Towards Makespan Minimization Task Allocation in Data Centers

Kangkang Li, Ziqi Wan, Jie Wu, and Adam Blaisse

Department of Computer and Information Sciences
Temple University
Philadelphia, Pennsylvania, 19122
Email: {kang.kang.li, ziqi.wan, jiewu, adam.blaisse}@temple.edu

Abstract—**Nowadays, data centers suffer from resource limitations in both the limited bandwidth resources on the links and the computing capability on the servers, which triggers a variety of resource management problems. In this paper, we discuss one classic resource allocation problem: task allocation in data centers. That is, given a set of tasks with different makespans, how to schedule these tasks into the data center to minimize the average makespan. Due to the tradeoff between locality and load balancing, along with the multi-layer topology of data centers, it is extremely time consuming to obtain an optimal result. To deal with the multi-layer topology, we first study a simple case of one-layer cluster and discuss the optimal solution. After that, we propose our hierarchical task allocation algorithm for multi-layer clusters. Evaluation results prove the high efficiency of our algorithm.**

Keywords–Task allocation; Data centers; Makespan.

I. Introduction

Modern data centers comprise tens of thousands of computers interconnected between each other with commodity switches. Nowadays, data centers suffer from resource limitations in both the limited bandwidth resources on the links and the computing capability on the servers, which triggers a variety of resource management problems. One classic issue is the task allocation problem, which involve various constraints, including performance, network, and cost.

Generally speaking, a task usually consists of two parts: the computation workloads and the communication traffic. Different tasks running in the data center will compete for computing capability on the servers and bandwidth resources on the links. Obviously, to minimize the duration of communication traffic, locality is one important factor that we need to take into consideration. That is, we need to place the tasks as close to each other as possible, in order to lower the hops of communication between tasks. Furthermore, the tasks running in the same server will not need communication traffic between each other due to the internal communication within the server.

However, the bandwidth capacity of each link is limited. More tasks running under the same link will lead to the lower average bandwidth allocated to each task, which will decrease the communication speed and increase the communication duration. What's worse, the computing capability of a server is limited. If tasks are packed together, it will also reduce the computing capability allocated to each task. With the degraded computation speed, the computation workloads will take more time to complete.

On the other hand, if we apply load-balancing and allocate the tasks evenly to the servers, the computing speed of each task can be maximized. However, the geographically separated tasks need more hops to communicate with each other. With the limited bandwidth resources on the links, load-balancing will considerably lengthen the communication time. Therefore, there is a tradeoff between the locality and load balancing.

The remainder of the paper is organized as follows: In Section II, we introduce some related work. In Section III, we present the task model discussed in this paper. In Section IV, we formulate the task allocation problem in the data center. In Section V, we study the simple case of a one-layer cluster and discuss the optimal solution. Section VI focuses on the multi-layer cluster and gives the hierarchical task allocation algorithm. Section VII conducts the simulations to validate the efficiency of our algorithm. Finally, conclusions are in Section VIII.

II. Related Work

Task allocation has been an open research topic since the traditional distributed computing era. Many task allocation issues are NP-hard, thus, we need to find a good heuristic algorithm to solve the problem, such as the first-fit and best-fit greedy algorithm used in [1].

Take the Mapreduce jobs as an example. One Mapreduce job consists of three tasks: map, shuffle and reduce. Many MapReduce schedulers have been proposed to try maximizing the resource utilization in the shared MapReduce clusters. In [2], the author introduced delay scheduling that speculatively postpones the scheduling of the head-of-line tasks and ameliorate the locality degradation in the default Hadoop Fair scheduler. In addition, in [3], the author also proposed Longest Approximate Time to End (LATE) scheduling policy to mitigate the deficiency of Hadoop scheduler in coping with the heterogeneity across virtual machines in a cloud environment. The author of [4] proposed a communication-aware placement and scheduling of MapTasks and predictive load-balancing for ReduceTasks to reduce the network traffic of Hadoop on heterogeneous clusters.

