Journal of Network and Computer Applications 69 (2016) 1–13

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Two-phase load balancing of In-Memory Key-Value Storages using Network Functions Virtualization (NFV)

Alex F.R. Trajano, Marcial P. Fernandez*

Universidade Estadual do Ceará (UECE), Av. Silas Munguba, 1700 Fortaleza, Ceará, Brazil

ARTICLE INFO

Article history:
Received 14 June 2015
Received in revised form
8 March 2016
Accepted 30 April 2016
Available online 3 May 2016

Keywords:
Software Defined Networks (SDN)
In-Memory Key-Value Storage (IMKVS)
Network Function Virtualization (NFV)

ABSTRACT

Social networks and other clouding applications should require fast responses from datacenter's infrastructure. One of the techniques that have been widely used for achieving such requirement is the employment of In-Memory Key-Value Storage (IMKVS) as caching mechanisms in order to improve overall user experience. Memcached and Redis are applications that use IMKVS approach. Commonly IMKVS systems use Consistent Hashing to decide where to store an object, which may cause network load imbalance due to its simplistic approach. Furthermore, these systems work only at the application layer, so network conditions are not considered to distribute user's accesses. This paper proposes a new cache architecture based on a two-phase load balancing to improve IMKVS performance, which has adopted a Network Function Virtualization (NFV) architecture to manage the load balancing mechanism. The proposal was evaluated in the Mininet and shows an improvement of 23% on the load of the caching servers and 5% on the load of the network compared to Consistent Hashing, which results in better resource usage and better user experience.

1. Introduction

The growing popularity and complexity of social networks and cloud applications require robust datacenters that handle a massive amount of data. Facebook, for instance, has a deployment scenario where a front-end web server is responsible for delivering requested content to users through Hypertext Transfer Protocol (HTTP). These servers must handle the requests and fetch data from different caches, databases and back-end servers in order to render the requested web page. It has been reported that a single HTTP page request required 88 cache lookups (consuming 648 KB), 35 database lookups (consuming 25.6 KB) and 392 back-end remote calls (consuming 257 KB), taking just a few seconds to completely load the page on the user's screen (Farrington and Andreyev, 2013).

In order to support the storage and the processing of large amounts of data, many cloud applications have adopted a simple but effective caching infrastructure that relies on In-Memory Key-Value Storage (IMKVS). These simple storages are able to provide fast access to any type of data that can be mapped by a key. Its data in-memory placement helps to avoid slow and expensive access to persistent storages on hard disk. Thus, IMKVS is often used to store and supply information that is cheaper to cache than to re-obtain, such as commonly accessed database queries or complex computations results. Several IMKVS cache implementations have been developed and deployed in large-scale cloud services, including Dynamo at Amazon (DeCandia et al., 2007); Redis at GitHub, Digg, and Blizzard (Redis, 2015); Memcached at Facebook, Zynga, and Twitter (Fitzpatrick, 2004); and Voldemort at LinkedIn (Sumbaly et al., 2012). Facebook has reported that there are tens of thousands Memcached instances operating on their datacenters (Nishtala et al., 2013).

These simple applications are basically Hash Maps that store either binary or text contents mapped by a unique key, usually a String of variable length, and they can be part of a huge caching layer designed to participate in applications' architectures in a distributed and independent way. In other words, each one of these IMKVS instances does not know about the existence of other IMKVS instances, forcing their client's applications to manage all issues of data partitioning and load balancing.

Most of Redis' (Redis, 2015) and Memcached's (Fitzpatrick, 2004) Application Programming Interface (API) clients use a technique proposed by Karger et al. (1997), called Consistent Hashing (CH). It consists in distributing the keys uniformly among the servers, as well preventing that neither server's addition nor removal causes a great impact on the keys' distribution. CH has been used successfully in several kinds of applications, like caching (Nishtala et al., 2013) and storage (Lakshman and Malik, 2010). Although CH is a very efficient technique, it is, basically, a special

* Corresponding author.
E-mail addresses: alex.ferreira@uece.br (A.F.R. Trajano), marcial.fernandez@uece.br (M.P. Fernandez).

http://dx.doi.org/10.1016/j.jnca.2016.04.024
1084-8045/© 2016 Elsevier Ltd. All rights reserved.

hash function that uses a key and a set of servers to select where to store the data. However, it may incur load imbalance in a production network, causing hot spots, since it does not consider any environmental aspects nor object's characteristics, like size, link congestion or object's popularity.

Moreover, in datacenter environment, it is necessary for some technique to perform network and server load balancing, either by application layer algorithms or network orchestration. In fact, over the last few years, it has been common to find specialized hardware appliances or applications capable of performing traffic load balance of specific types of service. Nevertheless, the load balancing task should not be coupled to specialized infrastructure items, since it should be an embedded feature of the network itself in order to ensure maximum performance. In-network load balancing is a way of providing more flexibility and near optimal performance. Besides, there is a wide set of Internet services that can be boosted using an in-network load balancing technique, which opens the opportunity for developing generic solutions focused on the adherence to current and future services at no cost.

