Computer Networks 112 (2017) 194–207

Contents lists available at ScienceDirect

Computer Networks

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

Optimal construction of virtual networks for Cloud-based MapReduce workflows

Cong Xu[a,b,*], Jiahai Yang[a,b], Kevin Yin[c], Hui Yu[a,b]

[a] *Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China*
[b] *Tsinghua National Laboratory for Information Science and Technology (TNList), Tsinghua University, Beijing 100084, China*
[c] *Chief Technology & Architecture Office, Cisco China, Beijing 100022, China*

A R T I C L E   I N F O

Article history:
Received 18 May 2015
Revised 21 July 2015
Accepted 3 November 2016
Available online 4 November 2016

Keywords:
Cloud computing
MapReduce
Virtual networks
OpenStack neutron
Optimal deployment

A B S T R A C T

Cloud-based big data platforms are being widely adopted in industry, due to their advantages of facilitating the implementation of big data processing and enabling elastic service frameworks. With the widespread adoption of cloud-based MapReduce frameworks, a series of solutions have been proposed to improve the performance of big data services over cloud. The majority of the existing studies concentrate on optimizing the task scheduling or resource provisioning mechanisms, to improve the data processing rate or data transmission rate of the platform separately, without an overall consideration of both the performance factors. Moreover, these studies seldom consider the impact of virtual network topologies on the performance of the cloud-based MapReduce workflows. The purpose of this work is to optimize the topologies of virtual networks used in cloud-based MapReduce frameworks. We formulate both the data transmission and data processing overhead of a specific cloud-based big data application, describe the optimal deployment of virtual networks as an optimization problem and then design algorithms to solve this problem. Experimental results show that our topology optimization mechanism improves the overall performance of cloud-based big data applications effectively.

1. Introduction

With the immense proliferation of cloud-based data intensive applications, rapid and efficient processing of large-scale datasets generated by cloud platforms has become a challenge. To address this problem, some large enterprises have deployed big data processing services on their cloud platforms, such as Google's BigQuery [1], Amazon's EMR [2], etc. The flexible virtual resources, as the fundamental advantage of cloud platforms, make the cloud-based big data applications more scalable and cost effective, by implementing an elastic service framework and enabling a pay-as-you-go manner to the users [3]. Moreover, these cloud-based big data platforms make it practical for smaller enterprises to access massive computing resources for short, semi-predictable time periods without having to deploy and manage their own big data platforms [4].

As one of the most widely used big data processing frameworks, MapReduce/Hadoop [5,6] has been adopted extensively in cloud-based big data systems. Correspondingly, optimizing these cloud-based MapReduce frameworks has attracted increasing attention and a series of solutions have been proposed to improve the performance of the MapReduce operations. However, the majority of these studies focused on optimizing the scheduling mechanisms of MapReduce jobs or tasks in local environment [7–12], only a few studies discussed the impact of cloud environment on MapReduce workflows.

The two dominant factors that affect the performance of a MapReduce workflow are data transmission latency between Virtual Machines (VMs) and data processing rate of a specific VM. To the best of our knowledge, the existing cloud platform optimization mechanisms concentrate on optimizing each performance factor (data transmission latency or data processing rate) separately, while they seldom take an overall consideration of both the performance factors. When considering the optimization of data processing rate, the researchers focus on analyzing the impact of cloud environment on the performance of each VM, and optimizing the provisioning or placement of VMs in a cloud platform [3, 13, 14]; while they hardly address the optimization of data transmission latency. Similarly, the data processing rate of each VM is also omitted in the existing data transmission latency optimization mechanisms [16, 19]. Moreover, different from the traditional MapReduce

* Corresponding author at: Room 1-211, FIT Building, Tsinghua University, Beijing 10084, China. Fax. +86 10 62785983.

