VirtualRack: Bandwidth-Aware Virtual Network Allocation for Multi-Tenant Datacenters

Tianlin Huang[+], Chao Rong[+], Yazhe Tang[+], Chengchen Hu[+], Jinming Li[*], Peng Zhang[+]
[+]Department of Computer Science and Technology
Xi'an Jiaotong University, Xi'an, China, 710049
[*]Huawei Technologies Co. Ltd
Shenzhen, China, 518129

***Abstract*—It has become a common practice that enterprises outsource their networks to the cloud by renting multiple virtual machines (VMs) in cloud datacenters. Due to the multi-tenant nature of cloud datacenter, how to efficiently share the network resources becomes an important issue. Recent studies, e.g., SecondNet and Oktopus, have taken network bandwidth into consideration when allocating VMs. However, these schemes are problematic in that the allocation is not that accurate, and can result in a low multiplexing rate. To this end, we present VirtualRack (VR), a new bandwidth-aware VM allocation scheme in multi-tenant datacenters. VR simultaneously considers intra-datacenter bandwidth and Internet-access-bandwidth requirements in the allocation process. In addition, we introduce a redundancy factor α that can be specified by tenants to accommodate their dynamic requirements. Simulation results show that VR can guarantee the network performance for each of the multiple tenants, and at the same time keep a high acceptance ratio without any false allocation.**

Keywords—Datacenter; Virtual Network; Bandwidth; Dynamic; Allocation

I. INTRODUCTION

Cloud datacenters are becoming increasingly popular due to their efficient use of computing resources with statistical multiplexing. Cloud tenants rent computation and storage resources from cloud datacenters by specifying the amount of CPU, memory and disk they need [1, 2, 3]. Currently, most cloud datacenters simply allocate these resources in the form of VMs, without any consideration of network bandwidth constraints. Which can hurt the performance of tenants' applications [4, 5, 6]. For instance, cloud datacenters usually allocate VMs in a locality-aware way by which VMs are placed as close as possible to each other without considering network issues [7]. As a result, some nodes in the network become hot-spots transmitting too much, while some tenants greedily use their network resources making others suffer from network congestions.

To solve this problem, recent studies propose some new bandwidth-aware resource allocation schemes, in which VMs are allocated with bandwidth guarantee. For instance, SecondNet [7] proposes a VDC (Virtual Data Center) model, in which tenants are mapped into different VDCs, and every VDC has its own SLA (Service Level Agreement) based on v2v (VM-to-VM pair) network requirements. The problem with SecondNet is that the v2v model is relatively complicated, and realizing the v2v model would consume much more resources than actually needed. This can reduce the amount of tenants a datacenter can serve at the same time, thereby hurts datacenters' revenue.

In Oktopus [6], VC (Virtual Cluster) and VOC (Virtual Oversubscribed Cluster) are proposed for VM allocation. VC allocates VMs in such a way that VMs connected into its virtual network (VC) receive a guaranteed bandwidth; VOC allocates VMs with one more parameter (the oversubscription ratio). By using access bandwidth guarantee it increases the multiplex rate of datacenter network. Yet, a problem with Oktopus is the proposed algorithm may miss the information of resources located two or more layers lower than the current layer. This may cause false allocation, i.e., allocate unavailable network resources to tenants. In addition, the oversubscription in VOC can cause chain reaction when multiple tenants dynamically change their requirements.

Besides the intra-datacenter bandwidth, Internet-access bandwidth can be equally important for cloud tenants, especially for that running web applications. Unfortunately, neither SecondNet nor Oktopus has taken the factor of Internet-access bandwidth requirement into account.