VM placement is similar to the task allocation problem under the environment of cloud computing. As data centers are becoming the mainframe of cloud services, the VM placement problem in data centers has been an open research area, considering both the network and servers. In [5], the author gave a heuristic algorithm to satisfy both the communication demands and physical resource restrictions. In [6], the author proposes minimizing the traffic cost through VM placement. Their objective is to place VMs that have large communication requirements close to each other, so as to reduce network capacity needs in the data center.

Oktopus [7] uses the hose model to abstract the tenant's bandwidth request, including both virtual cluster and oversubscribed virtual clusters. They propose a VM placement algorithm to deal with homogeneous bandwidth demands. The virtual cluster provides tenants with guarantees on the network bandwidth they demand, which, according to [8], can be interpreted as the min-guarantee requirements. However, this min-guarantee fails to consider the potential growth of a tenant's network demand. In order to alleviate this problem, in [9], we propose the concept of elasticity, favoring the on-demand scaling feature of cloud computing.

Our previous work in [9] also designed a recursive abstraction scheme and hierarchical VM placement algorithm, which is similar to the task allocation scheme in this paper. However, this paper focus on the objective of task makespan minimization and does not consider the environment of cloud computing. Furthermore, we focus on the study of tradeoff between locality and load balancing when doing task allocation. The communication model between tasks is different from the hose model for VMs' communications used in [9].

III. Task Model

In our model, each task can be separated into three parts: the pre-computation part, the computation part and the post-computation part, as shown in Figure 1. Here, we study the homogeneous task inputs. That is, all the tasks share the same pre-computation, communication, and post-computation workloads. To normalize these different types of workloads, we consider the situation that there are two tasks running in the data center and they are allocated to two servers under the same switch. Then we define the normalized time for each step as the time those two task go through each step, which is noted as α, 2β, and γ time units, respectively. Here we use 2β for two tasks communicating at the same time. If there is only one task fully using the bandwidth resource during the communication period, then its communication time is β.

Obviously, when $\alpha + \gamma \gg \beta$, the communication time can be neglected. Then the best choice is load-balancing and to evenly divide the input tasks into the servers. On the other hand, when $\alpha + \gamma \ll \beta$, the pre-computation and post-computation time can be neglected. Therefore, the best allocation scheme is to try to put all tasks into one server to minimize the communication cost.

We assume that all the tasks will start their pre-computation part at the same time. After finishing the pre-computation part, a $task_i$ will try to communicate with another $task_j$. However, due to the different completion times of the pre-computation parts of different tasks, $task_j$ might not finish its pre-computation part. Then, $task_i$ must wait for the $task_j$ to complete its pre-computation, and then, they can carry on the communication part with each other. After the communication part is finished, $task_i$ will start to complete its post-computation. Thus, the total makespan of a single $task_i$ is the sum of pre-computation time, waiting time, communication time, and post-computation time.

Pre-Compute
Communication
Post-Compute

Figure 1: Task model

IV. Problem Formulation

In this section, we formulate the average makespan minimization task allocation problem in data centers with the topology of a multi-layer binary tree. The data center configuration is *semi-homogeneous*, as shown in Figure 2. Each server has the same computing capability of C. Also, each link of the same layer has the same bandwidth capacity: L_k (the k^{th} layer link bandwidth capacity). However, the upper layer links have twice the bandwidth capacities than the lower layer links, i.e., $L_1 = 2L_2 = 4L_3$. The links capacities only differ between layers, we refer to this as the *semi-homogeneous* configuration, which is widely used to ease upper-layer link congestion. Our objective is to minimize the average makespan of all the input tasks, which can be expressed below:

$$makespan(i) = pre(i) + wait(i) + commun(i) + post(i) \quad (1)$$

$$\overline{makespan} = \frac{\sum_i^N makespan(i)}{N} \quad (2)$$

In Equation(2), $makespan(i)$ is the makespan of $task_i$, $pre(i)$ is the pre-computation time of $task_i$, $wait(i)$ is the waiting time between pre-computation and communication, $commun(i)$ is the communication time of $task_i$, $post(i)$ is the post-computation time of $task_i$. N is the total number of the tasks running in the system. Therefore, we tried to design a good task allocation scheme to minimize the average makespan of the input tasks.

