An efficient traffic-aware virtual machine placement algorithm (TVMP)

May 9, 2014
6.033 Design Project 2

Andrew Song (andrew90@mit.edu)
Daewook Kim (daewook@mit.edu)
Donggu Kang (donggu@mit.edu)
R01 Butler Lampson, Eirik Bakke

1. Introduction

As cloud computing is becoming more and more ubiquitous in enterprises and institutions, a lot of researches have been done on efficient uses of cloud computing resources in terms of financial cost and computation time. To increase the efficiency, virtual machines (VMs) that compute given tasks would have to be carefully placed, taking the several network conditions into account.

[1] suggested to optimize the placement of virtual machines (VM) within a data center rather than the topology of network architecture such as VL2 or B-Cube. [1] modeled the problem as a NP-hard quadratic assignment problem and solved the approximate version. However, it has two problems: First, the algorithm is traffic-aware but not available-bandwidth-aware; VMs with more traffic should be assigned to the faster section of the network. Second, the approximation ratio of the algorithm is as large as $O(\frac{k-1}{k}\ n)$ where n can be more than thousands and k is a small parameter for divide-and-conquer.

This paper proposes Traffic-aware Virtual Machine Placement (TVMP) system to solve a similar problem: given the amount of traffic to send between VMs, the final completion time should be minimized by the placement of VMs. Our solution uses the same approach as [1] but different in that 1) it is traffic-aware and also available-bandwidth-aware and 2) it improves the approximation ratio to $O(\sqrt{\lg n \lg k})$. The design is divided into two components; the measurement section explains how the current state of the network is captured by measuring the metrics needed for the placement component. The placement section explains how to use these metrics to find the best VM placement in $O(n^2)$ where n is the size of a data center.

2. Related Work

Significant part of our work is derived from [1]. [1] defined the following objective function:

$$\sum_{i,j} D_{i,j} C_{\pi(i)\pi(j)}$$

where $C_{\pi(i)\pi(j)}$ is a communication cost between the slot $\pi(i)$ and slot $\pi(j)$, $D_{i,j}$ is a traffic rate between the virtual machine i,j, and π is an 1:1 mapping of virtual machines to slots. Given the communication cost and the traffic rate, finding pi that minimizes the objective function is known to be NP-hard. [1] approximates the problem as the divide-and-conquer. First, all the slots are grouped into k slot-clusters with k-clustering algorithm. The size of slot-cluster may vary. Second, all the VMs are grouped into k VM-

cluster of the same sizes as k slot-clusters with the variant of balanced-minimum-k-cut algorithm presented in [1, 2]. The k VM-clusters and k slot-clusters are matched, and each pair of a VM-cluster and slot-cluster forms the sub-problem with the smaller number of VMs and slots, which would be recursively solved.

Although there is a large gap between the optimal solution and the divide-and-conquer solution, the empirical result by [1] shows the noticeable improvement over the random placement, and the benefit is larger as the traffic rate distribution is uneven.

3. Design Overview

1) Network Structure

The design considers the following data center architecture:

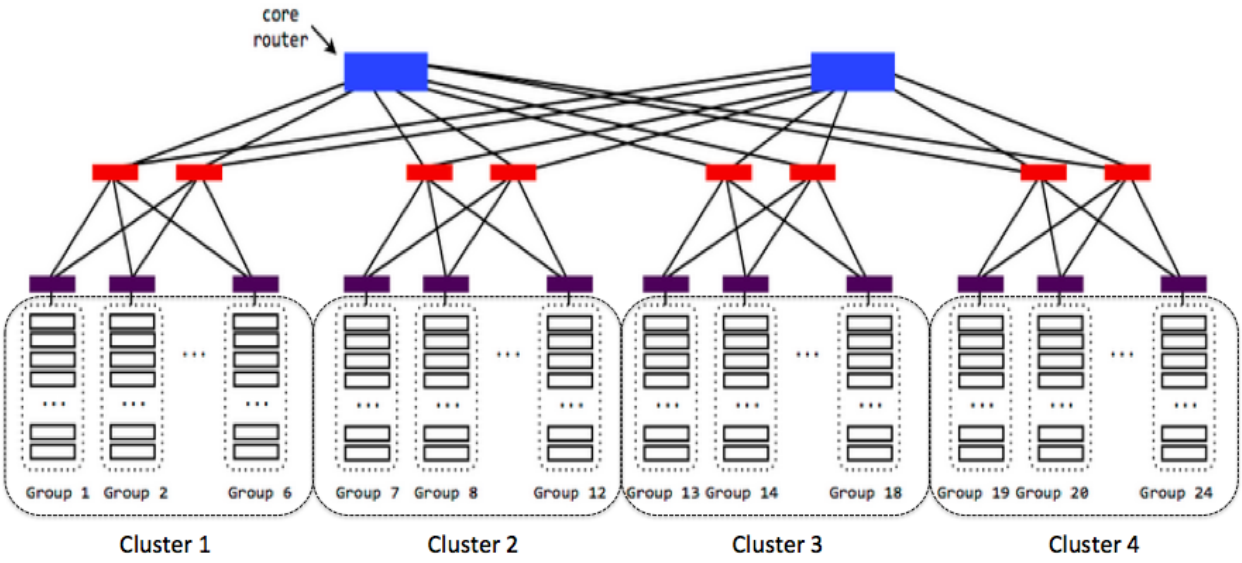

Fig 1. The data center network. A cluster consists of two routers and six groups.

