Evaluation of scalable, on-demand DNS-as-a-Service

Bruno Sousa[1,2], Vitor Fonseca[1], Paulo Simoes[2], Luis Cordeiro[1]
[1]OneSource, Consultoria Informatica Lda. Coimbra, Portugal
[2]CISUC-DEI, University of Coimbra, Portugal

***Abstract*—The Domain Name Service (DNS) is a vital service in the Internet. Much more than a simple translation mechanism, it also allows higher profile functionalities such as load balancing and enhanced content distribution. In the scope of cloud computing, DNS is foreseen as an elastic and robust service, supporting failover mechanisms, decentralised configuration and multi-tenant isolation.**

This paper presents and validates a cloud-based architecture for DNS as Service, considering the expected principles of security, scalability and elasticity. The obtained results reveal that the proposed architecture is capable of accommodating a high-load of DNS queries per second by dynamically managing the amount of used resources, enforcing a constraint of reduced query latency ($< 1s$). Additionally, it also incorporates failover monitoring mechanisms to ensure the stability of the system.

The performed assessment in Fed4FIRE testbeds shows that this approach allows for reduction of operational costs, managing used resources according to the service's load introduced by the simultaneous clients and by appropriately scaling in or out DNS servers independently of the underlying cloud infrastructure platform (e.g. OpenStack, Amazon Web Services).

***Index Terms*—DNSaaS, scaling, PowerDNS, on-demand instantiation**

I. Introduction

The Domain Name Service (DNS) is a vital service for the Internet and dates back to the primordials of ARPAnet, for name resolution purposes [1]. Since then, several extensions have been provided to accommodate new functionalities, such as security, load balancing and enhanced content distribution [2,3]. In addition, a hierarchical architecture has been specified and deployed to support the ever increasing demand for name resolution requests in the Internet.

The goal of reducing both Capital and Operational expenditure costs (CAPEX and OPEX, respectively) has long been pursued by Telcos and service providers. Evolving from virtualised mainframes into recent developments on cloud computing, supported by platforms such as OpenStack [4], Amazon Web Services [5], Windows Azure [6] or Google Cloud Services [7], services can now adapt more easily and take advantage of available features and infrastructures, such as parallel processing [8]. Bearing this flexibility in mind, the adoption of both vertical and horizontal scaling practices becomes increasingly more important when developing cloud-based services. With the goal of supporting these scaling practices, services resorting to cloud-computing orchestration platforms, such as Heat [9] in OpenStack, are enabled with on-demand elasticity but rely only on a restricted set of metrics such as CPU load and memory usage.

Due to the myriad of applications where DNS plays an important role (including the scope of private clouds), it becomes necessary to adapt it to the cloud-based paradigm, providing a configurable and multi-tenant environment capable of supporting multiple requirements and applications [10]. Nonetheless, the hierarchical and distributed organisation of common DNS infrastructures across multiple data-centres does not necessarily meet typical cloud principles, such as elasticity. This limitation motivates the definition of a new DNS as a Service (DNSaaS) architecture, flexible, robust, configurable and capable of meeting variable demand, by monitoring existing resources and triggering infrastructural changes whenever required. Flexibility and elasticity are the main goals achieved by this architecture, mostly by assuring its suitability for different cloud platforms, adopting common scaling practices for cloud-based services while always considering the specificities of the DNS service.

The contributions of this paper include the specification and validation of a DNS as a Service architecture, tailored for different cloud platforms and compliant with scaling practices of cloud-based services. In fact, the proposed DNSaaS architecture does not rely on specific functionalities of cloud platforms to support scalability (like the autoscaling features from HEAT), using instead platform-agnostic mechanisms. This work also presents the specification of key performance indicators alongside with their thresholds and measurements for each DNSaaS component, supporting constant performance monitoring via standard monitoring systems, such as Zabbix [11] and Graylog [12]. Additionally, the obtained evaluation results demonstrate that the proposed architecture is able to accommodate the high peak loads introduced by some data and VoIP applications. This validation has been performed using large-scale environments, such as iMinds virtual Wall2 and FuSeCo testbeds from Fed4Fire [13].