Considering that the server's computing capability is steady and limited, then the speed of computation is inversely proportional to the number of tasks computed at this time on this server.

Considering that the bandwidth is equally shared by different tasks communicating through the link, then the communication speed is also inversely proportional to the number of tasks communicating between the servers.

In this paper, we study a simple case, in which the bandwidth resources are equally allocated to each task. In that case, given the the link bandwidth capacity B, the bandwidth allocated to each task is $\frac{B}{x}$, where x is the number of tasks go through that link.

Take one server to analyze. Assume there are x tasks in the server. Take $f_1(i, x)$ to be the time it takes $task_i$ to finish its first step in the server, where x is the number of tasks running on this server. Then we have:

$$f_1(1, x) = f_1(2, x) = \cdots = f_1(x, x) = x\alpha \quad (3)$$

Equation (3) indicates that first step finish time of the first task finished in the first step is proportional to the the number of tasks running on the server. This is due to the tasks on the same server evenly sharing the computation resources. It also

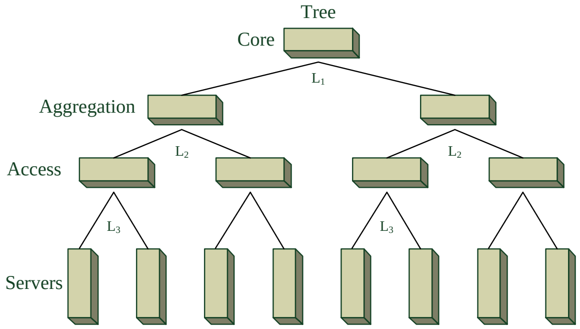


Figure 2: Tree-based network topology

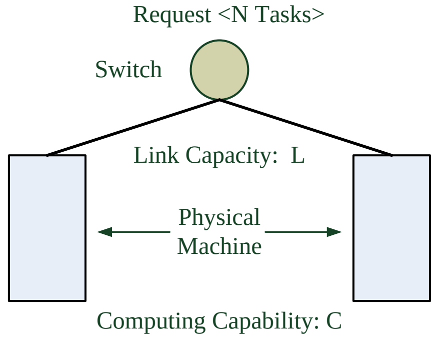


Figure 3: One-layer cluster

shows the finish time of the first step is equal for every task when the pre-computation workload is identical.

Define $f_2(t,x)$ as the number of tasks that have finished step 1 in the server at the time point of t. The we can get

$$f_2(t,x) = \begin{cases} 0 & \text{if } t < \alpha x \\ x & \text{if } t \geq \alpha x \end{cases} \tag{4}$$

t is the time from the the start of the pre-communication.

V. A ONE-LAYER CLUSTER STUDY

With N tasks accepted into the cluster, consider the simple case of two servers with a switch above them, as shown in Figure 3. Since all the tasks are homogeneous, we assume that, without loss of generality, we place x tasks in the left server, and leave $N-x$ tasks to the right server, and $x \geq N-x$. Due to the symmetry of binary-trees, there are, at most, $\lceil \frac{N}{2} \rceil$ different ways to allocate the tasks into two servers as the value of x increase from $\lceil \frac{N}{2} \rceil$ to N.

