 minutes duration. The contents were selected based on the types of typical YouTube videos described in [15]. VM images with a pre-installed NGINX web server configured in reverse proxy mode are stored in the OpenStack environment. These images are used to spawn virtual proxy VMs.

The price listings from three European cloud providers (CenturyLink, Host Europe and UpCloud) are used to emulate flavor costs for the testbed. Unlike other providers, they provide detailed pricing for vCPUs, RAM and storage resources. Similarly, egress network charges listed by Amazon, Google, Microsoft and Rackspace are used to emulate upload costs.

We used the httping tool [16] to estimate the RTTs between the origin server and caching proxies. ICMP can be problematic while measuring end-to-end delay as it can be

[3]The locations listed here were derived from the name of the corresponding site as provided by the OpenStack API. We do not know in which city the Spain site is located.

blocked on the path or rate limited. Moreover, RTTs at layer 3 do not reflect service response times, which are critical for multimedia communication [17]. An average value is calculated over 1000 RTTs between the origin server and a test proxy at each site and these values are assigned to the corresponding μ_{O,P_i} parameters. The $\mu_{P_i,U}$ parameters are assigned values based on statistics for CDN user latency, expressed as HTTP-based RTT, from the CDN performance report by CloudHarmony [18].

Default privileges assigned to our XIFI Cloud account prevent us from using more than 6 vCPUs and 3 VMs at one site. Consequently, we were not able to test the scalability of the system with large clusters of users. Our focus has been instead on the feasibility of the virtual CDN system and on quantifying various operational metrics that can be observed in small scale experiments.

The virtual CDN can be viewed as an example of network function virtualization (NFV). We have therefore selected four NFV performance metrics proposed by the ETSI Industry Specification Group[19] for measuring the service quality of a CDN running on virtualized infrastructure. Analysis of these metrics helps in estimating reliability and promptness of the virtual CDN service. The selected metrics are:

VM dead on arrival (DOA) ratio
: Tracks the liveliness of the VM. The DOA ratio indicates the ratio of number of faulty VMs provisioned to the total number of successfully spawned VMs.

VM provisioning latency
: Represents the time required for spawning a caching proxy in the virtual CDN. It is defined as the elapsed time between issuing a VM provisioning request and the successful spawning of the VM. This metric directly impacts the time required to scale a CDN system.

CDN operational latency
: Estimates how long time it takes for a caching proxy to be ready with cached video after the virtual CDN controller decides to spawn a proxy. It is the sum of VM provisioning latency and synchronization time with origin server (*i. e.*, time used by a virtual proxy for caching the video data from the origin server).

VM faulty shutdown ratio
: Defined as the ratio of number of faulty VM shutdowns to the total number of successfully shutdown VMs.

The demand from each user cluster is simulated with the help of a program running inside the virtual CDN controller. The controller executes the heuristic algorithm based on the simulated demand. At some point, the demand prompts the controller to start a small flavor caching proxy. The controller issues API calls to spawn the VM and then collects VM DOA ratio, VM provisioning latency and CDN operational latency data for the newly spawned caching proxy. Later, the demand is gradually decreased until the controller issues shutdown signal. VM faulty shutdown ratio is measured at this point. In a similar vein, the performance is observed for medium and large flavor caching proxies.

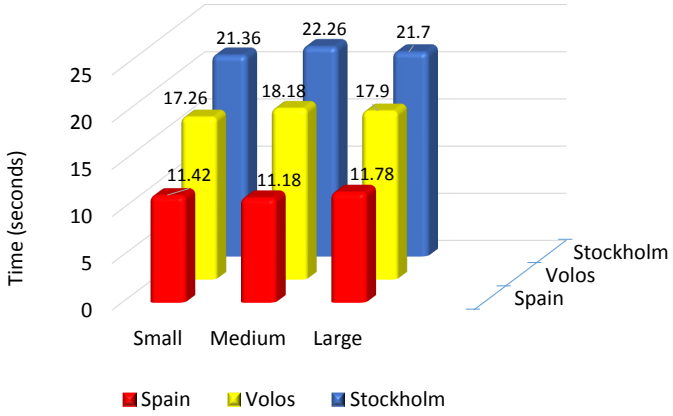

Fig. 2. VM provisioning latency

VI. Results and Performance Analysis

We performed 50 iterations of the experiment for each type of flavor and computed 95% confidence intervals for each metric. In all cases, the relative half-length of the confidence interaval was below 5%. However, we would like to point out that 50 iterations do not provide statistical guarantees for the level of reliability (*e. g.*, as in "five nines" availability) that can be expected from the CDN. Testing for high availability requires a very large number of iterations, magnitudes larger than 50 iterations.