Following an analysis and overview of DNS and cloud-related literature, in Section II, the definition and details of the DNSaaS architecture are presented in Section III. Section IV details the evaluation methodology, while Section V present the obtained performance and validation results for the proposed architecture, on Fed4Fire testbeds. Final thoughts and conclusions are outlined in Section VI.

II. Related Work

When dealing with scaling in the cloud, different approaches can be followed. For instance, prediction techniques can be

978-3-901882-89-0 @2017 IFIP

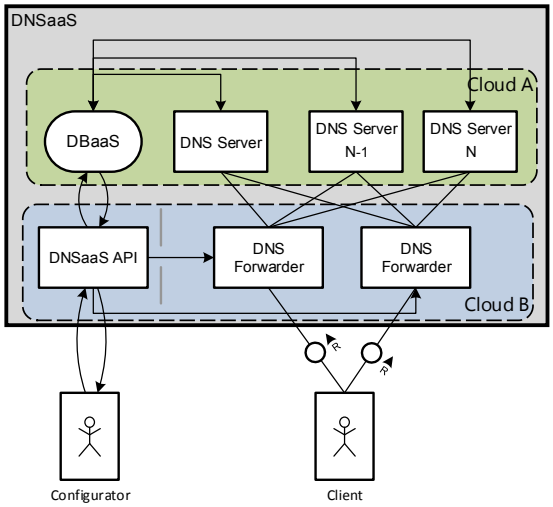

Fig. 1: Centralised Architecture

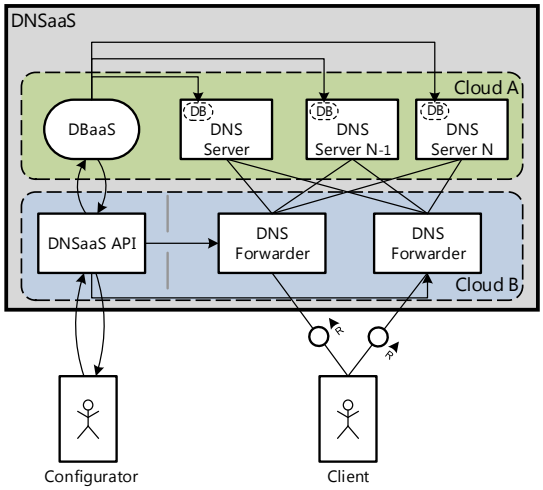

Fig. 2: Distributed Architecture

applied to determine, in advance, the amount of required resources for future operations. One existing proposal is PRESS [14], based on statistical learning algorithms to enable dynamic adjustments that aim at minimising resource waste and avoid Service Level Objects (SLOs) violations. Bearing this in mind, PRESS uses several metrics such as CPU usage, memory input/output (I/O) and network usage, being able to efficiently handle available resources in large-scale cloud-computing infrastructures. Another approach, BConf [15], also resorts to prediction techniques to enable dynamic balancing of multiple resources in cloud systems. Its main optimisation goal addresses the response time guarantee, for which BConf uses prediction algorithms and control mechanisms.

Reactive approaches for elasticity techniques, such as performing horizontal or vertical scaling, may fail to minimise costs due to SLO violations [16]. Taking this into account, proposals like AutoFlex combine reactive and proactive mechanisms to enable elasticity of services in the cloud. Similarly to PRESS, AutoFlex also relies on prediction mechanisms being able to reduce the number SLO violations.

The functionalities provided by orchestration services for scaling on cloud platforms, such as Heat [9] on OpenStack [4], fall into a reactive approach where base metrics such as CPU usage and memory I/O are monitored and scaling decisions (horizontal or vertical) are performed relying on user defined thresholds. Nonetheless, the support metrics are generic and not necessarily representative of the performance of specific services in the cloud, such as DNS.

An alternative resource management proposal, Anchor [17], formulates optimisation by considering both Operator and the Client sides, as opposed to common enhancement mechanisms that consider only operator metrics, such as CPU usage [14]. Anchor uses deferred acceptance algorithms to resolve the conflicts between client and operator interests.