It consists of the core level with two core routers, the aggregator level with four clusters, the group level with 24 groups, and the physical machine level with 1152 physical machines. Each physical machine can host up to four virtual machines. We call a place to host a virtual machine a *slot*. Note that the term cluster is distinguished from the term VM-cluster and slot-cluster.

We assume the traffic pattern of the network follows the one investigated in [1]. In more than 90% VMs, the traffic was within 1 standard deviation from the mean for more than 80% time. The fact that a large fraction of the VM's traffic rates are almost constant

implies it is a good idea to optimize the placement based on the past observation of traffic.

There are three types of VMs: Master VM, Application VM, and Probing VM.

Master VM: The central component in which the placement algorithm is run and necessary data are stored. Once deployed by the data center network, it stays in the same location until the entire application completes the task.

Application VM: The tasks of the application that the user runs are distributed among application VMs.

Probing VM: A lightweight VM intended solely for running measuring end-to-end bandwidth; probing VMs do not carry out any of the application's tasks.

2) System Design

The design consists of the measurement component and placement component. The measurement component investigates which slots are empty and what the available bandwidth are between groups and within groups. The placement component calculates the optimal placement of VMs from the metrics from the measurement component and places them accordingly.

In each "round" which lasts for T seconds (10 by default), the measurement and the placement is carried out once. At the end of each round, each VM will send messages to the master VM to report how much data is remaining. The periodic replacement enables the dynamic adjustment of VMs according to the non-static states of the network, thereby trying to minimize the overall completion time of the applications.

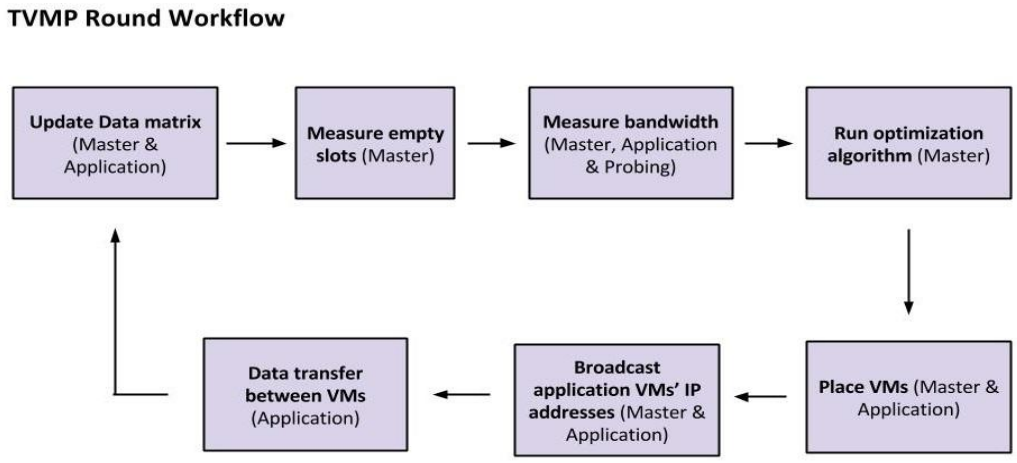

Fig 2. TVMP Workflow for a round

The algorithm used for the placement component is an improvement over [1]'s clustering algorithms and different in many aspects. Since we are aware of the network architecture, the k-clustering algorithm for slot-clustering is unnecessary. Slots under the same router are clustered together at each level. The VM-clustering algorithm using the gomory-cut is replaced with the altered version of an algorithm [3] for approximately-balanced-minimum-k-cut problem. Our system also utilizes the available bandwidth by reflecting the metric in the communication cost. This is contrary to the communication cost in [1], which is simply the number of routers between two physical machines.

4. Measurement Component

The key objective of the measurement component is to measure the "current state" of the network. The importance is even more emphasized by the fact that there are multiple users deploying their application simultaneously; the user has no knowledge of how other users have placed their VMs and how heavily congested each link is.

The component measures three metrics per each round: Remaining application data, empty slots in the network, and bandwidth between each group.

1) Measurement of remaining application data

The master VM calls *progress()* API for all the application VM pairs that still had remaining data to be transferred in the last round and updates the data matrix.

2) Measurement of empty slots in the network

The measurement procedure is very simple using the provided API, *machine_occupancy(m)*; the master VM simply calls *machine_occupancy(m)* for each physical machine *m* to get a list of empty slots.

