Affinity-aware Virtual Cluster Optimization for MapReduce Applications

Cairong Yan*†, Ming Zhu*, Xin Yang†, Ze Yu†, Min Li†, Youqun Shi*, Xiaolin Li‡†
*School of Computer Science and Technology, Donghua University, Shanghai, China
†Scalable Software Systems Laboratory, University of Florida, FL, USA

{cryan, zhuming}@dhu.edu.cn, {xinyang, zeyu, minli}@ufl.edu, yqshi@dhu.edu.cn, andyli@ece.ufl.edu

Abstract—Infrastructure-as-a-Service clouds are becoming ubiquitous for provisioning virtual machines on demand. Cloud service providers expect to use least resources to deliver best services. As users frequently request virtual machines to build virtual clusters and run MapReduce-like jobs for big data processing, cloud service providers intend to place virtual machines closely to minimize network latency and subsequently reduce data movement cost.

In this paper we focus on the virtual machine placement issue for provisioning virtual clusters with minimum network latency in clouds. We define distance as the latency between virtual machines and use it to measure the affinity of virtual clusters. Such metric of distance indicates the considerations of virtual machine placement and topology of physical nodes in clouds. Then we formulate our problem as the classical shortest distance problem and solve it by modeling to integer programming problem. A greedy virtual machine placement algorithm is designed to get a compact virtual cluster. Furthermore, an improved heuristic algorithm is also presented for achieving a global resource optimization. The simulation results verify our algorithms and the experiment results validate the improvement achieved by our approaches.

Keywords—Virtual cluster; MapReduce programming model; Provisioning; Shortest distance; Resource optimization

I. INTRODUCTION

As cloud computing services become popular, more and more academic, enterprise, and personal computing applications are deployed in the shared computing environment. Users of cloud services try to minimize the execution time of their submitted jobs without exceeding a given budget under the specified requirements, while cloud providers try to maximize the use of resources and achieve more profits. Infrastructure as a Service (IaaS) clouds have greatly reduced the investment risk of owning an infrastructure. It offers computers as physical or more often as virtual machines (VMs). A cluster of VMs, virtual cluster, is often requested as a platform for users to run parallel or distributed applications such as MapReduce and Dryad applications. In order to get high throughput, fast response, load balance, low cost, and low price, many topics on VM configuration [1], VM placement [2], VM consolidation [3], and VM migration [4] are explored.

The network topology of a virtual cluster has a significant impact on the execution of applications running on it because the physical nodes where VMs are located can be linked in different ways [5]. For example, some nodes are located in the same rack while others in different racks through a slow link.

Another special architecture is the hierarchical network where two physical nodes may lie in different local area networks. Further more, the characteristics of different applications have different requirements for the network topology. Some applications create tasks running on different VMs which need to exchange large amount of data frequently, while others create tasks which execute independently or exchange a little data.

MapReduce and MapReduce-like models are widely used to process ”Big Data” [6], [7], [8]. Applications based on such models place heavily data-dependency or communication on VMs, so network traffic becomes the bottleneck of jobs. The following are three phases of data exchange in the execution process of an application based on MapReduce model.

- DFS to map. ”Big Data”, stored in the distributed file system, is partitioned into splits as the input of one job, parallel tasks are created in VMs to operate on the corresponding splits to generate intermediate results. The results are partitioned according to the number of reduce tasks and stored in the form of key-value pairs in the local nodes.
- Map to reduce. Each partition of intermediate results of the map phase is transferred to the corresponding node performing the reduce operation. This is the so-called shuffle process.
- Reduce to DFS. Each reduce task aggregates the related data partitions belonging to it and stores its result in the distributed file system.

If there are $n$ servers in the cluster, using all $n$ servers to perform the computation and aggregation provides the highest parallelism. When all the servers participate in the map and reduce phase, the shuffle phase will have all-to-all traffic pattern with $O(n^2)$ flows. The large amount of intermediate results will take a long time to be copied from one node locating a VM which runs a map task to another in the private or public cloud computing platform. Therefore, the network latency becomes one of the main factors to affect the system performance.

To solve this problem, increasing the bandwidth is a direct approach. However, bandwidth is limited and the cost is very high. Other indirect approaches focus on decreasing the traffic. In Hadoop MapReduce runtime, combiner function is designed to aggregate the data of local nodes [9]. Camdoop pushes aggregation operation into the core network so as to decrease
the workload of aggregation of the edge [10].

Some job scheduling strategies decrease the data transformation workload by adopting the locality-based approach, e.g., allocating reduce tasks to the nodes that run map tasks and store related intermediate results in those nodes [11], [12], [13].