A different perspective on scaling issues is provided by RCM [18], assuming that several metrics need to be measured when addressing scalability in large-scale infrastructures, such as CPU, disk usage and inter-node network delay, among others. RCM aims at reducing the data collection cost by performing online compression of the measured metrics. However, despite the validation in real-monitored systems, no insights are provided regarding strategies for the optimisation of resources. Context-aware approaches are also employed to allow monitoring of network traffic by relying on different DNS classes, canonical, overloaded and unwanted [19,20], but the optimisation of resources is disregarded as well.

DNS performance has been characterised in the literature [21]–[25] considering the performance of DNS authoritative servers in the face of the load introduced by simultaneous clients and the volume of performed DNS queries. Although these studies identify the key performance indicators of traditional DNS services, they cannot be used to infer how the hierarchical architecture of DNS can be adapted to support the flexibility, elasticity and failover characteristics of cloud-based services. Moreover, they do not include insights on how the DNS record information can be managed to increase availability of servers (e.g. email or web servers) [26].

While many studies on elasticity and scaling on cloud-based services have been conducted, each individual service presents its own challenges and requirements. The DNSaaS architecture proposed in this paper includes support for cloud paradigms such as elasticity, scalability and failover mechanisms, without being tied to a specific cloud platform and with enough flexibility for allowing the splitting of functionalities (backends and frontends) in diverse data centres.

III. Domain Name Service as a Service

This section describes the architecture and main features of our DNSaaS platform.

A. DNSaaS Architecture

Following basic cloud principles, in addition to tenant isolation and appropriate configuration mechanisms, the defined DNSaaS architecture has been designed to be fault tolerant

and elastic. Moreover, in order to be scalable and cope with different utilisation levels, horizontal scaling was considered.

For the proposed architecture, within a single datacenter, two separate DNS Forwarders were used, being responsible for receiving all the DNS requests as primary and secondary name servers. This approach was foreseen not only as a mean to avoid multiple endpoints to a single service, but also to allow the failover approach of a typical DNS service, while also being able to use the forwarders as internal load-balancers. By having two dedicated endpoints for receiving DNS queries, the proposed architecture is able to support multiple DNS servers, whose IP addresses can be dynamically added or removed to the Forwarders list of available servers for load balancing. This aspect represents a crucial point in the architecture's ability to be elastic and react on-demand to increasing demands of load, while also being able to reduce the amount of needed resources whenever suitable.

Since the performance of the overall service is a major priority, ensuring an appropriate use of the DNSaaS database is crucial. However, due to the separation of the DNS service into multiple DNS servers, challenges regarding the consistency of records arise. For handling these issues, while always keeping performance in mind, two separate architectures were considered. One with a centralised database, resorting to DataBase as a Service solutions (such as Trove [27]), and another where each DNS Server has its own database engine, being all the databases consistent and synchronised in a master-slave paradigm, where the master database is managed directly by DNSaaS API, responsible for all the configurations of records.

A major concern from the bottleneck in the centralised architecture, depicted in Figure 1, is that concurrent database accesses may significantly increase latency, despite the in-memory caching mechanisms by each DNS Server. On the other hand, the decentralised approach, presented in Figure 2, may suffer from synchronisation challenges when assuring master-slave data replication.

B. DNSaaS Components

The different components depicted by both architectures have their own specificities. For instance, in the centralised approach, the DBaaS component serves as master database, where all the DNS information is stored. It is also responsible for keeping domains and records for all tenants. In the architecture with distributed database this component is also responsible for replicating the data to the other databases in the DNS Servers, acting as a master database. The slave databases are used only for read operations and contain optimisations, as indexes for the most common access operations (such as retrieving an IP for A records).

The DNS Server is the core component of the DNS service, handling the domain-address conversion and supporting geodns, among other possible DNS extensions (e.g. DNSSEC). The instantiation of typical DNSaaS architecture may begin with a single DNS Server, but it is able scale according to the registered needs, by considering a dynamic number of DNS servers. The optimal number of DNS servers may be found

