Future Generation Computer Systems 66 (2017) 1–10

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Clustering based virtual machines placement in distributed cloud computing

Jiangtao Zhang [a,b], Xuan Wang [a,c], Hejiao Huang [a,d,*], Shi Chen [a,d]

[a] School of Computer Science and Technology, Harbin Institute of Technology Shenzhen Graduate School, Shenzhen 518055, China
[b] Public Service Platform of Mobile Internet Application Security Industry, Shenzhen 518057, China
[c] Shenzhen Applied Technology Engineering Laboratory for Internet Multimedia Application, Shenzhen 518055, China
[d] Shenzhen Key Laboratory of Internet of Information Collaboration, Shenzhen 518055, China

HIGHLIGHTS

- A clustering based algorithm is recommended for data center selection problem.
- It is more applicable to large scale VM placement and runs faster.
- A more effective VM partition algorithm is presented utilizing a new model of VMs.
- All algorithms can address both homogeneous and heterogeneous requirements of VMs.

ARTICLE INFO

Article history:
Received 14 November 2013
Received in revised form
26 May 2016
Accepted 19 June 2016
Available online 29 June 2016

Keywords:
Virtual machines placement
Data center selection
Bandwidth minimizing
Cloud computing

ABSTRACT

Resource virtualization is one of the most prominent characteristics of cloud computing. The placement of virtual machines (VMs) in the physical machines determines the resource utilization efficiency and service quality. Especially for distributed cloud computing, where the data centers (DCs) span a large number of geographical areas and all DCs are connected by high speed internet, the placement of VMs of one big task or of one organization focuses on minimizing the distances and bandwidths between DCs. This minimizes communication latency and improves availability. A data center cluster should be found firstly to accommodate the requested VMs. The purpose is to minimize the maximum inter-DC distance. In contrast to existing method that only considers the distances between data centers, a more efficient clustering based 2-approximation algorithm is developed by taking full use of the topology and the density property of cloud network. The simulation shows the proposed algorithm is especially appropriate for very large scale problems. Then, the requested VMs should be partitioned to the DC cluster, so that the expensive inter-DC bandwidth is saved and the availability is improved. With the introduction of a half communication model, a novel heuristic algorithm which further cuts down the used bandwidths is presented to partition VMs. Its time complexity is reduced to $O(n^2)$ by a factor of $O(logn)$ and it runs 3 times faster than the existing method.

1. Introduction

Cloud computing has gained great popularity in recent years for the efficient resource usage and convenient service access [1, 2]. These competitive powers are attributable to the introduction of virtual technology and distributed networking of cloud. Based on the actual standard of virtualization industry, the cores of physical machines (PMs) can be virtualized into more virtual CPUs (vCPUs) [3]. Virtual machines (VMs) can be placed on the granularity of vCPUs and thus gain a more efficient resource utilization. It is also hoped VMs can be deployed closer to the end users in different geographical locations by distributed networking. Distributed cloud consists of a lot of data centers (DCs) and all DCs are connected by high speed internet [4]. Contrary to the counterparts of centralized cloud, distributed DCs have relatively small capability because they are planned according to the less traffic of the dispersed area they locate.

* Corresponding author at: School of Computer Science and Technology, Harbin Institute of Technology Shenzhen Graduate School, Shenzhen 518055, China.
E-mail addresses: jiangtaozhang@aliyun.com (J. Zhang), wangxuan@insun.hit.edu.cn (X. Wang), hjhuang@hitsz.edu.cn (H. Huang), honesty_chenshi@163.com (S. Chen).

http://dx.doi.org/10.1016/j.future.2016.06.018
0167-739X/

As more small cloud service providers enter cloud market, DC becomes much smaller [5]. But the distribution scheme has several benefits. The sites of DCs are selected according to the principle of proximity. Services of end users can be completed in the DC nearer to them. The reaction time is shortened and the overall long-distance bandwidth consumption is cut down. This means the service latency is reduced and availability is increased. Moreover, relatively small DC can be easily retreated/added based on the services traffic change in different areas. So distributed cloud is more scalable and more elastic. It has become the mainstream.

The VMs of some applications may span more than one DC for the relatively small capacity of DCs or some availability policy where an upper limit for the VMs in one DC is designated [6]. Most of such applications are geo-distributed in nature and can benefit from the utilization multiple DCs. One example is a big data stream processing system on a cloud for a huge supermarket chain across regions [7]. The processing systems consist of a large number of independent tasks. Because various data, such as information of customers and sales are produced continuously in different regions, the VMs to process the data and tasks are also distributed naturally. The VMs can be deployed in any DCs without violating the forced task semantics. The data stream will consume huge inter-DC bandwidth. Other applications, such as the thousands of virtual desktops (they are also a kind of VMs) for all branches of companies [8,9], processing-intensive tasks where the input of one processor is from another processor in another area [5,10] and logistic information systems [11], also need to span more than one DCs. Distributed cloud is the most eligible candidate.

We should select such a DC cluster firstly to accommodate the requested VMs which dedicate to one application and communicate with each other. In addition to the consideration of capacity matching, the inter-DC distance should be as small as possible to reduce the service latency. The primary objective is to minimize the maximum inter-DC distance. This prevents from the possibility of tasks running in VMs which are very far apart, so as not to delay the overall completion time of the user application [6].

After DC cluster is selected VMs should be partitioned to each DC of the cluster. On one side, the partition should not exceed the upper bound of DC capacity. On the other side, the important resource: network bandwidth between different DCs should be minimized [12]. This is because of two reasons: (1) Economical consideration. The long-distance line between DCs is very expensive. (2) Availability consideration. The more the long-distance line is used, the lower availability is possible. VMs with larger traffic can be agglomerated in one DC so that as many communications are completed inside the DC as possible.

This paper aims to investigate more efficient algorithms for the aforementioned data center selection and VM partition problems. The main contributions are summarized as follows:

1. A novel clustering based algorithm is recommended for data center selection problem. In addition to distance, this algorithm fully takes into consideration other networking information of cloud, such as topology, density integrated with DC capacity and thus improves the efficiency. The execution time is shortened 15%–27% for the random distribution scenario of DCs and 15%–72% for the clustering scenario respectively. Furthermore, it is more applicable to large scale VM placement.
2. With introduction of half communication model of VMs, the overall traffic of a VM can be considered in the decision process. Therefore, a new slightly more effective VM partition algorithm is presented. Its time complexity is reduced to $O(n^2)$ by a factor of $O(\log n)$ and the efficiency is improved about 4 times.
3. The two algorithms are designed on the basis of the actual standard of virtualization industry, i.e., the granularity of vCPUs. They can address both homogeneous and heterogeneous requirements of VMs.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 addresses the DC selection problem by means of a clustering based method and proves it as 2-approximation. Section 4 defines the half communication model of VMs and then presents a new faster algorithm. Section 5 introduces the test environment and evaluates both methods, especially on the algorithm efficiency. Finally the whole paper is wrapped up in Section 6 with some future work.

2. Related work

2.1. VM placement and current virtualization standard

Efficient resource usage lays the foundation for the service level assurance and thus makes the cloud service provider business a success and of maximum profitability. So the resource allocation problem is a key challenge for cloud computing [13,14]. Various aspects of resource allocation are explored, such as server integration [15], load balance [16] and energy [17]. But all these papers mainly concentrate on memory and CPU resources. DC selection and bandwidth saving are seldom studied in past years. The placement of VMs in one DC based on the traffic matrix is investigated in [18,19] and the objective is to improve the scalability of DCs.

