Blender: A Traffic-Aware Container Placement for Containerized Data Centers

1st Zhaorui Wu, 2nd Yuhui Deng, 3rd Hao Feng
Department of Computer Science, Jinan University, Guangzhou, China
diom_wu@163.com, tyhdeng@jnu.edu.cn

4th Yi Zhou
Department of Computer Science, Columbus State University, Columbus, American
zhou_yi@columbusstate.edu

5th Geyong Min
Department of Computer Science, University of Exeter, Exeter EX4 4QF, United Kingdom
g.min@exeter.ac.uk.

***Abstract*—Instantiated containers of an application are distributed across multiple Physical Machines (PMs) to achieve high parallel performance. Container placement plays a vital role in network traffic and the performance of containerized data centers. Existing container placement techniques do not consider the container traffic pattern, which is inadequate. To resolve this conflict, we investigate network traffic between containers and observe that it exhibits a Zipf-like distribution. We propose a novel container placement approach - *Blender* - by leveraging the Zipf-like distribution. Based on network traffic correlation, Blender employs *RefineAlg* and *SplitAlg* to divide containers of applications into blocks, and place these blocks across virtual machines. Blender exhibits two salient features: (i) it minimizes inter-block traffic by arranging the containers that communicate frequently in the same block. (ii) it achieves good load balancing by combining blocks according to the resource types they require and distributing them across multiple PMs. We compare Blender against two state-of-the-art methods SBP and CA-WFD. The experimental results show that Blender significantly reduces communication traffic. In particular, for the same number of PMs, Blender reduces the traffic of SBP and CA-WFD by 22% and 32%, respectively. Furthermore, with Blender in place, the physical resources of hosting PMs are well balanced and utilized.**

***Index Terms*—container placement, traffic-aware, load balancing, data centers, cloud computing.**

I. Introduction

In containerized data centers, the microservice architecture allocates the containers belonging to the same application in various component, these containers distributed across multiple Physical Machines (PMs) have to communicate with each other to provide desired applications [1], [2]. Such a communication between containers causes massive network traffic. For example, a web application may traverse hundreds of containers within a data center to obtain the result of a single query request. It was demonstrated that intra-data center traffic consumes nearly 70% of the network bandwidth, and 40-90% of traffic are across racks [3], [4].

Traditional strategies focus on consolidating containers for high availability, load balancing and financial cost saving [5], [6]. Most existing traffic solutions [7], [1], [8] distribute containers across different PMs to achieve load balancing and meet service continuity requirements. However, such a wide distribution of containers has two disadvantages. First, it causes a lot of unnecessary traffic. Second, it fails to leverage the traffic patterns for optimizing the network performance. It is reported that the network traffic generated by microservices belonging to the same application demonstrates a Zipf-like distribution [9], [10]. Because assigning one container for one microservice can achieve better performance [11], we can conclude that the network traffic incurred by containers belonging to the same application also demonstrates a Zipf-like distribution. Additionally, the network traffic pattern in data centers depends on the interactive behavior between different tasks instead of service architecture used (i.e., microservice architecture), we believe the above conclusion can be applied to the non-microservice architecture scenarios.

To address the aforementioned issues, we propose a novel traffic-aware container placement called Blender by leveraging the Zipf-like distribution of container traffic. We design a module called Refine&Split which relies on two sub modules - Refiner and Splitter - to restructure the containers belonging to the same application on a Descending Weight Sequence (DWS). Refiner assigns the frequently communicating containers close to each other. Splitter divides DWS into relatively independent blocks based on traffic correlation. It is worth noting that the "block" in this paper refers to a data structure composed of several containers, and the network traffic of intra- and inter-block are intensive and negligible. Furthermore, an iterate module is designed to facilitate load balancing by combining blocks with different resource requirements.

It is noteworthy that containers are assigned on Virtual Machines (VMs), and VMs are hosted on PMs. The main contributions of this study are summarized as below:

- We classify containers into blocks according to their traffic correlation. Specifically, the containers with strong traffic correlation between each other are assigned to a block. This container classification strategy generates intensive intra-block traffic yet negligible inter-block traffic, aiming to minimize the traffic cost of data centers.
- We formulate a mathematical container placement problem with the aim of minimizing the traffic cost of containerized data centers.
- We propose a traffic-aware container placement approach (i.e., Blender) that strikes to achieve a balance between traffic reduction and load balancing. Blender integrates two important modules, namely, Refine&Split and Iterate.