This paper proposes a new virtual network allocation framework named VirtualRack (VR). Compared with traditional allocation method (TAM for short), SecondNet and Oktopus allocation models, VR can guarantee tenants' intra-datacenter (VM-to-VM) bandwidth requirement and Internet-access (VM-to-Internet) bandwidth requirement. Moreover, VR is more user-friendly to cloud tenants. Unlike the v2v allocation model, tenants only need to specify the amount of VMs, the intra-datacenter bandwidth, the Internet-access bandwidth and a redundancy factor α (the usage of factor α will be explained later). Finally, VR can allocate VMs more accurately, without any false allocation caused by greedy allocation like Oktopus. To summarize, the differences of previous models and VR are listed in Table 1. Without considering bandwidth guarantee, TAM can allocate VMs freely, so it gets the highest flexibility of all. SecondNet and Oktopus take VM-to-VM bandwidth into account, but they both ignore the performance of VM-to-Internet and flexibility of tenant requirements. At the same time, they have complex user-interface and false allocation problem. VR takes all these factors into consideration. With a little sacrifice of flexibility, VR guarantees VM-to-VM and VM-to-Internet bandwidth achieving good application performance.

978-1-4799-2003-7/14/$31.00 ©2014 IEEE

	Bandwidth VM-to-VM	Bandwidth VM-to-Internet	User-friendly	Accuracy	Flexibility
TAM	No	Guaranteed	Media	High	High
SecondNet	Guaranteed	No	Low	High	Very low
Oktopus	Guaranteed	No	High	Low	Low
VR	Guaranteed	Guaranteed	High	High	Media

Table 1: TAM can allocate VM freely, but it fails to guarantee application performance. SecondNet and Oktopus guarantee bandwidth between VMs, but they both ignore the performance of VM-to-Internet and flexibility of tenant requirements. VR finds a trade-off between performance and flexibility.

We make following two contributions. First, we propose VR which can allocate virtual network to tenants to increase the service level of datacenter. Second, by some preliminary performance evaluation, we show that VR can increase the acceptance ratio of tenants' requirements by more than 30% over v2v allocation in SecondNet. At the same time, VR avoids false allocation caused by greedy allocation (like Oktopus) by 20% at a heavy-load datacenter. The communication between VMs in datacenters and Internet is more steady and fast. With different redundancy factor α, datacenters can be more elastic by 20%~80% when tenants dynamically change their requirements.

The rest of this paper is organized as follows. We present the architecture of VR in section 2. In section 3, we discuss how VR allocation algorithm works and present the problems some other existing methods have. We make simulation to evaluate VR allocation model in section 4. Finally, in section 5 we summarize the paper and provide an outlook to future work.

II. ARCHITECTURE

VirtualRack (VR) is a bandwidth-aware allocation model which appears to tenants like renting a rack from a datacenter. VR consists of VMs, VTS (virtual top-of-rack switch), links with intra-bandwidth (connecting VM with VTS) and links with inter-bandwidth (connecting VR with Internet). Conceptually VMs contain computation, memory and disk resources tenants can rent. For simplicity, we use the amount of VMs to represent tenants' VM requirements. VTS is not a real switch but a logical one, comprising switches and links, and providing the available network tenants need. In our model, the links connecting VM with VTS and links connecting VM with Internet are bandwidth guaranteed according to tenants' requirements. Fig. 1 illustrates the logical and real view of VR.

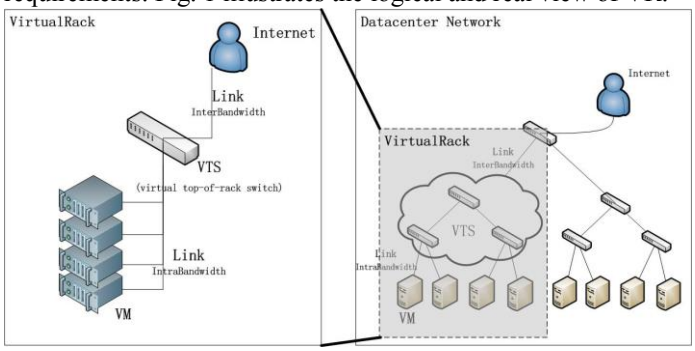

Figure 1: The left part is logical view of VR, and the right part is its real view

Intra-bandwidth is the bandwidth VM uses to connect with VTS. It promises that VM can communicate with other VMs in VR with intra-bandwidth at most, and the bandwidth will not be disturbed by other VMs neither in nor out of VR. Just like the 100Mbps NIC (network interface card) in our PC, 100Mbps is the fastest speed we can get to communicate with other PCs in the same LAN. With intra-bandwidth guaranteed, communication between VMs in the same VR can be steadier and the performance of applications hungry for network resources such as MapReduce [9] can be more predictable [6].