Different models are adopted to address the challenge. Bin packing [20,21] and graph theory [18,22,23] are two models widely used and they are selected based on the granularity of resources. The former assumes resource can be split arbitrarily to adapt to the diverse resource requirements of VMs. Statistical multiplexing is often utilized to compact more VMs into one PM. But it is not realistic for the current virtualization techniques. The resource can only be refined to the granularity of vCPUs [24]. Even VMWare, one of the leading virtualization technology corporations, claims that for its latest virtualization product: VMware vSphere 5.x, a virtual machine cannot have more vCPUs than the number of logical cores of the host. The number of logical cores is equal to the number if hyperthreading is disabled or at most twice of that number of the physical cores if hyperthreading is enabled [3]. The number of logical cores is just the most number of VMs that can be hosted in the PMs. In the VM instance types provided by Amazon, vCPU is used as the computing resource metric and it varies from 1 to 32 [25]. On the other side, bin packing based algorithm assumes the items have no relationship and fails to describe the situation when the packed VMs communicate with each other [26]. So we use the graph theory based model. Resource unit "slots" is often integrated with it [18,22,23]. The number of slots can be determined by some existing capacity tools [18]. Herein each slot corresponds to a vCPU. One slot can only be occupied by one VM. But one VM may require more than one slot.

Data center selection problem is firstly explored in [6]. After formulating as minimizing the diameter of a complete vertex weighted graph, the problem is proved as NP-hard. A FindMinStar algorithm is recommended to find a DC cluster around a certain DC and to calculate the corresponding diameter. Then in a 2-approximation algorithm MinDiameterGraph, FindMinStar is invoked for each DC. All the corresponding diameters are calculated and compared straight-forward. The DC cluster with the minimum diameter is selected as the solution. The time complexity of MinDiameterGraph is $O(n^3)$ and it is dominated by FindMinStar algorithm with time complexity $O(n^2)$. For the VM partition problem, a heuristic algorithm with $O(n^2 \log n)$ time complexity is also presented [6]. The simulation shows the method can produce better result than a random and a greedy algorithm.

But the paper assumes the DC capacity is measured by the number of VMs, so the algorithms can only address the homogeneous situation where all VMs require the same amount of resource. Moreover, normally the VM placement is online. It is necessary to explore lighter-weight algorithms.

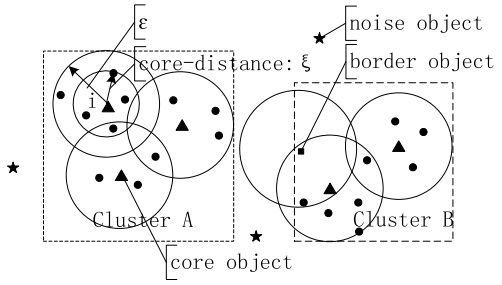


Fig. 1. Density-based clustering concepts where *MinPts* equals 5. Stars denote noise objects, triangles denote core objects and square denotes border objects. Core-distance$_{\epsilon,5}(i) = \xi$.

2.2. Density-based clustering

Density-based clustering method is extensively used in cluster analysis [27–29]. Its key idea is that, given a radius ϵ, the cluster consists of two kinds of objects. The first kind of object is "core object". For each core object i, its ϵ-neighborhood $N_\epsilon(i)$ (the closed disk whose core is i and radius is ϵ) contains at least *MinPts* points, i.e., the numbers of points in the neighborhood exceed a threshold. For other objects in the cluster, their ϵ-neighborhood contains fewer points than *MinPts*. These objects are called "border objects". If the ϵ-neighborhood of an object contains fewer points than *MinPts* and it is not included in any cluster, this object is a "noise". Intuitively, noise object is sparser. The whole point set can be viewed as clusters of points separated by noise points. Utilizing the aforementioned concepts, DBSCAN algorithm [30] can find the corresponding clusters based on the input ϵ and *MinPts*. The related concepts are illustrated in Fig. 1 for easy understanding.

Any change in the input parameters will lead to different clusters and DBSCAN algorithm must run again. With the introduction of core-distance and reachability-distance, OPTICS algorithm [31] is presented on the basis of DBSCAN. OPTICS finds core objects with respect to (w.r.t.) inputs ϵ and *MinPts* firstly. Then all points are ordered in reachability-distance after calculating the corresponding core-distance and reachability-distance. Thus, any cluster whose input radius is shorter than ϵ can be found easily.

3. Clustering based data center selection

3.1. Motivation

Data center selection problem aims to find a DC cluster to accommodate the requested VMs where each VM requires a certain number of vCPUs. The maximum inter-DC distance of the cluster needs to be minimized so that the latency of the application is reduced. This problem is formulated as a diameter minimizing problem of a complete vertices weighted graph $G = (N, E, w)$. The vertices N represent n DCs. The weight vector w of the vertices is the capacity of DCs, i.e., the number of vCPUs can be accommodated. E is the edge length matrix where E_{ij} is the edge length between vertices i and j. The diameter is the longest one of all edges. It is natural to suppose the distance between DCs (e.g., Euclidean distance) follows triangle inequality. The prerequisite is that the capacity of DCs can accommodate the requested VMs, otherwise the capacity needs to be expanded.

Though MinDiameterGraph in [6] is said to be network aware, it only considers the distances between DCs. In reality, the clustering nature of the distributed cloud is an important factor and should not be ignored. The location selection of DC is based mainly on communication traffic and population distribution which are usually both unbalanced. Some DCs are much denser and cluster together. While others are sparser and scatter widely. It is more possible to find a DC cluster in denser DCs in conjunction with the capacity of them, so that the diameter is as small as possible. MinDiameterGraph does not take advantage of this information. It can only use the straight-forward mechanism by comparing all DC clusters, one by one to find the solution. So despite the different densities, for the same problem, its execution time is very high and almost the same for different networks (refer Fig. 4 in Section 5.1 for details). MinDiameterGraph invokes FindMinStar one time for each DC to find a DC cluster and calculate the corresponding diameter by simply comparing the edges one by one. Because there are n^2 edges for n DCs, the diameter calculation is rather time-consuming and the time complexity of FindMinStar is $O(n^2)$.

In addition to the cluster nature of DCs, each DC can accommodate different numbers of vCPUs. After all, the designed capacity of each DC may not be the same. Moreover, resources allocated after delivery contrast greatly. Hence the remaining capacity is different. Integrating with the remaining capacity, if we can find a DC cluster, which can accommodate the requested VMs in a denser area, then some sparser DCs are unnecessary to be checked anymore and will be cut off directly. Further, we do not need to calculate their time-consuming diameter and avoid the straight-forward comparison of them. The efficiency will be greatly improved without deteriorating quality. The basic idea is to find a rather small radius ϵ, there exists at least one DC and whose ϵ-neighborhood contains enough DCs to accommodate the requested VMs. A corresponding diameter of these DCs is calculated. Any DC which is noise w.r.t. this diameter (acts as a new radius) and the number of vCPUs (acts as *MinPts*) required by some VMs is unnecessary to be considered.

3.2. Data center selection

In the notion of density-based clustering, every point is viewed as equivalent, except its varying distance to the core. But in cloud computing, even if a DC can be viewed as a point, it is still not equivalent because its capacity to accommodate VMs is different. So we cannot cluster simply according to the number of DCs.

Here we will introduce some concepts firstly in order to facilitate the later discussion.

We view a DC which can accommodate k vCPUs as a k points set and all inter-point distances are zero. If all DCs contained in the ϵ-neighborhood of a DC can accommodate the total m vCPUs required by some VMs, it means the ϵ-neighborhood of the DC contains m points. In this sense it is coincident with the density-based clustering notions. So we have the following definition:

Definition 1 (*Core Object*)**.** Given ϵ as a distance value, if DCs in the ϵ-neighborhood of a DC can accommodate at least m vCPUs, then this DC is a core object w.r.t. (ϵ, m).

