SEAL: Hybrid Resource Distribution for Multi-tenant Data Centers

Yang Gao*†, Lu Li*, Jingjie Jiang*, Baohua Yang*†, Yibo Xue†‡ and Jun Li†‡
*Department of Automation, Tsinghua University, Beijing, China
†Research Institute of Information Technology, Tsinghua University, Beijing, China
‡Tsinghua National Lab for Information Science and Technology, Beijing, China
{gaoyang11, lilu08, jjx08, ybh07}@mails.tsinghua.edu.cn, {yiboxue, junl}@tsinghua.edu.cn

***Abstract*—This paper presents SEAL, a hybrid resource distribution algorithm for data centers. SEAL is designed based on two well-known algorithms: the Best-Fit algorithm and the Next-Fit algorithm. With a dynamic hybrid mechanism, SEAL guarantees good performance in four important aspects simultaneously: Scalability, Efficiency, Agility and Low-Fragment. Experimental results show that the efficiency of the SEAL algorithm is almost as good as the Best-Fit algorithm, while keeping faster querying speed (about two to four times of the Best-Fit algorithm).**

***Index Terms*—resource distribution, data center, algorithm**

I. Introduction

The distribution of resources, *e.g.,* storage, bandwidth and CPUs has been a fundamental problem in modern data centers [1]. Today's data centers are required to provide QoS (Quality of Service) guarantee for the dynamic resource requests among multiple tenants [2]. For example, a tenant might need certain number of virtual machines (VMs) to finish a task, while demanding both high bandwidth and low latency among all running VMs.

With the growing scale of data centers, one controller may have to distribute switch ports and deal with more than thousands of VMs [3]. Therefore, the distribution of switch ports is vital to data centers.

The requirement of resource distribution is increasingly complicated today. Most of these requirements focus on the following aspects:

- *Scalability*. An algorithm should maintain its performance of resource distribution with the growing scale of the data centers.
- *Efficiency*. The efficiency means the ability of satisfying as many requests as possible, with only limited resources.
- *Agility*. Agility is also an essential factor. A good algorithm should be agile enough to achieve the resource request with acceptable latency.
- *Low-Fragment.* Fragments are the resource blocks that are too small to be utilized temporarily or permanently, thus large numbers of fragments is unfavorable. A desirable algorithm must consider avoiding the fragmentation.

In the paper, our focus is on the optimization of switch port distribution. Usually, a switch port is connected to a physical server with tens of VMs. To meet all those requirements above, we propose a hybrid algorithm named SEAL (Scalability, Efficiency, Agility and Low-fragment), which is designed based on ideas from two classical memory allocation algorithms: the Best-Fit (BF) algorithm and the Next-Fit (NF) algorithm [4]. The BF algorithm, demanding sorting and comparing process, is efficient but not agile, while the NF algorithm is inefficient despite agility. The SEAL algorithm achieves near-optimal efficiency and fast speed with a smart hybrid scheme.

Experimental results show that SEAL algorithm is almost as efficient as the BF algorithm and as fast as the NF algorithm.

The following part of the paper is structured as follows. In Section II, problem statement and the design of SEAL are presented. Section III shows the evaluation results. Related work is summarized in Section IV, and the last section concludes the paper with discussion of future work.

II. Problem Statement and Algorithm Design

Port distribution is critical to resource efficiency and overall performance in data centers. A good algorithm design must be based on in depth analysis of the problem and features advantages towards the optimization objectives.

A. Problem Analysis

The problem of port distribution in data centers can be described as: suppose there are N AS switches (Access Switches)[1] in the data center network, as $\{W_1, W_2, \ldots, W_N\}$, and each switch has M connection ports. Usually, each switch port can only connect to one physical host. Suppose each physical host only has C VMs running on it. Generally, C is a constant. Thus a physical host or AS switch port can be taken as a basic distribution unit for scalability reason. The task of port distribution is usually conducted by an out-of-bound management node, such as a controller or one of the similar mechanisms most of today's data centers already have [5], [6].

