Poster: Controlling Quality of Service of Container Networks in a Hyperconverged Platform

Sumitro Bhaumik
IIT Kharagpur, India 721302
Email: sumitro@iitkgp.ac.in

Kaustav Chanda
Manipal Institute of Technology, India 576104
Email: kaustav.chanda@learner.manipal.edu

Sandip Chakraborty
IIT Kharagpur, India 721302
Email: sandipc@cse.iitkgp.ac.in

***Abstract*—This poster characterizes the impact of storage traffic over containerized network workloads in a shared storage hyperconvergence architecture. We develop CONtrol, a control theoretic approach for assuring the performance of container networks in the presence of disk-intensive workloads. CONtrol uses a *proportional integral derivative* controller to dynamically decide the bandwidth redistribution among storage workloads and container networks. We have implemented CONtrol over a hyperconverged platform with 5 physical servers, and a thorough testing indicates that it can significantly improve the performance of various benchmark applications over a containerized hyperconverged platform.**

***Index Terms*—container, hyperconvergence, QoS**

I. Introduction

The data center (DC) technologies now-a-days are moving towards hyperconverged platforms [1] where the storage is directly coupled with the compute servers unlike network attached storage (NAS) or storage area network (SAN). Software defined storage (SDS) [2] platforms are normally used for storage management on this architecture, which provides a logically unified storage platform on top of the individually server-coupled storage disks. Consequently, the storage workload is distributed across the physical hosts, and any disk read-write operations initiate storage traffic among interconnection fabrics carrying block-level data through storage data transmission protocols (like iSCSI, FCIP, iFCP, FCoE etc.). Over such an architecture, it is challenging to provide consistent network connectivity with desirable QoS to a large volume of containers which primarily carry short-lived network workload, because of the following reason.

Storage traffic carries block-level data through persistent TCP connections which are typically long-term high-bandwidth flows. On the other hand, web traffics generate short-term low-bandwidth connections, like HTTP requests or responses. It is well known that long flows hurt the performance of short flows on a shared network bottleneck [3]. To alleviate this problem, existing studies use approaches to identify and control the bandwidth usage of long flows [4]. However, the distribution of storage-intensive and network-intensive containerized workloads over a single physical host is very dynamic. Therefore, a direct upper cap on the bandwidth usage for long flows may unnecessarily affect the storage access performance for the storage-intensive container workloads. Explicit TCP congestion management approaches like DCTCP [5] may solve this problem up to certain extent, but cannot eliminate it completely. This is primarily because they reduce the latency for short flows by giving them more priority through active queue management; however, they do not necessarily share the bandwidth among the flows depending on the application QoS requirement. Therefore, we argue that this problem cannot be tackled just by engineering the network traffic flows in an application-agnostic way.

In this poster, we propose CONtrol, a container orchestration framework to balance the network bandwidth usage among the storage traffic and the application network traffic in a shared-storage hyperconverged architecture. The objective of CONtrol is to "control" the storage traffic such the QoS for the container network is sustained. CONtrol monitors containers in real-time, and depending upon a container's current QoS level, it enforces network policies or migrates containers to reduce the network-storage conflict. For this purpose, a control-theoretic approach, based on *"Proportional Integral Derivative"* (PID), is used to decide the bandwidth redistribution among the storage traffic and the application network traffic generated from the containers. We implement and test the performance of CONtrol over a physical-testbed having 5 bare-metal servers. We observe that CONtrol improves the QoS of the container networks significantly, especially during heavy system loads, in comparison to different baselines.

II. CONtrol Resource Manager

We consider a set of containers $\mathbb{C} = \{c_1, c_2, c_3...c_i\}$, where each container runs over host machine h; the set of hosts is denoted by $\mathbb{H} = \{h_1, h_2, h_3...h_j\}$. Every container runs a specific type of workload; however, in CONtrol, we primarily focus on disk workloads and the network workloads. Containers which primarily execute high disk reads or writes are termed as disk containers, denoted by c^d, and similarly, containers which utilize high network bandwidth are termed as network containers, denoted by c^n. It can be noted that a container can have a mixed workload with both disk writes and network usage, and under such scenarios, we treat it with both the workload types.