Core-distance is modified accordingly from that in [31] as follows:

Definition 2 (*Core-Distance of an Object i*)**.** Given i as an object from DCs, ϵ as a distance value, m as a natural number. Denote $N_\epsilon(i)$ as the ϵ-neighborhood of i. Let m-*distance*(i) be the distance from i to its neighbor DC who just accommodates the mth vCPU and m is the total number of vCPUs required by some VMs. Let $Card(N_\epsilon(i))$ be the number of vCPUs accommodated by DCs contained in $N_\epsilon(i)$. Then, the core-distance of i is defined as *core-distance*$_{\epsilon,m}(i) =$

$$\begin{cases} UNDEFINED & \text{if } Card(N_\epsilon(i)) < m \\ m\text{-}distance(i) & \text{otherwise.} \end{cases}$$

For core object, core-distance is the smallest radius ξ, from i to its neighbor DC such that i is a core object w.r.t. (ξ, m). There is no definition for non-core object. Please refer Fig. 1 for illustration.

Definition 3 (*Feasible Subgraph*)**.** Given one vertex i, if i and its closest neighbors constitute a subgraph whose vertices (DCs) accommodate at least m vCPUs and the corresponding VMs. Then this subgraph is called a feasible subgraph centers around i.

Definition 4 (*Minimum Feasible Subgraph*)**.** Given one feasible subgraph, if any subgraph of it is not feasible subgraph, this feasible subgraph is called a minimum feasible subgraph.

Denote minimum feasible subgraph centers around i as F_i and the diameter of the graph as $D(F_i)$. For a core object i whose $\textit{core-distance}_{\epsilon,m}(i) = \xi$, F_i is just formed by all objects in $N_\xi(i)$. If radius of F_i is defined as the distance between i and the farthest neighbor in F_i, then the radius is just the same as the core distance.

The following method ModFindMinStar (Algorithm 1) can be used to find a minimum feasible subgraph. ModFindMinStar is modified from FindMinStar algorithm [6] except the diameter calculation process is deleted.

Algorithm 1 ModFindMinStar

Input: $G = (N, E, w)$: a complete graph with vertex, edge length matrix and weight
m: required weight of the subgraph, i: starting vertex
Output: Subgraph $G' = (N', E')$ of weight at least m formed by i and its closest neighbors

1: Let $e = (e_1, e_2, ..., e_{n-1})$ is the i-th column vector of E excluding E_{ii} sorted in non-decreasing order: $e_1 \leq e_2 \leq ... \leq e_{n-1}$. The corresponding vertices of e are $(N_{e_1}, N_{e_2}, ..., N_{e_{n-1}})$ and the weights are $(w_{e_1}, w_{e_2}, ..., w_{e_{n-1}})$
2: $N' \leftarrow i, m' \leftarrow w_i$
3: **for** $j = 1, \ldots, n-1$ **do**
4: **if** $m' < m$ **then**
5: $N' \leftarrow N' \bigcup N_{e_j}, m' \leftarrow m' + w_{e_j}$
6: **else**
7: $E' \leftarrow$ submatrix of E corresponding to N'
8: **return** $G' = (N', E')$
9: **end if**
10: **end for**
11: **if** $m' < m$ **then**
12: No subgraph of size m in G
13: **return** NULL
14: **end if**
15: $E' \leftarrow$ submatrix of E corresponding to N'
16: **return** $G' = (N', E')$

For a given DC, ModFindMinStar orders all other DCs in increasing distance to this DC, then the closest DC is added to the cluster. If the cluster can accommodate all the requested vCPU and the corresponding VMs, the program terminates. Otherwise the second closest DC is added until the cluster meets the VMs number requirement or all DCs have been added. Because of the heterogeneity of VMs, the feasibility to accommodate the concrete VMs combination is necessary. But for a real DC which can supply thousands of vCPUs, it can be negligible.

Proposition 3.1. *Suppose there exist one object i, $D(F_i) = d$ and $\epsilon \geq d$, then for any core object j w.r.t. (ϵ, m) whose core-distance is bigger than d, we have $D(F_j) > d$.*

Proof. Suppose the core-distance of j is ξ. Because $\xi > d$, there must exist at least two objects in $N_\xi(j)$ and they are at least d apart. F_j is just formed by all objects in $N_\xi(j)$. So diameter $D(F_j)$, the maximum edge length of F_j, is bigger than d.

Proposition 3.1 enlightens us, if there is one feasible subgraph and its diameter is d, then we only need to judge whether other objects are core objects w.r.t. (d, m). All noise objects will be cut off directly because the minimum feasible subgraphs that center around these objects cannot produce a more optimal solution. The smaller the d is, the more points will be cut off and hence improves the efficiency.

A new algorithm ClusteringBasedMinDiameter (CBMinDia, Algorithm 2) is presented to find a DC cluster with minimum diameter. CBMinDia includes three phases: phase 1 (line 1) selects an arbitrary vertex and uses the radius of its minimum feasible subgraph (all the minimal feasible subgraphs in this algorithm are found by ModFindMinStar) as the initial radius ϵ. Phase 2 (lines 2–9) judges whether each vertex is core object w.r.t. (ϵ, m) and then labels. Lines 10–25 are phase 3. In this phase vertices which cannot be more optimal are cut off by a criterion and the vertex with the smallest diameter is found. The criterion is d, i.e., the diameter of the minimum feasible subgraph with the smallest core distance. This feasible subgraph is a feasible solution and is used as the initial value. We hope d is rather small, so that more vertices can be excluded from it. Based on Proposition 3.1, for core objects w.r.t. (ϵ, m), only the one with core distance smaller than d is checked. The solution is updated when a smaller diameter is found. The process is reflected in lines 15–18. For non-core objects w.r.t. (ϵ, m), it is checked only when the initial radius ϵ is smaller than d and its neighborhood contains at least m objects (line 20), because a non-core object w.r.t. a bigger ϵ than d cannot be more optimal. Furthermore, the neighborhood should contain enough objects. Otherwise it is not a feasible solution.

Algorithm 2 ClusteringBasedMinDiameter (CBMinDia)

Input: $G = (N, E, w)$: a complete graph with vertex, edge length matrix and weight
m: required weight of the subgraph
Output: Subgraph $G' = (N', E')$ of weight at least m with minimum diameter d

1: Select any vertex k from N, initial radius $\epsilon \leftarrow$ radius of F_k
2: **for** $i = 1, \ldots, n$ **do**
3: **if** $Card(N_\epsilon(i)) \geq m$ **then**
4: Label i as core object
5: Sort points in $N_\epsilon(i)$ in increasing distance to i and calculate $\textit{core-distance}_{\epsilon,m}(i)$
6: **else**
7: Label i as non-core object
8: **end if**
9: **end for**
10: Let k is the vertex with the minimum core-distance.
11: $d \leftarrow D(F_k)$, feasible subgraph center: $\textit{center} \leftarrow k$
12: $N' \leftarrow$ vertices of F_k, $E' \leftarrow$ edge matrix of F_k
13: **for** $j = 1, \ldots, n$ **do**
14: **if** j is labeled as core object **then**
15: **if** $(\textit{core-dist}_{\epsilon,m}(j) \leq d)$ and $(D(F_j) \leq d)$ **then**
16: $d \leftarrow D(F_j)$, $\textit{center} \leftarrow j$
17: $N' \leftarrow$ vertices of F_{center}, $E' \leftarrow$ edge matrix of F_{center}
18: **end if**
19: **else**
20: **if** $(\epsilon < d)$ and $(Card(N_d(j)) \geq m)$ and $(D(F_j) \leq d)$ **then**
21: $d \leftarrow D(F_j)$, $\textit{center} \leftarrow j$
22: $N' \leftarrow$ vertices of F_{center}, $E' \leftarrow$ edge matrix of F_{center}
23: **end if**
24: **end if**
25: **end for**
26: **return** $G' = (N', E')$ and d

3.3. Complexity analysis and effectiveness proof

Phase 1 finds a feasible subgraph after sorting all vertices and takes $O(n \log n)$ time. In phase 2, core-distance is calculated by sorting after vertices within the neighborhood are found. The execution time is $O(n \log n)$. This process is traversed for each vertex and takes $O(n^2 \log n)$. Plus the time $O(n)$ to get the minimum core-distance, the worst execution time is still $O(n^2 \log n)$. Phase 3 calculates diameters for the non-excluded vertices and each vertex needs to compare the diameters at most $O(n^2)$ times. There are total $O(n)$ vertices giving the worst execution time $O(n^3)$.