The tenants' requests, as $R = \{R_1, R_2, \ldots\}$, is a discrete series in time domain. Each request involves Num_i of hosts. Here we assume that Num_i is always smaller than or equal to the switch port number so that each request can fit into a single AS switch, because data exchange between ports on the same AS switch is much faster and therefore provides high bandwidth and low latency connections among the VMs of the same tenant[7]. We define available ports on the same AS as

[1]Access Switch is a switch that connects physical hosts directly. It provides connections between the end-hosts and the upper network.

a resource block and use the Block Information Base (BIB) to record the state of the AS switches.

When a request is received, the controller will check the BIB and executes the resource distribution algorithm, to find which switch should respond this request. After the distribution, the controller updates the BIB and continues to wait for the next request.

B. Algorithm Design

1) Observation: We observe that the algorithm of resource distribution in data centers has three similarities with the traditional memory allocation algorithm. First, both of them are request-driven. Second, they are both required to dynamically locate an available resource block with sufficient size. Finally, they have similar metrics: scalability, efficiency, agility and fragmentation.

Nevertheless, the memory allocation algorithm cannot be directly applied to port distribution, due to the following reason. Unlike memory segmentation which introduces discrete fragments for memory blocks, there are no discrete fragments in port distribution, as ports of an AS switch is connected together inherently as a whole resource block. Consequently, the most widely deployed memory allocation technique, Buddy Algorithm [8] is unsuitable for port distribution. Also, the classical memory allocation algorithms, BF and NF, have their respective weaknesses: the BF algorithm has good efficiency at the cost of low agility and high-fragment; the NF algorithm responses quickly but its efficiency is undesirable.

2) SEAL Overview: Based on the observations above, mature solutions to memory allocation can help tackle the port distribution problem, but the classical algorithms each has some weaknesses in solving the problem. In order to take advantages of both BF and NF smartly, we propose the SEAL algorithm as a hybrid scheme.

In SEAL, we configure an optimization threshold *opt* to divide requests into two categories according to their size: small requests and large requests, and apply BF and NF respectively to achieve scalability, efficiency, agility and low-fragment at the same time.

3) Procedure of SEAL: SEAL consists of two vital steps to fulfill the task of distributing resources:

- *Request Classification.* We use *opt* to determine the type of a request and apply different algorithms accordingly. Thus, the selection of *opt* is critical to SEAL's performance. Classification criteria include the number of available resource blocks, the ratio of average request size to average resource block size and the frequency of relatively large requests compared with average requests [9]. Since all the three factors change from time to time, *opt* should be also selected dynamically to optimize the classification.
- *Distribution Strategy.* Once requests are classified, an appropriate algorithm should be used to distribute resources. For small requests, the BF algorithm can locate the most suitable resource block at an acceptable speed since it probably ranks top in the BIB. Meanwhile, applying BF algorithm here can achieve high efficiency. However, responding large requests with BF will reduce the agility considerably. Thus, we apply NF instead for its fast response. Besides, since saving large free blocks are meant to satisfy large requests, using NF here will not render degradation of efficiency.

To sum up, the procedure of SEAL is as follows: Based on a given *opt*, the category of an incoming request is decided. If it is a large request, we attempt to distribute resources in a NF-like way. We use a pointer reference to record where the search should begin and start to search till finding the first suitable resource block. After the distribution succeeds, the location of the current block is recorded and assigned to the pointer reference and the state table is updated. Otherwise, if the request is small, we apply a BF-like method. Before searching, the BF algorithm is designed to sort all available resource blocks from the least to the largest and generates the BIB. Then the system starts searching and compares the request size with the size of each free resource block from the least to the largest. Once a block's free size is larger than the request size, the system will distribute this most suitable block to the requesting tenant and then update the BIB.