This paper targets at provisioning a virtual cluster according to the position relationship between VMs so as to decrease the network traffic and improve the performance of MapReduce and MapReduce-like applications rather than modifying the job scheduling strategies or VM configurations. By optimizing the architecture of virtual clusters, cloud users can get a more efficient platform with the same resource request and cost, and cloud providers can obtain a higher resource utilization ratio. The main contributions of this paper are summarized below.

- We measure the affinity by defining the distance of a virtual cluster. The shorter the distance, the closer the virtual cluster. The shortest distance problem is presented to obtain the closest virtual cluster.
- We solve the shortest distance problem by formulating it into an integer linear programming. A heuristic VM placement algorithm is put forward to provision a virtual cluster. It is designed for MapReduce applications to improve the shuffle speed and accelerate the execution.
- We also optimize the virtual cluster from the global angle, i.e., provisioning virtual clusters for a request queue rather than a single request.
- The online heuristic VM placement algorithm and the global sub-optimization algorithm are compared by simulations. The former has lower time complexity while the latter returns shorter average distance for multiple requests.
- We analyze the performance of our approach through experiments. In the experiment, we adopt different virtual cluster architectures to test different MapReduce applications. Two metrics of application runtime and cluster affinity show the efficiency of virtual cluster optimization.

The rest of the paper is organized as follows. Section II describes the model of virtual cluster provisioning. In section III we define the shortest distance problem, formalize it as an integer linear programming problem, and discuss it from the global optimization angle. Section IV presents an online heuristic VM placement algorithm and a global sub-optimization algorithm for affinity-aware virtual cluster provisioning. Section V analyzes the two algorithms by simulations and demonstrates the model by experiments. Section VI presents related works and section VII concludes our paper.

II. MODEL DESCRIPTION

A cluster is a group of servers and other resources that act like a single system and enable high availability, load balancing, and parallel processing. A virtual cluster consists of a number of virtual nodes or virtual machines which are hosted in the same or different physical nodes. It generally acts as IaaS in private or public cloud computing environments. Each VM is allocated some capabilities such as CPU, memory, and storage space. VMs are classified according to the capabilities they feature. As an example, Table I lists three types of VMs (called instances) available in Amazon EC2 and Table II shows the relationship among servers and VMs where node \( N_1 \) and node \( N_2 \) are in the same rack \( R_1 \), and \( N_1 \) can provide two \( V_1 \)s and three \( V_2 \)s. Each user requests several VMs at one time.

### TABLE I

<table>
<thead>
<tr>
<th>Instance type</th>
<th>Memory (GB)</th>
<th>CPU (compute unit)</th>
<th>Storage (GB)</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>( V_1 )(small)</td>
<td>1.7</td>
<td>1</td>
<td>160</td>
<td>32-bit</td>
</tr>
<tr>
<td>( V_2 )(medium)</td>
<td>3.75</td>
<td>2</td>
<td>410</td>
<td>64-bit</td>
</tr>
<tr>
<td>( V_3 )(large)</td>
<td>7.5</td>
<td>4</td>
<td>850</td>
<td>64-bit</td>
</tr>
</tbody>
</table>

### TABLE II

<table>
<thead>
<tr>
<th>Rack</th>
<th>Node</th>
<th>VM type</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R_1 )</td>
<td>( N_1 )</td>
<td>( V_1 )</td>
<td>2</td>
</tr>
<tr>
<td>( R_1 )</td>
<td>( N_2 )</td>
<td>( V_1 )</td>
<td>3</td>
</tr>
<tr>
<td>( R_2 )</td>
<td>( N_3 )</td>
<td>( V_2 )</td>
<td>2</td>
</tr>
</tbody>
</table>

Let \( n \) be the number of nodes and \( m \) be the number of VM types. We assume VM set is \( \{ V_0, V_1, ..., V_{m-1} \} \) and node set is \( \{ N_0, N_1, ..., N_{n-1} \} \). The following data structures are needed to make a decision for selecting a cluster.

1. \( R \) : a vector of length \( m \) which indicates the number of resources of each type requested by a user. If \( R_i = k \), \( k \) instances of the VM type \( V_i \) are requested.
2. \( A \) : a vector of length \( m \) which indicates the number of available resources of each type. If \( A_i = k \), there are \( k \) instances of the VM type \( V_i \) available totally.
3. \( M \) : a matrix of \( n \times m \) which defines the maximum number of VMs provided by each node. If \( M_{ij} = k \), \( N_i \) may provide at most \( k \) instances of the VM type \( V_j \).
4. \( C \) : a matrix of \( n \times m \) which defines the number of resources of each type currently allocated by each node. If \( C_{ij} = k \), node \( N_i \) is currently allocated \( k \) instances of the VM type \( V_j \).
5. \( L \) : a matrix of \( n \times m \) which indicates the remaining resource provided by each node. If \( L_{ij} = k \), \( N_i \) may provide \( k \) instances of the VM type \( V_j \) to a request. Note:
   - \( L = M - C \), \( A_j = \sum_{i=0}^{n} L_{ij} \).
   - \( \forall j, 0 \leq j \leq m, R_j \leq A_j, R_j = \sum_{i=0}^{n} C_{ij} \).
6. \( D \) : A matrix of \( n \times n \) which defines the distance between physical nodes. According to the network latency, we can define the distance between two physical nodes in the same rack is \( d_1 \), the distance between two physical nodes in