Over the last few years, Network Functions Virtualization (NFV) (Guerzoni et al., 2012) has been becoming one of the most promising study areas for developing modern computer network technologies. NFV poses a novel way to develop network services, by using software and virtualization aiming the replacement of proprietary hardware appliances that run network functions, like Domain Name Service (DNS), Network Address Translation (NAT), Intrusion Detection System (IDS), caching, etc. In NFV, these services, called Virtualized Network Function (VNFs), are implemented through software and deployed in Virtual Machine (VMs), allowing new and efficient ways of network deploying. It may permit us to customize and manage such services, enabling a tremendous cost saving and more agility to serve the daily changes that computing network is susceptible.

Simultaneously with NFV, another networking approach that has been gaining popularity is the Software-Defined Network (SDN) (McKeown). SDN allows network management by creating an abstraction of the lower-level functionalities of the networking, by separating the control plane (decision making) from the data plane (forwarding). SDN is often confused with OpenFlow (McKeown et al., 2008) beeing the most popular and promising SDN technology. OpenFlow, beyond have the data and control planes separated, have a centralized architecture that simplifies the development of different kinds of network services, which consists in creating a unique programmable controller that can be deployed to a commodity server, maintaining SSL channels to all switches within the network, programming all aspects of packet forwarding of the network. Although SDN and NFV have many common aspects, both are neither competitors nor do they conflict. When both are used together, the whole network tends to benefit from it, since it has the best aspects of data and control plane separation coexisting with the virtualization power, allowing a more efficient control of the network. SDN contributes to better traffic orchestration while NFV focuses on service delivery.

This work aims to propose a novel VNF that will be responsible for load balancing IMKVS requests, focusing on the datacenter environmental characteristics. This new VNF is going to consider various network's and servers' features in order to minimize load from both datacenter network and servers, providing high scalability, better resource usage and replication mechanisms to alleviate load generated by popular objects within the caching layer. In addition to that, a generic OpenFlow-based load balancing will be proposed to avoid creation of bottlenecks into the network, serving as a *front-end broker* for the VNF. Both OpenFlow-based load balancing (First-Phase) and the cache redirector (Second-Phase) compose the proposed Two-Phase Load Balancer.

This article is an extended version of the paper presented in The Twentieth IEEE Symposium on Computers and Communications (ISCC 2015) (Trajano and Fernandez, 2015). Comparing to the original conference paper, this one shows a deeper theoretical formulation, more details in the proposal include more evaluation analysis to validate the proposal.

The rest of the paper is structured as follows. In Section 2, we present some related work about the load balancing of IMKVS. Section 3 introduces the concepts. In Sections 4–6 we present the Two-Phase Load Balancer architecture. Section 7 gives some experimental results and in Section 8 we conclude the paper and present some future works.

2. Related work

Li and Pan (2013) proposed an OpenFlow-based load balancer for Fat-Tree networks that supports multipath forwarding. Their proposal aims to recursively find the current best path from a source to a destination, load balancing the network by enabling the use of alternate paths at runtime, minimizing network congestion. Their algorithm works only on networks that operate on the Fat-Tree topology and use network metrics for choosing the optimal path.

Wang et al. (2011) proposed an interesting load balancing approach that aims to proactively load balance traffic from clients to servers by slicing the IP address space into trees that isolates a set of clients to a set of servers. The work uses the concept of server weighting, which defines a fixed portion of the clients to a server on the network. To do so, it is proposed the extensive use of wildcards, which may reduce forwarding performance and create management issues, as can be seen in Lopes Alcantara Batista et al. (2014). Furthermore, the proposed solution requires that in certain conditions (network topology changes or server weight updates); a part of the network traffic passes through the controller, which could cause serious scalability problems that may lead the network controller to collapse. Network metrics is not considered.

The work from Koerner and Kao proposes an architecture to enable in-network load balancing of multiple services using OpenFlow (Lopes Alcantara Koerner and Kao, 2012). Their proposal relies on a set of SDN controllers on top of a FlowVisor instance (Sherwood et al., 2012), where each controller is responsible for load balancing the traffic of a specific service. The authors have focused on the architecture, so there is no information about particular service implementation, while the experiment does not fit real-world scenarios. The idea of using a set of controllers to handle exact services might be interesting in some specific cases, but has the drawback of not permitting multiple services to be handled by a single controller, which is the most common situation of SDN deployment.

Handigol et al. (2009) present Plug-n-Serve, a module residing within an OpenFlow controller that is capable of performing load balancing over unstructured networks, aiming to minimize average response time of HTTP servers. Plug-n-Serve load balance HTTP requests by gathering metrics about CPU consumption and network congestion on the network links, which makes its load balancing algorithm to select the appropriate server to direct requests, while controlling the path taken by packets on the network.

De Cesaris et al. (2014) proposed Network-Assisted Lookups (NAL), a method to do rapid load balance of key-value storages through the existing IP infrastructure. The proposed solution consists in assigning multiple IP addresses to each server of the caching layer, being each IP address mapped to a bucket of objects. The NAL Controller is responsible for collecting the load of the buckets in order to notice performance degradation of servers. Since the controller recognizes that a bucket is an issue, through