Let $p(t,x)$ be the probability of a task available for a communication step at time point t, with x tasks in the server.

$$p(t,x) = \frac{f_2(t,x)}{x} \tag{5}$$

We assume a task will randomly choose another task to communicate with. Then the probability of $task_i$ and $task_j$, both being available to communicate with each other at time point t is $p_i(t) \times p_j(t)$. In the one-layer binary tree model, there are only 2 possible relationships of $task_i$ and $task_j$, either in the same server, or in the different servers. Let $f_3(t,x)$ be the number of tasks able to start communication at time t in this one-layer cluster, then we can figure out:

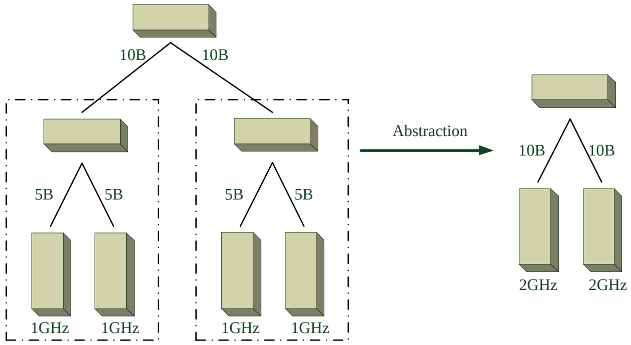


Figure 4: Abstraction process

$$f_3(t,x) = x \times p(t,x) + (N-x) \times p(t, N-x) \tag{6}$$

The internal traffic of tasks running on the same server will introduce no communication cost. If two tasks communicate in the same server, then their communication time will be ignored. Therefore, our focus is located on the traffic between tasks on different servers. Since the allocated bandwidth of tasks on the two servers might be different, we adopt the minimal one as the bottleneck of communication. However, if there are only two servers in the cluster, all the traffic outside the servers go through both sides of the switch. Then the outbound link bandwidth will be equally shared for both links, when they have the same communication capacities.

Let $f_4(t,x)$ be the number of tasks the have finished the communication part at time t, with x tasks in the left server, and $N-x$ tasks in the right server. The average number of tasks communicating between the two servers at time t is $f_3(t,x) - f_4(t,x)$. For a $task_i$ communicating with a task on the other server, let $t_{sc}(i)$ be the communication starting time of $task_i$, $t_{fc}(i)$ be the communication finishing time of $task_i$. Then we can get the constrains as follow

$$f_3(t_{sc}(i), x) = i \tag{7}$$

$$f_4(t_{fc}(i), x) = i \tag{8}$$

$$\int_{t_{sc}(i)}^{t_{fc}(i)} \frac{1}{f_3(t,x) - f_4(t,x)} dt = \int_0^{\beta} \frac{1}{1} dt = \beta \tag{9}$$

Let $f_5(t,x)$ be the number of tasks that have finished all the three parts (pre-computation, communication and post-computation) at time t. We further assume that $f_{5L}(t,x)$ is the number of tasks that have finished on the left server at time t, and let $f_{5R}(t,x)$ be the number of tasks that have finished on the right server at time t. Similarly, we can split $f_4(t,x)$ into $f_{4L}(t,x)$ and $f_{4R}(t,x)$. Likewise, let $t_{spc}(i)$ be the post-computation starting time of $task_i$, and let $t_{fpc}(i)$ be the post-computation time of $task_i$, and $task_j$ can be either in the left or right server. Then we can get

$$f_{4L}(t,x) = \frac{f_4(t,x) \times x}{N} \tag{10}$$

$$f_{4R}(t,x) = \frac{f_4(t,x) \times (N-x)}{N} \tag{11}$$

$$\int_{t_{spc}(i)}^{t_{fpc}(i)} \frac{1}{f_{4L}(t,x) - f_{5L}(t,x)} dt = \int_0^{\gamma} \frac{1}{1} dt = \gamma \tag{12}$$

or

$$\int_{t_{spc}(i)}^{t_{fpc}(i)} \frac{1}{f_{4R}(t,x) - f_{5R}(t,x)} dt = \int_0^{\gamma} \frac{1}{1} dt = \gamma \tag{13}$$