3) Measuring end-to-end bandwidth

The Master VM measures two kinds of end-to-end bandwidth: The bandwidth within a group and bandwidth between different groups. The measurement procedure is almost identical for the two metrics; the only difference is the variables that these metrics get stored in.

The available bandwidth between any pair of virtual machines within the same group can be assumed identical, because the links between a machine and the corresponding group router has the extremely high bandwidth (100Gb/s) and therefore the bottleneck is likely to be the group router. With such assumption, the problem of measuring the bandwidth between the groups now becomes a simpler one in which the bandwidth between the VMs that act as a representative of each group.

Below are the steps for the measurement for bandwidth between groups. Different configurations for the measurement of bandwidth in a group, if there are any, are specified for each step.

a. Deploying probing VMs

In order to measure the bandwidth between groups, there has to be at least one VM per each group to be able to receive and send probing packets. In other words, the groups that do not have some of the user's VMs deployed will need probing VMs for the measurement.

For *k*-th group without user's application VM, *place* method will be called on the machine with the lowest possible index possible in the *k*-th group (i.e. 2nd group will start with machine 49) until successful placement. This operation terminates when there is at least one VM per group for all groups. For *k*-th group with user's application VMs, a VM will be randomly selected to represent the group.

(Bandwidth within group) There should be at least two VMs, one for source VM and one for destination VM.

b. Sending probing packets

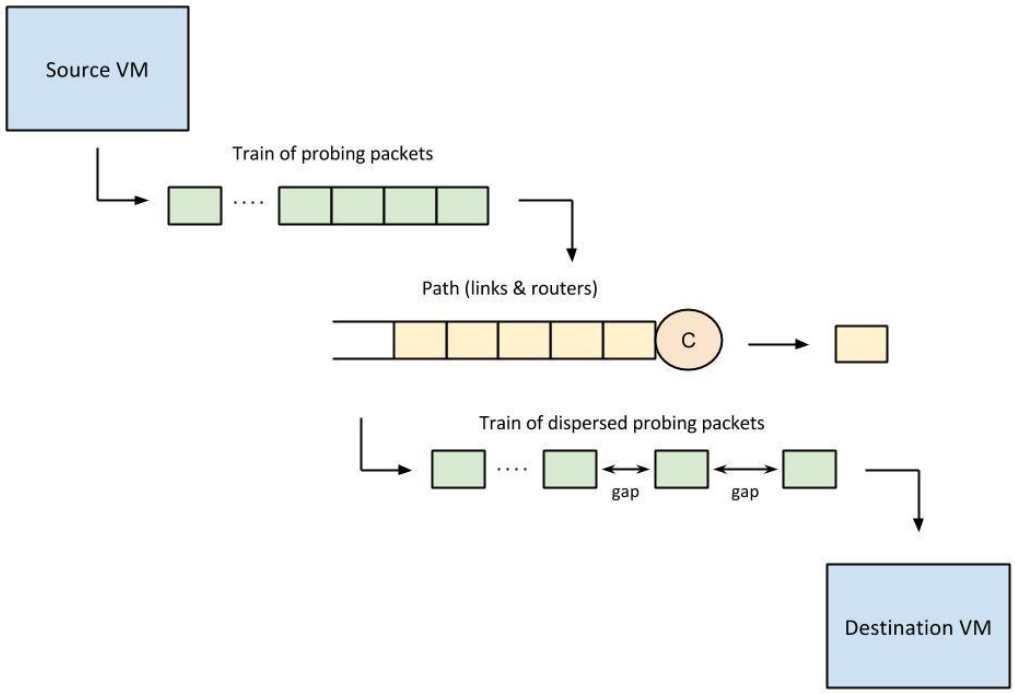

Fig 3. Probe Gap Model (PGM) used by Spruce

With the measurement environment set up, the system uses a lightweight bandwidth estimation tool called Spruce [4]. Spruce is based on the Probe Gap Model (PGM), the model that calculates the effective end-to-end bandwidth by exploiting dispersed gaps that appear between probing packets while traveling a path from one end to other end. In the default setting for Spruce, an end VM continuously sends probing packets at 240 Kb/s to the other end VM.

Spruce tool at the destination VM calculates the available bandwidth every second and updates the values locally every second.

c. Report to Master VM
Every T seconds, the Master VM requests each probing VM (or application VM) to report available bandwidth with each VM's neighbors. Upon receiving these messages, the Master VM de-serializes them and updates the throughput values accordingly.

d. Release Probing VMs
Probing VMs deployed for measurement purpose will now be released to prevent them from further consuming financial resources.

e. Manual measurement of available bandwidth vs tcp_throughput
An alternative method for measuring the group throughput is to use *tcp_throughput* API call provided by the data center. However, the API has a few drawbacks: it might not give us available bandwidth if the transferred data within in 100ms does not fully utilize the available bandwidth. Moreover, the time and traffic cost of the API call are not specified.