The total time complexity is dominated by phase 3. Hence the worst case complexity is $O(n^3)$. Because many objects are cut off and their most time-consuming diameter computing is saved, the execution time of CBMinDia is much shorter than that of MinDiameterGraph [6]. This is demonstrated in Section 5.1.

Theorem 3.2. *Algorithm CBMinDia finds a 2-approximation solution when VMs are homogeneous.*

Proof. Suppose the longest edge in the optimum subgraph is AB and its length is l. After running the algorithm we affirm A and B will not be cut off by taking d as the criterion. This is because d is the diameter of a feasible subgraph, l is the optimum solution and hence the minimum. This gives $l \leq d$. But only objects with diameter bigger than d is cut off in the program.

Now we take A as an example to prove the approximation property. If A is a core object, suppose its core-distance computed by the algorithm is l'. Then l' is the smallest distance, from this object to its neighbor DC who just accommodates the mth vCPU and the corresponding VMs. So we have $l' < l$. The diameter of the feasible subgraph centers around A is at most $2l'$ due to triangle inequality, hence is not bigger than $2l$.

If A is not a core object, suppose the radius of the minimum feasible subgraph centers around A is l'. This distance is the shortest one if objects centers around A can accommodate m vCPUs and the corresponding VMs. This gives $l' \leq l$. Similarly, the diameter of the feasible subgraph centers around A $< 2l' \leq 2l$.

Actually, the result of CBMinDia is just the same as that of algorithm MinDiameterGraph. This is because the new algorithm only cuts off the objects less optimal than a criterion. Because this criterion is a feasible solution, the operation will not deteriorate the optimum.

4. Virtual machines partition to selected data centers

Virtual machines partition problem aims to assign the requested VMs to the corresponding DCs so that the total inter-DC bandwidth used is as small as possible. This is because the inter-DC line is usually long-distance link and hence very expensive. The saving of long-distance bandwidth saves capital expenditures. Reliability is another consideration because it deteriorates as the distance becomes longer.

The traffic between n VMs can be denoted by a symmetric $n * n$ traffic matrix T [18], where T_{ij} is the traffic between VMs i and j and $T_{ii} = 0$. For a selected DC cluster $(D_1, D_2, \ldots, Dr)$, each $D_k (k = 1, 2, \ldots, r)$ can accommodate s_k vCPUs. All VMs are partitioned to disjoint sets: $P_1, P_2, \ldots, P_r$. It is required that the assigned VMs in each partition component P_k commensurate with the capacity of the corresponding D_k. So the objective of VM partition problem is to minimize $\sum_{k=1}^{r} \sum_{i,j=1, i \in P_k, j \notin P_k}^{n} T_{ij}$. It subjects to $|P_k| \leq s_k, k = 1, 2, \ldots, r$. Here $|P_k|$ is the number of vCPUs in component P_k.

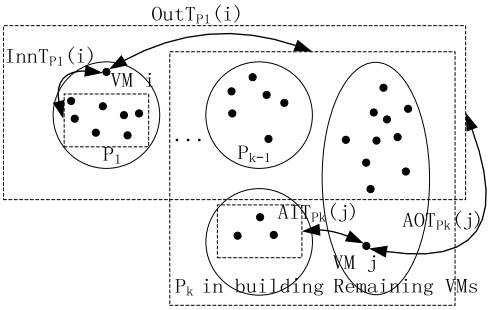


Fig. 2. Four traffic definitions of Half Communication Model (HCM) for different partition components $P_1, \ldots, P_k$. Arrows between VM i (VM j) and the dotted squares denote traffic between this VM and VMs included in the squares.

4.1. Half communication model (HCM) of VMs

For a given partition component P_k whose all members have been determined, i.e., all VMs belong to the component have been found and the component is commensurate with the capacity of a DC. We have definitions:

Definition 5 (*Inner Traffic (InnT) and Outer Traffic (OutT)*)**.** Given the requested VMs set as V, the partition component as $P_k \subseteq V$ and traffic matrix as T. $\forall i \in P_k$, inner traffic of i w.r.t. P_k is defined as the total communication traffic between i and other members in the component. Formally: $InnT_{P_k}(i) = \sum_{j \in P_k, 1 \neq j} T_{ij}$.

Similarly, outer traffic of i w.r.t. P_k is defined as the total communication traffic between i and the non-members of the component. Formally: $OutT_{P_k}(i) = \sum_{j \notin P_k} T_{ij}$.

Given a partition, *InnT* and *OutT* of any VM in the partition are determined values. Ideally, we hope any VM in each partition component has the largest *InnT* and the smallest *OutT*. However the member is found one by one in the building of the objective partition component. We cannot know all the members in advance and so do *InnT* and *OutT*. What we can know are the following two kinds of approximate traffic for a candidate VM:

Definition 6 (*AIT and AOT*)**.** For the objective partition component P_k, $\forall$ candidate $i \notin P_k$, approximate inner traffic: $AIT_{P_k}(i) = \sum_{j \in P_k} T_{ij}$, and approximate outer traffic: $AOT_{P_k}(i) = \sum_{j \notin P_k, 1 \neq j} T_{ij}$.

AIT and AOT are traffic in the process of building the partition. They can be calculated and viewed as the approximation of InnT and OutT, so as to illustrate which VM should be selected and added to the partition in priority.

The sum of InnT and OutT equals the sum of AIT and AOT once a candidate VM becomes a member of a partition component. It is just the overall traffic between this VM and other VMs. The aforementioned four definitions reveal the traffic of one VM from the inner part and the outer one of a partition component. Thus making it possible to explore the traffic of a VM from a point view of the component. They are summarized as half communication model (HCM) and illustrated in Fig. 2.

We have the following observations:

Proposition 4.1. *For any candidate VM i and partition component P_k, considering P_k in building and after P_k is determined, we have,*

1. $InnT_{P_k}(i) \geq AIT_{P_k}(i)$, $OutT_{P_k}(i) \leq AOT_{P_k}(i)$.
2. *AOT of a previously added member dominates the overall traffic between this member and the later added ones. Moreover the overall AOT of the previously added members dominates the AIT of the later added one. Formally, $\forall i, j$, j is added after i to P_k, $AOT_{P_k}(i) \geq \sum_j T_{ij}$. Moreover, $\sum_{i \in P_k} AOT_{P_k}(i) \geq AIT_{P_k}(j)$.*

3. *AIT and AOT of a later added member approximates more to its InnT and OutT than the previously added member does. Formally, $\forall i, j$, j is added after i to P_k, $InnT_{P_k}(i) - AIT_{P_k}(i) \geq InnT_{P_k}(j) - AIT_{P_k}(j)$, $AOT_{P_k}(i) - OutT_{P_k}(i) \geq AOT_{P_k}(j) - OutT_{P_k}(j)$.*
4. *$InnT_{P_k}(i) = AIT_{P_k}(i)$, $OutT_{P_k}(i) = AOT_{P_k}(i)$ if i is the last added member of P_k.*

Though we want to turn larger inter-VM traffic into intra-DC traffic, so that the inter-DC traffic becomes as small as possible. It cannot ensure the VM added to the partition component is the best one or has the largest InnT and smallest OutT. Intuitively, a candidate should have larger AIT and smaller AOT as much as possible in the building process of the partition component. However, based on item 2 of Proposition 4.1, the AOT of the previously added member limits the traffic of the later candidates, hence the selection of them. If the AOT of a previous member is too small, the traffic of this VM and the later added ones cannot be too large. Furthermore, the total AOT of all the formerly added members dominates AIT of the later ones added. A too small AOT will decrease the overall intra-DC traffic. We need an "appropriate" small AOT for the previously added VM so that VMs with much larger AIT can be selected in the building process. Item 3 of Proposition 4.1 indicates AOT has less deviation from OutT in the closing stage of building. So we require AOT becomes smaller as the process draws to end. For the last candidate, we select the smallest AOT based on item 4 of Proposition 4.1.