$$f_{5L}(t,x) = f_{5L}(t,x) + f_{5R}(t,x) \tag{14}$$

Apparently, by solving $f_5(t,x) = i$, $i = 1, 2, ..., N$, we can get the finish time of each task, say $t_1(x), t_2(x), ..., t_N(x)$. Then the object is to minimize the average makespan of tasks, which is $\frac{t_1(x)+t_2(x)+...+t_N(x)}{N}$. Thus, given a one-layer cluster with 2 servers and N tasks, traverse all the possibilities to partition the N tasks into 2 servers. Since two servers are identical, we just need to choose x from $\lceil \frac{N}{2} \rceil$ to N for one server, and $N-x$ for the other one. Since all the tasks are identical, then the complexity will be $O(N)$. Then we choose the best number to be put in the left server as x, remaining $N-x$ tasks will be put in the right server in the one-layer cluster.

VI. Multi-layer cluster Study

Given a binary tree multi-layer cluster with M servers and N task requests, we can get the optimal allocation scheme by traversing all the possibilities to partition the N tasks into M servers; however, it will be extremely time-consuming. Due to the NP-hardness of this problem, there is no optimal solution in polynomial time. However, based on the optimal results of the one-layer cluster, we can generalize this solution to multi-layer clusters, which has a considerably low time complexity.

For the bottom-layer access switches, we can view them as the root of a one-layer binary cluster, and try to abstract it into a single node. Each server under the access switch shares the same computing capability of C, and the sum of computing capabilities under the access switch is $2C$. Since the upper-layer links have twice the bandwidth resources as the lower layers, therefore, we can view this $2C$ as the accumulative computing capability of the abstraction node.

Based on this abstraction of the bottom-layer switch, we are able to abstract the entire multi-layer cluster to a one-layer cluster in a similar way. For each layer's switch connecting two sub-trees from the bottom to top, we can view it as the root of a one-layer binary cluster, and try to abstract it into a single node. Upon reaching the root switch at the top, the whole multi-layer cluster is abstracted into a one-layer cluster. We can see that our abstraction process misses no information. For abstraction nodes with the same accumulative computing capability, the inner structure of the original sub-trees are the same. We can see in Fig. 4, two servers with two lower-layer links are abstracted into a single node of accumulative computing capability of 2GHz. Both abstraction nodes with a computing capability of 2GHz share the same configuration of the original abstracted one-layer sub-tree.

Algorithm 1 Hierarchical Task Allocation Algorithm

Input: The links capacity and servers computing capability; Task requests $\langle N \rangle$

```
for layer $i$=1 to N do
    for all switches in layer $i$ do
        Calculate the accumulative capacity for each switch connecting two sub-trees in the layer
if input VMs could be accepted then
    for layer $j$=N to 1 do
        for all switches in layer $j$ do
            Optimally allocate the given number of tasks to this subtree
```

1) Computing capability Sharing of Multi-layer Clusters: The accumulative computing capability of the abstraction node is equally divided by all tasks allocated in this node.

2) Bandwidth Sharing of Multi-layer Clusters: For multi-layer clusters, the upper-layer link is connecting with a subtree with multiple servers. The link bandwidth would be equally divided by all tasks under that sub-tree.

3) Hierarchical Task Allocation Algorithm: With the abstraction of a multi-layer cluster into a one-layer cluster, we can use the optimal solution for a one-layer cluster. Based on that result, we propose our hierarchical task placement algorithm.