5. Placement Component

The placement component finds an efficient placement of virtual machines. The data given by the measurement component is the list of empty slots, the throughput matrix T that represents through between each pair of groups, and the remaining traffic matrix D. The basic idea is to use a divide and conquer algorithm to break the problem into smaller ones and solving them.

Overall, There are two stages of divide and conquer algorithm:

First stage: An approximation algorithm is used to partition a set V of n VMs into four subsets $V_1, \ldots, V_4$ so that VMs in V_i are assigned to the empty slots under cluster i.

Second stage: For each V_i partitioned in the first stage, we use the same approximate algorithm again to partition it into six subsets, such that each VM subset is assigned to one of the six groups in cluster i.

After VMs are completely partitioned into $4 \times 6 = 24$ sets, we randomly deploy VMs in each set on their corresponding groups. The approximate algorithm and the divide and conquer algorithm is described in more detail below.

1) Balanced Minimum K-cut Problem

Having an undirected weighted graph G as an input, the minimum k-cut problem is to partition G into k components so that the total weight of edges connecting different components is minimized. The balanced minimum k-cut problem (BMKP) is to partition G into the equally sized k components. BMKP is NP-hard, so the approximate version (k,v) that partitions G into k components such that no component has the size greater than $(1+\epsilon) \cdot \left\lceil \frac{n}{k} \right\rceil$ has been researched for various k and v.

For v=1+ϵ, [3] (KNS algorithm) gave an approximation algorithm that has the approximation ratio $O(\sqrt{\log n \cdot \log k})$and runs in$O(nk/\epsilon)$ time. As we pick a smaller value for ϵ, we get a tighter upper bound on the size of partitioned components, but also get a longer running time. We use $\epsilon = 0.1$ in our design to attain both a reasonable running time and a reasonable upper bound.

2) Representing VMs and traffic between them as a graph

Regard all VMs as vertices and traffic between two VMs as the weight of the edge between them. Then, we can represent VMs a graph

3) Divide and conquer algorithm

Denote the traffic matrix B, the numbers of empty slots in each group $n_1, \ldots, n_{24}$, and the group throughput matrix T. We want to deploy n application VMs on physical machines of 24 different groups. The divide and conquer algorithm used in TVMP goes through two stages.

a. Preparation stage

In the preparation stage, the number of empty slots and the number of VMs are made equal. This is done by the following step. If the number of empty slots is larger than the number of VMs, *filling-VMs* with no traffic is created to make two numbers equal. Otherwise, if the number of empty slots is smaller than the number of VMs, VMs that have small traffic are abandoned so that the numbers are equal. Call this number of VMs, which is also the number of empty slots, *n*.

b. The first stage

In the first stage, n application VMs are partitioned into four sets each of which corresponds to one of the four clusters. The basic idea is to add balancing VMs and

represent all VMs with a graph representation discussed above to change the problem into the BMKP. Then, the KNS algorithm is used to solve the BMKP problem. These steps are explained below by an example.

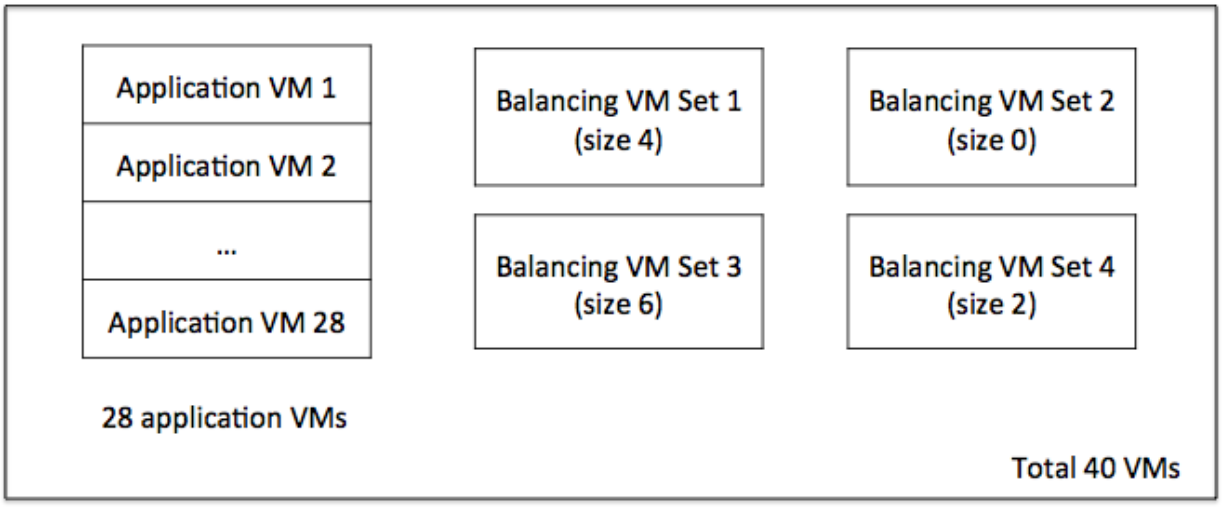

Fig 4. The composition of 40 VMs

Suppose that we want to place 28 application VMs on four clusters each having 6, 10, 4, and 8 empty slots. We first create **balancing-VM sets** of sizes 4, 0, 6, and 2 so that the sum of the number of empty slots and balancing-VMs are equal to the maximum number of empty slots, which is 10 (6+4=10, 10+0=10, 4+6=10, 8+2-10). The resulting 40 VMs contain balancing and application VMs as in Fig 4.