This idea is implemented by a Weighted Combination of AIT and AOT: For candidate VM i, $WCT(i) =$

$$\begin{cases} AOT_{P_k}(i) & \lambda = 0 \\ (1/\lambda) * AIT_{P_k}(i) + (1 - 2\lambda) * AOT_{P_k}(i) & \lambda \in (0, 1] \end{cases}$$

where $\lambda = \sum_{i,j \in P_k} T_{ij} / (s_k / average|C_k| * (\max_{i,j \in C_k} T_{ij} + \min_{i,j \in C_k} T_{ij}))$, where s_k is the required size of P_k, C_k is the candidate VMs set for P_k. For example, C_1 is all VMs set: V, C_2 is the remaining VMs: $V \setminus P_1$ and so on. $|C_k|$ is the number of vCPUs of VMs in C_k. So the numerator is the current traffic between VMs partitioned to P_k. The denominator is the estimated total traffic between VMs in P_k supposing P_k is determined.

For each component, we will select the VM with the maximum WCT from the candidates in sequence. The change of λ indicates the progress of the partition component building. We hope λ can facilitate the selection of a VM with a maximum AOT and an appropriate AIT in each step. $\lambda = 0$ implies i is the first member to be added. This ensures the first selected VM has the largest AOT, i.e., the largest traffic with all the other VMs. $\lambda \to 1$ as the process progresses. VM with the largest AIT and the "appropriate small" AOT is selected. At the same time, this VM has a rather large overall traffic. This will not prevent the selection of the VM that has a large traffic with the formerly added ones based on item 2 of HCM. As the process progresses, λ approaches 1, WCT approaches $AIT_{P_k}(i) - \lambda * AOT_{P_k}(i)$. This ensures VM with smaller AOT is selected and it approximates the expected OutT according to item 3 of HCM. $\lambda = 1$ if i is the last member to be added. VM with the smallest AOT is selected for the reason of item 4 of Proposition 4.1. WCT considers the overall traffic of a VM to make the decision. It is different from [6] which only considers the traffic between this VM and the selected ones. Thus it achieves a better quality.

4.2. Virtual machines partition

Based on HCM property we give the following scheme to partition VMs more efficiently.

Algorithm 3 HCMPartition

Input: V: VMs to be partitioned
T: $n * n$ traffic matrix of n VMs
$s_1, s_2, ..., s_r$: vCPU number of required partition components
Output: A partition with components $P_1, P_2, ..., P_r$ such that $|P_k| \leq s_k (k = 1, \ldots, r)$

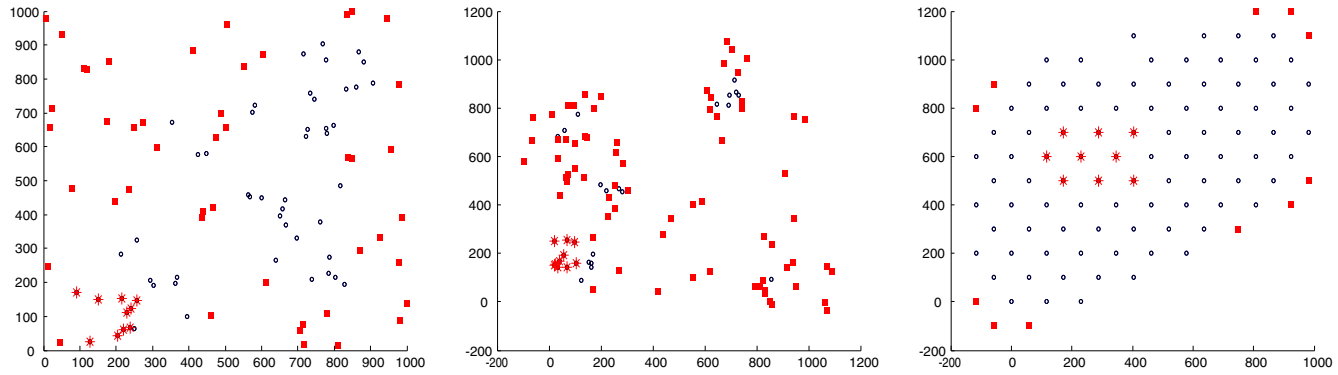
Let $s_1, s_2, ..., s_r$ be in non-increasing order
$V' \leftarrow \emptyset$
VMs total traffic vector: $VT = (VT(1), VT(2), ..., VT(n))$ where $VT(j) = \sum_{i=1}^{n} T_{ij}$
Candidate VMs total traffic vector: $CVT \leftarrow VT$
for $k = 1, \ldots, r$ **do**
  $P_k \leftarrow HCMFindCluster(T, VT, CVT, V, V', s_k)$
  $V \leftarrow V \setminus P_k$
  $V' \leftarrow V' \cup P_k$
  $CVT \leftarrow CVT \setminus CVT(j), j \in P_k$
end for
return $P_1, P_2, ..., P_r$

Algorithm 4 HCMFindCluster

Input: T: $n * n$ traffic matrix of n VMs
VT: VMs total traffic vector
CVT: candidate VMs total traffic vector
V: candidate VMs, V': partitioned VMs
s: required number of vCPUs of the partition component.
Output: Partition component P

$P \leftarrow \emptyset$, $AOT \leftarrow CVT$, $currentTraff \leftarrow 0$, $x \leftarrow 0$(Equals 1 if all vCPUs required by the VM can be accommodated by the DC, and 0, otherwise)
if $|V| > s$ **then**
  while $s > |P|$ **do**
    if $|P| = 0$ **then**
      $estimatedTraff \leftarrow (s/average|V| * (max_{i,j \in V} T_{ij} + min_{i,j \in V} T_{ij}))$
      $u \leftarrow$ VM with maximum AOT (This is the first member and $\lambda = 0$, so $AOT = WCT$)
      if $|u| + |P| \leq s$ **then**
        $x \leftarrow 1$
      end if
    else
      $currentTraff \leftarrow currentTraff + \sum_{i \in P} T_{iu}$
      $\lambda \leftarrow currentTraff / estimatedTraff$
      $AOT' \leftarrow VT - T_u$ (The u-th row of T)
      $AOT \leftarrow AOT' \setminus AOT(j), j \in V'$
      $AIT \leftarrow CVT - AOT$
      $WCT \leftarrow (1/\lambda) * AIT + (1 - 2\lambda) * AOT$
      $u \leftarrow$ VM i with $max(WCT(i))$
      if $|u| + |P| \leq s$ **then**
        $x \leftarrow 1$
      end if
    end if
    if $x == 1$ **then**
      $P \leftarrow P \cup u$, $V' \leftarrow V' \cup u$
    else
      Find other candidate VM which can be accommodated. Update P and V'
    end if
  end while
else
  $P \leftarrow V$
end if
return P

Algorithms HCMPartition and HCMFindCluster present a heuristic scheme. In HCMPartition, firstly, the objective partition



(a) Uniformly random distributed DCs. (b) Clustering distributed DCs. (c) Equivalent-distance distributed DCs.