Inter-bandwidth is the bandwidth with which VTS transfers packets to outside of datacenters, i.e. the Internet. It guarantees a steady and available path from VR to Internet. This is helpful for applications working with more traffic from Internet like web servers [8], because they must handle users' requests and send back the responds quickly.

VTS is a logical switch acting as a top-of-rack switch. VTS can consist of a group of switches or just one switch. The core idea of VTS is to provide an available inter-connected network with bandwidth guarantee to VMs in VR. To realize this core idea, we use an optimization method, which will be described later in this paper, to increase the multiplexing rate of infrastructure in datacenters.

Redundancy factor α is introduced to make it easy for datacenters to adapt to tenants' dynamic requirement modification. By reserving some available resources around VR can be easily extended with nearby resources inside or outside current VR when tenants ask for more resources. Extending VR locally can shorten the path through which VMs communicate, and save the scarce network bandwidth at higher level of topology in datacenters. Therefore, the factor α is defined as V1/(V1+V2), where V1 means the amount of VMs the tenant requires and V2 means VMs reserved for that tenant.

As we discuss above, the interface VR exposes to tenants can be written as 4-tuple [N, Intra-bandwidth, Inter-bandwidth, α]. N is the amount of VMs tenants require. Intra-bandwidth is the bandwidth with which VM communicates with each other. Inter-bandwidth specifies the communication bandwidth between VR and Internet. Factor α defines how much resources will be reserved for further extension. Apparently this interface is so user-friendly that tenants can understand and use it conveniently.

VMs and switches can be treated as nodes and links act as edges connecting them, then allocating VR in datacenter network can be modeled as searching sub-graph with edge constraints in a graph. Since this searching problem has been proved to be NP-hard [17], people have been trying to solve it with heuristic methods. VDC allocation algorithm presented in SecondNet [7] and greedy algorithm presented in Oktopus [6] are two of them that we mentioned before. But these two solutions both have some sort of drawbacks. We will compare them with VR later.

III. ALLOCATION ALGORITHM OF VR

Today, most of datacenters deploy their network as tree topology [13]. To increase the throughput, datacenters usually adopt high-performance devices or new techniques like

aggregated link. For the same purpose, researchers try to design efficient network structures such as fat-tree [14], VL2 [15], BCube [16] and HULL [20]. Our allocation algorithm focuses on switch-centric topology like fat-tree and VL2. This kind of topologies resembles traditional tree topology, which are hierarchical and recursively made of sub-trees at each level. Different from the traditional one, our topology has rich links across intra-network. On the basis of this topology, we design an intuitive heuristic algorithm.

When a tenant's requirement comes as [N, intra-bandwidth, inter-bandwidth, α], VR will basically try to find a sub-tree of the datacenter's network with n (n=N/α) available VMs, VTS and a path from VTS to Internet with inter-bandwidth guarantee, like Fig. 2.

This allocation algorithm has three steps.

First, find available VMs. VR allocation algorithm marks each VM to display it is available or not. Switches, in turn, count the available VMs located in the downlink port areas as illustrated in Fig. 2. Then, we search switches one by one to find a switch whose available VM amount is the least among all the switches while bigger than n. This switch will be the root of a sub-tree for next step. Searching VMs connected with the same root can keep VMs as close as possible to each other, and then shorten the way VMs communicate and save higher level network resources. In some rich connectivity networks, higher level switch might repeatedly count the same available VMs. However, it will not lead to the failure of the allocation algorithm because with each VM marked, the repeat-counted VM will be detected at the last step and the whole algorithm will backtrack to find another root.