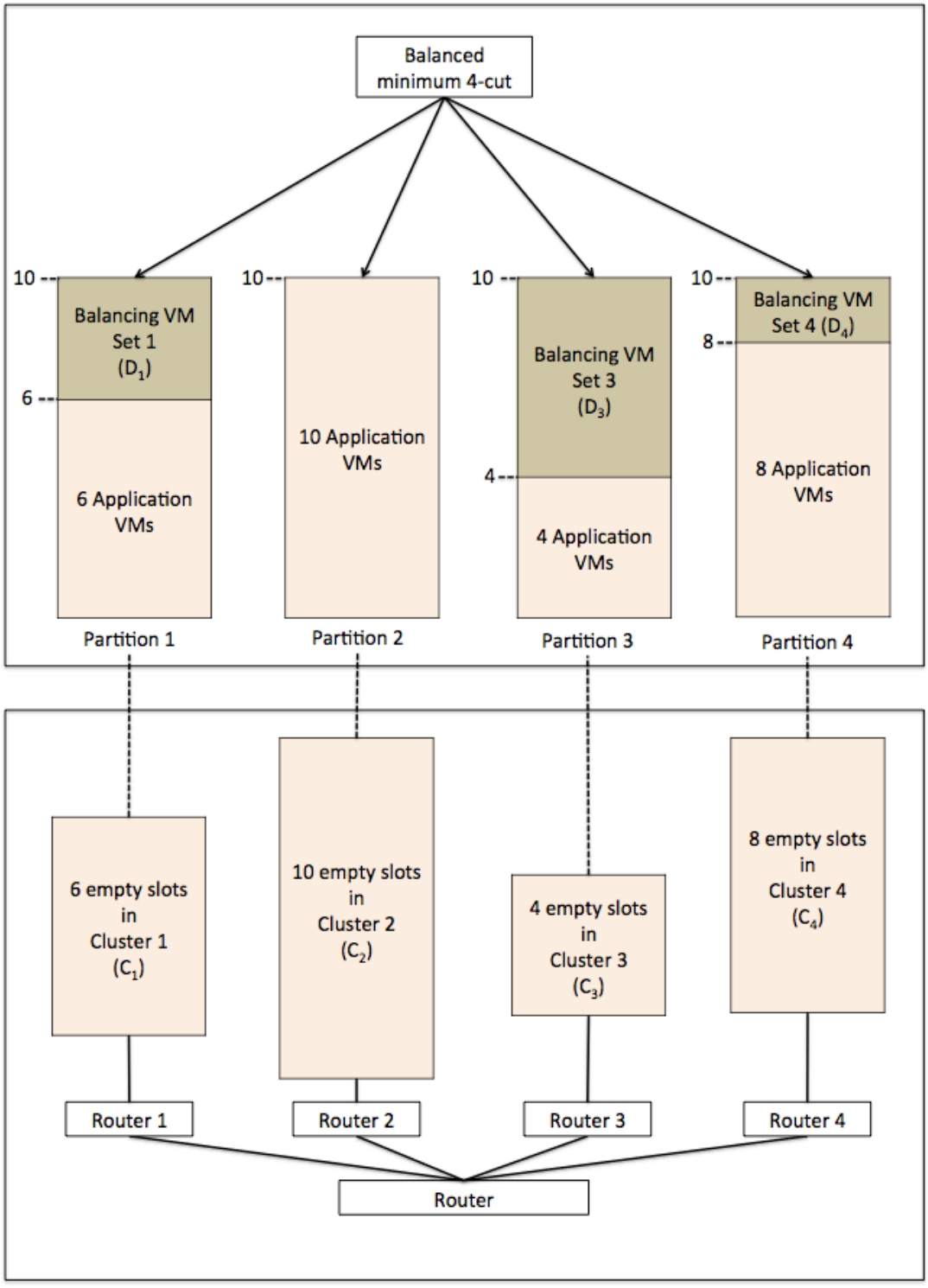

Fig 5. The BMKP and assignments of partition to empty slots

Now, we set traffic between balancing VMs so that 1) all balancing VMs in the same balancing VM set end up in the same partition, and 2) all balancing VMs in different balancing VM set end up in different partitions. This can be done by setting a large

enough positive number to traffic between balancing VMs in the same balancing VM set, and setting a large enough negative number to traffic between balancing VMs in different balancing VM sets. Now, if we create a graph G with the union of application and balancing VMs and solve the balanced 4-cut problem on G, the partitions will look like partitions in the upper part of Fig. 5.

c. Considering the available bandwidth

The available bandwidth between clusters can be considered by adding traffic between balancing VMs and application VMs. It is generally better to place heavy traffic VMs in the physical machines that have high throughput. Hence, we assign higher traffic between heavy traffic application VMs and high throughput balancing VMs, and lower traffic between light traffic application VMs and low throughput balancing VMs so that the heavy traffic VMs likely to belong to high throughput clusters.