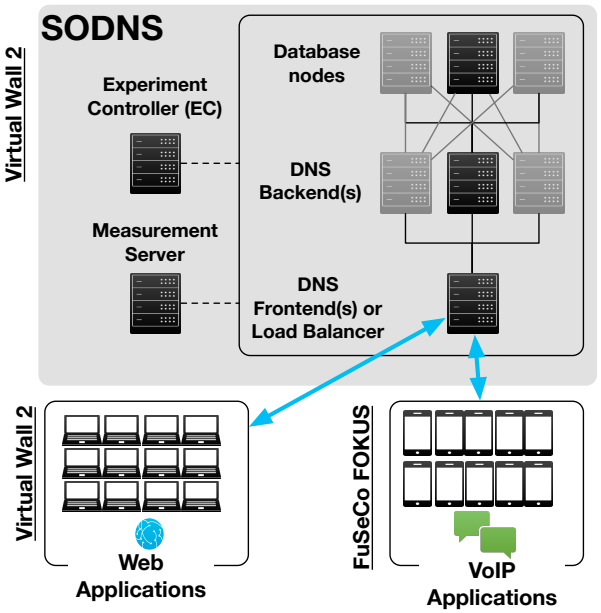

Fig. 3: Evaluation Scenario

by considering multiple criteria decision, resorting to simple metrics such as latency or the ratio between successful and unsuccessful queries, used to infer the performance of DNS servers. Heuristic metrics and prediction algorithms may also be applied, as previously discussed in the related work section.

The DNS Forwarder is the front-end of the service for a client, being the main task of this component is to forward DNS requests to an existing DNS server, using round-robin fashion algorithm, or even by adopting more complex load-balancing mechanisms.

The API acts as an endpoint for the configurator (or enterprise end-user), allowing an authenticated and tenant-based configuration of the DNSaaS service. This component provides simple CRUD operations for domains and records, keeping DBaaS updated according to the available services and servers. This component is also responsible for maintaining the available list of DNS Servers up-to-date, informing the forwarders so they can distribute the load appropriately.

One of the design goals of the proposed DNSaaS includes reliability, allowing to distribute DNSaaS components in diverse cloud platforms that can be geographically distant from each other, as demonstrated in the centralised and distributed architectures, Figure 1 and Figure 2, respectively. The solution can be deployed on Amazon Web Services, OpenStack based-clouds since DNSaaS employs cloud-specific agnostic mechanisms for scaling. For instance, does not require specific metrics from Neutron service on OpenStack to gather performance data.

IV. Evaluation Methodology

This section discusses the methodology we used to evaluate the scaling performance of DNSaaS, presenting the evaluation scenario and associated metrics. The validation has been

performed using iMinds vWall2 and FuSeCo testbeds from Fed4Fire [13].

A. Scenario

The considered evaluation scenarios focus on the distributed approach, since our preliminary validations results [28,29] focused mainly on the centralised approach. Nonetheless, none of the previous works included a large-scale evaluation, as the one targeted in this paper (e.g. 12.5M DNS requests in a scale of minutes). The scenario has been deployed in virtual vWall2 and FuSeco FOKUS testbeds as depicted in Fig. 3.

The database nodes have been configured using a MySQL cluster as the database engine for DNSaaS [28]. A Load Balancer (LB) has been introduced to enable sharing the load between the database nodes and is configured with HAProxy to enable an efficient, scalable load sharing and tolerant to failures regarding the reachability or the service status. Thus, avoiding the use of nodes not reachable or with down status.

For performance evaluation and validation purposes, a single DNS Forwarder was used, as depicted in Fig. 3. This option was considered to maintain the repeatability of the scenarios' conditions, avoiding unnecessary uncertainty factors from load-balancing mechanisms, focusing on the main purpose of validating the architecture for scaling and assessing the behaviour of the DNS Servers upon receiving a high-load of queries requests, from simultaneous clients. In this regard, a variable number of DNS Servers (*n_servers*) was considered, from one to three (*1, 3*), all of them connected to the DNS Forwarder. It is also important to consider that all the servers, including the DNS Servers and the DNS Forwarders, were configured to not perform any caching of DNS queries or answers, guaranteeing an adequate evaluation on each run with the same initial conditions and ensuring repeatability. This configuration also allows to evaluate the service's performance in a worst case scenario. The number of DNS Forwarders can be easily incremented using the proposed architecture, requiring only the reconfiguration of DNS clients (with a new IP) and the respective server (information of the DNS Servers to perform load balancing). The DNS Forwarder considered two main technical solutions, one based on PowerDNS Recursor [30] and another one based on dnsdist [31], with the goal of assessing different load balancing mechanisms between the DNS Servers. Other mechanisms could be pursued, such as load balancing with Neutron service, but would introduce OpenStack lock-in. The dnsdist was configured with two policies, the *leastOutstanding* which load requests to the DNS Server with less load and the *firstAvailable* which employs a load bellow a certain QPS threshold (e.g. configured with a value 1250).