\[ \ldots \]
different racks is \(d_2\), the distance between two physical nodes in different clouds is \(d_3(0 < d_1 < d_2 < d_3)\), etc. The shorter the distance, the faster the data transferring speed. The distance between two VMs is the distance between the two nodes where the VMs are located, i.e., the distance between two VMs in the same node is 0.

Resources can be allocated to a request \(R\) only if the following conditions are satisfied:

- \(\forall j, R_{ij} \leq \sum_{i=0}^{n} M_{ij}(0 \leq j \leq m)\), otherwise, the request will be refused.
- \(\forall j, R_{ij} \leq \lambda_{ij}\), otherwise, the request needs to wait in the queue until resources are available.

Under the constraints of the number and the distance between nodes, how to select available resources from the resource pool is the key to provide better quality of service. Note that we do not consider the problem of server reconfiguration and related information is obtained from the cloud provider.

III. AFFINITY-AWARE CLUSTER PROVISIONING

A. Shortest Distance Problem

In order to provision a virtual cluster to minimize the data transferring overhead for MapReduce applications, we need to resolve the following two questions: (1) what virtual cluster can minimize the network traffic? (2) can it be measured? (3) how should we measure such a virtual cluster? Definition 1 and Definition 2 are presented to valuate a virtual cluster.

**Definition 1. Distance of a virtual cluster (DC).** Given an allocation matrix \(C\) which represents a virtual cluster and a distance matrix \(D\) which defines the distance between nodes. The distance \(DC\) of this virtual cluster is chosen to minimize the sum of distance of all cluster nodes from a distinguished central node.

\[
DC(C) = \min_k \left( \sum_{i=1}^{n} \left( \sum_{j=1}^{m} C_{ij} \right) * D_{ik} \right)
\]

\(N_k\) is the central node \((1 \leq k \leq n)\) and \(\sum_{j=1}^{m} C_{ij}\) is the number of VMs provided by the node \(N_i\).

**Definition 2. Shortest distance (SD) problem.** Given a request vector \(R\), a remaining resource matrix \(L\), and a distance matrix \(D\) find an allocation matrix \(C\) with \(R_{ij} = \sum_{i=1}^{n} C_{ij}\) and \(C_{ij} \leq L_{ij}\) such as to minimize \(DC(C)\). The shortest distance of a request is denoted by \(SD(R)\) and the related virtual cluster is denoted by \(CSD(R)\) which is a matrix. Suppose all possible allocation matrix set is \(C(C^1, C^2, ..., C^s)\) and \(s\) is the number of all possible allocations.

\[
SD(R) = \min_h (DC(C^h)) = \min_h \left( \sum_{i=1}^{n} \left( \sum_{j=1}^{m} C_{ij}^h \right) * D_{ik} \right)
\]

Then we can get \(CSD(R)=C^h (1 \leq h \leq s)\). \(N_k\) is the central node \((1 \leq k \leq n)\) of \(C^h\). \(\sum_{j=1}^{m} C_{ij}^h\) is the number of VMs provided by the node \(N_i\) in allocation \(C^h\).

Fig. 1 presents an instances of a virtual cluster request that is two \(V_1\), four \(V_2\), and one \(V_3\). Rack1 and Rack2 are two different racks provided in a cloud environment.

There are many possible allocations. The following are four available virtual cluster allocations and their distances. \(DC_1\), \(DC_2\), \(DC_3\), and \(DC_4\) are distances of four different provisioning choices.

\[
DC_1 = \begin{bmatrix} 2 & 2 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} = 2d_1 + d_2 \text{ with central node } N_1.
\]

\[
DC_2 = \begin{bmatrix} 2 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} = 2d_1 + d_2 \text{ with central node } N_2.
\]

\[
DC_3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} = 2d_2 \text{ with central node } N_1.
\]

\[
DC_4 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} = d_1 + 2d_2 \text{ with central node } N_1.
\]

We can get \(SD(R)\) and \(CSD(R)\) according to the distance computation of all possible allocations.