Each physical host is associated with a shared storage system. We consider RAID-1 based shared storage replication; however, CONtrol can work with any RAID level replications

Annex to ISBN 978-3-903176-28-7

with minor tuning in a few hyper-parameters. CONtrol looks for the QoS drops at different containers based on their disk and network resource usage. Once CONtrol detects a QoS drop, the system uses a control-theoretic approach based on PID controllers to decide about the required resource allocation, such that the QoS of all the containers over a host can be balanced. PID is a simple and very lightweight but efficient control technique for closed control space (like the total amount of available resources at a host is fixed) [6], [7], so the management overhead is low. CONtrol spawns a PID controller for every container initiated in the system.

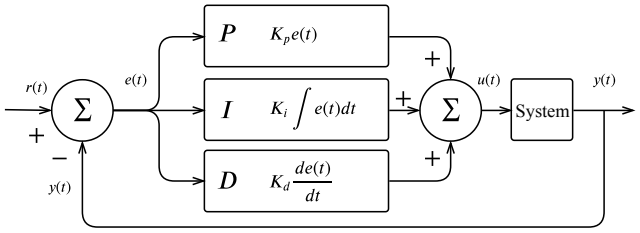

Fig. 1: PID controller - Block diagram

A PID controller is a control-loop feedback mechanism [6], [7] which is very effective in systems that need continuous monitoring. A standard PID controller is shown in Fig. 1. Let $r(t)$ be the value of the desired metric m of the system which is dependent on some parameter or resource b that we can adjust. A PID controller computes the error function $e(t)$, which is the difference between $r(t)$ and $y(t)$, where $y(t)$ is the current value of metric m. In CONtrol design, the error function is modeled as the QoS drop at the container at time t. The difference tells us how far the system is currently away from the optimal operating condition. Based on $e(t)$, we need to adjust the resource allocation.

We feed the value of $e(t)$ into three sub-modules as follows. A proportional component (denoted by P) simply watches the magnitude of $e(t)$. If $e(t)$ is large enough, then the P component requests for resources *proportional* to $e(t)$. The integral component (denoted by I), integrates over the $e(t)$, and thus takes into account the past values of history of $e(t)$. The derivative component (denoted by D) takes the derivative of $e(t)$ and checks the rapidity of change of the value of $e(t)$, and thus takes into account the possible future values of error. The output of the three components are added to form $u(t)$ which is the new resource allocation to be submitted to the system.

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt} \tag{1}$$

The constant coefficients K_p, K_i, & K_d are gain factors for the components P, I and D respectively, and are three hyperparameters in our design. CONtrol provides a flexibility in the choice of $y(t)$; any metric that gives an indication of the desired QoS can be used in place of $y(t)$. In our actual implementation of CONtrol, we define the storage and the network QoS based on the system performance counters captured by the DM.

Now, the output of the PID is a unitless value, which is simply a magnitude of the action to be taken. If it's positive, we should give resources to the container, and vice-versa if it's negative. So, the output cannot be directly used to adjust the resources of the container. Hence, we have passed the PID output though a sigmoid function, shown in Eq. (2) to get the appropriate resource value (here, the network bandwidth).

$$S(u(t)) = c\frac{e^{au(t)+b}}{1+e^{au(t)+b}} \tag{2}$$

Here a, b, & c are the control parameters decided as follows. c denotes the maximum resource that can be given to a container; the magnitude of a denotes the responsiveness of the system while allocating resources, and b denotes the minimum resource needs to be provided to the container in the event of a QoS drop.