In order to create load on the DNS service, several clients were employed, each one performing *500K* DNS queries, with diverse send rates of requests per second ($sendRate = \{100, 250, 500, 1000\}$) to assess the impact of diverse types of load in the DNSaaS. The clients read files with *500K* records and perform the DNS queries by using the *dnsperf* tool [32], a synchronization mechanism has been implemented to manage the parallel execution of requests from the diverse clients. These queries are sequential and independently configured according to the number of outstanding requests, which is similar to the sendRate. Thus within this number each query is sent immediately after the other without waiting for any query reply, resulting in a burst of queries in DNS servers. The total number of clients (*n_clients*) however, was varied from a single client up to 25 concurrent clients (*1, 3, 5, 25*), leading to a different load in terms of DNS queries ($\sum$ *DNS queries*). This approach allows an assessment of the performance of the proposed DNSaaS architecture under different levels of load, up to a maximum of 12.5M DNS requests.

TABLE I: Configuration parameters

Test	DNS Frontend	Number clients	clients
1-8	1 recursor	1	type: wired sendRate:{100,250,500,1000} records:{A,NAPTR}
9-16	1 recursor	3	type: all sendRate:{100,250,500,1000} records:{A,NAPTR}
17-20	1 recursor	5	type: all sendRate:{100,250,500,1000} records:{A,NAPTR}
21-24	1 dnsdist least-Outstanding	5	type: all sendRate:{500,1000} records:{A,NAPTR}
25-28	1 dnsdist firstAvailable	5	type:all sendRate:{500,1000} records:{A,NAPTR}
29-36	1 recursor	1	type: wireless sendRate:{100,250,500,1000} records:{A,NAPTR}
37-40	1 recursor	25	type: all sendRate:{500,1000} records:{A,NAPTR}
41-44	1 recursor	25	type: all sendRate:{500,1000} records:{A,NAPTR}
45-48	1 dnsdist least-Outstanding	25	type: all sendRate:{500,1000} records:{A,NAPTR}

Given the variation possibilities of the different presented parameters, multiple evaluation scenarios exist, providing a thorough performance comparison of distinct DNSaaS configurations. All the available evaluation tests combinations are summarised in Table I. Tests consider Data applications to request information for "A" records and VoIP applications require the information provided in "NAPTR" record, which is required for successful SIP signalling. Besides the type of record, the main difference between data and VoIP applications relies in the configured timeout for DNS replies. The former assumes a timeout of 5s, while the last requires a timeout of 2s. The differences between each test lie in the configured sendRate and/or the solution used for the DNS Frontend. For instance, test 1 includes a sendRate=100, while test22 includes a sendRate of 1000 and uses the dnsdist solution.