B. Integer Programming Formulation of the SD Problem

We formulate the SD problem as follows. Given:

- a set of \(n\) nodes \(N = N_1, ..., N_n\)
- a set of \(m\) virtual machine types \(V = V_1, ..., V_m\)
- a distance matrix \(D(n * n)\)
- an available resource matrix \(L(n * m)\)
- a request vector \(R\) with the length of \(m\)

Variables:

- \(x_{ij}\), integer variables, for \(i = 1, ..., n\), \(j = 1, ..., m\), and \(0 \leq x_{ij} \leq R_{ij}\). \(x_{ij}\) means that \(N_i\) allocates \(x_{ij}\) VMs of type \(V_j\) to form a virtual cluster.
- \(k\), integer variable, \(k = 1, ..., n\), represents the central node of this virtual cluster.

The optimization problem can be written as the follows. Minimize:

\[
\sum_{i=1}^{n} \left( \sum_{j=1}^{m} x_{ij} \right) * D_{ik}
\]
subject to:

\[ \sum_{i=1}^{n} x_{ij} = R_j \]
\[ x_{ij} \leq L_{ij}, \quad k = 1, \ldots, n \]
\[ R_j \leq \sum_{i=0}^{m} L_{ij} \]

The first constraint guarantees the request resources can be satisfied.

The second constraint guarantees the request resources are less than the total available resources.

The provisioning condition is that the number of requested resources do not exceed the number of the available resources.

**Definition 3. Decision version of the SD problem.** Given a request vector \( R \), a remaining resource matrix \( L \), a distance matrix \( D \), and a value \( d \), find an allocation matrix \( C \) with \( R_j = \sum_{i=1}^{n} C_{ij} \) and \( C_{ij} \leq L_{ij} \) such that \( DC(C) \leq d \).

For Definition 4, consider that we are given a certificate. That is, we are given an allocation and a value \( d \). We can verify in polynomial time whether the distance of this allocation is less than \( d \) under the constraints of \( R_j = \sum_{i=1}^{n} C_{ij} \) and \( C_{ij} \leq L_{ij} \). So \( SD \in NP \).

**C. Global Optimization**

By integer programming formulation, we can get the virtual cluster with the shortest distance every time a request is served.

However, when a request arrives and there are not enough resources to allocate to it, the request will wait in a queue until some resources in the clouds are released.

If resources are enough for several requests in the queue to get resources at the same time, does there exist a global optimization strategy for all these requests? Definition 4 describes the problem.

**Definition 4. Global shortest distance (GSD) problem.** Suppose there are enough resources for a request set \( \tilde{R}(\tilde{R}^1, \tilde{R}^2, \ldots, \tilde{R}^p) \) and the available resource matrix is \( L \). The goal is to get a global optimization strategy for all these \( p \) requests, i.e., a global shortest distance GSD and an allocation set \( \tilde{C}(\tilde{C}^1, \tilde{C}^2, \ldots, \tilde{C}^p) \) with \( \tilde{R}^k_j = \sum_{i=1}^{n} \tilde{C}^k_{ij} \) such as to minimize GSD (1 \( \leq k \leq p \), 1 \( \leq i \leq n \), 1 \( \leq j \leq m \)).

\[ GSD(\tilde{R}) = \sum_{k=1}^{p} DC(\tilde{C}^k) \quad (3) \]

We formulate the GSD problem as follows. It can also be formulated as integer programming problem.

Given:

- a set of \( n \) nodes \( N = N_1, \ldots, N_n \)
- a set of \( m \) virtual machine types \( V = V_1, \ldots, V_m \)
- a distance matrix \( D(n \times n) \)
- an available resource matrix \( L(n \times m) \)
- a request matrix \( \tilde{R}(p \times m) \)

Variables:

- \( x_{ij}^k \), integer variables, for \( k = 1, \ldots, p \), \( i = 1, \ldots, n \), and \( j = 1, \ldots, m \); \( 0 \leq x_{ij}^k \leq \tilde{R}_{kj} \). \( x_{ij}^k \) means \( N_i \) allocate \( x_{ij}^k \) of type \( V_j \) for request \( \tilde{R}_{kj} \) to form a virtual cluster.
- \( T_k \), integer variable, \( k = 1, \ldots, p \), represents the central node of request \( \tilde{R}_{kj} \).

The global optimization problem can be written as the follows.

Minimize:

\[ \sum_{k=1}^{p} \left( \sum_{i=1}^{n} \sum_{j=1}^{m} x_{ij}^k \right) * D(\tilde{T}_k) \]

subject to:

\[ \sum_{i=1}^{n} \sum_{j=1}^{m} x_{ij}^k = \tilde{R}_{kj} \]
\[ x_{ij}^k \leq L_{ij} \]
\[ 1 \leq T_k \leq n \]
\[ \sum_{k=0}^{p} \tilde{R}_{kj} \leq \sum_{i=0}^{m} L_{ij} \]

The service time of a request cannot be determined except that users adopt the reservation way and tell the cloud provider how long the resources will be occupied. Furthermore, the length of the wait queue is limited and requests will be served according to some scheduling strategies such as priority-based or FIFO. Users can also cancel their jobs. It is very hard to get a global optimal allocation set.