Fig. 3. Three simulated distribution scenarios of distributed DCs. Amplified stars denote solution found by algorithm CBMinDia and the solid squares denote DCs cut off by the algorithm.

Table 1
Number of vCPUs and proportion for VMs instance.

Number of vCPUs	1	2	4	8	16	32
Proportion (%)	15	30	30	20	4	1

components are sorted in non-increasing order of their size. Then candidates VMs are selected by HCMFindCluster for each partition in sequence until all components are determined. In the building process of each component in HCMFindCluster, VM with maximum WCT is selected one by one from the candidates and added to the component until this component is determined.

The while loop is executed one time for the selection of each VM. In the selection process of each VM, *AOT* and *AIT* of each candidate are calculated only once simply by the traffic matrix vector addition and subtraction. There are at most n candidates. So the worst time complexity is $O(n^2)$, which is smaller than the complexity of algorithm Partition: $O(n^2 \log n)$ [6]. The efficiency is improved greatly and a slightly better result is obtained as illustrated in Section 5.2.

5. Experiments

5.1. Clustering based data centers selection

In order to evaluate the effectiveness and time complexity, we use the algorithms in [6] as the baseline. Because the baseline can only address the homogeneous scenario, all the algorithms are adapted to address the heterogeneous VMs. Except that the traffic between VMs follows Zipf distribution [32] instead of uniform distribution to capture the long tail characteristics [18], and DC capacity is measured in number of vCPUs rather than VMs, the test method is almost the same as that in [6].

As illustrated in Table 1, these VMs are randomly selected from Amazon instance with certain proportions based on the 80/20 principle. Each instance requires a different number of vCPUs [25].

The distance between DCs is E-distance and the capacity (numbers of vCPUs can be accommodated) of each DC follows $U(200-450)$. DC is selected from a $1000*1000$ grid in x–y plane by uniform random distribution. In addition, in this paper, two other different topologies of DCs are added to verify the efficiency in different scenarios. One scenario is that DCs have certain clustering property. 80% DCs follow a normal distribution and 20% DCs are selected uniformly randomly from the grids. Another is an ideal scenario. DC locates just on the vertices and centers of congruent planar hexagons. All distances between DCs are equivalent. It will be demonstrated that algorithm CBMinDia can obtain a rather good result, even in the ideal equivalent-distance environment, though

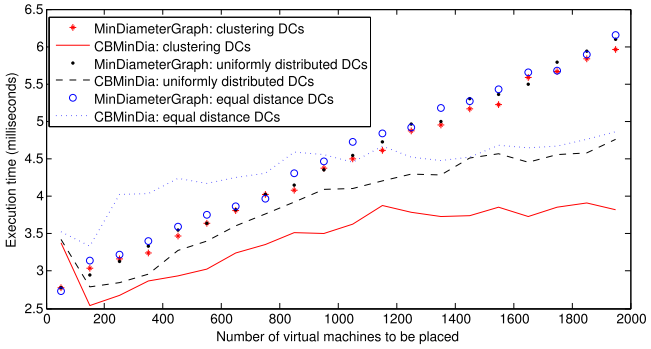


Fig. 4. Comparing algorithms for three scenarios when selecting data cluster to accommodate different number of VMs (increases from 50 to 2000) in 800 DCs.

the algorithm is designed based on the ideas of clustering method. The aforementioned three topologies are illustrated in Fig. 3. There are 100 DCs in the network and the requested number of VMs is 800.

Because DCs are randomly distributed except for the ideal situation and the capacity of each DC also follows random distribution, we produce DCs and the capacity randomly for each run. The test result is the mean value of 100 executions. We simulate in Matlab on a personal computer (Think Centre M4350t, Intel(R) Core(TM) i3-2120 CPU @ 3.30 GHz, 4G RAM).

Fig. 4 shows the differences and variations of execution time of algorithm MinDiameterGraph and CBMinDia when the requested number of VMs to be placed in 100 DCs increases. The execution time of MinDiameterGraph is almost the same in all three scenarios and reveals its weak awareness of the cloud network. It does not utilize fully the density property and capability information of DCs. On the other side, it grows linearly with the increment of VMs in all scenarios. For the algorithm CBMinDia, the time consumed grows more slowly compared to the corresponding scenario of MinDiameterGraph. CBMinDia outperforms MinDiameterGraph distinctly for the clustering distribution. Efficiency increases 15%–72%. Though for random distribution it takes more time. The cost is considerably lower than MinDiameterGraph. Efficiency increases 15%–27%. Even in the ideal equivalent-distance environment, the new algorithm runs faster than the old one when the requested VMs exceed 1000.

One point that needs explanation is that the CBMinDia runs slower than MinDiameterGraph when the number of VMs is smaller than about 75 due to the check of core objects and the calculation of core-distance. If CBMinDia can cut off enough points to compensate for the payment of core object checking and core-distance calculating, the efficiency will improve obviously.

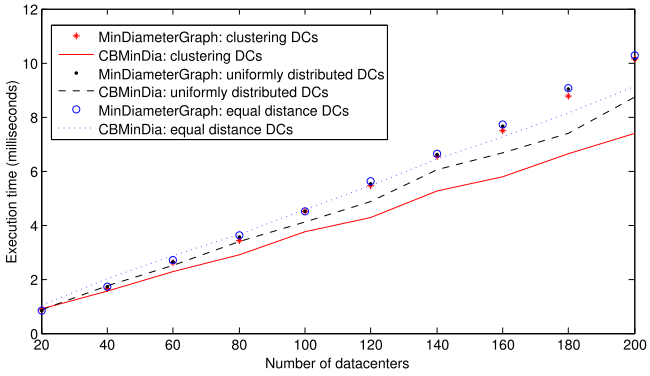


Fig. 5. Comparing algorithms for three scenarios when selecting data cluster to accommodate 1000 VMs in different number of DCs (increases from 20 to 200).

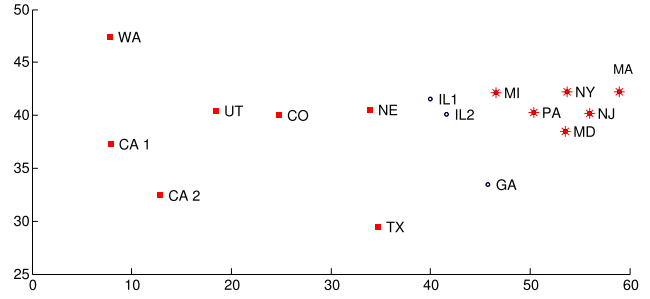


Fig. 6. NSFNET T3 network with 16 datacenters. Amplified stars denote solution found by algorithm CBMinDia when 800 VMa are placed and the solid squares denote DCs cut off by the algorithm.

The phenomenon can be complemented by Fig. 3. Obviously the points cut are the most for clustering distribution, least for equivalent-distance scenario and median for uniform random distribution. So algorithm CBMinDia is the most efficient for clustering scenario, then uniform random distribution, and worst for equivalent-distance scenario. We can discover the solutions denoted by amplified stars do not always lie in the densest DCs. The reason is that the different capacity of each DC makes its contribution to the optimum solution different. This also explains why some points can be excluded by the algorithm even for the ideal equivalent-distance scenario in Fig. 3(c).

Fig. 5 demonstrates the execution time difference and variation with increasing of DCs when there are 1000 VMs to be deployed. Similarly, CBMinDia is more efficient than MinDiameterGraph for uniform distribution and performs most efficiently for clustering conditions. Even for the ideal equivalent-distance scenario, it begins to outperform MinDiameterGraph when VMs exceed 100.