SEAL makes a good trade-off among the four basic metrics: scalability, efficiency, agility and low-fragment. For a series of sorted resources, a small request is more possible to be satisfied at the first few resource blocks, whereas a large request will always take more steps of searching and comparing. We utilize the NF algorithm to tackle the large request. Thus fewer steps are required to finish searching, which augments agility. On the other hand, to solve the problem of efficiency degradation brought by NF, BF is used to guarantee that the small requests utilize the small free blocks and therefore large free blocks are saved for future requests.

C. Discussion

Here we discuss two aspects of SEAL. One is sorting expenditure. Using the BF algorithm, we have to maintain the BIB and run sorting and comparing on all resource blocks each time before searching. In SEAL, we only have to record and update the BIB of resource blocks which have smaller size than *opt*. The storage cost of the BIB is therefore reduced significantly and the response time is cut down.

The other is the selection of *opt*, which will affect the performance of SEAL significantly. If *opt* is too large, more requests will be defined as small requests and proceeded by BF. Consequently, the responding time will increase. On the contrary, if *opt* is too small, more requests are defined as large requests, which causes over-utilization of NF, leading to degradation of efficiency. The value of *opt* also need to change dynamically. With many resource blocks being distributed, run-time response of all the algorithms will reduce, since it need more time to search for a suitable block. Then *opt* should be adjusted to a smaller value since the agility of BF will become unacceptable. In an opposite process, some VMs will be migrated or relieved and thus the number of available

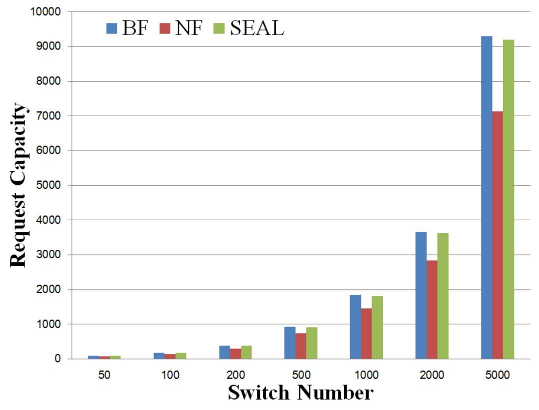

Fig. 1. Comparison on scalability between BF, NF and SEAL algorithm

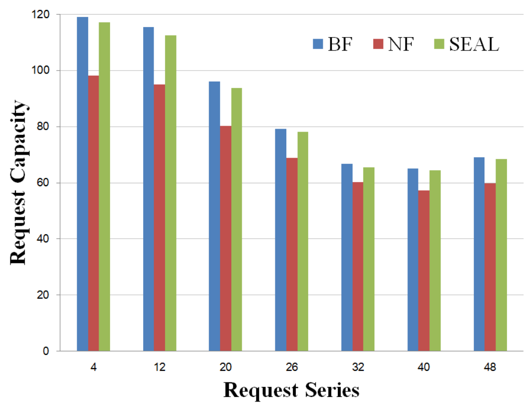

Fig. 2. Comparison on efficiency among BF, NF and SEAL Algorithm

blocks will increase. Then we could select a larger *opt* to achieve better efficiency.

III. Evaluation

SEAL is evaluated in comparison of BF and NF with the four metrics: scalability, efficiency, agility and low-fragment.

A. Scalability

In this section we use these algorithms to execute the distribution with seven different switch numbers. The switch number rises from 50 to 5000 and the request series is a normal random series. To test the normal situation, we fixate *opc* to the average between requests of maximum and minimum. The increase of request capacity reflects the scalability of different algorithms. Fig.1 shows the ability of the three algorithms to handle different resource scales. We can see all the three algorithms' request capacity increases at almost the same rate. And when the switch number becomes larger, the NF algorithm's capacity is much smaller than the other two algorithms. But the BF algorithm and the SEAL algorithm's capacity are still very close.

