第48卷第6期 中国科学技术大学学报 Vol.48,No.6
2018年6月 JOURNAL OF UNIVERSITY OF SCIENCE AND TECHNOLOGY OF CHINA Jun. 2018

文章编号:0253-2778(2018)06-0495-09

面向带宽保障的云中虚拟集群调度算法

徐 华,李 京

(中国科学技术大学计算机科学与技术学院,合肥安徽 230026)

摘要:在多租户的云数据中心,由于网络资源的共享,最小带宽保障已经成为保障云应用性能的重要方法之一.高效的云中虚拟网络划分能够有助于容纳更多的虚拟集群,提高数据中心资源利用率.为此面向用户的网络带宽保障需求,提出了一种基于回溯的虚拟集群调度算法.针对典型的树形数据中心网络拓扑,首先逐层判定网络拓扑中每棵子树内部是否存在调度解,随后基于回溯的算法在子树内部递归搜索具体的放置方案,从而避免已有研究中存在的假性成功分配或者错误地拒绝请求的问题.实验表明,基于回溯的精确搜索能够有助于接受更多的虚拟集群请求,相对于已有算法,请求拒绝率降低了10%,有利于提高数据中心的资源利用率.

关键词:云计算;带宽保障;虚拟集群调度;回溯算法

中图分类号:TP393 **文献标识码**:A doi:10.3969/j.issn.0253-2778.2018.06.008

引用格式:徐华,李京. 面向带宽保障的云中虚拟集群调度算法[J]. 中国科学技术大学学报,2018,48(6):495-503.
XU Hua, LI Jing. A scheduling algorithm towards bandwidth guarantee for virtual cluster in the cloud [J]. Journal of University of Science and Technology of China, 2018,48(6):495-503.

A scheduling algorithm towards bandwidth guarantee for virtual cluster in the cloud

XU Hua, LI Jing

(*Department of Computer Science and Technology, University of Science and Technology of China, Anhui* 230026, *China*)

Abstract: Due to the sharing of network resources in multi-tenant cloud data centers, minimum bandwidth guarantee has become one of the important methods to provide predictable performance for cloud applications. Efficient virtual network allocation helps accommodate a larger number of virtual clusters and improve the resource utilization in the cloud. Towards the demand for network bandwidth guarantee, this paper proposes a novel backtracking algorithm for scheduling virtual clusters in the cloud. For the typical tree network topology of a data center, this algorithm firstly judges whether there exists a valid solution inside each sub-tree in the network topology. After determining the sub-tree for the virtual cluster, it recursively searches for the detailed solution inside the sub-tree based on a backtracking algorithm, thus avoiding the problems of fake allocation or rejecting the request by mistake existing in related works. The experimental results show that the exact search based on backtracking can help increase the acceptance ratio of virtual cluster requests. Compared with existing algorithms, it can reduce the rejection ratio by 10% on average, which can contribute to improving the resource utilization in the cloud.

Key words: cloud computing; bandwidth guarantee; virtual cluster scheduling; backtracking algorithm

收稿日期:2017-09-25;**修回日期**:2018-04-10

基金项目:科技部国家重点研发计划项目(2016YFB021402)资助;赛尔网络下一代互联网技术创新项目(NGII20150110)资助.

作者简介:徐华,男,1990年,博士研究生,研究方向:分布式系统,云计算,大数据处理等.E-mail:boxwh1@gmail.com

通讯作者:李京,博士/教授.E-mail:lj@ustc.edu.cn

0 引言

随着海量搜索、并行计算等数据敏感类应用的增多,“东西”向网络流量逐步在云数据中心占据主导地位①.这主要是因为 Hadoop②、Spark[1]等分布式系统往往需要大量的服务器组成集群协同工作,使得服务器之间的内部网络传输流量逐渐增加.由于云中网络资源的共享,不同租户的虚拟集群之间必然会有资源竞争,进而导致网络性能的波动,最终影响到服务质量或应用性能[2].目前解决该类问题的核心思想是通过带宽预留的方式来为云租户的虚拟集群提供最小带宽保障[3-4].类似于 CPU 和内存,云租户可以指定需求的网络带宽配额,云数据中心会将网络带宽需求考虑在内进行资源调度,并且在集群运行时提供需求的最小带宽保障.不同于单个虚拟机对外的网络带宽保障,虚拟集群的带宽保障更加复杂,刻画租户对集群网络性能的需求,并且在复杂的网络拓扑中为其寻找满足需求的调度方案是其中的关键,因此它也成为了云计算中的研究热点;现有的解决方案包括 SecondNet[5]、Oktopus[6]和 VirtualRack[7]等.

SecondNet[5]提出了虚拟数据中心(VDC)的模型,提供虚拟机(VM)两两之间的带宽保障.由于模型需要云租户精确地指出任意两个虚拟机之间的带宽需求,对于云租户来说复杂度过高,因此需要用户具备丰富的经验,不够友好.Oktopus[6]提出了虚拟集群 VC 模型,通过模拟物理的连接方式,提供多个 VM 到虚拟交换机的带宽保障,用户只需指定虚拟机数量以及它们到虚拟交换机的上连带宽即可.该模型能够简化用户对网络带宽的需求,同时能够有效地提高数据中心的网络利用率.