In order to get an efficient and practical optimization strategy, the following will consider the related suboptimal problems.

**IV. HEURISTIC VM PLACEMENT**

**A. Online Heuristic Algorithm**

To solve the linear programming SD problem above, we put forward the following online heuristic algorithm.

**Theorem 1.** Given two virtual clusters with allocation matrices \( C^1 \) and \( C^2 \), and the same central node \( N_x \). All values in \( C^1 \) and \( C^2 \) are the same except that \( C^1_{pr} = C^2_{pr} - 1 \) and \( C^1_{qr} = C^2_{qr} + 1 \). \( N_p \) and \( N_q \) represent two nodes. \( V_r \) represents one VM type \( (1 \leq x, p, q \leq n, 1 \leq r \leq m, x \neq p, x \neq q, p \neq q) \). If \( D_{xp} \) is less than \( D_{xq} \), \( DC(C^1) \) is less than \( DC(C^2) \).

**Proof:**

\[
DC(C^1) - DC(C^2) = \sum_{i=1}^{n} \left( \sum_{j=1}^{m} C^1_{ij} \right) * D_{xz} - \sum_{i=1}^{n} \left( \sum_{j=1}^{m} C^2_{ij} \right) * D_{xz} = D_{xp} - D_{xq} < 0
\]

From Theorem 1, we can conclude that when \( N_k \) is nearer to the central node \( N_x \) than \( N_i \), if we delete one \( V_j \) in node
$N_i$ and add one $V_j$ in node $N_k$, the distance of the allocation will become shorter. So we should try to place VMs in the neighboring nodes of the central node and maximize the use of those nodes.

The following gives a description about the process of VM placement. Whenever a request arrives, the scheduler will select VMs to form a virtual cluster according to the request type and the number so as to get the shortest distance for this request.

Method $\text{com}(\text{int}[m] A, \text{int}[m] B)$ returns a vector with the element value $\text{min}(A[j], B[j])$. If $\text{com}(A, B) = B$, each element value of vector $A$ is no less than that of vector $B$, i.e., $A$ can provide all resources for this vector $B$.

Method $\text{getList}(\text{int}[n] D, \text{int} x, \text{int} \text{flag})$ returns a node list. Parameter $D$ is the distance matrix, $x$ represents node $N_x$. Parameter $\text{flag}$ remarks the relationship between node $N_x$ and other nodes. If $\text{flag}$ is 0, $\text{getList}()$ returns all the nodes that are in the same rack with $N_x$. If $\text{flag}$ is 1, $\text{getList}()$ returns all the nodes that are not in the same rack with $N_x$. Further more, each node $N_i$ in the list will be sorted according to $\Sigma_{j=0}^{m} \text{com}(L[x], L[i])[j]$ in descending order.

Method $\text{getDist}(\text{int}[n][n] C, \text{int} x)$ returns the distance of an allocation. Parameter $C$ is an allocation matrix and $x$ represents the central node.

A greedy VM placement algorithm is shown as Algorithm 1. First of all, we choose one central node randomly. Then, we select those that are nearer to the central node. The closer away from the central node, the greater chance of being selected. For nodes with the same distance, the more resources they provide, the greater chance of being selected. This algorithm is simplified by configuring the distance to $d_1$ between nodes in the same rack and $d_2$ between nodes in different racks. The central node is not unique. For instance, if all the VMs are allocated from the same rack and no two VMs are hosted on the same node, then any of the allocated nodes could be the central one. It does not impact the algorithm.

The time complexity of this greedy algorithm is $O(n^2 \times m)$ where $n$ is the number of physical nodes in clouds and $m$ is the number of the types of VMs.

### B. Global Sub-optimization Algorithm

Since online heuristic algorithm provides a strategy to obtain the sub-optimization distance for one request each time, if the resources are enough for more requests, a global optimization strategy should be considered.

**Theorem 2.** Given two virtual clusters. One allocation matrix is $C^1$ with central node $N_x$ and the other is $C^2$ with central node $N_y$ ($x \neq y$). If $\exists j, 1 \leq j \leq m, C_{1y}^j > 0$ and $C_{2y}^j < R_y^j$, we can deduce that there exists $k, 1 \leq k \leq n, C_{yk}^2 > 0$. Then we can execute $C_{ky}^1 = - -$ and $C_{kj}^1 = + +$; a new allocation $\hat{C}^1$ is produced. Similarly, we execute $C_{yj}^2 = + +$ and $C_{kj}^2 = - -$; and a new allocation $\hat{C}^2$ is produced. If the relationship among nodes $N_x$, $N_y$, and $N_k$ is $D_{xy} + D_{yk} > D_{xk}$, the distance sum of $D(C^1)$ and $D(C^2)$ is bigger than that of $D(\hat{C}^1)$ and $D(\hat{C}^2)$.