*Corresponding author: Yuhui Deng, tyhdeng@jnu.edu.cn

978-3-9819263-5-4/DATE21/©2021 EDAA

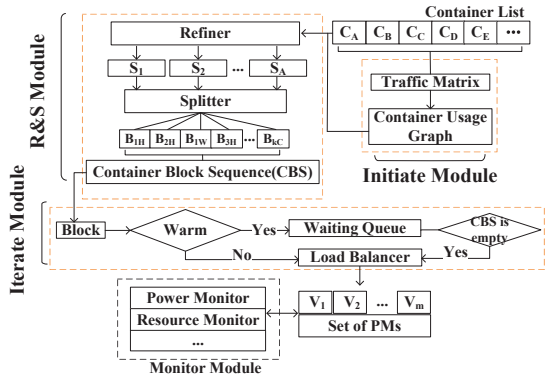

Fig. 1. Architecture overview of the container placement system

Refine&Split module focuses on reducing the overall traffic cost by refining the containers into several DWS according to the application ID; Iterate module targets at enhancing load balancing by distributing the blocks across different PMs.

The rest of the paper is organized as follows. Section II introduces related work. The design and implementation of Blender are given in Section III. Section IV presents quantitative performance evaluation. Last, Section V concludes this paper.

II. Related Work

In the past few years, there has been a lot of work focusing on network traffic and load balancing in modern data centers. Li *et al.* designed a Virtual Machine (VM) placement approach SBP to minimize the two costs. SBP recursively splits a group of VMs into sub-groups and places these sub-groups on PMs in a first-fit manner based on the amount of resources required by each request. Lv *et al.* proposed a container placement called CA-WFD to achieve traffic cost reduction and load balancing [1]. Dzmitry *et al.* presented a task scheduler e-STAB that emphasizing the role of communication fabric, to achieve traffic load balancing and prevent network congestion [12]. Xie *et al.* proposed a network topology structure Totoro that satisfies the design goals of scalability, high network capacity and robustness [13]. Zhou *et al.* applied a workload skewness strategy to reduce data loading traffic, aiming to improve the energy-efficiency of clusters [14].

Summarizing the related work, there are some strategies have been proposed to balance the traffic cost reduction and load balancing, but they only provide some simple placement schemes that fail to consider the traffic correlation between containers. As the number of containers increases, the pattern of inter-container traffic become more ane more apparent, especially in microservice architecture. Different from the existing works, Blender reduces the traffic cost and achieves load balancing by considering the container traffic pattern.

III. Design and Implementation

A. Architecture Overview

Blender consists of four modules, namely, the Initiate Module, the Refine&Split Module, the Iterate Module and the Monitor Module (Fig. 1). The Refine&Split Module focuses on distributing the containers into various blocks aiming to reduce traffic cost. Then, the Iterate Module strives to blend and place these candidate blocks on VMs according to the types of blocks, aiming to achieve load balancing. For example, *CPU-intensive* blocks can be aggregated with *memory-intensive* blocks. The Monitor Module tracks status information (i.e., resource utilization.) and feeds this information back to the scheduler to adjust container placement decisions.

B. Formulating Traffic Reduction

To provide guidelines for Blender to place containers, we mathematically formulate container placement. Container pairs have communication only when they meet the condition $\mathbf{A}(C_i) = \mathbf{A}(C_j), i \neq j$, where C_i indicates the ith container and $\mathbf{A}(C_i)$ represents the application ID to which the C_i belongs. And we assume that traffic between containers is mutual (i.e., $D_{ij} = D_{ij}$), containers that do not serve the same application have no traffic. Given the physical link distance l_{ij} between two communicating containers C_i and C_j, the total traffic cost can be calculated as:

$$Cost_T = \sum_{\forall C_i, C_j \in \mathbf{C}} l_{ij} * D_{ij} * \alpha_{ij}, \tag{1}$$

where α_{ij} = 1 if $\mathbf{A}(C_i) = \mathbf{A}(C_j), i \neq j$, else α_{ij} = 0. Recall that Blender aims to (i) reduce traffic cost reduction (ii) while achieving load balancing. Load balancing strategies should be designed to flatten as many resources as possible onto the host. In our model, the efficiency of load balancing is evaluated by the resource utilization.