Once the PID controller has decided on how much resource to allocate to the container, the container forwards this instruction to a network resource manager, which is in charge of creating network policies for individual containers that they are bound to follow. A policy is of the format {`container ID, maximum upload rate, maximum download rate`} which are set at the network interface of the host. We can use a software defined networking (SDN) based approach here to embed the policy at the switch level.

III. Implementation and Results

We have implemented CONtrol over a server cluster with five Hewlett-Packard HP BL460C Gen9 servers $\mathcal{S}_1$ to $\mathcal{S}_5$, each having 256 GB RAM and two Intel Xeon(R) E-5 2698 processors with a total of 80 cores with hyperthreading, along with 1.1TB usable storage. The servers are interconnected with a top-of-rack switch 10Gbps downlink and 40Gbps uplink bandwidth. Each server is configured with Ubuntu 18.04 server edition along with Docker 19.03. To create the shared storage environment, we have used `GlusterFS`, which is a scalable network file system. We have created a 900GB Gluster partition in each server and have set the replication policy as "distributed replicated". In distributed replication, all disks have the same content; any write performed in one server is replicated to all other servers. This enables high data redundancy and fast migration of containers. For generating the system load, we have used containerized-version of standard benchmarking toolboxes. `fio` is used to perform large disk writes; whereas to generate the network load, we have used `apache-http` server to host multiple files and `apache-benchmark` to fetch the files from the web-servers to emulate the network traffic. The `apache-benchmark` runs on a separate client server, and generates network traffic load for the containers running the web servers.

We run 10 `fio` containers in $\mathcal{S}_1$ to emulate the disk load, and execute varying numbers of `apache-http` servers in $\mathcal{S}_2$ and $\mathcal{S}_3$ to emulate an outbound network load. Meanwhile, $\mathcal{S}_4$ and $\mathcal{S}_5$ run `apache-benchmark` to generate inbound network load by fetching data from the `apache-http`

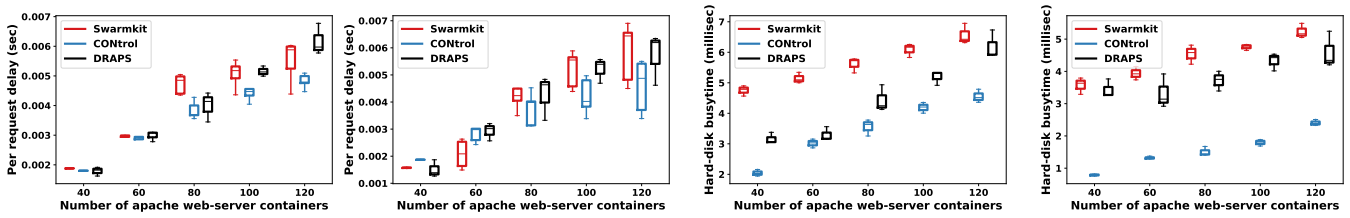

Fig. 2: CONTrol Performance: Avg. web server request delay with respect to number of web servers for `Cubic` (a) & `DCTCP` (b), Avg. disk-write busytime with respect to number of web servers for `Cubic` (c) & `DCTCP` (d)

servers running over $\mathcal{S}_2$ and $\mathcal{S}_3$. It can be noted that although we initially keep all the 10 disk workload-intensive containers over $\mathcal{S}_1$, the storage access affects all the servers due to the "distributed replicated" setup of the `GlusterFS` file system. The number of network workload-intensive containers in each server is varied from 40 to 120. The `apache-http` servers contain files from 0.25MB to 4MB, which are fetched by the `apache-benchmark` clients. In `fio`, the write block-size, which is the amount of data to be written at every write request, is set as 256MB. With this setup, every disk workload-intensive container consumes around 600Mbps of network bandwidth. We run each experiment multiple times (5 runs), where each run is executed for 180 minutes. The purpose of this experiment is to observe how well our CONtrol model handles a large number of network containers in the presence of disk workload.