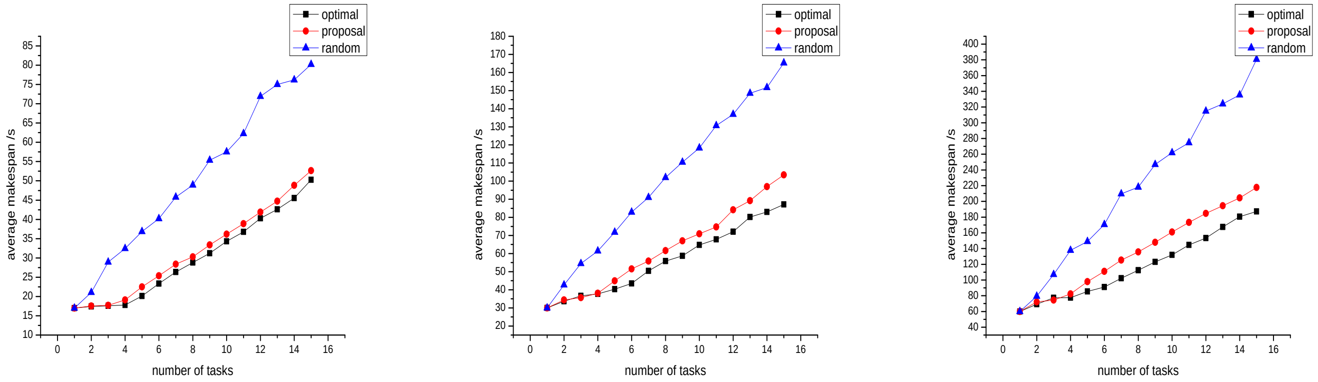
Given N input tasks, our algorithm can be divided into two steps. First, for each switch at each layer from bottom to top, the accumulative computing capability of the abstraction node rooted at that switch is calculated.

Second, for each switch connecting two sub-trees at each layer from top to bottom, recursively allocate the input tasks into the its two sub-trees according to our optimal one-layer solution. Upon finishing the bottom-layer switch (access switch), all the tasks are allocated into the servers. We summarize our algorithm in Algorithm 1.

Our algorithm takes two loops. The first is the abstraction from bottom layer to top, and the second is the allocation from top to bottom. For the first loop, each switch at each layer is abstracted. For a K layers cluster, the total number of switch is $\sum_{k=1}^{K-1} k$. Suppose we have M servers at the bottom, then $M = 2^K$. Therefore, it takes $O(M)$ for the first loop of abstraction process. In the second loop, for each layer, our algorithm takes $O(N)$ time to calculate the optimal solution, based on the discussion in Section V. In the K-layer cluster, it takes $O(KN)$ for the allocation process. In sum, the total time complexity of our algorithm is $O(KN + M)$, which is very efficient.

VII. Evaluations

In this section, we evaluate our proposed algorithm in the case of a 2-layer binary tree data center. We made comparisons with our proposed algorithm with the optimal brute force algorithm and the random allocation algorithm. We produce the optimal solution by programs that traverse all the possibilities dividing the N inputs into M servers. The random allocation algorithm is generated by putting random number of tasks into different servers.