The clients rely on pcgen3 nodes (CPU with 2.4GHz, 24GB

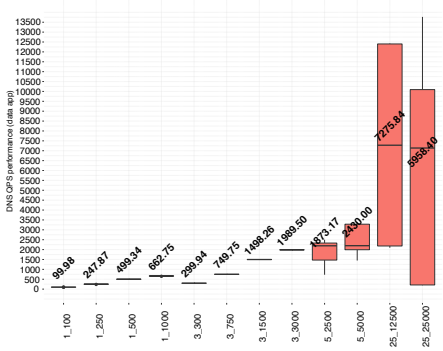

Fig. 4: QPS performance of Data Apps

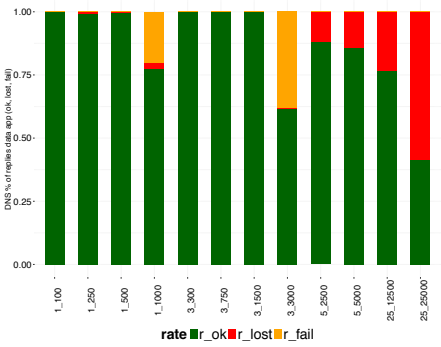

Fig. 5: Ratio of Successful requests of Data Apps

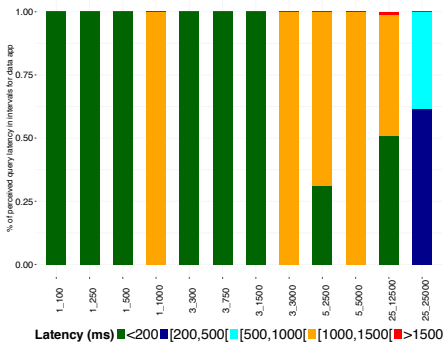

Fig. 6: Ratio of intervals latency of Data Apps [25]

of RAM), while the servers rely on pcgen5 nodes (CPU with 3.1GHz, 16GB of RAM) of Virtual vWall2.

B. Performance Metrics

Having defined all the relevant evaluation scenarios, it is necessary to determine appropriate performance metrics that reflect the service's response to different demands. These metrics concern the different components that compose the DNSaaS architecture, assessing mainly the capacity of answering DNS queries in a timely fashion. The analysed metrics include: the queries throughput (qps), as measured by each client; the query loss ratio; and the query failure ratios, due to loss of requests/replies or due to ServFail (requests not answered by DNS Servers). The Query latency is considered in five intervals [25] (in ms): < 200; $[200, 500[$; $[500, 1000[$; $[1000, 1500[$ and > 1500. These metrics are collected in graylog [12] without imposing additional overhead on the DNSaaS components.

V. Results

This section presents the obtained results during the validation and performance assessment of the defined DNSaaS architecture. These results are discussed based upon a 95% confidence interval, of the average from ten runs of each test.

A. Data App Perspective

The results of the measured metrics for the data applications are depicted in Figures 4, 5 and 6. The depicted results are based on both types of clients (i.e. wireless and wired). With lower sendRates (≤ 500), clients are able to have a throughput similar to the sendRate. Nonetheless, with higher sendRates (> 500) the achieved throughput is lower and is not close to the maximum theoretical value. For instance, a single client issuing requests at a rate of 1,000 requests per second is only able to get a nominal performance around 665 QPS. Such fact is associated with the full resolution process of DNSaaS, that has an higher delay in these cases, with a measured *qa-latency*[1] in the frontend higher than 1 second. This can be noticed with the higher failure ratios for these test cases (see Fig. 5).

Another relevant aspect to analyse includes the simultaneous load that is supported. Indeed, with more than 5 clients, the achieved performance presents a high variation in the perceived QPS, which never reaches the theoretical values. The query loss ratio is higher in these cases, leading to 50% of loss with 25 simultaneous clients issuing 1,000 requests per second.

[1]Latency of query answers measured in the Frontend

B. VoIP App Perspective

The results of the measure metrics for the VoIP applications are depicted in Figures 7, 8 and 9.

Besides the query loss ratios, the latency has also an high impact in VoIP applications. The performance in terms of achieved DNS requests throughput follows the same trends as data apps. In particular, QPS is not similar to the sendRates with high load ($> 1,000$) and with a high number of requests (5 * 500k = 2.5M). With VoIP apps, the timeout relies in values of 2s, leading to higher loss ratios (75%) in extreme load cases (25 clients issuing 1,000 requests per second), in comparison to data apps.

C. On-demand instantiation and disposal

The deployment of DNSaaS solution, include DNS Forwarder, DNS Servers and Database nodes relies in the order of seconds (around 330s), when considering an OpenStack Infrastructure and a platform based on OpenShift, as reported in previous work [29]. The disposal of resources also relies in the order of seconds (around 5s), which includes deletion of virtual machines and updates in configurations (e.g. IP addresses of DNS Backends in Frontends).

The process of loading the database nodes with the necessary information for DNS resolution, relies in the order of minutes. Indeed, databases were populated with millions of records, due to the use of 100 domains, having each one more than 500k records of type A and 500k for NAPTR type.

VI. Conclusion

The elastic and scalable architecture for DNS as a Service, DNSaaS, has been validated in high load conditions, with a high number of simultaneous clients. The proposed architecture does not rely on specific metrics, or mechanisms to assess the performance levels for scaling, which does not tie the solution to a particular cloud-computing platform to support elasticity.