C. Initiate Module

The Initiate module constructs a container usage graph based on the traffic matrix that created by leveraging the Zipf-like distribution. The container usage graph illustrates resource requirements of containers and the communication relationships between the containers. For an application i, its graph is represented as $G_i(V, E)$. Each vertex corresponds to a container, and the vertex weight indicates its resource demands. The initiate module initializes edge weight as the traffic volume between containers, $E\langle C_A, C_B\rangle$ represents the traffic volume between containers *A* and *B*. After the container placement is completed, Equation (1) is leveraged to calculate the traffic cost of the data center.

D. Refine&Split Module

We delve into Refine&Split module which relies on two sub-modules: Refiner and Splitter. Refiner first identifies, sorts, and merges containers with the same application ID into a DWS (such as S_i in Fig. 1, where i indicates the ith application) by using RefineAlg algorithm (see Lines 4 - 9 in Algorithm 1). Second, Splitter divides each DWS into three tagged blocks,

Algorithm 1 Refine&Split

```
1: function getIndex(start,end):
2: return i where (T_i − T_{i−1}) is MAX, i ∈ (start, end)
3: for all A ∈ Application do
4:     Summing all container's degree value up
5:     DWS ← DWS initialization function
6:     LCPDWS ← DWS
7:     for all c ∈ C and A(c) == A do
8:         Update the DWS and LCPDWS
9:     end for
10:    x ← getIndex(0, 0.3 * Len_DWS) // 30% of containers
11:    y ← getIndex(x, 0.5 * Len_DWS) // 50% of containers
12:    Hot block ← [C_1, C_x]
13:    Warm block ← (C_x, C_y]
14:    Cold block ← (C_y, C_Len]
15:    CBS ← CBS ∪ Hotblock ∪ Warmblock ∪ Coldblock
16: end for
17: return CBS
```

namely, *Hot*, *Warm*, and *Cold* block by using SplitAlg algorithm (see Lines 10 - 14 in Algorithm 1), according to traffic correlation of containers. In Fig. 1, B_{iH}, B_{iW}, and B_{iC} indicate the *Hot*, *Warm* and *Cold* block for application i, respectively. Third, the tagged blocks of all applications are combined into a Container Block Sequence (CBS).

1) Refiner: Given a set of containers, we first calculate the total traffic by summing up all the traffic volume related to each container. Second, we initialize a DWS by selecting the top two containers with the highest traffic proportion. That is, assume that the traffic proportion of the top two containers C_i and C_j are $\frac{D_{ij}}{T_i} = A$ and $\frac{D_{ji}}{T_j} = B$, respectively, where T_i denotes the total traffic generated by C_i. In the meantime, the Last Container Pair in DWS (LCPDWS) is initialized as $[C_i, C_j]$ if $A > B$. Last, we identify the next container pair with the second-highest traffic by traversing containers that communicate frequently with the container pair stored in LCPDWS. Once the pair of containers with the second-highest traffic are located, we update the DWS. In the same breath, we set the current LCPDWS to the newly found container pair for the next iteration. Such a search and update iteration continues until all containers of an application are sorted out (see Lines 7 - 9 in Algorithm 1).

2) Splitter: Guided by the traffic pattern of the Zipf-like distribution, Splitter encapsulates the top x% (any value in (0,30%]) frequently accessed containers as a *Hot* block. The next y% (any value in (x%, 50%]) containers and remaining containers are constructed as a *Warm* block and a *Cold* block, respectively. The values of x and y vary for each DWS (see Lines 10 - 14 in Algorithm 1). In this way, blocks of various applications with different tags can be placed anywhere without any worries in terms of traffic reduction because of the negligible inter-block traffic. After the refining and splitting phase, all application blocks are merged into a CBS, which is used for block placement by the Iterate module.

E. Iterate Module

To place the containers in a well-balanced load manner, another tag is assigned to each block to indicate its primary type of the required computing resources (e.g., *CPU intensive*, *memory intensive*, etc.). Given a block, the amount of MIPS requests and the amount of the memory request from all containers are divided by the unit of system resources, for example, 18637 for MIPS and 2048 for memory, respectively. Then, the block is tagged *CPU intensive* if the weighted CPU request is larger than the weighted memory request, and vice versa.