Oktopus 在 VC 模型的基础上提出了相应的调度算法.面向带宽保障的虚拟集群调度不仅需要考虑物理机上的计算资源,同时需要考虑该 VC 内部通信时经过的物理交换机和链路的可用带宽.Oktopus 在已知数据中心网络拓扑的条件下,分两步对 VC 进行调度.首先检测每个网络节点的上连链路剩余带宽和以该节点为顶点的子树内部剩余可用的槽数,寻找能够容纳整个 VC 的子树.在确定子树之后,再使用贪心算法将 VC 所需的虚拟机依次放置到该子树内部的节点中.由于在贪心寻找的过程中,它并没有再次考虑每个节点是否满足 VC 的带宽通信需求,导致可能存在假性成功分配现象[7],即子树的顶层节点满足需求,但是其子节点的通信带宽不一定满足需求的情况.

VirtualRack[7]对此提出了改进算法,通过资源分配后从子树的顶层节点往底层节点逐层再次检查,检测带宽需求是否满足,如果不满足则跳过该子树,尝试重新选择其他子树并且重新调度所有虚拟机.这种方案可能会跳过拓扑中唯一的可行解,最终引起虚拟集群请求的拒绝.

本文设计提出了一种基于回溯的虚拟集群调度算法.算法基于 VC 模型,捕捉云租户的虚拟集群对 VM 数量和上连带宽的需求,进而对虚拟集群请求进行调度.在对虚拟集群进行调度时,为了尽量接受更多的用户请求,提高数据中心资源利用率,算法尽量将整个 VC 放置到低层的子节点内部,因此算法从底往上逐层地计算每个节点内部可容纳的该类型虚拟机的数目,确定能够容纳整个 VC 的子树.在具体的虚拟机分配时,引入了回溯的方式组合匹配该子树内部每个子节点可容纳的虚拟机数目,达到精确调度虚拟集群请求的目的,避免因为贪心策略导致错过可行解,造成请求拒绝的情况.实验结果表明,本文提出的算法能够有效地避免假性成功分配现象,并且能够降低对 VC 请求的拒绝率,其相对值约为 10%,有利于提高云数据中心的资源利用率.

1 相关工作

对云租户应用的虚拟集群的网络需求模型进行刻画,以适应不同场景下的需求,是为云租户提供网络带宽保障的基础. VDC[5]、VC[6]、FGVC[8]和 TAG[9]等都是针对用户虚拟集群网络性能保障提出的模型.VDC 的目标是保障集群中任意两个虚拟机之间的网络带宽,而 VC 模型则是模拟现实中物理服务器的互连方式,保障虚拟机与虚拟交换机之间的带宽.FGVC 模型在 VC 模型的基础上,细化了租户对部分两两虚拟机对之间的网络传输带宽需求,使得用户能够更好地表达和定制应用对网络的需求.TAG 则模拟了云中应用的不同组织结构,保障不同应用组件之间的连接带宽,从而保障应用的网络性能.不同的模型有不同的适用场景,对云租户经验的要求也不尽相同.本文基于 VC 模型进行研

① http://www.cisco.com/c/dam/m/zh_cn/solutions/service-provider/sp_gciwhitepaper_whitepaper_cn.pdf. 2017 September.

② Apache Hadoop. http://hadoop. apache. org/. 2017 September.

究，VC 模型对用户经验的要求最低，实际部署时最简单，所有应用均能适用，尤其适合当前主流的大数据处理类应用，因此有较高的实用价值.

基于虚拟集群的网络需求模型，研究人员致力于研究其调度算法.部分学者提出在保障虚拟集群最小带宽需求的同时，提高云数据中心的利用率. Oktopus[6]、VirtualRack[7]均以此为目标，都是基于 VC 模型.本文针对 Oktopus 的假性分配和 VirtualRack 可能错过可行解的问题，提出了基于回溯查找的方法，在引入少量的时间开销的情况下，进一步降低了对云数据中心虚拟集群请求的拒绝率，从而提高了数据中心的资源利用率.

此外，研究人员还对虚拟集群的扩展性、服务稳定性等目标进行了考量.文献[10]研究了基于 VC 模型时如何根据用户对虚拟集群的规模和带宽需求变化，在快速地扩展集群的同时，最小化虚拟机迁移代价.文献[11]指出，在调度虚拟集群时，需要预留足够的带宽资源，以便在物理机故障后快速地从备份资源中恢复虚拟集群的服务，保证虚拟集群服务的高可用性.这一类工作在本文的研究目标范围之外，两者能够相互补充，使得虚拟集群在云中运行的网络性能得到更好的保障.

2　问题模型描述

不同于单个虚拟机的调度问题，面向带宽保障的虚拟集群的调度问题需要考虑整个集群内部通信的网络带宽和计算资源需求.本章先介绍云数据中心模型，然后描述虚拟集群的模型，最后介绍面向带宽保障的虚拟集群调度问题模型.

2.1　云数据中心模型

在云数据中心，海量带有计算资源的服务器通过网络和交换机等网络设备互连形成了整个数据中心，其网络拓扑复杂多变[12].本文以典型的树形拓扑为例对云数据中心进行建模.