E-mail addresses: xucong10@mails.tsinghua.edu.cn, xc19861230@126.com (C. Xu), yang@cernet.edu.cn (J. Yang), kyin@cisco.com (K. Yin), yuhui7red@outlook.com (H. Yu).

http://dx.doi.org/10.1016/j.comnet.2016.11.001
1389-1286/© 2016 Elsevier B.V. All rights reserved.

framework, the data transmission latency of a cloud-based MapReduce workflow is determined not only by the physical network topology, but also by the topologies of virtual networks in a cloud platform: after transmitted to an end node of a physical network (a server), a data chunk will suffer an additional transmission delay in virtual networks to reach its destination slave (a VM). The existing data transmission latency optimization mechanisms, which only minimize data transmission time in physical networks, are not applicable to cloud-based MapReduce frameworks since they neglect the transmission time spent in virtual networks.

This paper focuses on optimizing the topologies of virtual networks in data centers to improve the overall performance of cloud-based MapReduce frameworks. We deploy a MapReduce framework which is based on our cloud platform built using open source software OpenStack Havana [20], and analyze the detailed operating procedure of a cloud-based MapReduce workflow. To evaluate the performance of a cloud-based MapReduce framework, we present a performance model that precisely capture the expected data transmission latency and data processing rate in the Map and Reduce operations. Based on the modeling results, we further propose an optimization model and corresponding algorithms to determine the optimal topologies of virtual networks embedded in physical cloud platforms. Our focus is to strike the right balance between data transmission latency and data processing rate to improve the overall performance of cloud-based MapReduce workflows.

The main contributions of this paper are as follows:

- We propose a novel mechanism that considers the optimization of both the data transmission latency and data processing rate, and strikes the right balance between the two performance factors.
- We optimize the topologies of virtual networks embedded in physical data centers, which can be considered as an extension of the existing data transmission latency optimization mechanisms.
- We take the first step towards providing optimal deployment mechanism of multi-host virtual networks based on OpenStack Neutron (of releases Grizzly and Havana).

The rest of this paper is organized as follows. Section 2 summarizes the existing performance optimization mechanisms of MapReduce frameworks, and introduces multi-host based virtual networks built using OpenStack software. Section 3 studies the detailed operating procedure of a cloud-based MapReduce workflow and puts forward a performance model to formulate the data transmission and data processing latency. Section 4 proposes an optimization model to describe the topology optimization problem, and designs a novel mechanism TOMON to determine the optimal virtual network topology based on the modeling results. Section 5 evaluates our topology optimization mechanism in simulation environment as well as in a real cloud computing platform. Section 6 concludes the paper and gives directions for our future work.

2. Related work

2.1. Performance optimization of MapReduce workflows

Improving the performance of MapReduce applications has been attracting the attentions of researchers. The majority of the existing studies focus on optimizing the scheduling mechanisms of MapReduce jobs or tasks. M. Zaharia et al. [7] present an improved task scheduling scheme, LATE, which reduces the application completion time by executing tasks that will finish farthest into the future. H. Chang et al. [8] devise approximation algorithms which generate feasible schedules of MapReduce jobs, and keep a job's completion time within a small constant factor of the speculative optimal value. F. Chen et al. [9] delve into task level and develop constant factor approximation algorithms for minimizing the weighted task completion time. Network bottlenecks of the MapReduce clusters have also been considered in [10–12], and judicious task placement and scheduling methods have been proposed to further improve the data transmission time of MapReduce jobs. However, these solutions omit the impacts of cloud environment on the performance of MapReduce operations, which are more applicable to big data applications in local environment.

There are also a few researches which address the practical performance of MapReduce frameworks on cloud platforms. K. Kambatla et al. [13] compare resource consumption of different cloud-based applications, and optimize the configurations of cloud resources for these applications. Y. Geng et al. [14] devise a model to theoretically analyze data allocation problems in virtual environment, and design a location-aware file block allocation strategy to retain the compatibility of cloud platforms with native Hadoop frameworks. H. Herodotou et al. [21] analyze the impact of virtual cluster scale on the data processing rate of cloud applications, and further optimize the scale of virtual clusters to improve the performance of data-intensive cloud-based applications. Y. Yuan et al. [3] re-model the resource provisioning problem in cloud-based big data systems and present an interference-aware solution. Z. Zhang et al. [22] focus on predicting the completion time of MapReduce jobs in heterogeneous environment.