B. Efficiency

Here we use these algorithms to execute the distribution for seven different request series with different request trend.

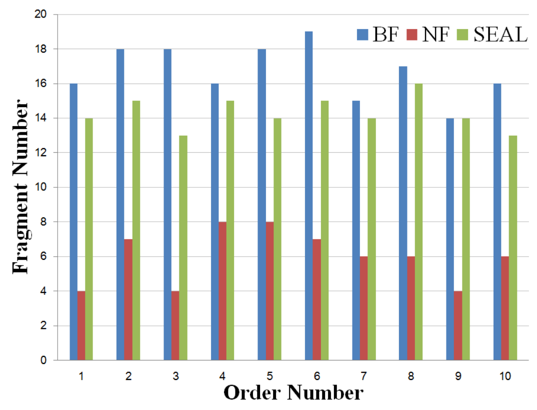

Fig. 3. Comparison on fragment number among BF, NF and SEAL algorithm

Request numbers have a larger probability to appear around the number below the column. To simulate the network environment of data centers, we set up 50 switches, each of which has 48 free ports on it and fixate *opc* to the average between requests of maximum and minimum. Each test of the three algorithms stops when all the switches' free port number is not large enough for distribution, and therefore the largest request number can reflect the efficiency. Fig.2 shows the results of the experiments. We can see the BF algorithm and the SEAL algorithm satisfy almost the same number of requests dealing with different types of request series, but the NF algorithm is much worse. This shows that our SEAL algorithm's efficiency is nearly as good as the BF algorithm.

C. Low-Fragment

The data of this part is collected from the experiments of the efficiency and is used to analyze the fragment of three algorithms. As mentioned above, we define fragments as the resources that cannot be used. In the simulation, the minimum of a request size is defined as 4. Thus, if free ports of one block are fewer than the minimum, this block would become a fragment and cannot be employed for resource distribution. We evaluate the fragment by the number of switches that have fragments and cannot be used. In Fig.3 we can see that the BF algorithm has the maximum fragment number, the NF algorithm has the minimum and the fragment number of the SEAL algorithm is moderate. Through calculating we can see that the BF algorithm and the NF algorithm both have a standard deviation of about 1.56 while the SEAL algorithm's standard deviation is about 0.95. This indicates the stability of the SEAL algorithm in fragmentation.

D. Agility

In this part we use the request series of the same trend to perform the test, but the number of request series is limited to 50, making sure all the three algorithms will not reach saturation. We calculate the overall querying latency for each algorithm to evaluate the cost of time (We suppose the system takes 1 ms per querying action). From Fig.4 we can see the test result. The querying latency of the NF algorithm is surely

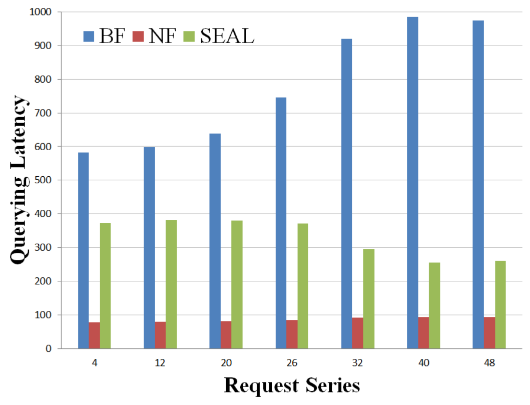

Fig. 4. Comparison on agility among BF, NF and SEAL algorithm

the least because if we still have unused free blocks, the algorithm won't take more than two querying actions to finish the distribution. On the contrary, the BF algorithm's latency is more than 10 times of the NF algorithm's because every time its querying action begins at the smallest free block. The SEAL algorithm's agility is just in-between: compared with the BF algorithm, its querying latency is averagely reduced by half.