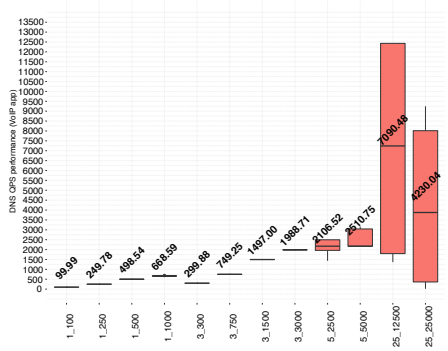

Fig. 7: QPS performance of VoIP Apps

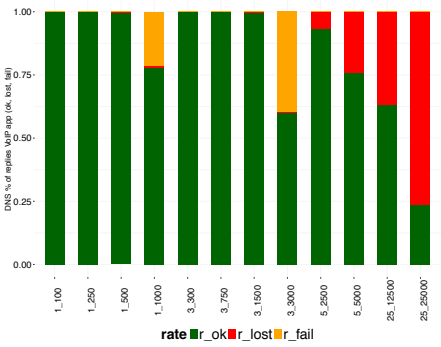

Fig. 8: Ratio of Successful requests of VoIP Apps

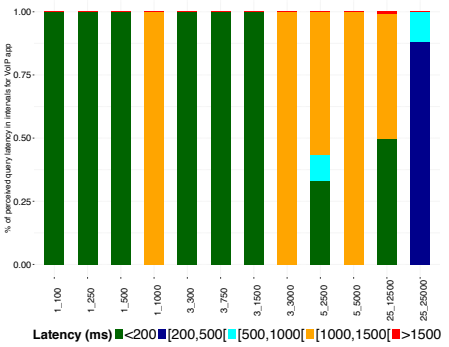

Fig. 9: Ratio of intervals latency of VoIP Apps [25]

Regarding the performance evaluation of the presented architecture, it became clear that the service is able to accommodate variable loads of DNS queries per second, always keeping satisfactory levels of performance in terms of DNS queries throughput and low latency DNS answers. This performance level was maintained by DNSaaS resorting to a horizontal scaling approach, instantiating additional resources whenever required.

Acknowledgements

This work was carried out with the support of the Mobile Cloud Networking project (FP7-ICT-318109) and the Fed4FIRE project (FP7-ICT-2011-8), both funded by the European Commission through the 7th ICT Framework Program.

References

[1] C. Liu and P. Albitz, *DNS and BIND*. O'Reilly Media, 2006. [Online]. Available: https://books.google.pt/books?id=HggtWI1ShvMC

[2] M. Pathan, R. Sitaraman, and D. Robinson, *Advanced Content Delivery, Streaming, and Cloud Services*, ser. Wiley Series on Parallel and Distributed Computing. Wiley, 2014. [Online]. Available: https://books.google.pt/books?id=3yaUBAAAQBAJ

[3] Bernhard Ager and Wolfgang M?hlbauer and Georgios Smaragdakis and Steve Uhlig, "Comparing DNS resolvers in the wild," in *Internet Measurement Conference*, 2010, pp. 15–21.

[4] OpenStack, "Openstack cloud software," https://www.openstack.org [Last Visit: 02-October-2016].

[5] Amazon, "Amazon web services," https://aws.amazon.com/ [Last Visit: 02-October-2016].

[6] Microsoft, "Windows azure," http://azure.microsoft.com/en-us/ [Last Visit: 02-October-2016].

[7] Google, "Google cloud computing," https://cloud.google.com/ [Last Visit: 02-October-2016].

[8] Amir Basirat and Asad Khan and Balasubramaniam Srinivasan, "Highly Distributable Associative Memory Based Computational Framework for Parallel Data Processing in Cloud," 12 2014.

[9] OpenStack, "Heat - openstack orchestration," https://wiki.openstack.org/wiki/Heat [Last Visit: 02-October-2016].

[10] ——, "Designate, a DNSaaS component for OpenStack," http://designate.readthedocs.org/en/latest/ [Last Visit: 02-October-2016].

[11] Zabbix, "Zabbix - the enterprise-class monitoring solution for everyone," https://www.zabbix.com/ [Last Visit: 02-October-2016].