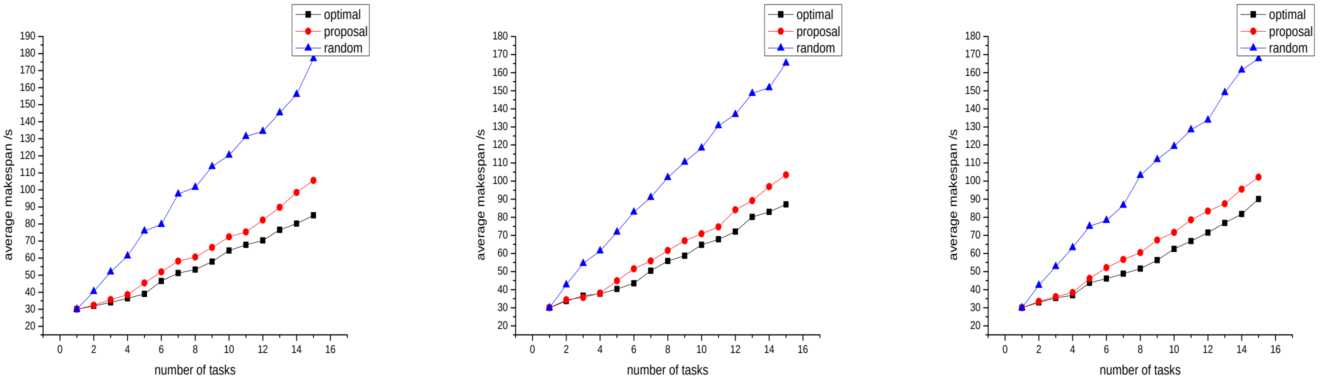


(a) $\alpha = 2, \beta = 2, \gamma = 15$

(b) $\alpha = 15, \beta = 2, \gamma = 15$

(c) $\alpha = 45, \beta = 2, \gamma = 15$

Figure 5: Performance comparisons of average makespan vs. the value of α



(a) $\alpha = 15, \beta = 1, \gamma = 15$

(b) $\alpha = 15, \beta = 2, \gamma = 15$

(c) $\alpha = 15, \beta = 4, \gamma = 15$

Figure 6: Performance comparisons of average makespan vs. the value of β

We evaluate the average makespan of our algorithm under three groups of simulations on the average total completion time. As shown in Figures 5, 6 and 7, our proposed algorithm is very close to the optimal one, which is much better than the random algorithm.

A. *Simulation Settings:*

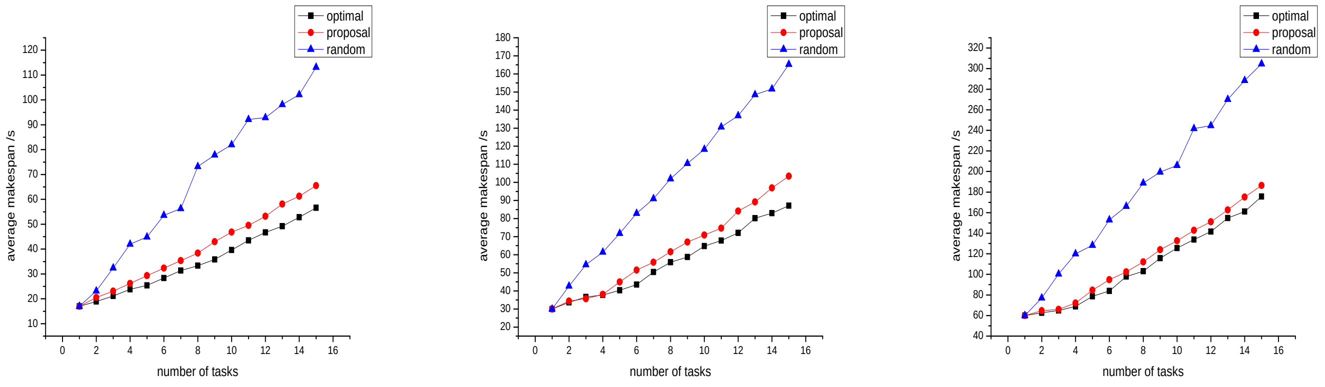
- Group 1: We select the α to be 2, 15 and 45 separately. And we set β, γ as 2 and 15. The bandwidth of links in each layer is equal. Task numbers go from 1 to 15.
- Group 2: We select the β to be 1, 2 and 4 separately. We also set α, γ as 15 and 15. The bandwidth of links in each layer is equal. Task numbers go from 1 to 15.
- Group 3: We select the γ to be 2, 15 and 45 separately. We also set β, α as 2 and 15. The bandwidth of links in each layer is equal. Task numbers go from 1 to 15.