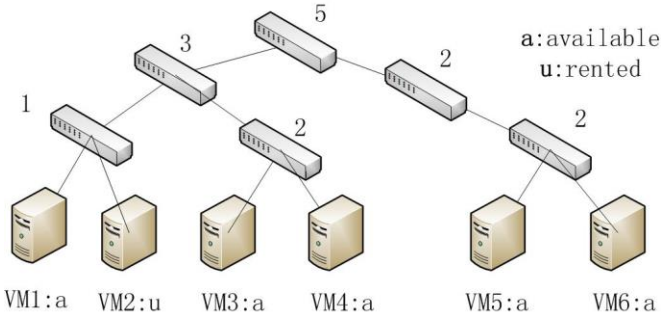

Figure 2: Number around switch show the available VMs in downlink port area of it

Second, construct VTS in the network with the consideration of intra-bandwidth. Intuitively, VTS is a sub-tree in datacenters' topology. VMs communicate with each other with intra-bandwidth guarantee through VTS. Intra-bandwidth is the highest speed VM can use without disturbing other VMs inside or outside VR. To guarantee bandwidth, VR constructs VTS to reserve bandwidth in each link connecting each VM in VR. However, reserving bandwidth in each link does not mean, for instance, that the link by which one VM communicate with ten VMs should reserve 10*intra-bandwidth. We use Hose model [19] to optimize the links forming VTS. Back to the previous case, if one VM communicates with ten VMs, the link connecting them only needs to reserve 1*intra-bandwidth to satisfy tenants' requests. Because no matter how many VMs are there at one side of the link (in this case, 10 VMs), the highest communication speed is determined by the smaller side of the link (in this case, 1 VM). This saves more network resources than v2v in SecondNet. Consequently, constructing VTS is to try to find a sub-tree with highest bandwidth guarantee to each VM.

VR takes the switch found in step one as the root of the sub-tree, and then checks if the links in it can meet the need of the tenants' requirements. Each downlink port of the switches in VTS is checked to find how many VMs this link can satisfy. The amount of VMs can be allocated to that link is:

$$\begin{aligned} &\text{maximize } k \\ &\text{s.t.} \\ &k \le \min(n, m) \\ &\min(k, n-k)\text{*intra-bandwidth} \le \mathrm{B}_l \end{aligned}$$

where *min(a,b)* means the minimum of a and b. Then, the actual bandwidth required by that link is:

$$\mathrm{B}_r = \min(k, n-k)\text{*intra-bandwidth}.$$

The parameters are detailed as follows: n is the total amount of VMs the tenant requires and m is the amount of VMs that destination switch of the link can support at that time. Then k is the amount of VMs that can be allocated to that link for the current tenant. Obviously k is smaller than the minimum value of n and m. B_l is the bandwidth left (unallocated) on that link while B_r is the bandwidth finally required. Note here we choose minimum value of k and n-k times intra-bandwidth as the actual required bandwidth by simply following the Hose model [19]. This link check will be done switch by switch until reaching the boundary of network.

Third, check for bandwidth guarantee path from VR to Internet. Because core switches can be seen as boundary switches that connect all VMs inside a datacenter with Internet, we use core switches to represent the Internet. We can enumerate uplink port layer by layer to find a path from VTS to the core switch. In the extreme case, there will be four levels including vswitch, top-of-rack switch, aggregation switch and core switch. To scale well with large datacenters, we design a 3-tuple matrix which stores the connection relationship between two switches via a third switch. The matrix is:

$$M = \begin{array}{c} \\ r_1 \\ r_2 \\ \dots \\ r_n \end{array} \begin{array}{c} \begin{array}{cccc} c_1 & c_2 & \dots\dots & c_n \end{array} \\ \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & \dots \\ \dots & \dots & \dots & \dots \\ a_{n,1} & \dots & \dots & a_{n,n} \end{bmatrix} \end{array}.$$

r_k ($0<k<n+1$) represents kth top-of-rack switch, c_k ($0<k<n+1$) represents kth core switch, while $a_{i,j}$ ($0<i<n+1$, $0<j<n+1$) logically represents the path consisting of a list of switch names via which r_i and c_j are connected. If we want to find a path from switch vs1 to core switch cs1, for example, we just need to enumerate top-of-rack switches vs1 connects with and check the paths from those top-of-rack switches to cs1 in the matrix. If such a path does exist, the whole allocation algorithm finishes successfully; otherwise, go back to step 1.

The three steps will be done again and again until VR cannot find the resources and reject the tenant's requirements.