将云数据中心模型定义为树 $T=<D, E>$.D 表示其节点的集合，包括两类节点：叶子节点 P 和非叶子节点 S.其中 P 表示物理服务器的集合，数量为 n；S 表示物理交换机的集合，数量为 q；E 代表节点之间连接的边的集合；$E_{i,j}$ 表示两个物理设备 D_i 和 D_j 之间的链路，它们之间的连接带宽表示为 $\mathrm{bw\_pl}_{i,j}$.对于每个物理机 P_i 中的计算资源，由于本文主要是面向带宽保障的调度，所以我们将其简化为槽数 slot_i（包括 CPU、内存和磁盘资源），代表该物理机可以容纳的虚拟机个数.一个典型的树形拓扑模型如图 1 所示.

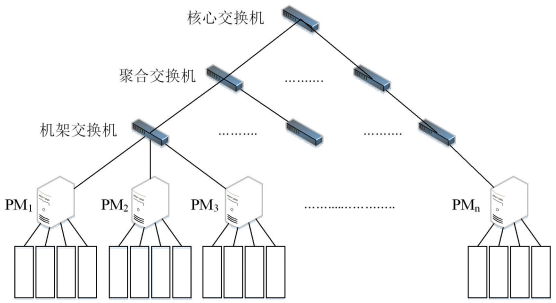

图 1　数据中心拓扑示例

Fig.1　An example of network topology of data center

2.2　虚拟集群模型

一个虚拟集群请求可以刻画为一个 VC$<m$，$B>$，m 代表该虚拟集群需要的虚拟机数目，B 则代表虚拟机与虚拟交换机的连接带宽需求，与物理的连接方式类似，每个虚拟机可以无阻塞地与内部的其余虚拟机以带宽 B 进行通信.该虚拟集群拓扑可以被概括为一层的树形拓扑，如图 2 所示.

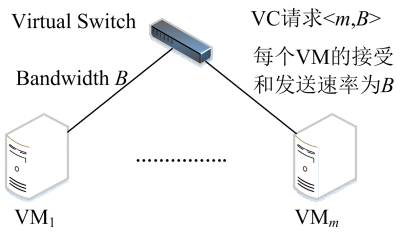

图 2　虚拟集群的树形拓扑

Fig.2　Tree topology of virtual cluster

2.3　调度问题模型

面向带宽保障的虚拟集群调度的目的是在云数据中心网络拓扑模型中寻找满足虚拟集群需求的物理机位置进行调度，并且满足其对网络带宽的需求.具体地，假定在某一时刻有若干个虚拟集群存在，此时物理机的剩余资源状态可表示为（$\mathrm{avail\_slot}_1$，$\mathrm{avail\_slot}_2$，…，$\mathrm{avail\_slot}_n$），而物理链路 e 的剩余带宽为 $\mathrm{avail\_bw}_e$.此时若有一个虚拟集群请求到来，其需求为$<m, B>$，调度目标则是将该 m 个虚拟机放置到 n 个物理宿主机的可用槽中，因此我们将此调度问题形式化描述为寻找一个映射：

$$\pi:[0,1,\cdots,m-1]\rightarrow[0,1,2,\cdots,n-1] \quad (1)$$

同时该问题需要满足计算和网络资源的限制.对于计算资源，每个物理机内部的可用槽数必须大于等于分配给该 VC 的虚拟机数目，可表示为

$$\forall k\in[0,n),\mathrm{avail\_slot}_k\geqslant\sum_{\{i\in[0,m);\pi(i)=k\}}1 \quad (2)$$

对于网络带宽资源，每个物理链路的剩余带宽必须大于等于该 VC 的虚拟机内部通信需求的最小带宽.对于给定的映射关系，可以通过映射结果计算出以每个节点 D' 为顶点的子树内部放置的该 VC 的虚拟机个数 v .若 e 是 D' 的上连链路，它把 VC 的 m 个虚拟机分割为两部分(v 和 $m-v$)，那么该 VC 经过链路 e 的最大网络传输速率限制为 $\min(v,m-v)*B$ ，因此我们将对网络带宽的限制条件定义为

$$\forall e \in E, \text{avail\_bw}_e \geqslant \min(v,m-v)*B \quad (3)$$

3 基于回溯的虚拟集群调度算法

为了解决此调度问题，并且避免假性分配和错误的拒绝请求，本文的算法思路是从底往上逐步搜索各节点，精确计算以该节点为顶点的子树内部可容纳的“此种类型的虚拟机数量”的集合 M，从而确定能够容纳整个虚拟集群的子树顶点.如果请求的 VM 数量 m 在某个节点的集合 M 中，则基于回溯的方法递归地在该顶层节点的儿子节点中组合匹配寻找每个儿子节点放置的虚拟机数量，最终确定该 VC 的 m 个 VM 在 n 个物理机上的映射.如果遍历所有节点都没有找到满足要求的顶层节点则拒绝该请求.我们将算法简称为 bt_search，其算法描述如算法 3.1 所示.