The combination of Figs. 4 and 5 illustrates CBMinDia has more advantages than MinDiameterGraph with the growth of DCs and requested VMs. The execution time grows much more slowly. So algorithm CBMinDia is more appropriate for large scale DC selection problem.

In addition to the simulated DC topology, a realistic DC topology is utilized to verify the performance of the proposed algorithm. Since the locations of DCs of the public could service providers are secret and are not available, US NSFNET T3 network [33,34] is considered here. NSFNET T3 has 16 DCs across US and is projected onto an x–y plane as illustrated in Fig. 6. Suppose that all the DCs are interconnected and the capacity of each DC is a little bigger than the simulated one. The numbers of vCPUs that can be accommodated by each DC follow $U(400 - 900)$. The distance between DCs is estimated by the latitude and longitude of the city where it locates in. Fig. 7 compares the execution time of the algorithms when different number of VMs are placed.

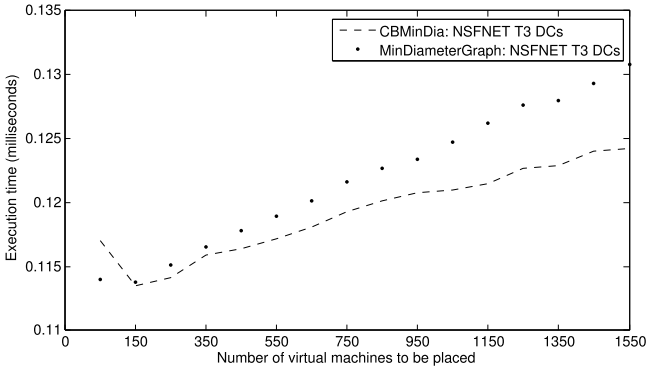


Fig. 7. Comparing algorithms when selecting data cluster to accommodate different number of VMs (increases from 50 to 1500) in 16 DCs of NSFNET T3 network. .

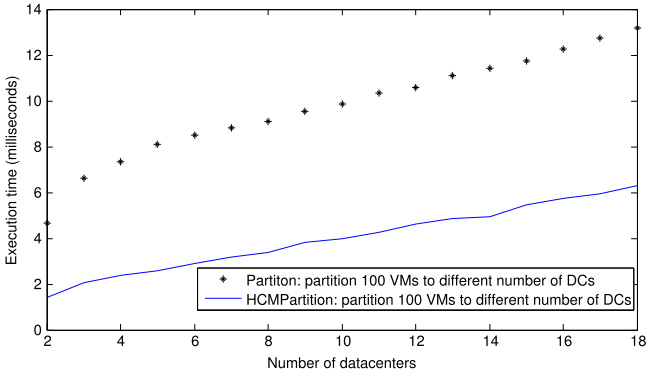


Fig. 8. Comparing algorithms when partitioning VMs to different number of DCs.

It demonstrates a similar phenomenon to that of the simulated scenario in Fig. 4: CBMinDia is faster than MinDiameterGraph. Because there are only 16 DCs the execution time is much shorter than the time when there are 800 DCs in Fig. 4.

5.2. Virtual machines partition

We will assign 100 VMs to a certain number of DCs. The traffic between VMs follows a Zipf distribution of 0–1 Mbps. The number of vCPUs supported by each DC is a quotient where the denominator is the total number of vCPUs required and the numerator is the number of DCs. So each DC accommodates more VMs when there are fewer DCs and inversely, accommodates fewer VMs.

Algorithm Partition has already shown that it outperformed a random and a greedy algorithm in partition quality and its time complexity is $O(n^2 \log n)$ [6]. Here we only compare our new algorithm with it. When the DC number grows from 2 to 18, Fig. 8 plots the efficiency improvement of 100 VMs partition. With the introduction of HCM, the efficiency of partition is improved about 3 times compared to Partition.

Fig. 9 demonstrates that HCMPartition produces slightly better quality at a considerably shorter execution time.

6. Conclusions and future work

By means of the notions of clustering methods, this paper presents a more efficient algorithm, CBMinDia, for the DC selection problem. CBMinDia keeps the 2-approximation property and is more appropriate for large scale DCs or requested VMs. Because the algorithm takes full use of the density and DC capacity information of the network, it cuts off the sub-optimum DCs compared to a rather good feasible solution. The computing effort is greatly decreased and the simulation reveals that it is the most efficient for clustering DC distribution.

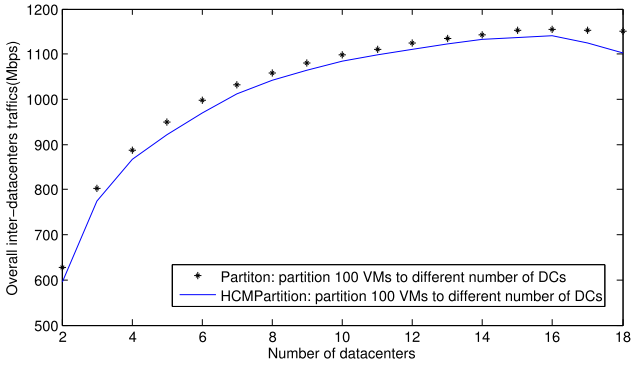


Fig. 9. Overall Inter-DCs traffic comparing of two algorithms.

For VM partition problem, a slightly more effective algorithm is investigated with the introduction of HCM concept. This algorithm determines an appropriate pair of AOT and AIT for each selection of a VM. The value of AOT and AIT permits maximizing the intra-DC traffic and minimizing the inter-DC traffic. More importantly, the concept can facilitate the convenient selection of VMs by means of simple vector addition and subtraction calculation of the traffic matrix. Hence the time complexity is reduced to $O(n^2)$ by a factor of $O(\log n)$ and the efficiency is improved about 3 times.

All algorithms consider the actual standard of virtualization industry and are designed on the granularity of vCPUs. They can address both homogeneous and heterogeneous requirements of VMs.

Our future work is a natural extension of the DC selection problem. We aim to address the DC selection problem for a large company which has multiple geo-distributed branches in the distributed clouds. We need a distributed mechanism which can offer services to users in closer data centers with minimum diameter. This, not only shortens the reaction time of a user request, but also minimizes the consumption of the network bandwidth. However, the closer are the data centers to the branches, the farther are the distances between the data centers, hence enlarge the communication distances between branches. On the other side, the limited data centers will be competed by more than one branch. How to select the appropriate data centers for the multiple branches while balancing the aforementioned contradictions is a great challenge.

Acknowledgments

This work was financially supported by National High Technology Research and Development Program of China (No. 2015AA016008), National Science and Technology Major Project (No. JC201104210032A), National Natural Science Foundation of China (Nos. 11371004, 61402136), Natural Science Foundation of Guangdong Province, China (No. 2014A030313697), International Exchange and Cooperation Foundation of Shenzhen City, China (No. GJHZ20140422173959303), Shenzhen Strategic Emerging Industries Program (No. ZDSY20120613125016389), Shenzhen Overseas High Level Talent Innovation and Entrepreneurship Special Funds (No. KQCX20150326141251370), Shenzhen Applied Technology Engineering Laboratory for Internet Multimedia Application of Shenzhen Development and Reform Commission (No. [2012]720), Public Service Platform of Mobile Internet Application Security Industry of Shenzhen Development and Reform Commission (No. [2012]900).