A. VM DOA Ratio

During all the experiments, no flaws were observed while spawning caching proxies, which resulted in a zero VM DOA ratio. Zero values at all locations indicate that content delivery service at the three clusters was not affected by faulty or inoperable caching proxies. In production systems, VM DOA ratios similar to those obtained here helps MPs in rendering more reliable services to users. Furthermore, zero DOA ratio at each location implies that the virtual CDN system has potential for scaling efficiently without any glitches when there is a sudden surge in user demand from any cluster.

B. VM Provisioning Latency

On average, low VM provisioning latencies were observed at the site in Spain and high latencies were observed at the Stockholm site. It can be observed from Figure 2[4] that provisioning latencies are directly related to the location from where the caching proxies were spawned. Compared to Stockholm, the VM provisioning latencies at the site in Volos were 17–19% lower and VM provisioning latencies in Spain were smaller by 45–49%. It was also observed that VM provisioning latency of small, medium and large flavors at a particular site were almost equivalent, with at most 1 s variations. The results indicate that during the event of sudden surge in demand from a user cluster, a large flavor proxy can be spawned as quickly as a small flavor one.

[4]On the x-axis in the figure, the keywords Small, Medium and Large denote VM flavors, which define resources allocated to the VM.

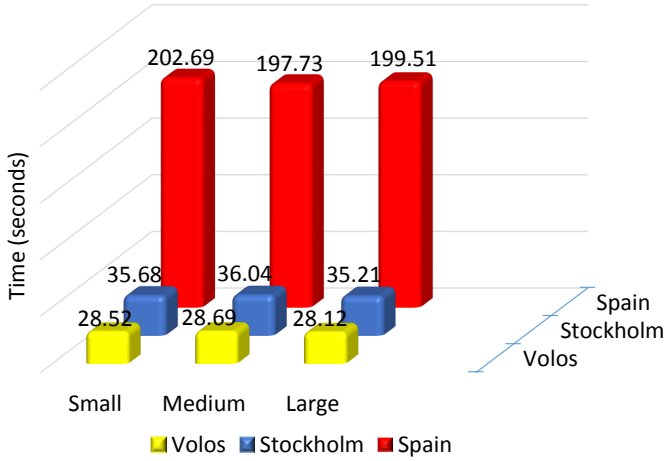

Fig. 3. CDN operational latency

TABLE II
AVERAGE RTT BETWEEN ORIGIN SERVER AND VIRTUAL PROXY

Spain RTT (ms)			Stockholm RTT (ms)			Volos RTT (ms)		
Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
228.7	314.3	1320.6	204.5	214.0	282.4	159.0	162.1	187.4

TABLE III
AVERAGE BITRATE BETWEEN ORIGIN SERVER AND VIRTUAL PROXY

Spain	Stockholm	Volos
1.26 Mbps	16.97 Mbps	21.26 Mbps

C. CDN Operational Latency

Although VM provisioning latency at the Spain site was low, as described in the previous section, its CDN operational latency was very high as it can be observed from Figure 3. High values can be attributed to time required for caching video from origin server. In spite of sufficient bandwidth capabilities and enough computational resources, communication across geographic regions typically suffers from wide-area latencies as pointed out in [20]. We observe in Table II that RTTs between Budapest and Spain varied from 228.7 ms to 1320.6 ms, which indicates that high network latency is indeed present. The latency effect on transfer rates between the origin server and a proxy in Spain can be observed in Table III, which shows a meagre average bitrate of 1.26 Mbps.

Low VM provisioning latency, fair bandwidth and low network latency were the main reasons for low CDN operational latency at Volos. In production systems, low CDN operational latencies are advantageous to MPs. This is particularly true during flash crowds when virtual proxies must be brought online rapidly to serve users and prevent their requests from swamping the origin server.

D. Faulty Shutdown Ratio

No flaws were observed while shutting down caching proxies and thus VM faulty shutdown ratio was zero. This implies that the virtual CDN has potential for shrinking its resource usage without any glitches when user demand declines.