**Proof:**

$$
\begin{align*}
D(C^1) + DC(C^2) & - (DC(\hat{C}^1) + DC(\hat{C}^2)) \\
& = DC(C^1) + DC(C^2) - \\
& (DC(C^1) - D_{xy} + D_{yk}) + (DC(C^2) - D_{yk}) \\
& = D_{xy} + D_{yk} - D_{xk} > 0
\end{align*}
$$

According to the Theorem 2, we can optimize the global allocations like the follows. If two requests share the same central node, do nothing. If two requests have different central

**Algorithm 1 : Online Heuristic Algorithm**

**Input:** $R$: Request list

**A:** Available resource list

**L:** Remaining resource matrix

**D:** Distance of resources

**$n$:** The number of nodes

**Output:** $C$: Allocation matrix

1. for $(i = 1; i += 1; i < n + 1)$ do
2. if $(R[j] > A[j])$ then
3. return $\emptyset$
4. end if
5. end for
6. List $\text{rackList} \leftarrow \emptyset$, $\text{nRackList} \leftarrow \emptyset$
7. int $cn = 0$
8. $L1:$
9. for $(i = 1; i += 1; i < n)$ do
10. if $(\text{com}(L[i], R) = = R$ then
11. $C[i] = R$; $cn = i$;
12. return $C$
13. end if
14. end for
15. List $\text{tempC} \leftarrow 0$, $\text{tempR} \leftarrow 0$
16. $\text{tempC}[i] \leftarrow \text{com}(L[i], R)$
17. $\text{tempR} \leftarrow \text{tempR} - \text{com}(L[i], \text{tempR})$
18. if $(\text{tempR} == 0$ and $\text{getDist}(C, cn) > \text{getDist}(\text{tempC}, i))$ then
19. $C \leftarrow \text{tempC}$; $cn = i$; break $L1$;
20. end if
21. end for
22. $\text{nRackList} \leftarrow \text{getList}(D, i, 1)$
23. for $(j$ in $\text{nRackList})$ do
24. $\text{tempC}[j] \leftarrow \text{com}(L[j], \text{tempR}[j])$
25. $\text{tempR} \leftarrow \text{tempR} - \text{com}(L[j], \text{tempR})$
26. if $(\text{tempR} == 0$ and $\text{getDist}(C, cn) > \text{getDist}(\text{tempC}, i))$ then
27. $C \leftarrow \text{tempC}$; $cn = i$; break $L1$;
28. end if
29. end for
30. end for
31. end for
32. end for
33. return $C$
Algorithm 2 : Global Sub-optimization Algorithm

**Input:** Q: Requests queue
A: Available resource list
L: Remaining resource list
D: Distance of resources
n: The number of nodes

**Output:** Cs: Allocation matrix set

1: Rs ← getRequests(Q, A)
2: for (i = 1; i + +; i < Rs.length()) do
3: (Cs[i]) ← OnlineAlgo(Rs[i], A, L, D, n)
4:  A ← A − Cs[i]
5: end for
6: for (i = 1; i + +; i < Cs.length()) do
7:  for (j = 1; j + +; j < Cs.length()) do
8:       if (i < j and center(Cs[i]) ≠ center(Cs[j])) then
9:           transfer(Cs, Rs, i, j)
10: end if
11: end for
12: end for
13: return Cs

The number of physical nodes in clouds and another will be executed. Method transfer operation of transferring some VMs from one virtual cluster but share some features as the Theory 2 describes. Then the allocation pairs which do not share the same central nodes, to minimize the sum of the distance of all the allocations. If there will be not enough resources for another request, the available resources will become less. At the end, for each request. Each time the algorithm is applied on a global sub-optimization algorithm shown as Algorithm 2. There are two steps to process.

Step 1: Obtain all the requests that the cloud resources can meet.

Method getRequests(int q, m, A) returns all the requests in the queue Q that the resources A can meet according to some related priority strategies based on the queue, e.g., FIFO. Parameter q is the length of Q.

Step 2: Apply online heuristic algorithm on these requests one by one and get an allocation set.

Algorithm 1 returns the allocation with the shortest distance for each request. Each time the algorithm is applied on a request, the available resources will become less. At the end, there will be not enough resources for another request.

Step 3: Adjust the allocation set slightly based on Theorem 2 to minimize the sum of the distance of all the allocations.