[12] I. Graylog, "graylog - Open source log management that actually works," https://www.graylog.org, 2016.

[13] Fed4Fire., "Federation for Future Internet Research and Experimentation," http://www.fed4fire.eu/testbeds/, 2016.

[14] Z. Gong, X. Gu, and J. Wilkes, "PRESS: PRedictive elastic ReSource scaling for cloud systems," in *2010 International Conference on Network and Service Management (CNSM)*, Oct. 2010, pp. 9–16.

[15] Y. Wei and C.-Z. Xu, "Dynamic Balanced Configuration of Multi-resources in Virtualized Clusters," in *2013 IEEE 21st International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS)*, Aug. 2013, pp. 60–69.

[16] F. Almeida Morais, F. Vilar Brasileiro, R. Vigolvino Lopes, R. Araujo Santos, W. Satterfield, and L. Rosa, "Autoflex: Service Agnostic Auto-scaling Framework for IaaS Deployment Models," in *2013 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, May 2013, pp. 42–49.

[17] H. Xu and B. Li, "Anchor: A versatile and efficient framework for resource management in the cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1066–1076, Jun. 2013.

[18] Y. Tan, V. Venkatesh, and X. Gu, "Resilient Self-Compressive Monitoring for Large-Scale Hosting Infrastructures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 3, pp. 576–586, Mar. 2013.

[19] David Plonka and Paul Barford, "Context-aware clustering of DNS query traffic," in *Internet Measurement Conference*, 2008, pp. 217–230.

[20] Moheeb Abu Rajab and Fabian Monrose and Andreas Terzis and Niels Provos, *Peeking Through the Cloud: DNS-Based Estimation and Its Applications*. Springer, 2008.

[21] S. H. da Mata, J. M. Magalhaes, A. Cardoso, P. R. Guardieiro, and H. A. Carvalho, "Performance comparison of enum name servers," in *Computer Communications and Networks (ICCCN), 2013*. IEEE, 2013, pp. 1–5.

[22] Y. Yu, D. Wessels, M. Larson, and L. Zhang, "Authority server selection in dns caching resolvers," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 2, pp. 80–86, 2012.

[23] D. Migault, C. Girard, and M. Laurent, "A performance view on dnssec migration," in *Network and Service Management (CNSM), 2010*. IEEE, 2010, pp. 469–474.

[24] J. Rudinsky, "Private enum based number portability administrative system evaluation," in *Ultra Modern Telecommunications & Workshops, 2009. ICUMT'09. International Conference on*. IEEE, 2009, pp. 1–7.

[25] Vulimiri, Ashish and Godfrey, Philip Brighten and Mittal, Radhika and Sherry, Justine and Ratnasamy, Sylvia and Shenker, Scott, "Low Latency via Redundancy," *Proceedings. of CoNEXT*, pp. 283–294, 2013.

[26] Andrew J. Kalafut and Craig A. Shue and Minaxi Gupta, "Understanding implications of DNS zone provisioning," in *Internet Measurement Conference*, 2008, pp. 211–216.

[27] OpenStack, "Trove - database as a service for openstack," https://wiki.openstack.org/wiki/Trove [Last Visit: 02-October-2016].

[28] Bruno Sousa, Claudio Marques, David Palma, João Gonçalves, Paulo Simões, Thomas Bohnert, Luis Cordeiro , "Towards a High Performance DNSaaS Deployment," in *proceedings of the 6th International Conference on Mobile Networks and Management MONAMI'14*. Springer, September 2014.

[29] B. Sousa, L. Cordeiro, P. Sim?es, A. Edmonds, S. Ruiz, G. A. Carella, M. Corici, N. Nikaein, A. S. Gomes, E. Schiller, T. Braun, and T. M. Bohnert, "Toward a fully cloudified mobile network infrastructure," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 547–563, Sept 2016.

[30] I. PowerDNS, "PowerDNS Recursor," https://powerdns.com/recursor.html, 2016.

[31] ——, "dnsdist - highly DNS-, DoS- and abuse-aware loadbalancer," http://dnsdist.org/, 2016.

[32] Nomium, "Network measurement tools," http://nominum.com/support/measurement-tools [Last Visit: 02-October-2016].