Different from the former studies, some researchers focus on improving the data transmission performance of cloud-based applications. M. Alicherry et al. [23] introduce “VM communication latency” as a new performance factor to the traditional resource provisioning mechanisms, and put forward a novel network aware resource provisioning mechanism. M. Li et al. [15] couple the data placement, VM placement, and task placement to systematically improve data locality of cloud-based MapReduce applications. Novel VM placement and resource provisioning mechanisms are proposed in [16–18] to improve the data transmission or processing latencies of cloud-based big data applications on the basis of workloads. There are also some researches concentrating on studying the “Virtual Network Embedding (VNE)” problem: M. Yu et al. [24] design greedy embedding policy to embed VMs to the physical server with the lightest load first; X. Cheng et al. [25] and S. Zhang et al. [26] design “topology-aware” virtual network embedding mechanisms to improve the communication performance of virtual clusters; J. Lu et al. [27] concentrate on embedding virtual networks of backbone-star topologies into physical clusters; L. Gong et al. [28] propose a novel metric –global resource capacity (GRC), to quantify the embedding potential of each substrate node, and propose an efficient heuristic virtual network embedding (VNE) algorithm; I. Houidi et al. [29,30] propose distributed virtual network embedding mechanisms to improve the scalability of the traditional centralized embedding mechanisms.

These studies optimize data processing or data transmission performance separately, without an overall consideration of both the performance factors.

2.2. Multi-host virtual network based on OpenStack neutron

OpenStack [20] has become one of the most widely used open sources to build cloud platform, since it shows great advantages in layered architecture, SOA (Service Oriented Architecture), componentization, openness, etc. Neutron is an OpenStack component to provide “networking as a service” between interface devices (e.g., vNICs (Virtual Network Interface Cards)), which enables the establishment of virtual networks in physical data centers.

Using the single-host based virtual network architecture in OpenStack release Folsom, the communication agent (L3 agent,

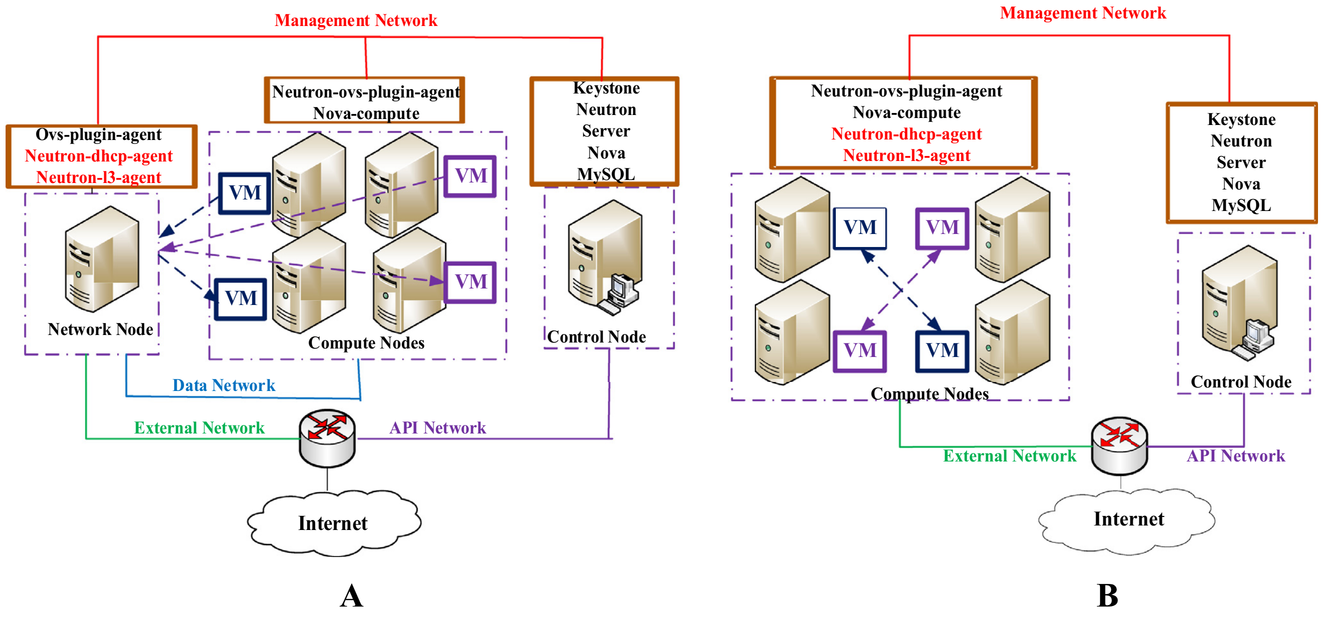


Fig. 1. Single-host and multi-host deployment of virtual networks using OpenStack neutron.