Recall in first step, we introduce factor α to make VR more flexible. Ideally, tenants may modify their requirements, and datacenters should react to this change quickly. Generally, there are two kinds of modifications. One is decreasing the resources they have. The other is increasing the resources they want. For the first case, datacenters can just get back resources which are allocated to tenants. For the second, it is complex. If resources are allocated in an incremental way, the VMs newly assigned may be far away from the ones the tenant currently has. This will increase the communication delay between VMs and waste higher level network resources. On the other hand, if resources are allocated in a reallocating way, two problems may occur. One is that current available resources are not enough to allocate tenants' whole requirements; the other is that reallocating all the VMs a tenant has need to carry out VM migrations, causing much overheads. While in reallocating way, the reallocation will fail because the available resources left in the datacenter are not enough. However, if we provide tenants a factor α to predict their possible future requirements, we can get a more flexible VR. For instance, if tenant A rents VM1 and VM2 with α=2/3, then the datacenter will very likely reserve one more VM (VM3) for A. When A wants another VM later, the datacenter will certainly allocate VM3 very quickly. Figure 3 illustrates the pseudo code of VR allocation algorithm.

Algorithm : Allocation algorithm of VR

Input: amount of VMs *N*, intra-bandwidth of VR *ia*, inter-bandwidth of VR *ie*, redundancy factor *α*
Output: tenant's request can be allocated or not.

```
1.  n=N/α
2.  for topo-level i = low to high do
3.    sort by amount of available VMs in ascending order
4.    for switch s = begin to end of i and begin.vm > n do
5.      //check ia in the tree where s serves as root
6.      for link in s:
7.        find k best suit to link
8.        n=n-k
9.        if n<0
10.         break
11.       end if
12.     end for
13.     //check ie of VR
14.     for path in M: //M is the Matrix in step3
15.       if path available
16.         break
17.       end if
18.     end for
19.   if ia and ie are both satisfied
20.     return true
21.   else
22.     return false
23.   end if
24. end for
```

Figure 3: Pseudo code of VR

IV. Performance Evaluation

We evaluate VR in four aspects. First, we use *requirement acceptance ratio* to verify that VR can maintain a higher multiplexing rate of the infrastructure, compared with v2v allocation policy like SecondNet. Second, we measure the RTT (Round Trip Time) from switches to VMs, and calculate the mean time and arithmetic root to test whether VR can maintain a stable communication between VMs and Internet. Third, we test the effect of α to show the flexibility of VR when facing tenants' dynamic requirements of different load and request frequency. Finally, we demonstrate that VR can avoid false allocation posed in greedy allocation policy like Oktopus.

Our experiment setup is as follows: we use three-level tree-topology with 1/10 oversubscription rate that is consistent with the observation made in [15]. The core switch is connected with the aggregation switch, which in turn is connected with all top-of-rack switches. Every rack has 5 servers and each server contains 4 VMs. Totally, there are 3200 VMs in the network. We use Mininet [18] to simulate this network. Fig. 4 illustrates the logical view of the network configuration used in our experiments.

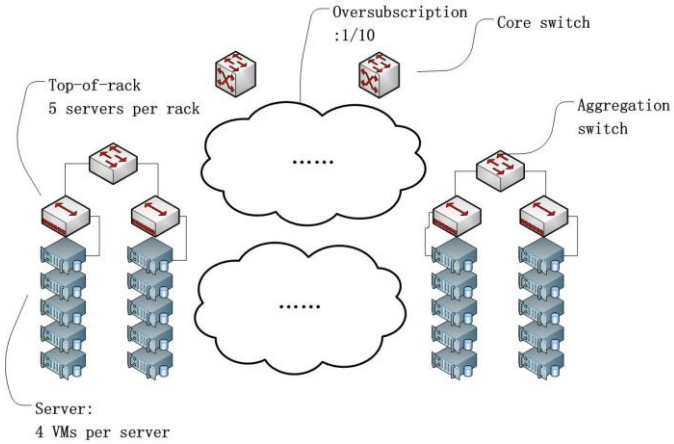

Figure 4: Platform of Simulation

In the simulation, VR will be compared with other allocation models. To get precise results, we generate the same list of tenant requirements for different allocation models. We assume the number of VMs required by each tenants follows an exponential distribution with the mean being 49, consistent with observation made in real datacenters [5]. We also assume that the bandwidth required by each tenant is exponential distributed.