Given a set of requests and a set of allocations, we find the allocation pairs which do not share the same central nodes, but share some features as the Theory 2 describes. Then the operation of transferring some VMs from one virtual cluster to another will be executed. Method transfer(int p, m, Cs, int i, int j) will exchange the VMs’ positions of allocation i and allocation j so as to return two new allocations with smaller sum of the distance. Method center(int[n][m] C) returns the central node of allocation C.

The time complexity of step 1 is O(n^2 × m) where n is the number of physical nodes in clouds and m is the number of the types of VMs. Because the number of requests that can be served is less than the length of the queue, the time complexity of step 2 is less than O(q^2) where q is the length of the waiting queue.

V. PERFORMANCE EVALUATION

A. Simulations

In this section, we evaluate the provisioning model of virtual clusters based on the characteristic of affinity and two heuristic algorithms are compared. We simulate a cloud with the following configurations.

- The number of racks is 3. The distances between nodes in different racks are the same.
- Each rack has 10 nodes. The distances between nodes in the same rack are the same.
- Each node provide different VM types and numbers. The distance between VMs in the same node is considered to 0.
- Twenty requests are simulated. The simulated requests will arrive and their job will finish randomly.

The greedy VM placement algorithms are implemented in Java language. The instances on each physical node are distributed randomly. The types and numbers of the twenty requests are also generated randomly.

Fig. 2 shows the distance difference between two central node selection strategies. Heuristic distance is mapped to the virtual cluster with the most appropriate central node built by our online VM placement algorithm. Shortest distance with a random central node is mapped to the same virtual cluster, however, the central node is chosen randomly. It is clear that even if we select the same virtual cluster, but not the same position of the central node, the distance difference is also great. For MapReduce applications, the selection of the central node is the same important with the architecture of the virtual cluster.

Fig. 3 shows the variation of central nodes with different requests under the constraint of the shortest distance. Each physical node has different capability and each request has different requirement. For MapReduce applications, it is very important to match the request with a virtual cluster and appropriate central node so as to make full use of the advantage of data-locality and reduce the network traffic.

Fig. 4 shows the different distances under the different central nodes. We can see that the choice of the central node has a important impact on the distance for one request. This is because MapReduce and Mapreduce-like models are based on master-slaves network topology.

When the requests arrive randomly, their service time are also random, and the cloud resources are enough to meet multiple requests at one time, we can get the virtual cluster provisioning according to our online heuristic algorithm and global sub-optimization algorithm. We simulate two scenarios of requests. One uses the same request configurations as the previous simulations and the other uses a request sequence with a relatively small number of VMs. Fig. 5 and Fig. 6 show the distance variation of two algorithms under the two scenarios. Compared with the online heuristic algorithm, the...
global sub-optimization algorithm can get shorter distances. In the former scenario, it makes the sum of distances decrease by 2%, and in the latter, it makes the sum of distances decrease by 12%. The difference reflects that the global sub-optimization algorithm is more suitable for the requests with a small number of the VMs and the online heuristic algorithm is very competitive for its lower time complexity.

**B. Experimental Evaluation**

We conducted an experiment on the high performance center at University of Florida. The test cases and the deployment of environment are as follows.

1. **Benchmarks.** WordCount is a typical application where Hadoop developers get hands on. It is intent to count the number of occurrence of each word in the provided input files.

2. **Network Topology Setup.** There are three data transferring phases between VMs. First, the data stored in the distributed file systems as blocks needs to be partitioned and sent to the nodes running map tasks. Second, the intermediate results of map tasks need to be copied to the nodes running reduce tasks. Third, the results of reduce tasks need to be stored in the distributed file system. As the number of VMs increases and the network topology changes, the runtime of applications will be affected. Different deployments of VMs also affect the execute of applications [14], [15]. In our experiment, each rack is made up of different number of physical nodes and each physical node includes different number of VMs.

3. **Metrics:** We use the following metrics to compare the performance of virtual clusters with different distances:

   - **Application runtime.** This metric provides a measurement of the application performance which is one of the most useful parameters to evaluate a platform.

   - **Cluster affinity.** This metric describes the distance of virtual clusters. Here we configure the distance between VMs
in the same node is 0, the distance between nodes in the same rack is 1, the distance between nodes in different racks is 2.

Fig. 7 shows the runtime of WordCount in the clusters with different distances. We can see that the runtime is smaller when the distance is shorter. There is an exception where the runtime of distance 14 is larger than that of distance 16. It can be explained in Fig. 8. When a MapReduce job is submitted, many map tasks and reduce tasks are distributed in different VMs to run. The placement of tasks is determined by the job scheduler and affected by the running environment. There are 32 map tasks and 1 reduce task in this experiment. We find that the non data-local map tasks and non local shuffle processes are smaller when the distance of cluster is 16 so that its runtime decreases.