which enables the data transmission between VMs) must be deployed on a single server (named network node). Thus, only centralized virtual network topology can be realized based on OpenStack Folsom as is shown in Fig. 1A, and the performance and availability of these virtual networks are affected by the performance bottleneck and the SpoF (Single point of Failure) problem of the single communication agent.

To counteract the problems caused by single-host virtual networks, multi-host deployment of communication agents are enabled in the later OpenStack releases Grizzly and Havana. As shown in Fig. 1B, the additional network node is unnecessary in multi-host based virtual networks, and the communication agents can be deployed on any of the compute nodes. Therefore any compute node can act as a network node, which improves the scalability and performance of virtual networks. However, detailed virtual network deployment mechanisms have not been provided in the latest releases of OpenStack, and our research takes the first step towards providing optimal deployment mechanism of multi-host virtual networks based on OpenStack Neutron.

3. Performance evaluation of cloud-based MapReduce workflows

In this section, we present the detailed operating procedure of a cloud-based MapReduce application in a multi-host virtual cluster, and propose a corresponding model to formulate the performance of the cloud-based MapReduce framework. Some important notations and definitions used in the model are illustrated in Table 1. For analytic tractability, we assume that the physical cloud platform satisfies centralized architecture, and our focus is to optimize the topologies of the overlying virtual networks.

3.1. Baseline experiment

To accurately capture the performance of a cloud-based MapReduce workflow, firstly we need to analyze the impact of cloud environment on the data processing rate and data transmission latency. Fig. 2A and B show the impact of server load and VM location on VMs' data processing and communication performance respectively.

First, we analyze the impact of server load on VMs' data processing performance. We gradually increase the number of co-located VMs on the same server while processing the same input data, and record the average data processing rate of these co-located VMs in different scenarios. We can see from Fig. 2A that the average data processing rate of the co-located VMs shows a linear decline as the server load increases, which is consistent with the result shown in [7]. Moreover, if the VMs co-locate with a communication agent (L3 agent in our experiment), they will suffer performance degradation at a constant rate. Performance function of VM j on server i can be approximately expressed as:

$$\mu_{ij} = \left[\mu_0 - \frac{\mu_0}{n_i^{\max}+1}(n_i - 1)\right] \cdot [1 - x_i(1-\gamma)] \tag{1}$$

As is shown in Fig. 2A, our performance function accurately describes the changing trend of average data processing rate, which can be used in the following performance model.

Next, we analyze the impacts of VM locations and virtual network topologies on the data transmission rate between VMs. Fig. 2B shows the data transmission latencies between two VMs at different locations, under different virtual network topologies. Similar to the result shown in [23], we find that the communication overhead between co-located VMs can be neglected compared with VMs located on different servers. Thus, to improve the overall communication performance of a virtual cluster, an optimal deployment mechanism should avoid cross-server data transmissions.

Comparing the results shown in Fig. 2A and B, we find that conflicts exist for optimizing the two performance factors (data transmission latency and data processing rate): according to Fig. 2A, to maximize the data processing rate of each VM, the optimal deployment mechanism should allocate VMs on different servers to reduce the amount of co-located VMs; however, according to Fig. 2B, to minimize the total data transmission latency, the VMs should be placed closely to each other to reduce the cross-server communications, which is a worst deployment mechanism to optimize the data processing performance according to Fig. 2A. Therefore, a novel mechanism is needed to strike the right balance between the two performance factors, and to improve the overall performance of cloud-based MapReduce applications.

Table 1
Summary of key notations and definitions.