The Iterate module makes use of a two-round iteration method to place blocks across VMs. In the first iteration, *Hot* and *Cold* blocks that require different primary types of resources are mixed to boost the overall resource utilization of the VMs. *Warm* blocks from the CBS are temporarily pushed into a waiting queue until the first iteration is accomplished. Furthermore, a resource reservation mechanism is adopted to guarantee there is enough capacity in the VMs to host the *Warm* blocks in the second iteration. The second iteration is triggered after the CBS is empty, the *Warm* blocks in the waiting queue are inserted into the VMs, which improves the load balancing.

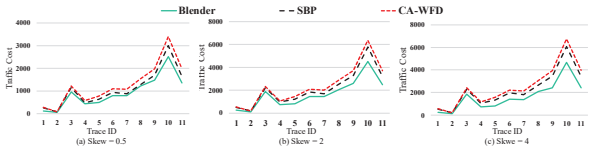

Fig. 2. Comparison of the traffic cost with different algorithms across different traces

IV. Performance Evaluation

A. Experiment Environment

In our extensive experiments, We implement Blender in ContainerCloudSim [15]. We build a testbed which consists of 250 heterogeneous PMs and 1,100 VMs with various types of resources. We adopt an Alibaba cluster trace [16] and split them into 11 parts (from Trace 1 to Trace 11) to validate our Blender. To evaluate Blender, we compare our Blender with two state-of-the-art strategies SBP [8] and CA-WFD [1], because these two strategies are the most relevant to our Blender in terms of application scenarios.

B. Performance Evaluation

Now, we are in the position to evaluate the performance of our Blender. We compare the traffic cost and traffic skew sensitivity of our Blender with existing schemes SBP and CA-WFD. Fig. 2 plots the traffic cost of the three container placement schemes on 11 traces, with Skew is set to 0.5, 2, and 4, respectively.

Traffic Cost Reduction. Fig.2 reveals that Blender outperforms SBP and CA-WFD in terms of traffic cost; this trend is consistent under the three skew conditions. For instance, Figs.2 (a) (b) and (c) show a similar trend, Blender achieves an average traffic cost reduction of 20% and 30% compared with SBP and

CA-WFD. Fig.2 also shows that regardless of the skew value, Blender performs better on trace 1 than it does on traces 2 to 11. For example, compared with SBP and CA-WFD, Blender reduces the traffic cost ranging from 52% to 55% and from 54% to 59%, respectively. On the other hand, with the increase of the skew value, Blender outperforms SBP and CA-WFD. This is because Blender integrates the frequently communicated containers, thus reducing the price for communicating through physical distances.

These results confirm that Blender successfully arranges frequently communicated containers into a set of blocks and distributes these blocks across VMs, which reduces the traffic path length between the containers, thereby significantly cutting off the traffic cost.

TABLE I
RESOURCE USAGE ACROSS DIFFERENT APPROACHES

	Blender	SBP	CA-WFD
Average of CPU	62%	60%	52%
Variances of CPU	0.0088	0.0370	0.0145
Average of Memory	67%	67%	53%
Variances of Memory	0.0096	0.0148	0.0145
Number of active PMs	232	234	240

Load Balancing. Because the resource utilization of PMs is a very good indication for the load balancing evaluation, we investigate the CPU and memory utilization incurred by Blender, SBP, and CA-WFD, with respect to different PM number. Table I shows that the average CPU utilization of Blender, SBP, and CA-WFD are 62%, 60%, and 52%, respectively. The memory utilization of the three approaches are 67%, 67% and 53%, respectively. Moreover, the number of active PMs are 232, 234 and 240, respectively. In contrast to SBP and CA-WFD, Blender improves the resource utilization of PMs by using the resource reservation mechanism for the first placement iteration. Thus the *Warm* blocks with typical resource tag (*CPU intensive* or *memory intensive*) can be inserted into PMs to improve the resource utilization. Hence, the number of PMs activated by Blender is lower than that of SBP and CA-WFD. Additionally, we have noticed that the average memory utilization of Blender and SBP are identical. This is because we use the same traces to evaluate the three approaches, and the VMs and PMs are heterogeneous. Therefore, the containers that are placed in VMs with different processing ability incur different resource utilization of PMs.