算法 3.1 bt_search

```
Input: the topology tree T, the request of VC(m,b)
Output : whether there exists a possible solution
1 for i=3 to 0 do
2    nodes Ā get_nodes_at_level(i)
3    sort nodes by the sum of intraband width from the lower
     to the higher, of i is not 3
4    for each node in nodes do
5       m_set Ā cal_m(m,b,node)
6       if m in m_set then
7          ScheduleVc (m,b,node)
8          update the available solts of physical nodes and the
           available bandwidth of links
9          return true
10       end if
11    end for
12 end for
13 return false
```

首先，算法从底往上逐层搜索节点内部是否能够容纳下整个 VC 请求.在对交换机节点(算法中条件 i 不等于 3)进行搜索时，首先计算该层节点内部剩余带宽之和，然后对它们进行排序，剩余带宽低的节点优先进行搜索，这样会尽量将 VC 集中化放置，有利于容纳更多的 VC.

随后，算法使用 cal_m 函数计算节点内部可容纳的“此种类型的虚拟机数量”的集合 M.在描述其具体的算法之前，我们先给出 M 的计算公式.对于叶子节点(即物理机节点)，计算方式如下：

$$M_{\text{leaf}} = \{i \in [0, \min(\text{avail\_slot}, m)] \quad \text{s.t.} \quad \min(i, m-i)*B \leqslant \text{avail\_bw}_e\} \quad (4)$$

式中，avail_slot 是该节点剩余的槽数，m 是该 VC 需求的 VM 数量，B 是需求的上连带宽，avail_bw_e 则是该节点上连链路的剩余带宽.由此可以看出，如果放置 k 个 VM($k \in M$)在该节点内部，那么它必定满足限制条件(2)和(3)，前半部分对应于计算资源的限制，后半部分则满足了带宽资源的限制.值得一提的是，由于带宽的限制，M_{leaf} 不一定是连续整数的集合，有可能整个 VC 都能放置到该节点内部，也有可能只能放置一部分的节点.

对于非叶子节点(交换机节点)，计算方式如下：

$$M_{\text{nonleaf}} = \{i \in \text{combinations\_children} \quad \text{s.t.} \quad \min(i, m-i)*B \leqslant \text{avail\_bw}_e\} \quad (5)$$

与公式(4)不同，i 的取值范围不再是由该节点内部剩余的槽数决定，而是由其所有儿子节点可容纳的 VM 数量的组合形成的一个集合 combinations_children 决定.因为如果只关心该节点内部剩余的槽数和节点上连链路的剩余带宽，则有可能出现 Oktopus 中的假性分配现象，即寻找到的顶层节点满足需求，但是其内部节点没有满足需求的调度方案的现象.考虑内部各个儿子节点可容纳的数量的组合数，则能精确判断该节点内部是否能够容纳下整个 VC 的所有 VM.

具体地，计算集合 M 的 cal_m 函数的算法描述如算法 3.2 所示.对于两种不同类型的节点，算法对 num_set 的初始值设置了不同的计算方式.对于物理机节点，num_set 的取值范围根据公式(4)可以直接得到，而对于交换机节点，算法计算其子节点的集合 M 中的数的任意组合相加的和，得到的值存储在 num_set 中.值得一提的是，由于组合的可能非常多，因此算法只记录最终组合相加的结果的可能值，它们互相不重复，这样有利于减少空间复杂度和时间复杂度.随后根据公式(4-5)的后半部分限制，算法对 num_set 中的数逐一判断其是否满足 VC 带宽需求.最后算法缓存低层节点的集合 M，避免高层节

点判定时的重复计算.

算法 3.2　cal_m

```
1 function Cal_m(node,vm_num,b)
2   num_set Ā [0,1,…,min(node.avail_slot,vm_num)]
3   if type of node is switch then
4     for each child in node.get_children do
5        children_m_sets.app end (child.get_m_set)
6      end for
7      num_set Ā the any combination of the number in
       the sets of children_m_sets
8   end if
9   for each num in num_set do
10       if min(num,vm_num-num) * b 6 node.up_link.
         avail_band then
11         m_set.append(num)
12       end if
13     end for
14     node.m_set Ā m_set
15 return m_set
16 end function
```

在计算得到节点的集合 M 后,如果该 VC 需求的虚拟机数量 m 在此集合中,则说明以该节点为顶点的子树内部能够容纳下整个 VC 的虚拟机并且满足其带宽需求.随后我们需要找出在顶层节点内部各个子节点中的具体分配方案,如算法 3.1 中的第 7 行所示,算法使用 ScheduleVc 函数在该节点内部寻找 m 个 VM 的具体放置方案,使得放置后每个链路均满足 VC 的带宽通信需求,其具体的分配算法如算法 3.3 所示.