Experiment 1: Acceptance ratio. We use the *requirement acceptance ratio* (defined as the ratio between the number of satisfied requirements and number of total requirements) to demonstrate the benefits VR gains compared with v2v policy of SecondNet. To make the comparison reasonable, we use the same configurations for both simulations. That is, we use the same tenant requirements, the same distribution for the number of VMs required, and the bandwidth demanded. We run the simulation for ten times, and obtain the average results. As shown in figure 5. VR-500Mbps (100Mbps) represents the case of VR with 500Mbps (100Mbps) bandwidth requirement;

SecondNet-500Mbps (100Mbps) represents the case of v2v policy with 500Mbps (100Mbps) bandwidth requirement. We observe that when the lower bandwidth requirements are, the higher the acceptance ratios, which are intuitive correct. We also observe that VR can always accept 20% more requirements over v2v policy, whatever the requirements are. In addition, it can be seen from Fig. 5 that the acceptance ratios drop sharply when the number of required VMs exceeds 120. The reason is that the network in our simulation has 3200 VMs and the mean amount of VMs a tenant requires is 49. When the requirements exceed 120, all resources in the network would be completely allocated (120*49=5880).

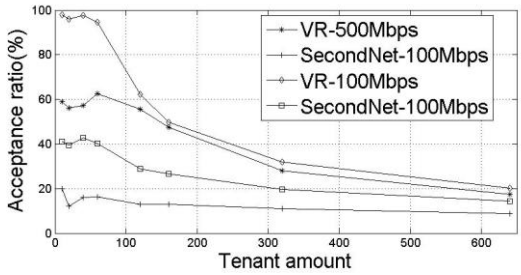

Figure 5: VR accept more tenant requirements than SecondNet

Experiment 2: Communication speed and stability. In this experiment, we use a host connected with the core switch to simulate the Internet access point. Again, we use the same configurations for each simulation and generate the same tenants' requirements for different allocation models. Then, we randomly choose 10 VMs from network. To make our experiment realistic, we let VMs in each VR generate steady flows to simulate the tasks running on them. By measuring the RTTs from the core switch to VMs and calculating the mean and arithmetic root of these RTTs, we obtain the results for VR, TAM, SecondNet and Oktopus, as shown in Figure 6. For VR, the mean RTT obtained from the randomly chosen VMs is lower than TAM and Oktopus, but a little higher than SecondNet. This is because in SecondNet, a tenant's requirement is relatively hard to satisfy, thus there are fewer tenants in the network, resulting lower RTT. On the contrary, VR, TAM and Oktopus can accept much more tenants' requirements, so their network loads are much heavier than that of SecondNet. Similarly, the arithmetic root of RTT in VR is smaller than all others but SecondNet, meaning that VR can achieve a more stable performance. Note that the arithmetic root of RTT in TAM is the biggest of all. The reason may be that in TAM, tenants' VMs are allocated without network consideration, hence VMs are placed unevenly: some VMs are placed too close to disturb each other, while others are placed far from each other and thus enjoy more network resources. Oktopus has a small mean RTT value, but the false allocation allocates VMs with wrong information of network. Consequently, Oktopus has a bigger arithmetic root than VR causing the problem like TAM has.

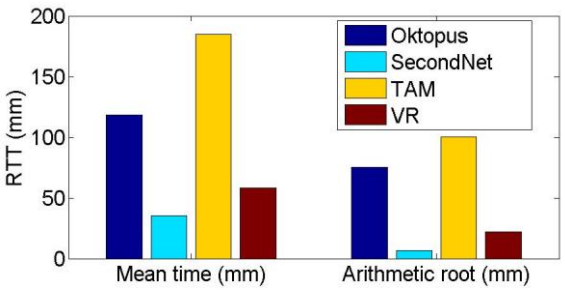

Figure 6: Mean value and arithmetic root of RTT between VMs and Internet shows the communication quality in different allocation models