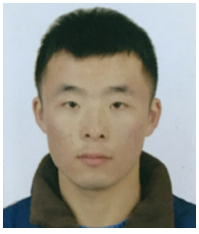
References

[1] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud computing and emerging {IT} platforms: Vision, hype, and reality for delivering computing as the 5th utility, Future Gener. Comput. Syst. 25 (6) (2009) 599–616.
[2] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia, A view of cloud computing, Commun. ACM 53 (4) (2010) 50–58.
[3] VMware, vcpu. URL http://pubs.vmware.com/vsphere-50/index.jsp#com.vmware.vsphere.vm_admin.doc_50/GUID-13AD347E-3B77-4A67-B3F4-4AC2230E4509.html.
[4] S. Alliance, Telecom grade cloud computing v1.0. URL http://scope-alliance.org/sites/default/files/documents/CloudComputing_Scope_1.0.pdf.
[5] E. Ahvar, S. Ahvar, N. Crespi, J. Garcia-Alfaro, Nacer: A network-aware cost-efficient resource allocation method for processing-intensive tasks in distributed clouds, in: IEEE International Symposium on Network Computing and Applications, 2015, pp. 90–97.
[6] M. Alicherry, T. Lakshman, Network aware resource allocation in distributed clouds, in: INFOCOM, 2012 Proceedings IEEE, IEEE, Orlando, USA, 2012, pp. 963–971.
[7] D. Zeng, L. Gu, S. Guo, A general communication cost optimization framework for big data stream processing in geo-distributed data centers, IEEE Trans. Comput. 65 (1) (2016) 19–29.
[8] Huawei, Huawei virtual desktops. URL http://e.huawei.com/hk/solutions/technical/cloud-computing/desktop-cloud.
[9] J. Zhang, L. Zhang, H. Huang, X. Wang, C. Gu, Z. He, A unified algorithm for virtual desktops placement in distributed cloud computing, Math. Probl. Eng. 2016 (1) (2016) 1–15.
[10] D. Irwin, P. Shenoy, E. Cecchet, M. Zink, Resource management in data-intensive clouds: Opportunities and challenges, in: Local & Metropolitan Area Networks IEEE Workshop on Vol. 12, 2010, pp. 1–6.
[11] S. Dubey, S. Jain, A new algorithm for improving latency in distributed data center for logistics information system over cloud, Int. J. Comput. Appl. 130 (2015) 16–20.
[12] P. Mell, T. Grance, The nist definition of cloud computing (draft), NIST Spec. Publ. 800 (2011) 145.
[13] P.T. Endo, A.V. de Almeida Palhares, N.N. Pereira, G.E. Goncalves, D. Sadok, J. Kelner, B. Melander, J. Mangs, Resource allocation for distributed cloud: Concepts and research challenges, IEEE Netw. 25 (4) (2011) 42–46.
[14] J. Zhang, H. Huang, X. Wang, Resource provision algorithms in cloud computing: A survey, J. Netw. Comput. Appl. 64 (2016) 23–42.
[15] Z. Xiao, W. Song, Q. Chen, Dynamic resource allocation using virtual machines for cloud computing environment, IEEE Trans. Parallel Distrib. Syst. 24 (6) (2013) 1107–1117.
[16] Y. Guo, A.L. Stolyar, A. Walid, Shadow-routing based dynamic algorithms for virtual machine placement in a network cloud, in: INFOCOM, 2013 Proceedings IEEE, IEEE, Turin, Italy, 2013, pp. 620–628.
[17] A. Beloglazov, J. Abawajy, R. Buyya, Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing, Future Gener. Comput. Syst. 28 (5) (2012) 755–768.
[18] X. Meng, V. Pappas, L. Zhang, Improving the scalability of data center networks with traffic-aware virtual machine placement, in: INFOCOM, 2010 Proceedings IEEE, IEEE, San Diego, USA, 2010, pp. 1–9.
[19] J. Zhang, Z. He, H. Huang, X. Wang, C. Gu, L. Zhang, Sla aware cost efficient virtual machines placement in cloud computing, in: Performance Computing and Communications Conference (IPCCC), 2014 IEEE International, IEEE, 2014, pp. 1–8.
[20] I. Hwang, M. Pedram, Hierarchical virtual machine consolidation in a cloud computing system, in: 2013 IEEE Sixth International Conference on Cloud Computing (CLOUD), IEEE, Santa Clara Marriott, USA, 2013, pp. 196–203.
[21] W. Song, H. Luo, Z. Xiao, Q. Chen, Adaptive resource provisioning for the cloud using online bin packing, IEEE Trans. Comput. (2013) 1.
[22] T. Verbelen, T. Stevens, F. De Turck, B. Dhoedt, Graph partitioning algorithms for optimizing software deployment in mobile cloud computing, Future Gener. Comput. Syst. 29 (2) (2013) 451–459.
[23] X. Li, J. Wu, S. Tang, S. Lu, Let's stay together: Towards traffic aware virtual machine placement in data centers, in: INFOCOM, 2014 Proceedings IEEE, IEEE, Toronto, Canada, 2014, pp. 1842–1850.
[24] A. Corradi, M. Fanelli, L. Foschini, Vm consolidation: A real case based on openstack cloud, Future Gener. Comput. Syst. 32 (2014) 118–127.
[25] Amazon, Amazonvminstance. URL http://aws.amazon.com/ec2/instance-types/.
[26] E.C. man Jr., M. Garey, D. Johnson, Approximation algorithms for bin packing: A survey, in: Approximation Algorithms for NP-Hard Problems, 1996, pp. 46–93.
[27] A.K. Jain, M.N. Murty, P.J. Flynn, Data clustering: A review, ACM Comput. Surv. 31 (3) (1999) 264–323.
[28] R. Xu, D. C. Wunsch Ii, IEEE, Survey of clustering algorithms, IEEE Trans. Neural Netw. 16 (3) (2005) 645–678.
[29] J. Han, M. Kamber, J. Pei, Data Mining: Concepts and Techniques, Morgan kaufmann, San Francisco, 2006.
[30] M. Ester, H.P. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, in: Proceedings of the 2nd International Conference Knowledge Discovery and Data Mining, AAAI, Portland, USA, 1996.
[31] M. Ankerst, M.M. Breunig, H.-P. Kriegel, J. Sander, Optics: Ordering points to identify the clustering structure, SIGMOD Rec. 28 (2) (1999) 49–60.
[32] W.J. Reed, The pareto, zipf and other power laws, Econom. Lett. 74 (1) (2001) 15–19.
[33] NSFNET, Nsfnet t3 network. URL https://en.wikipedia.org/wiki/National_Science_Foundation_Network.
[34] B. Chinoy, H.W. Braun, The national science foundation network, Technical Report GA-A21029, SDSC, 1992.

Jiangtao Zhang received the M.S. degree in applied mathematics from Xidian University, China, in 1999. Then as a senior engineer in Huawei, he engaged in research and development of mobile communication networks and cloud computing. He was also in charge of communication network planning and optimization techniques research. In 2012, he became a Ph.D. candidate. His research interests lie in the fields of mathematical programming, cloud computing and distributed computing, especially architecture, protocols and algorithms.

Xuan Wang received M.S. and Ph.D. degrees in Computer Sciences from the Harbin Institute of Technology, Harbin, China, in 1994 and 1997, respectively. He is a Professor and Dean of the school of Computer Science and Technology in Harbin Institute of Technology, Shenzhen Graduate School, ShenZhen, China. His research interests include Artificial Intelligence, Computer Network Security, Computational Linguistics, and Computer Vision.

Hejiao Huang graduated from the City University of Hong Kong and received the Ph.D. degree in computer science in 2004. She is currently a professor in Harbin Institute of Technology Shenzhen Graduate School, China, and previously was an invited professor at INRIA, France. Her research interests include cloud computing, trustworthy computing, formal methods for system design and wireless networks.



Shi Chen has been pursuing the M.S. degree of computer science and technology in Harbin Institute of Science and Technology, Shenzhen graduate school since 2014. His research interests lies in the fields of cloud computing, green scheduling in cloud data center and spammer detecting in e-commerce.