算法 3.3　ScheduleVc

```
1 procedure ScheduleVc(m,b,node)
2   if the type of node is server and min node. m_set then
3       place mvm sin this node
4   else
5       solution Ā GenS (m,b,node.children)
6       for each child_solution in solution do
7         ScheduleVc (child_solution.m,b,child)
8       end for
9   end if
10 end procedure
11
12 function GenS (m,b,nodes)
13.  if len (nodes)==land min nodes[0].M_set then
14       solution [node[0]]Ā m
15       return true, solution
16   else
17       split the nodes into two part symmetrically: left
         and right
18       possible_set Ā compute the set of the number of
         VM which can be placed in left
19       while true do
20         if possible_set in null then
21           return false, N one
22       end if
23       max_le_set is null then
24       possible_set. remove(max_left)
25       if s_left Ā GenS(max_left,b,left) and s_right Ā
         GenS(m_max_left,b,right) then
26         return true,s_left+s_right
27       end if
28     end while
29   end while
30 end function
```

在算法 3.3 中,首先对该节点的类型进行判断,如果是物理机节点(叶子节点),则可以直接将 m 个虚拟机分配到该节点内部,分配完成.如果该节点是交换机节点(非叶子节点),则使用 GenS 函数在其所有的儿子节点中寻找可行解.GenS 以递归的方式在儿子节点中寻找分配方案,其思路是使用二分查找,将儿子节点尽量对称地分成左右两个部分,并且分别计算两部分可容纳的此类型虚拟机数量的最大值之和,从而得到左边部分可容纳的数量的范围集合.在可容纳的数量范围内,从大到小地尝试将 max_left个虚拟机放置到左边部分的儿子节点中.如果成功,则继续尝试将 m-max_left 个虚拟机放置到右边部分的儿子节点中.如果这两次尝试中有一次不成功,则进行回溯,让 max_left 值减少继续尝试,直到没有可尝试的值,则再往上层回溯.如果找到可行解,则返回 m 个虚拟机的放置位置信息.在 GenS 函数得到可行解 solution 后,则按照 solution 在该节点的儿子节点中使用 ScheduleVc 函数逐层递归地放置这 m 个虚拟机.

4　实验评估

为了评估本文提出的算法 bt_search 的效率,我们模拟云数据中心的虚拟集群分配过程,进行了一系列的实验评估.我们选取了 Oktopus 和 VirtualRack 的调度算法作为对比算法,它们均是基于 VC 模型的调度算法.实验在两个方面对算法进行评估:VC 请求的拒绝率和算法的时间开销.

4.1 实验环境设置及评价指标

云数据中心环境:由于缺少足够数量的物理设备来构成真实的测试环境,实验基于 Python 开发了一个模拟器来模拟云数据中心对虚拟集群请求的调度.实验环境模拟了一个典型的 3 层树形架构网络拓扑,并且参考相关数据中心的过度订阅率[13],底层链路与核心层的过度订阅率被设置为 1∶10.在具体的树形拓扑中,每 40 个物理机通过 1 Gbps(简称为 G)的链路连接到机架交换机,每 10 个机架交换机通过 10 G 的链路连接到聚合交换机,最终每 10 个聚合交换机通过 40 G 的链路连接到核心交换机,因此一共设置了 100 个交换机,4 000 个物理机节点,并且每个物理机的最大可用槽数设置为 4,则总共可容纳的虚拟机个数为 16 000.

虚拟集群请求设置:为了模拟真实情况下虚拟集群请求的到达和离开,实验假定虚拟集群的请求服从到达速率为 λ 的泊松分布.而其离开的时间则为其任务完成的时间,平均任务完成时间被设置为 T_c,服从均匀分布,波动范围设置为 $0.4 * T_c$.另外参考相关文献中观察到的数据中心请求数据[14],虚拟集群的大小 m 被设置为满足平均值为 49 的均匀分布,波动范围设置为 15.每组实验模拟了 2 000 个 VC 请求的响应情况,单位时间到达的 VC 请求数量 λ 设置为 4.

控制变量:①虚拟集群的带宽需求.由数据中心的设置可以看出,由于每个物理机上连带宽约为 1 000 Mbps(简称为 M),内部的槽数为 4,所以理论上平均为每个槽可用带宽为 250 M.当每个虚拟集群需求的带宽值 B 低于 250 M 时,计算资源的合理分配是关键,反之网络带宽的调度是关键,因此对于虚拟集群请求的带宽值 B,其平均值被设置为 100 M、200 M、250 M、300 M、400 M 和 500 M,递进地反映云数据中心在网络资源轻负载、适中的负载和高负载的场景下算法的效率.特别地,我们在其中加入了 250 M 这一临界点的实验评测,以便观测不同算法对 VC 请求拒绝率的变化.在每组实验中,不同虚拟集群之间带宽服从均匀分布,平均带宽值为 B,波动范围为 $0.4 * B$,以此模拟云租户的带宽需求的差异性.②云数据中心的负载.由于虚拟集群请求的到达服从泊松分布,在任务完成后会离开并释放资源,而云数据中心的计算资源是有限的,所以我们可以通过下式计算出云数据中心在稳定状态的负载,实验通过变化云数据中心的负载来评估算法的效率.

$$\mathrm{Load}=\frac{\lambda * m * T_c}{\mathrm{slot}_{\mathrm{total}}} \tag{6}$$

式中,λ 是单位时间内到达的 VC 请求的平均值,m 代表每个 VC 请求的平均虚拟机数量,T_c 则为 VC 的平均任务完成时间,$\mathrm{slot}_{\mathrm{total}}$ 则是数据中心可承载的虚拟机总量.实验通过调整虚拟集群的平均计算时间 T_c,来调整云数据中心负载,以此来评估算法的效率.实验中云数据中心的负载被递进地设置为 10%,20%至 90%.