Experiment 3: Allocation flexibility. In this experiment, we assign each tenant with a parameter α. By varying the number of tenant requirements, we simulate datacenters with different load. Tenants' dynamic VM requests are modeled with standard exponential distribution. Without loss of generality, the mean value is simply set to half of the mean in tenant VM requirement distribution. With different factor α, we measure the acceptance ratio of tenants' dynamic VM requests. The result shows that with bigger α, VR can accept more tenants' dynamic requirements without additional allocation. For different tenants, VR can provide diverse service level to satisfy their various requirements as shown in Fig. 7. We would advise that batch tasks like data-analysis be deployed in datacenters with less support of α since this kind of tasks is steady with few requirements for dynamic modification. While for applications with more variety like web applications, situation is different. Those applications consume VMs and bandwidth with big variety in terms of time, for example, a web server will be extremely busy when thousands and thousands of people access its service synchronously; It also can be extremely free at midnight when people are all asleep. In this case, factor α should be set to support it.

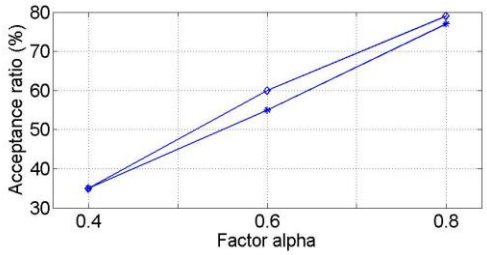

Figure 7: Factor α gives flexibility to datacenters

Analysis: False allocation. In previous part of this paper, we refer false allocation in Oktopus several times. Now we try to explain this problem more clear by a detailed presentation. To make it easily to understand, we setup a simple scenario, which we illustrate in Fig. 8.

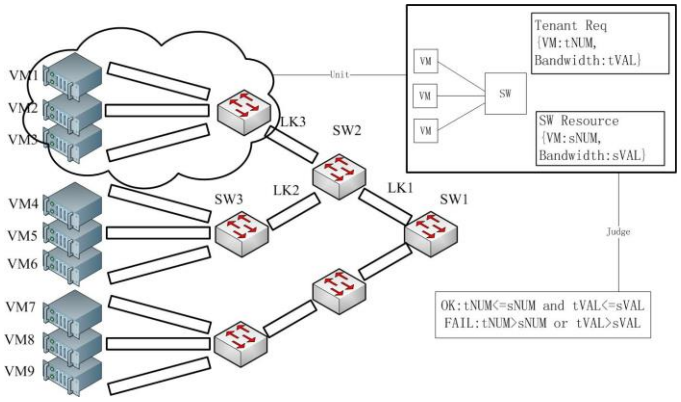

Figure 8: Scenario of false allocation (each link has bandwidth of 1Gbps.)

Oktopus allocates tenants' requests by VM amount and bandwidth. When a tenant request comes, it will search the network and find out the available resources to satisfy the tenant's request. The search unit is a switch and its available bandwidth in each link. It can search the network from lower level to higher level until finding the suitable unit which illustrates in Fig. 8.

When a tenant request comes with 4 VMs and 600Mbps bandwidth, Oktopus will find SW2 which is satisfied with tenant's need and allocate VM1, VM2, VM3, VM4 and bandwidth resources to tenant illustrated in Fig. 9. Due to Oktopus bandwidth allocation policy, LK2 need to provide 600Mbps bandwidth. While, if there comes another tenant's request which requests 4 VMs and 500Mbps bandwidth, Oktopus finds SW1 as an available resource unit and allocates VM5,VM6, VM7,VM8 and bandwidth resources to tenant (5 available VMs under SW1, each link with 1Gbps bandwidth available). But there is only 400Mbps (400Mbps=1Gbps-600Mbps) bandwidth left in LK2. This cannot satisfy the tenant request, and we call it false allocation. The cause of this problem is that Oktopus fails to validate the available bandwidth resources in lower level. In contrast, VR can avoid this problem by more check steps. Instead of false allocation, VR allocates VM6, VM7, VM8, VM9 and available bandwidth resources to tenant without hurting tenant's benefit.