IV. Related work

The increasing prominence of multi-tenant data centers has caused the network resource distribution problem. As mentioned above, this problem can be tackled by analogy with traditional memory allocation algorithm. Nevertheless, memory allocation algorithm cannot be directly utilized in network resource distribution scene. Thus, network resource distribution demands research of some new or improved algorithms. There have been some studies on this issue. Researchers mainly concentrate on presenting some communication models for network architecture of data centers. [7] abstracts two virtual network models exploring the trade-off between tenant cost and provider revenue. Concrete models such as VL2[5] and FatTree[10] are implemented to meet some particular goals.

The network bandwidth distribution has also arisen interest of researchers. Seawall[11] and NetShare[12] implement network weights to share the bandwidth among tenants. Hedera[13] describes a flowing scheduling system scheme to efficiently utilize aggregate network resources. SPAIN[14] constructs multiple path forwarding over the underlying physical component and allows end-hosts to specify the path to use for their packets.

The focus of the research above is quite different from what we are studying. But if available, these technologies and studying can be implemented together to enforce network distribution of data centers.

V. Conclusion and future work

In this paper, we proposed SEAL, a novel resource distribution algorithm for data centers. SEAL is designed based on two classical resource distribution algorithms BF and NF. All experimental results show that the SEAL algorithm can make a balance at the scalability and efficiency which BF algorithm achieves well and the agility which NF algorithm achieves well to meet our requirements and solve our resource distribution problems in data center network.

Our algorithm behaves well in terms of scalability, efficiency and agility when distributing switch ports. But more realistic simulations are still demanded to verify the performance of SEAL. Also, we consider implementing SEAL in other resource distribution of data centers. Moreover, there are still many interesting issues to touch in future. For example, if the number of requested hosts is more than the ports on a single AS switch, or fragment blocks across AS switches are used to satisfy a request, the situation will become more complicated as higher level switches would be involved. Our future work will focus on further developing novel algorithms for various resource requirements in more real-life data centers.

References

[1] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types," *8th Networked Systems Design & Implementation*, pp. 323–336, 2011.

[2] J. Mudigonda, P. Yalagandula, J. Mogul, B. Stiekes, and Y. Pouffary, "Netlord: A scalable multi-tenant network architecture for virtualized datacenters," in *ACM SIGCOMM*, 2011.

[3] Amazon ec2. [Online]. Available: http://aws.amazon.com/ec2/

[4] P. Wilson, M. Johnstone, M. Neely, and D. Boles, "Dynamic storage allocation: A survey and critical review," *Lecture Notes in Computer Science*, pp. 1–1, 1995.

[5] A. Greenberg, J. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta, "Vl2: A scalable and flexible data center network," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, pp. 51–62, 2009.

[6] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: A scalable fault-tolerant layer 2 data center network fabric," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, pp. 39–50, 2009.

[7] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," Technical Report MSR-TR-2011-72, Microsoft Research, Tech. Rep., 2011.

[8] K. Knowlton, "A fast storage allocator," *Communications of the ACM*, vol. 8, no. 10, pp. 623–624, 1965.

[9] J. Shore, "On the external storage fragmentation produced by first-fit and best-fit allocation strategies," *Communications of the ACM*, vol. 18, no. 8, pp. 433–440, 1975.

[10] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*. ACM, 2008, pp. 63–74.

[11] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha, "Sharing the data center network," in *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI). USENIX*, 2011.

[12] T. Lam, S. Radhakrishnan, A. Vahdat, and G. Varghese, "Netshare: Virtualizing data center networks across services," *University of California, San Deigo, Tech. Rep. CS2010-0957*, 2010.

[13] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proceedings of the 7th USENIX conference on Networked systems design and implementation*. USENIX Association, 2010, pp. 19–19.

[14] J. Mudigonda, P. Yalagandula, M. Al-Fares, and J. Mogul, "Spain: Cots data-center ethernet for multipathing over arbitrary topologies," in *Proceedings of the 7th USENIX conference on Networked systems design and implementation*. USENIX Association, 2010, pp. 18–18.