评价指标:为了评估本文提出的算法,我们选择了两个性能指标:VC 请求的拒绝率和算法的时间开销.①VC 请求的拒绝率,即数据中心对 VC 请求数量的拒绝次数与收到 VC 请求的总量的比值,它能够反映调度算法对每次 VC 请求的调度的合理性.拒绝率越低,则说明算法效率越高,对数据中心资源的利用率就越高,同样成本下也就意味着更高的利润.②算法的时间开销,它能够说明算法对虚拟集群请求响应的及时性,当数据中心接收到云租户的 VC 创建请求时,需要及时(秒级)响应请求并且为其调度虚拟集群.

4.2 实验结果及分析

4.2.1 VC 请求的拒绝率

实验在不同的云数据中心负载和不同的带宽需求条件下评估了 3 种算法对 VC 请求的拒绝率,按带宽不同分类,其结果分别如图 3 至图 6 所示(在带宽需求为 100 M 和 200 M 时,三个算法对 VC 请求的拒绝率均为 0,故不在图中展示).

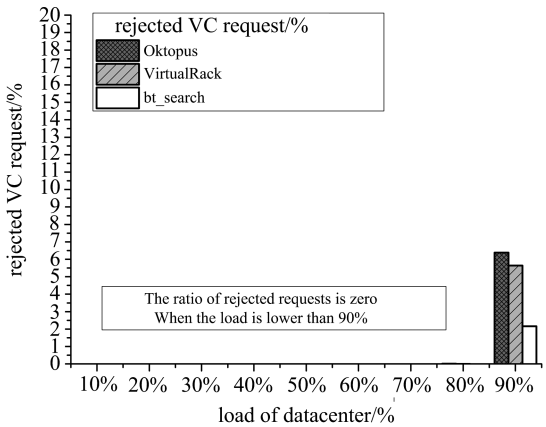

图 3 VC 请求的拒绝率(250 M)

Fig.3 Ratio of rejected requests (250 M)

从结果中可以看出:①随着 VC 请求带宽的增加,拒绝率是逐渐增加的,这是因为网络带宽资源有限,带宽需求值越高,则 VC 跨高层交换机节点所需

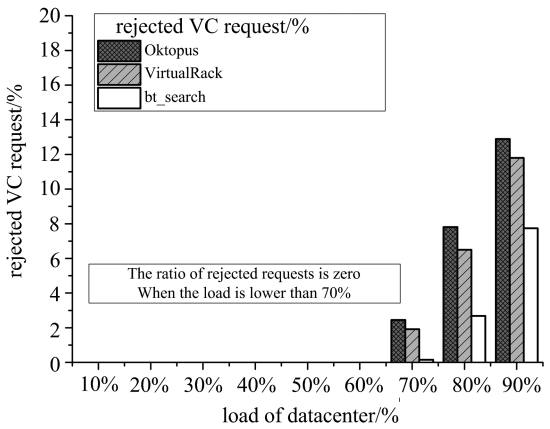

图 4　VC 请求的拒绝率(300 M)

Fig.4　Ratio of rejected requests (300 M)

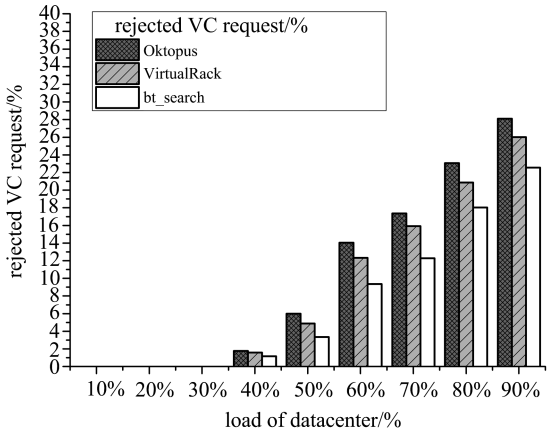

图 5　VC 请求的拒绝率(400 M)

Fig.5　Ratio of rejected requests (400 M)

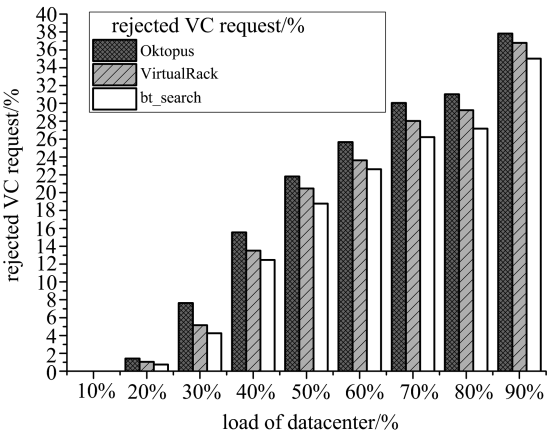

图 6　VC 请求的拒绝率(500 M)

Fig.6　Ratio of rejected requests (500 M)