VII. CONCLUSIONS

This paper presents a model for dynamically scaling and releasing virtual proxies on a federated cloud environment.The main purpose of the algorithm is to trigger the instantiation and release of caching proxies according to variations in demand. A video streaming systems can use the algorithm to spawn caching proxies at data centres located in close proximity to users, thus potentially lowering latency and jitter related to streaming. An implementation of this model was operated in the XIFI cloud and various latencies were examined. Our preliminary results indicate that spawning virtual proxies is an elegant solution to address user demand by dynamic provisioning of caching proxies. This solution is also time efficient. For example, at the site in Volos we measured a CDN operational latency of 28.12 s when large flavor caching proxies are spawn. We think that our solution can be of interest to MPs as well as end-users looking to operate their own cost-aware CDN in order to lower operational costs.

REFERENCES

[1] F. Dudouet, P. Harsh, S. Ruiz, A. Gomes, and T. M. Bohnert, "A case for CDN-as-a-service in the cloud: A Mobile Cloud Networking argument," in Proc. of ICACCI, Delphi, India, Sep. 2014.
[2] OpenStack [Online]. Available: http://www.openstack.org.
[3] XIFI [Online]. Available: https://www.fi-xifi.eu/home.html.
[4] L. Al-Jobouri, M. Fleury, and M. Ghanbari, "Broadband wireless video streaming: achieving unicast and multicast IPTV in a practical manner," Multimed. Tools Appl., pp. 1–28, Apr. 2015.
[5] A. J. Cahill, "An efficient CDN placement algorithm for high-quality TV content," in Proc. from Internet and Multimedia Systems and Applications - EuroIMSA, Grindelwald, New York, NY, USA, Oct. 2005.
[6] A. K. Pathan and R. Buya, "Chapter 2: A Taxonomy and Survey of Content Delivery Network," Content Delivery Network, Springer-Verlag, ISBN: 978-3-540-77886-8, 2008.
[7] T. M. T. Anh, H. P. Huu, T. Nguyen, and C. S. Kim, "Enhance Performance of Content Delivery Network Using Provider Oriented Hierarchical Corporative Proposal," Int. J. Control Autom., vol. 7, no. 6, pp. 317–330, Mar. 2014.
[8] B. Frank, I. Poese, Y. Lin, G. Smaragdakis, A. Feldmann, B. Maggs, J. Rake, S. Uhlig, and R. Weber, "Pushing CDN-ISP Collaboration to the Limit," SIGCOMM Comput Commun Rev, vol. 43, no. 3, pp. 34–44, Jul. 2013.
[9] ETSI GS NFV 001 v1.1.1 (2013-10): "Network Function Virtualisation (NFV): Use Cases." www.etsi.org, Oct. 2013
[10] F. Chen, K. Guo, J. Lin, and T. La Porta, "Intra-cloud lightning: Building CDNs in the cloud," in Proc. of IEEE INFOCOM, Orlando, FL, USA, Mar. 2012.
[11] K. Obraczka and F. Silva, "Network latency metrics for server proximity," in Proc. of IEEE GLOBECOM, vol. 1, San Francisco, CA, USA, Nov. 2000.
[12] M. J. Freedman,"Experiences with CoralCDN: A Five-Year Operational View," in In Proc. of USENIX NSDI, San Jose, CA, USA, Apr. 2010.
[13] "nginx documentation." [Online]. Available: http://nginx.org/en/docs/.
[14] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao, "Energy-aware Server Provisioning and Load Dispatching for Connection-intensive Internet Services," in Proc. of USENIX NSDI, San Francisco, CA, USA, Apr. 2008.
[15] H. Chang, H. Liu, Y.-W. Leung, and X. Chu, "Minimum latency server selection for heterogeneous cloud services, " in Proc. of IEEE GLOBECOM, Austin, TX, USA, Dec. 2014.
[16] "httping manual." [Online]. Available: http://linux.die.net/man/1/httping.
[17] V. Aivazov and R. Samkharadze,"End-To-End Packet Delay In The Network," Automated Control Systems, no. 2(13), pp. 128–134, 2012.
[18] "CDN Performance Report," Cloud Reports–CloudHarmony, Sep. 2014.
[19] ETSI GS NFV-INF 010 V1.1.1 (2014-12): "Network Functions Virtualisation (NFV): Service Quality Metrics." www.etsi.org, Dec. 2014.
[20] A. Chandra, J. Weissman, and B. Heintz, "Decentralized Edge Clouds," IEEE Internet Comput., vol. 17, no. 5, pp. 70–73, Sep. 2013.