Notations	Definitions
N	Total number of avaiable VMs in a cloud platform
m	Total number of available physical servers
n_i	Number of available VMs on server i
$n_i^{\max}$	Maximum number of VMs can be co-located on server i
S	Expected total size of input big data during a fixed time period
B	Average data transmission rate between any two servers
t_{ij}^{map}	Execution latency of VM j located on server i in map phase
t_{ij}^{reduce}	Execution latency of VM j located on server i in reduce phase
x_i	Binary variable, set to 1 if an agent is deployed on server i, set to 0 otherwise
μ_0	Average data processing rate of a VM at empty load
γ	Average performance degradation rate of a VM, when co-located with a communication agent on a server
μ_{ij}	Average data processing rate of VM j located on server i
s_{ij}	Total size of data chunks allocated to VM j located on server i during a fixed time period
n_r	Total number of selected reducers
S_r	Average size of intermediate results generated on a VM after map phase
k	Total number of deployed communication agents
$l(i)$	Physical location of agent i
$V(i)$	Set of VMs assigned to agent located on server i
A_{ij}	Agent responsible for the communication of VM j located on server i
N_a	Total number of VMs co-located with communication agents

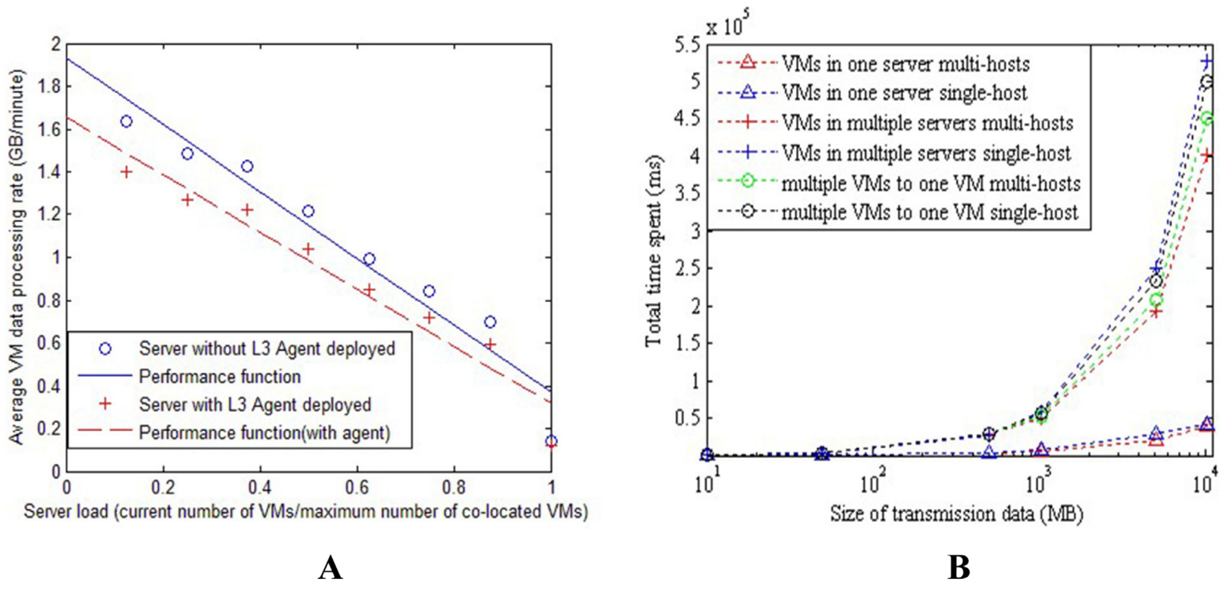


Fig. 2. Impact of cloud environment on VMs' data processing and communication performance.

3.2. Performance model for the map phase

In the map phase, the input data is split into small chunks and assigned to the mappers (VMs located on servers) for processing. The mappers read the data chunks and produce intermediate results. Fig. 3 describes the detailed operation procedure of the map phase in a multi-host virtual network based on OpenStack Neutron.

The procedure of the map operation can be divided into three sub-phases:

- The input data is split into small chunks and assigned to the communication agents (L3 agent in our scenario).
- Communication agents forward the small data chunks to the destination VMs.
- VMs (mappers) process the data blocks and produce intermediate results.