First, for each VM, the total data *d* to transfer and receive is computed. Then, for each cluster, an average network throughput *t* is computed using the group throughput matrix T. Then for each cluster, pick one of balancing VMs in its corresponding balancing VM set (If there is a cluster of which the corresponding balancing set is empty, we can simply add one balancing VMs to all clusters), and assign traffic between it and all of the application VMs. This amount of traffic is determined by an increasing function of the remaining traffic of the application VM and the average throughput of the cluster. An example of this function is $f(t,d) = td^2$ which makes it more likely for the heavier traffic VMs to belong high throughput clusters. However, it is hard to choose this function without experiments, and should be adjusted after the actual implementation.

d. Using the KNS algorithm [3]

After setting traffic values as described above, if we immediately run the KNS algorithm, the size of each partitioned set can be at most $(1+\epsilon)\cdot\frac{\mathrm{n}}{\mathrm{k}} = 1.1\cdot 10 = 11$. It is true that we might end up with a partitioned set having more VMs than the number of empty slots (10 in this example) in its corresponding cluster. To fix this issue, $[\frac{\epsilon}{1+\epsilon}\mathrm{n}\,] = 3$ relatively unimportant application VMs are temporally removed. These relatively unimportant application VMs are chosen by selecting ones with smallest traffic. Thus, 28-3=25 application VMs and 12 balancing VMs are left now, and there are total 37 VMs.

Now, these VMs are partitioned by [3] into four sets and, the size of each partitioned set can be at most $\left\lceil(1+\epsilon)\cdot\frac{\mathrm{n}}{\mathrm{k}}\right\rceil = \left\lceil(1+0.1)\cdot\frac{37}{4}\right\rceil = [\,10.175] = 10$. Then, the number of application VMs, which are grouped with each balancing VM set is going to be at most 10-4=6, 10-0=10, 10-6=4, and 10-2=8, and they can all fit into their corresponding clusters. Now, previously removed unimportant application VMs are placed on arbitrary

empty slots. Since the portion of removed relatively unimportant application VMs are at most $\frac{\epsilon}{1+\epsilon} = 0.09$ (9%), it does not affect the overall performance much.

e. The second stage

After the first stage, VMs are partitioned into four components $V_1, \ldots, V_4$. In the second stage, TVMP repeats exactly the same algorithm used in the first stage to partition each set V_i to six components that is going to be deployed on the six groups of C_i. The only difference from the first stage is that in the second stage each VM component is partitioned into six components, and that the group throughput matrix can be used directly. After this partition is done, the placement component simply deploys application VMs to any empty slots in its corresponding group. There is an error that might happen during this phase. A slot that was available in the measurement component may become unavailable after the computation in the placement component. We first try to find alternate empty slots in the same group, and if we fail, we place it in any empty slot.

f. After VM placement

After the new placement, the master VM broadcasts the updated list of VM locations to other VMs. The application VM is blocked from transferring data to its neighbor VMs until it has received the most recent list of VMs. Otherwise, the data might end up in old IP addresses and get lost.

6. Analysis

1) Choosing the appropriate Round time T

As the round time T gets smaller, the system can adjust to the network status more dynamically. However, if the value of T is reduced indefinitely, a new problem will arise as application data from VMs might not reach their intended destinations due to queuing & processing delay before VMs get replaced; the data will get lost. Thus, T cannot be too small, as it will move VMs too frequently. As can be seen from the running time $O(nk)$ where n = 4608 and k = 24 in our case, we can expect the placement algorithm to run in less than a second. Thus, we decided to use 10 seconds for the round time T, which is not too frequent, but frequent enough to adjust to the network status.

2) Bandwidth consumption due to measurement

In this section, we analyze how much congestion the probing packets generated by Spruce tool incurs on the network links.