Furthermore, the variances of CPU utilization with Blender, SBP, and CA-WFD are 0.0088, 0.0370, and 0.0145, respectively, and the variances of memory utilization with the three methods are 0.0096, 0.0148, and 0.0145, respectively. It shows that the variances of resource utilization when using Blender are much smaller than that of SBP and CA-WFD. This indicates that in contrast to SBP and CA-WFD, the physical resources of hosting PMs are well balanced and utilized when applying Blender. This is because Blender is capable of combining *CPU-intensive* and *memory-intensive* blocks into a single suitable VM, which not only improves the resource utilization but also achieves load balancing. This is consistent with the variances of resource utilization.

V. CONCLUSIONS

In this paper, we proposed Blender, a holistic framework that strikes to achieve a balance between traffic cost reduction and load balancing by leveraging a Zipf-like distribution. We conducted extensive experiments to evaluate the performance of Blender. The evaluation results show that Blender significantly outperforms two state-of-the-art methods SBP and CA-WFD in terms of traffic cost, resource utilization, and robustly.

VI. ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China under Grant No. 62072214 and Grant No. 61572232, and the International Cooperation Project of Guangdong Province under Grant No. 2020A0505100040. The corresponding author of this paper is Yuhui Deng.

REFERENCES

[1] L. Lv, Zhang *et al.*, "Communication-aware container placement and reassignment in large-scale internet data centers," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 540–555, 2019.
[2] Y. Zhang, Xu *et al.*, "Going fast and fair: Latency optimization for cloud-based service chains," *IEEE Network*, vol. 32, no. 2, pp. 138–143, 2017.
[3] T. Benson, Akella *et al.*, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010, pp. 267–280.
[4] K. Bilal, Khan *et al.*, "A survey on green communications using adaptive link rate," *Cluster Computing*, vol. 16, no. 3, pp. 575–589, 2013.
[5] W. Zhou, White *et al.*, "Improving short job latency performance in hybrid job schedulers with dice," in *Proceedings of the 48th International Conference on Parallel Processing*, 2019, pp. 1–10.
[6] Y. Deng, "What is the future of disk drives, death or rebirth?" *ACM Computing Surveys (CSUR)*, vol. 43, no. 3, pp. 1–27, 2011.
[7] X. Meng, Pappas *et al.*, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *Proceedings of the 2010 IEEE INFOCOM*. IEEE, 2010, pp. 1–9.
[8] X. Li, Wu *et al.*, "Let's stay together: Towards traffic aware virtual machine placement in data centers," in *Proceedings of the IEEE INFOCOM 2014-IEEE conference on computer communications*. IEEE, 2014, pp. 1842–1850.
[9] H. Aragon, Braganza *et al.*, "Workload characterization of a software-as-a-service web application implemented with a microservices architecture," in *Companion Proceedings of The 2019 World Wide Web Conference*, 2019, pp. 746–750.
[10] L. Durbeck, Tront *et al.*, "Energy efficiency of zipf traffic distributions within facebook's data center fabric architecture," in *Proceedings of the 25th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. IEEE, 2015, pp. 152–160.
[11] F. H. L. Buzato, A. Goldman, and D. Batista, "Efficient resources utilization by different microservices deployment models," in *Proceedings of the 2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, 2018, pp. 1–4.
[12] D. Kliazovich, Arzo *et al.*, "e-stab: Energy-efficient scheduling for cloud computing applications with traffic load balancing," in *Proceedings of the 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*. IEEE, 2013, pp. 7–13.
[13] J. Xie, Y. Deng *et al.*, "An incrementally scalable and cost-efficient interconnection structure for data centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 6, pp. 1578–1592, 2016.
[14] Y. Zhou, S. Taneja, C. Zhang, and X. Qin, "Greendb: Energy-efficient prefetching and caching in database clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 5, pp. 1091–1104, 2018.
[15] S. F. Piraghaj and a. o. Dastjerdi, "Containercloudsim: An environment for modeling and simulation of containers in cloud data centers," *Software: Practice and Experience*, vol. 47, no. 4, pp. 505–521, 2017.
[16] A. Inc., "Alibaba production cluster data v2018." https://github.com/alibaba/clusterdata, 2018.