Thus the execution latency of VM j located on server i in the map phase can be expressed as:

$$t_{ij}^{map} = t_{ij}^{assign} + t_{ij}^{forward} + t_{ij}^{process} \tag{2}$$

The time spent in the assignment sub-phase is determined by the size of input data and the data transmission latency of each server. Assume the total size of the input data is S during a fixed time period in the steady state, then we get $t_{ij}^{assign} = S/B$.

A VM's data processing latency is determined by the total size of data chunks forwarded to the VM and the VM's data processing rate. Under the most commonly used FIFO (first in first out) task scheduler, the total size of data chunks assigned to a VM is proportional to the VM's processing rate. Therefore, a VM's data processing latency in the map phase can be formulated as:

$$t_{ij}^{process} = s_{ij}/\mu_{ij} = \left(S \cdot \mu_{ij} \Big/ \sum_{i=1}^{m} \sum_{j=1}^{n_i} \mu_{ij} \right) \Big/ \mu_{ij} = S \Big/ \sum_{i=1}^{m} \sum_{j=1}^{n_i} \mu_{ij}$$

As aforementioned, an optimal virtual network topology should minimize cross-server communications. In order to minimize cross-server communications between VMs and agents, if a communication agent is deployed on a server, all the co-located VMs on the same server should be assigned to this agent. Thus, if a VM is co-located with a communication agent on the same server, the data forwarding overhead of this VM can be neglected according

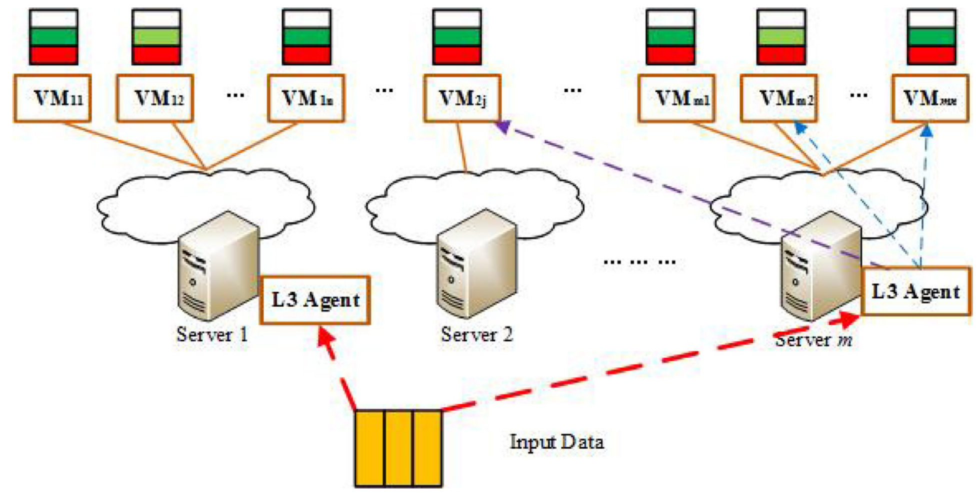


Fig. 3. Operating procedure of map phase in a multi-host virtual network.

to the result shown in Fig. 2B. Thus, the lower bound of the transmission overhead is determined by the total size of data chunks in cross-server transmissions. So, we get:

$$t_{ij}^{forward} = (1 - x_i) \cdot \left(\sum_{u,v}^{V_{uv} \in V(A_{ij})} s_{uv} - \sum_{v=1}^{n_{l(A_{ij})}} s_{uv} \right) / B$$

Substituting the expressions t_{ij}^{assign}, $t_{ij}^{forward}$ and $t_{ij}^{process}$ into (2), we can calculate the total execution latency of a VM in the map phase. Moreover, by adding the execution latency of each VM, we get the overall performance of a cloud-based MapReduce cluster with the scale of m servers, N VMs and k agents:

$$\begin{aligned} T_{total}^{map} &= \sum_{i=1}^{m} \sum_{j=1}^{n_i} t_{ij}^{map} = N \cdot \frac{S}{B} + \frac{\sum_{i}^{x_i=0} \sum_{j=1}^{n_i} \mu_{ij}}{\sum_{i=1}^{m} \sum_{j=1}^{n_i} \mu_{ij}} \cdot \frac{S}{kB} \\ &+ \frac{NS}{\sum_{i=1}^{m} \sum_{j=1}^{n_i} \mu_{ij}} = N \cdot \frac{S}{B} + \frac{SN_a}{kBN\gamma} + \frac{NS}{\sum_{i=1}^{m} \sum_{j=1}^{n_i} \mu_{ij}} \end{aligned} \quad (3)$$