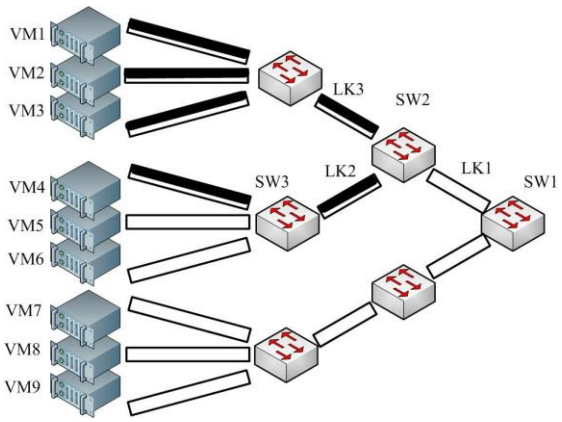

Figure 9: False allocation happens in LK2

V. Conclusion and Future Work

In this paper, we present VR, a new VM allocation algorithm to satisfy tenants' requirement in cloud datacenters. The core idea of VR is to give more consideration on network bandwidth constraints. In other words, we allocate a virtual network to tenants. By reserving bandwidth and backup VMs, VR can guarantee the performance of tenants' applications and keep a high allocation flexibility. At the same time, VR avoids some problems of SecondNet and Oktopus, and gives tenants a simpler user-interface and more accurate allocation plan. Our future work includes improving VR to adapt to more variable network topology, and designing new algorithms to deal with the chain reactions in VOC.

References

[1] Amazon EC2. http://aws.amazon.com/ec2/.

[2] Windows Azure.https://www.microsoft.com/windowsazure/.

[3] Michael Armburst et al. Above the Clouds: A Berkeley View of Cloud Computing. Technical report, University of California, Berkeley, 2009.

[4] Bodík, P., Menache, I., Chowdhury, M., Mani, P., Maltz, D. A., and Stoica, I. Surviving failures in bandwidth-constrained datacenters. In Proc. of the ACM SIGCOMM, 2012

[5] Niu, D., Xu, H., Li, B., and Zhao, S. Quality-assured cloud bandwidth auto-scaling for video-on-demand applications. In INFOCOM, 2012

[6] A.Shieh, S. Kandula, A. Greenberg, and C. Kim. Sharing the Datacenter Network. In Proc. Of USENIX NSDI, 2011.

[7] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards Predictable Datacenter Networks.In Proc. Of ACM SIGCOMM,2011.

[8] Wikipedia. Web development usage. http://en.wikipedia.org/wiki/Three-tier (computing).

[9] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In OSDI, 2004.

[10] http://rapgenius.com/James-somers-herokus-ugly-secret-lyrics

[11] Q. He, S. Zhou, B. Kobler, D. Duffy, and T. McGlynn.Case study for running HPC applications in public clouds.In Proc. of ACM Symposium on High Performance Distributed Computing, 2010.

[12] Amazon Cluster Compute, Jan. 2011. http://aws.amazon.com/ec2/hpc-applications/.

[13] Cisco. Data center Ethernet. http://www.cisco.com/go/dce.

[14] M. Al-Fares, A. Loukissas, and A. Vahdat. A Scalable, Commodity Data Center Network Architecture. In SIGCOMM, 2008.

[15] A. Greenberg et al. VL2: A Scalable and Flexible Data Center Network. In SIGCOMM, 2009.

[16] C. Guo et al. BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers. In SIGCOMM, 2009.

[17] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K.Ramakrishnan, and J. E. van der Merive. A flexible model for resource management in virtual private networks. In Proc. of ACM SIGCOMM, 1999.

[18] Bob Lantz, Brandon Heller, and Nick McKeown. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. 9th ACM Workshop on Hot Topics in Networks, October 20-21, 2010, Monterey, CA.

[19] N. G. Duffield, P. Goyal, A. G. Greenberg, P. P. Mishra, K. K. Ramakrishnan, J. E. van der Merwe, "A flexible model for resource management in virtual private network", ACM SIGCOMM 1999, August 1999.

[20] Alizadeh, M., Kabbani, A., Edsall, T., Prabhakar, B., Vahdat, A., and Yasuda, M. Less is more: Trading a little bandwidth for ultra-low latency in the data center. In Proc. of USENIX NSDI,2012