We compare the performance of CONtrol with the default docker-based container management (Swarmkit) and DRAPS [8] which is a dynamic resource-aware container placement strategy based on the heterogeneity of available resources. DRAPS uses a methodology to identify the dominant resource type considering the demands from different services and accordingly places the containers by balancing the resource usage across devices. As the underlying TCP congestion control also influences the bandwidth redistribution at the shared network link, we further evaluate the performance of the proposed approaches with two different TCP variants – TCP Cubic and DCTCP [5]. DCTCP is particularly designed to handle the large network flows over a datacenter.

Figs. 2(a) and 2(b) indicate the delay from the first experiment for TCP Cubic and DCTCP. We observe that when the number of containers is low, the network request-response delay per request is almost similar for both CONtrol and the baselines. However, as the system gets stressed with more number of network workload-intensive containers, the delay per request increases rapidly for the other baselines, whereas it rises slowly for CONtrol. This is because as the number of `apache-http` containers are increased, the amount of network traffic increases, which causes high traffic contention among the network and the disk containers. Such an underlying network contention is oblivious to Swarmkit and DRAPS, as they check the resource usage entirely from the application perspective and do not have information about the network resource utilization by the `GlusterFS` storage management. Further, we observe that DCTCP yields a low delay because of the TCP congestion control management. However, it cannot eliminate the problem completely, and CONtrol over DCTCP performs best among all the competing heuristics.

Next, we analyze the QoS performance for the storage or disk workload, which is shown in Figs. 2(c) and (d). The average disk-write busytime indicates the latency for completing a disk-write operation. Therefore, a low value of this metric indicates better disk QoS. From the figure, we observe that the busytime is far less in CONtrol compared to other baselines.

IV. Conclusion

In this poster, we develop an architecture for container QoS management in a shared-storage hyperconverged environment, where storage workload-intensive containers and network workload-intensive containers interfere due to the shared network bandwidth. We developed CONtrol, a PID based storage-network conflict mitigation model, which improves the QoS of containers running in the aforementioned environment. From initial performance investigation, we observe that CONtrol outperforms other baselines in terms of application QoS while indulging less management overhead.

References

[1] R. S. C. Siddalingaiah, M. Vanninen, R. G. Costea, R. Carter, and E. Chiu, "Hyperconverged infrastructure supporting storage and compute capabilities," jun 2019, uS Patent App. 10/318,393.

[2] A. Darabseh, M. Al-Ayyoub, Y. Jararweh, E. Benkhelifa, M. Vouk, and A. Rindos, "Sdstorage: a software defined storage experimental framework," in *IEEE ICCE*, 2015, pp. 341–346.

[3] L. Guo and I. Matta, "The war between mice and elephants," in *IEEE ICNP*, 2001, pp. 180–188.

[4] L. A. D. Knob, R. P. Esteves, L. Z. Granville, and L. M. R. Tarouco, "SDEFIX – identifying elephant flows in SDN-based IXP networks," in *IEEE/IFIP NOMS*, 2016, pp. 19–26.

[5] A. M. Abdelmoniem and B. Bensaou, "Hysteresis-based active queue management for TCP traffic in data centers," in *IEEE INFOCOM*, 2019, pp. 1621–1629.

[6] I. Stathopoulos, F. Papandreou, and S. Manesis, "Position tracking and control of lightweight flexible joint manipulator robolink® using kinect sensor," in *IEEE MED*, 2018, pp. 788–793.

[7] H. Zhao, A. Cui, S. A. Cullen, B. Paden, M. Laskey, and K. Goldberg, "FLUIDS: A first-order lightweight urban intersection driving simulator," in *IEEE CASE*, 2018, pp. 697–704.

[8] Y. Mao, J. Oak, A. Pompili, D. Beer, T. Han, and P. Hu, "DRAPS: Dynamic and resource-aware placement scheme for docker containers in a heterogeneous cluster," in *2017 IEEE 36th IPCCC*. IEEE, 2017, pp. 1–8.