3.3. Performance model for the reduce phase

In the reduce phase, the MapReduce framework shuffles the intermediate data generated in the map phase and moves them to the destination VMs (reducers) for processing. After that, the reducers process the intermediate data and generate final results. Detailed operating procedure of the reduce phase in a multi-host virtual network is shown in Fig. 4.

The data forwarding process in the reduce phase can be divided into three steps: step 1, the mappers transmit the intermediate data to their communication agents; step 2, all the communication agents forward their data to the destination agents which handle the communications of reducers; step 3, the destination agents transmit their data to the reducers for processing.

Data transmission in each step can be an intra-server transmission or a cross-server transmission, depending on the locations of the source and destination VMs. Correspondingly, the transmission latency of a specific VM in the reduce phase has four possible values. For the best case, the source VM, destination VM and their communication agents are co-located on the same server, and the transmission latency is close to 0. For the worst case, the source and destination VMs are located on different servers without agents deployed, and handled by different agents, and then they will have to suffer three cross-server transmissions.

Correspondingly, the transmission latency from VM j located on server i, to reducer v located on server u can be formulated as:

$$t_{ij}^{reduce} = \begin{cases} \frac{3S_r}{B} & if \;\; A_{ij} \neq i \;\; and \;\; A_{uv} \neq u \;\; and \;\; l(A_{ij}) \neq l(A_{uv}) \\ \frac{2S_r}{B} & if \;\; A_{ij} \neq i \;\; and \;\; A_{uv} \neq u \;\; and \;\; l(A_{ij}) = l(A_{uv}) \\ & or \;\; A_{ij} = i \;\; and \;\; A_{uv} \neq u \;\; and \;\; l(A_{ij}) \neq l(A_{uv}) \\ & or \;\; A_{ij} \neq i \;\; and \;\; A_{uv} = u \;\; and \;\; l(A_{ij}) \neq l(A_{uv}) \\ \frac{S_r}{B} & if \;\; A_{ij} \neq i \;\; and \;\; A_{uv} = u \;\; and \;\; l(A_{ij}) = l(A_{uv}) \\ & or \;\; A_{ij} = i \;\; and \;\; A_{uv} \neq u \;\; and \;\; l(A_{ij}) = l(A_{uv}) \\ & or \;\; A_{ij} = i \;\; and \;\; A_{uv} = u \;\; and \;\; l(A_{ij}) \neq l(A_{uv}) \\ 0 & if \;\; A_{ij} = i \;\; and \;\; A_{uv} = u \;\; and \;\; l(A_{ij}) = l(A_{uv}) \end{cases} \quad (4)$$

Suppose the reducers are randomly selected among the existing VMs, to get the accurate total execution latency, first we need to calculate the expected number of communication agents that handle the communications of the reducers, and the expected number of the reducers co-located with their agents.

The total data transmission latency of step 1 is determined by the number of VMs co-located with their communication agents, since these VMs need not to suffer additional cross-server communications in step 1. Suppose the number of VMs co-located with their agents is N_a, then the expression of N_a is:

$$N_a = n_{l(1)} + n_{l(2)} + \ldots + n_{l(k)}$$

The total data transmission latency of step 2 is determined by the number of the reducers' communication agents. These agents are the destination agents of the transmissions. The probability of each one of the k agents to be chosen as the destination agent can be calculated as:

$$p = 1 - C_{n_r+(k-1)-1}^{(k-1)-1} / C_{n_r+k-1}^{k-1} = \frac{n_r}{n_r + k - 1}$$

Let k_r be the number of the VMs co-located with the destination agents in step 2. Then, these VMs do not suffer the additional cross-server communications in step 2. The expected value of k_r is:

$$E(k_r) = \sum_{i=1}^{k} n_{l(i)} \cdot p = \frac{n_r N_a}{n_r + k - 1} \quad (5)$$