Fig. 7. Runtime of WordCount under the four different topologies of virtual clusters with the same capability.

Fig. 8. Data and shuffle locality under the four different topologies of virtual clusters with the same capacity.

The reason why the distance is not continuous is that the topology of a virtual cluster has a great impact on the value of the distance. We perform the experiment on the HPC of UF and each time it returns a different cluster topology according to the configuration of MyHadoop.

VI. RELATED WORK

(1) VM management. When a virtual machine is requested to be deployed in the cloud environment, the process of selecting the most suitable host for the virtual machine is known as virtual machine placement. Generally hosts are rated based on the virtual machines hardware and resource requirements, and the anticipated usage of resources [2]. VM scheduling based on load balancing to improve the system throughput and response time is widely researched by using Best-Fit or other related strategies [16]. Cost-aware system can provide efficient support for elasticity in the cloud. Optimizing the selection of a virtual machine configuration can minimize the the cost and maximize the profit [17], [18]. VM consolidation schemes focus on CPU, memory, disk I/O, or bandwidth prediction to pack VMs tightly so as to save energy and make full use of the resources [19], [3], [20]. Virtual machines can instead be used to fully utilize the system resources, ease the management of such systems, improve the reliability, and save the power.

(2) Virtual cluster. A virtual cluster is composed of a number of VMs interconnected by a virtual or physical network over underlying physical resources commonly existed in the cloud computing environment. Each VM acts as a virtual node of a computing cluster. To build a virtual cluster, under-provisioning and over-provisioning are taken into consideration to reduce the cost, optimize the resources, and satisfy the customer requirements [21], [22]. Virtual cluster’s building is very important for applications running on it, e.g., MapReduce applications. Some evaluations are made for MapReduce model in the virtual cluster [23]. It is proved that data shuffling is very important for system performance. Affinity-aware virtual cluster VM migration technology is used to minimize the communication overhead [24], [4]. This paper focus on how to form a virtual cluster for MapReduce applications.

(3) MapReduce job scheduling. It is a hot topic to improve the performance of MapReduce applications. Traditional and efficient locality-aware scheduling strategies can reduce processing time and increase throughput by decreasing the data traffic between nodes and balancing the loads of the cluster [25], [13], [12]. The shuffle operation of MapReduce model is proved to be as much as a third of the completion time of a MapReduce job with others are map operation and reduce operation [26]. Chen et al. also presents the effectiveness of the relationship of task precedence and dependence [11]. Camdoop, a MapReduce-like system, uses a direct-connect network topology with servers directly linked to other servers so as to reduce traffic by performing in-network aggregation of data during the shuffle phase [10]. The strategy of "grey boxes" is presented to optimize data shuffling by treating user-defined functions as "grey boxes" so that users can adjust related parameters according to the feedback of job execution [27].

(4) Virtual cluster provisioning for MapReduce jobs [28], [1], [8]. Park proposes a dynamic VM reconfiguration technique for MapReduce applications. It assigns tasks on the nodes containing their input data to maximize the utilization of resources and improve the performance by executing data-locality tasks. It needs resource reservation in advance for scaling up VMs. Palanisamy proposes a method to prove the data-locality of map and reduce tasks by exploiting prior knowledge about the characteristics of MapReduce overloads. Liu implements a MapReduce model on clouds platform so as to make full use of the VM resources instead of physical nodes.

Virtual cluster is a promising platform for MapReduce
applications. Since many Cloud providers such as Amazon EC2 and FutureGrid provide static VM types to be selected, we present a heuristic algorithm to help them provide affinity-aware virtual cluster without any prior knowledge of the jobs and without scaling up or scaling down the instances. It does not affect the execution of jobs or scheduling. It targets at the optimization of virtual cluster by placing VM on suitable physical nodes so as to improve the response time of jobs.

VII. CONCLUSION

IaaS clouds allow users to request computational resources in the form of VMs. A high efficient virtual cluster can minimize the network overhead and maximize the system performance. The simulation demonstrates the feasibility of the online heuristic algorithm and global sub-optimization algorithm. Meanwhile the experiment results show the improvement of virtual cluster optimization for MapReduce applications. The optimization method can be extended to MapReduce-like applications where a master node distributes tasks on several different slave nodes and these slave nodes work collaboratively.

In the future, we first plan to investigate the distance between physical nodes. It is measured and configured statically in this paper. How to compute their values when some VMs are down or reconfigured is critical for the VM placement policy. Second, from the experiments, we find that the data-locality and the shuffle-locality are two important factors to affect the execution time in addition to the affinity of the virtual cluster. The integration of more fine-grained virtual cluster provisioning methods and MapReduce scheduling strategies needs to be explored.

REFERENCES