In the default implementation of Spruce, probing packets from each VM are sent at the rate of 240Kb/s. We take a look at the cost of measurement in two different links: a link between core and aggregator router and a link between aggregator and group router.

a. Core - Aggregator router link

The core-aggregator router link serves the traffic between clusters of 6 groups. For simplicity, let's measure the bandwidth of links going in and out of cluster 1. Each group in cluster 1 will send the probe packets to 18 groups in other clusters, which $\text{is } 18 \times 240\text{Kb/s} = 4320\text{Kb/s} \simeq 4.5\text{Mb/s}$. Since a cluster has 6 groups, the total traffic for outgoing probing packets from the cluster is $4.5\text{Mb/s} * 6 \simeq 30\text{Mb/s}$. Since there are two core routers and two aggregator routers, there are 4 links that can equally serve the traffic. Thus, the bandwidth used for each link is 7.5Mb/s. Symmetrically, we can also reach the same 7.5Mb/s for incoming probing packet traffic to cluster 1 sent by other clusters. Combining incoming and outgoing measurement traffic, we reach the conclusion that about the measurement cost is 15Mb/s for each link of 10Gb/s capacity or about 0.15%of the maximum bandwidth.

b. Aggregator - Group router link

There are two kinds of probing packet traffic: One for measurement within a cluster and one for measurement between clusters. The latter one is simply $15\text{Mb/s} \div 6 = 2.5\text{Mb/s}$ because a group router is connected to 6 group routers.

For the earlier metric, $5 \times 240\text{Kb/s} = 1.2\text{Mb/s}$is generated by a group router as there are 5 other groups within a cluster. Considering the incoming and outgoing traffic, the number becomes 2.4Mb/s, but considering the fact that there are 2 aggregator routers, the traffic would still be 1.2Mb/s.

Summing up the two values, we get 3.7Mb/s, which is about 0.04% of the link bandwidth.

Combining these two results we conclude that Spruce is feasible as the bandwidth consumption is really low.

3) Use cases

Among many use cases, we take a look at three most probable cases and analyze how our system will perform in these cases.

a. 100VMs, the same end-to-end throughput, traffic

Even when the end-to-end throughput is all the same, it still is a good idea to place the virtual machines close to each other. Distributing the virtual machines requires extra routers and shared links.

If each physical machine has already three virtual machines, there are 1152 available slots in total. At the aggregator level, each cluster has 288 (=1152/4) slots. Assigning all 100 virtual machines into one cluster gives us the zero inter-cluster traffic. Any cluster-level router will be selected and assigned 100 virtual machines. Now there are 6 groups under the cluster, each with 48 slots. The best way to minimize the inter-group traffic is to assign 100 virtual machines into three groups with 48, 48, 4 slots.

We can see that our design usually results in the most compact assignment.

b. 10 VMs placed in congested Group 1~6, and Group 19~24 are newly available

The master VM checks the available slots every T seconds, so it may take up T seconds to notice the new slots in the group 19-24. When it detects the new slots, each group is assigned the probing VM to investigate the network status. Then the placement algorithm is run to find the new solution, and the application VMs are placed accordingly.

c. Every user uses this design

As seen in the case of 100VMs, the algorithm usually tends to place the VMs for one application into the smallest region as possible. This means that even if all the users use the same strategy, the localized assignments will minimize the interference between users.

The fairness (per VM) may be a problem because the user who occupied the network first can choose the best placement for him but users who come in later have less options in terms of placement. The only unfair situation is when the application 1 shares the link with heavy-traffic applications and the application 2 shares the link with light-traffic applications. However, it is application 2's overutilization and the application 1 is still guaranteed to have the bandwidth as much as the total bandwidth divided by the number of maximum virtual machines.

An inefficient stalemate state is [6] because of the frequent assignment and release can be relieved by allowing the trade of VM placements between users. The trade is beneficial for both because it increases the localization of both users' VMs.

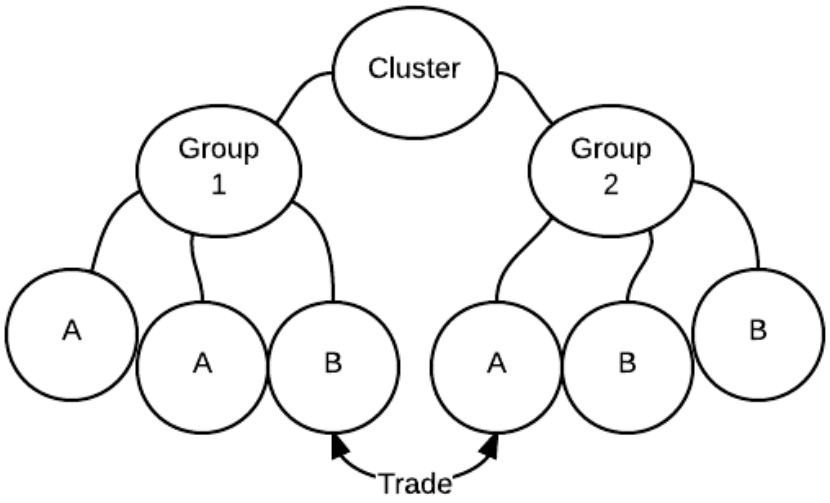

Fig 6. The stalemate state

If the design is deployed by a data center rather than each user, it is possible to achieve a better result by matching all the VMs used by each user altogether with all the available slots at once. For a case of 17,000 VMs in the data warehouse hosted by IBM Global Services in [1], it only takes a few seconds to solve the placement problem by our $O(n^2)$ algorithm.