要的带宽就越高,最终造成拒绝率会增加.②在带宽值比较低时(如 100 M 和 200 M),3 类算法的拒绝率都为 0,这是因为理论上每个槽的平均可用带宽为 250 M,当低于这个值时,带宽的负载低于计算资源的负载,网络资源的分配没有太多的限制,所以只要有剩余的槽,VC 就有极大概率被接受.③当平均需求的带宽值达到 250 M 左右时,开始出现请求拒绝的情况.如图 3 所示,在 250 M 带宽条件下且云数据中心处于 90%负载时,不同算法均开始出现 VC 请求的拒绝.④在 VC 请求开始出现拒绝的所有场景中,Oktopus 算法拒绝率最高,而本文提出的算法性能相比于 Oktopus 算法和 VirtualRack 算法的性能都有所提升,能够有效降低数据中心对 VC 请求的拒绝率.以 VirtualRack 为基准,拒绝率下降的相对值平均约 10%,绝对值最大超过 2%.

4.2.2　算法的平均响应时间

实验在不同的云数据中心负载和不同的带宽需求条件下评估了 3 种不同的算法对 VC 请求的平均响应时间.下面选取了 100 M、250 M 和 500 M 带宽作为网络资源低负载、适中负载和高负载场景的代表,按带宽不同分类,其响应时间分别如图 7、图 8 和图 9 所示.

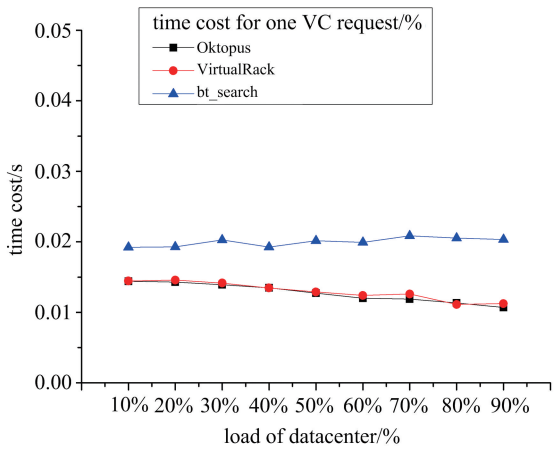

图 7　VC 请求的平均响应时间(100 M)

Fig.7　Average time cost for one VC request (100 M)

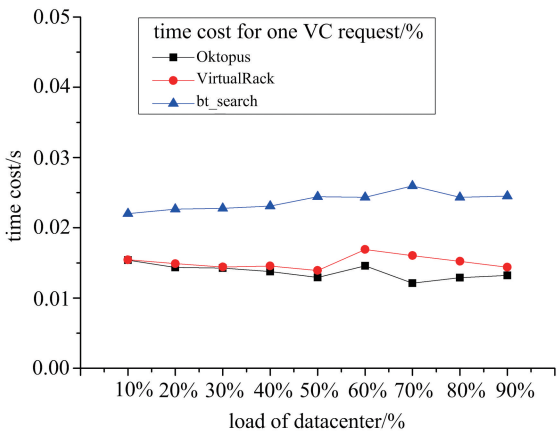

图 8　VC 请求的平均响应时间(250 M)

Fig.8　Average time cost for one VC request (250 M)

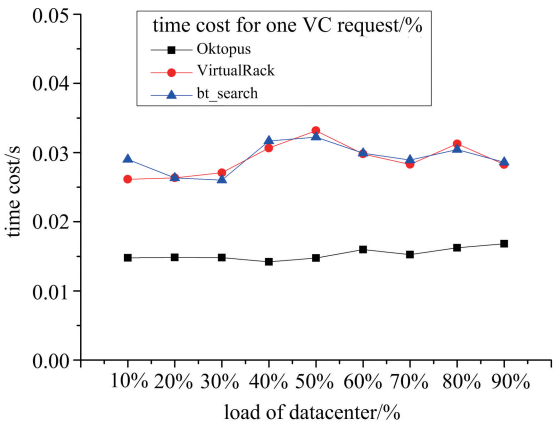

图 9 VC 请求的平均响应时间(500 M)

Fig.9 Average time cost for one VC request (500 M)

从图 7,图 8 和图 9 中可以看出:①3 种算法在不同的云数据中心负载条件下,对 VC 请求的响应时间均低于 0.1 秒,都能够满足数据中心对 VC 请求调度的及时性要求.②3 种算法中,Oktopus 算法时间开销最少,这是因为它没有对假性分配现象进行处理,也就不会进行额外的搜索.而 VirtualRack 会进行检查并在下一个节点继续进行搜索.本文算法则会在确定好放置 VC 的顶层节点后,采用回溯的方法在内部进行搜索正确的放置方案,因此会更耗时.③虽然 bt_search 算法为了确定具体的放置方案进行了回溯搜索,但是其额外的时间开销仍在可接受范围之内.这是因为在确定好放置 VC 的顶层节点后,bt_search 算法会逐层地采用二分查找在其儿子节点内部进行搜索,其时间开销只与每个节点的儿子结点数量有关,并不会随着数据中心规模的增加而成爆炸式增长,因此不会增加过多的搜索时间.④3 类算法的时间开销都随着网络带宽需求的增加而有所增长.这是因为在低带宽时,将整个 VC 放置到较低层的节点内部的可能性更大,反之在高层节点内部可能性更大.高层节点由于搜索空间大于低层节点,所以造成搜索时间开销的增加.⑤对比 VirtualRack 和 bt_search 算法,在低带宽时,VirtualRack 较优,而在高带宽(500 M)时两者的时间开销几乎相等.这是因为 VirtualRack 算法可能存在错过可行解的情况,会进行大量额外的搜索,而高层节点的搜索开销会较大,所以额外的搜索会更加耗时.相比之下,bt_search 算法在第一次找到可放置的顶层节点后便能正确地找到具体的放置方案,所以能够有效减少多余无用的搜索.