B. *Simulation Results:*

The results for the three groups of simulations are shown in Figures 5, 6, and 7. From those, we can see that when the number of tasks grows, the proposed algorithm's performance will deviate from that of the optimal one. However, the proposed solution does not deviate far from the optimal solution, and still follows the growing pattern of the optimal solution. Besides that, we still have the following observations:

1) For different pre-computation times (α), as shown in Figure 5, when the pre-computation time grows, the proposed algorithm's performance will deviate farther from that of the optimal one.
2) For different communication times (β), as shown in Figure 6, when the communication time grows, the difference between the proposed algorithm's average makespan and that of the optimal one will grow slowly.
3) For different post-computation times (γ), as shown in Figure 7, when the post-computation time grows, the the proposed algorithm's performance will be closer to that of the optimal one.

As it is shown, the post-computation does not have the same influence as the pre-computation time or communication time does. One reason is that the post-computation is independent with the waiting time. Notably, the waiting time can inevitably increase the makespan of a task. Therefore, it



(a) $\alpha = 15, \beta = 2, \gamma = 2$

(b) $\alpha = 15, \beta = 2, \gamma = 15$

(c) $\alpha = 15, \beta = 2, \gamma = 45$

Figure 7: Performance comparisons of average makespan vs. the value of γ

makes sense for the pre-computation time to contribute more deviation than the post-computation time does. The smaller portion of the pre-computation time the better performance will be. The other reason is that, our proposed algorithm will loss some information about the bandwidth during the abstraction step. And the more congestion in the bandwidth resources, the larger the difference between the proposed solution and the optimal solution will be. However, even the performance varies with the ratio of $\alpha : \beta : \gamma$, our proposed algorithm is still much better than the random one.

VIII. Conclusions and Future Work

In this paper, we study the classic task allocation problem in the data center, which is now suffering from both the limited bandwidth resources and computing capabilities. Compared to the previous work, we focus on the tradeoff between locality and load balancing. To minimize the makespan of input tasks into the multi-layer data center, we first study the one-layer cluster and discuss the optimal solution. After that, we propose our hierarchical task allocation algorithm to deal with the multi-layer cluster. The evaluation results show the high efficiency of our algorithm.

In this paper, we only study the homogeneous task model under the semi-homogeneous data center configuration. In our future work, we will first extend the task model into heterogeneous settings. That is, each task will have different values of α, β, and γ.

Furthermore, the heterogeneous configuration of data centers will also be studied. We will consider two sets of heterogeneous data center configurations. The first is that, we let the servers' computing capabilities heterogeneous, while keeping the links capacities semi-homogeneous as before. The second will be that, we let all the links capacities, along with the servers' computing capabilities heterogeneous. We will evaluate the efficiency of algorithm both theoretically and experimentally.

References

[1] J. Xu and J. Fortes, "Multi-objective virtual machine placement in virtualized data center environments," in Proc. of CPSCom 2010, pp. 179 –188.

[2] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling," in Proceedings of the 5th European Conference on Computer Systems, ser. EuroSys '10. New York, NY, USA: ACM, pp. 265–278.

[3] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation, Berkeley, CA, USA, pp. 29–42.

[4] F. Ahmad, S. T. Chakradhar, A. Raghunathan, and T. N. Vijaykumar, "Tarazu: Optimizing mapreduce on heterogeneous clusters," in Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems, ser. ASPLOS XVII. New York, NY, USA: ACM, pp. 61–74.

[5] J. T. Piao and J. Yan, "A network-aware virtual machine placement and migration approach in cloud computing," in Proc. of GCC 2010, pp. 87 –92.

[6] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in Proc. of INFOCOM 2010, pp. 1 –9.

[7] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in Proc. of the ACM SIGCOMM 2011, pp. 242–253.

[8] L. Popa, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "Faircloud: sharing the network in cloud computing," in Proc. of ACM HotNets-X 2011, pp. 22:1–22:6.

[9] K. Li, J. Wu, and A. Blaisse, "Elasticity-aware virtual machine placement for cloud datacenters," in Cloud Networking (CloudNet), 2013 IEEE 2nd International Conference on, Nov 2013, pp. 99–107.