7. Conclusion

[1] suggested using the graph partition algorithm with $O\left(\frac{k-1}{k}\ n\right)$ -approximation ratio to solve the TVMP problem. Our design improved this to $O\left(\sqrt{\log n \log k}\right)$-approximation with a novel dummy VM technique and at expense of the most insignificant 10% of virtual machines. The space complexity of the placement algorithm is $O(n^2)$ and the time complexity is $O(nk/\epsilon)$ where n is the number of slots and up to 4608, k is 4 and ϵ is 0.1 in our network architecture. Considering that the size of IBM enterprise data center is 17000 VMs and that the frequent VM replacement is not necessary, the placement algorithm is very fast (a few seconds) and can be ignored.

The design satisfies all the requirements described in the design project 2. However, the design is based on some unrealistic assumptions, such as no cost in VM replacement. Further work on how to relax such assumptions is desired. Moreover, a network simulation or an experiment in real world data centers should be performed to support the result with the empirical evidence.

8. References

[1] X. M. X. Meng, V. Pappas, and L. Z. L. Zhang, “Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement,” *INFOCOM, 2010 Proc. IEEE*, 2010.

[2] H. Saran and V. V. Vazirani, “Finding k-cuts within twice the optimal,” *[1991] Proc. 32nd Annu. Symp. Found. Comput. Sci.*, pp. 743–751, 1991.

[3] R. Krauthgamer, J. S. Naor, and R. Schwartz, “Partitioning Graphs into Balanced Components,” in *SODA*, 2009, pp. 942–949.

[4] J. Strauss, D. Katabi, and F. Kaashoek, “A measurement study of available bandwidth estimation tools,” in *Proceedings of the conference on Internet measurement conference - IMC ’03*, 2003, pp. 39 – 44.

[5] A. E. Feldmann and L. Foschini, “Balanced Partitions of Trees and Applications,” in *STACS*, 2012, pp. 100–111.

[6] N. Guttmann-Beck and R. Hassin, “Approximation Algorithms for Minimum K -Cut,” *Algorithmica*, vol. 27, no. 2, pp. 198–207, 2000.

[7] A. Amir, J. Ficler, R. Krauthgamer, L. Roditty, and O. S. Shalom, “Multiply Balanced k-Partitioning.” [Online]. Available: http://www.wisdom.weizmann.ac.il/~robi/papers/AFKRS-MultiplyBalanced-LATIN14.pdf. [Accessed: 04-May-2014].

[8] T. Benson, A. Anand, A. Akella, and M. Zhang, “Understanding data center traffic characteristics,” *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1. p. 92, 2010.

[9] S. Chaisiri, B.-S. L. B.-S. Lee, and D. Niyato, “Optimal virtual machine placement across multiple cloud providers,” *2009 IEEE Asia-Pacific Serv. Comput. Conf.*, 2009.

[10] D. S. Dias and L. H. M. K. Costa, “Online traffic-aware virtual machine placement in data center networks,” in *2012 Global Information Infrastructure and Networking Symposium (GIIS)*, 2012, pp. 1–8.

[11] N. Hu and P. Steenkiste, “Estimating Available Bandwidth Using Packet Pair Probing,” 2002.

[12] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, “The Nature of Datacenter Traffic : Measurements & Analysis,” *Provider*, no. Microsoft, pp. 202–208, 2009.

[13] K. Andreev and H. Räcke, “Balanced graph partitioning,” *ACM Symp. Parallel Algorithms Archit.*, p. 120, 2004.

[14] X. Wen, K. Chen, Y. Chen, Y. Liu, Y. Xia, and C. Hu, “VirtualKnotter: Online Virtual Machine Shuffling for Congestion Resolving in Virtualized Datacenter,” in *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, 2012, pp. 12–21.

[15] R. S. Prasad, C. Dovrolis, and B. A. Mah, “The effect of layer-2 store-and-forward devices on per-hop capacity estimation,” *IEEE INFOCOM 2003. Twenty-second Annu. Jt. Conf. IEEE Comput. Commun. Soc. (IEEE Cat. No.03CH37428)*, vol. 3, 2003.

[16] R. S. Prasad, M. Murray, C. Dovrolis, K. Claffy, R. Prasad, and C. D. Georgia, “Bandwidth Estimation: Metrics, Measurement Techniques, and Tools,” *IEEE Netw.*, vol. 17, pp. 27–35, 2003.
[17] N. Hu and P. Steenkiste, “Estimating Available Bandwidth Using Packet Pair Probing,” 2002.
[18] R. S. Prasad, M. Murray, C. Dovrolis, K. Claffy, R. Prasad, and C. D. Georgia, “Bandwidth Estimation: Metrics, Measurement Techniques, and Tools,” *IEEE Netw.*, vol. 17, pp. 27–35, 2003.

Word count: 4699