从实验结果中可以看出,相比于已有算法,本文提出的算法能够有效降低云数据中心对 VC 请求的拒绝率,有利于提高数据中心资源利用率.在时间开销上,虽然算法引入了额外的回溯算法,但是其额外增加的时间开销在可接受范围之内,总的时间开销能够满足 VC 请求响应的及时性需求.

5 结论

网络性能是云租户对云中资源性能要求的重要指标,因此为云租户的虚拟集群提供可靠的带宽保障显得十分重要.本文面向租户的带宽保障需求,针对已有虚拟集群调度算法的不足之处,提出了一种新型的基于回溯的虚拟集群调度算法.算法在避免假性分配的基础上,能够有效减少对用户请求的拒绝率,有助于提高云数据中心的资源利用率.

参考文献(References)

[1] ZAHARIA M, CHOWDHURY M, FRANKLIN M J, et al. Spark: Cluster computing with working sets [C]// Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing. Berkeley, CA, USA: USENIX Association, 2010:10.

[2] AIDA K, ABDUL-RAHMAN O, SAKANE E, et al. Evaluation on the performance fluctuation of Hadoop jobs in the cloud [C]// IEEE 16th International Conference on Computational Science and Engineering. Piscataway, NY, USA: IEEE Press, 2013: 159-166.

[3] SHIEH A, KANDULA S, GREENBERG A, et al. Seawall: Performance isolation for cloud datacenter networks [C]// Proceedings of the 2nd USENIX conference on Hot topics in cloud computing. Berkeley, CA, USA: USENIX Association, 2010:1.

[4] POPA L, YALAGANDULA P, BANERJEE S, et al. Elasticswitch: Practical work-conserving bandwidth guarantees for cloud computing[J]. ACM SIGCOMM Computer Communication Review, 2013, 43 (4): 351-362.

[5] GUO Chuanxiong, LU Guohan, WANG Helen, et al. SecondNet: A data center network virtualization architecture with bandwidth guarantees[C]// The 6th International Conference on emerging Networking Experiments and Technologies. New York: ACM, 2010: 15.

[6] BALLANI H, COSTA P, KARAGIANNIS T, et al. Towards predictable datacenter networks [J]. ACM SIGCOMM Computer Communication Review, 2011, 41(4): 242-253.

[7] 荣 超,唐亚哲,胡成臣,等.基于带宽感知的多租户云数

据中心虚拟网络分配算法[J].小型微型计算机系统，2015,36(1):7-12.
RONG Chao, TANG Yazhe, HU Chengchen, et al. Bandwidth-aware virtual network allocation in multi-tenant cloud datacenters [J]. Journal of Chinese Computer System, 2015, 36(1): 7-12.

[8] YU Hui, YANG Jiahai, XU Cong, et al. SpongeNet: Towards bandwidth guarantees of cloud datacenter with two-phase VM placement[C]// 2016 IEEE/IFIP Network Operations and Management Symposium. Piscataway, NY, USA: IEEE Press, 2016: 410-417.

[9] LEE J, TURNER Y, LEE M, et al. Application-driven bandwidth guarantees in datacenters[J]. ACM SIGCOMM Computer Communication Review, 2014, 44(4): 467-478.

[10] YU Lei, CAI Zhipeng. Dynamic scaling of virtual clusters with bandwidth guarantee in cloud datacenters [C]// IEEE International Conference on Computer Communications. Piscataway, NY, USA: IEEE Press, 2016: 1-9.

[11] YU Ruozhou, XUE Guoliang, ZHANG Xiang, et al. Survivable and bandwidth-guaranteed embedding of virtual clusters in cloud data centers [C]// IEEE International Conference on Computer Communications. Piscataway, NY, USA: IEEE Press, 2017.

[12] 李丹,陈贵海,任丰原,等.数据中心网络的研究进展与趋势[J].计算机学报,2014,37(2):259-274.
LI Dan, CHEN Guihai, REN Fengyuan, et al[J]. Chinese Journal of Computers, 2014, 37(2): 259-274.

[13] GREENBERG A, HAMILTON J R, JAIN N, et al. VL2: A scalable and flexible data center network[J]. ACM SIGCOMM Computer Communication Review, 2009, 39(4): 51-62.

[14] NIU Di, XU Hong, LI Baochun, et al. Quality-assured cloud bandwidth auto-scaling for video-on-demand applications[C]// IEEE International Conference on Computer Communications. Piscataway, NY, USA: IEEE Press, 2012: 460-468.