The total data transmission latency of step 3 is determined by the number of the reducers co-located with their agents, as the transmission latency between these reducers and agents can be neglected. Since the reducers are randomly selected, each reducer has the probability N_a/N to co-locate with its agent. We define n_a as the number of the reducers co-located with their agents among

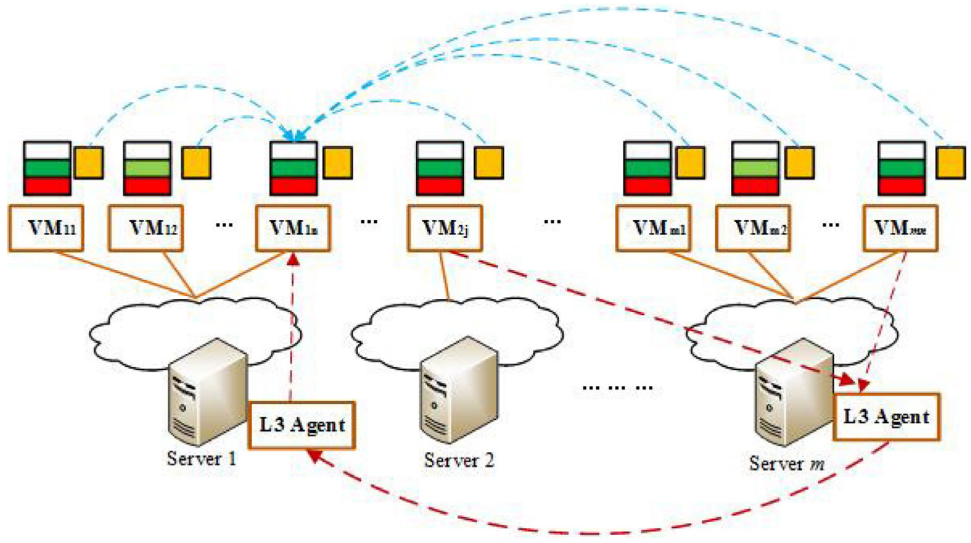


Fig. 4. Operating procedure of reduce phase in a multi-host virtual network.

all the n_r reducers. The expected value of n_a can be calculated as:

$$E(n_a) = \frac{N_a n_r}{N} \tag{6}$$

Combining (4)–(6), we can calculate the accurate execution latency of each step. Furthermore, based on the inclusion-exclusion principle, we get the total execution latency of all the reduce tasks:

$$\begin{aligned} T_{total}^{reduce} &= \sum_{i=1}^{m}\sum_{j=1}^{n_i} t_{ij}^{reduce} \\ &= \frac{(N-N_a)S_r n_r}{kB} + \frac{NS_r}{B}\cdot(1-\frac{N_a}{N})n_r \\ &\quad + \left[\frac{NS_r n_r}{B} - \frac{(N-N_a)S_r}{kB}\cdot\frac{n_r k}{n_r+k-1} - \frac{N_a S_r n_r}{B(n_r+k-1)}\right] \\ &= \frac{[(2k+1)N-(k+1)N_a]S_r n_r}{kB} - \frac{(N-2N_a)S_r n_r}{B(n_r+k-1)} \end{aligned} \tag{7}$$

Note that a MapReduce workflow will suffer an additional execution latency, waiting for the reducers to process the intermediate data and produce the final results. However, the communication agents do not work in this sub-phase and hence the performance of the reducers are not affected by the distribution of the agents, which means that the total execution latency of this sub-phase is irrelevant to the virtual network topology. Therefore, we omit this sub-phase in our performance model.

4. Topology optimization mechanism

Based on the modeling results shown in Section 3, we put forward a virtual network topology optimization mechanism – TOMON (Topology Optimization Mechanism based on OpenStack Neutron) to improve the performance of cloud-based MapReduce frameworks. TOMON is composed of three strategies, determining the following factors in a data center respectively:

- The optimal number of communication agents
- The optimal placement of each communication agent
- The optimal matching strategy between VMs and agents

4.1. Performance optimization model for cloud-based MapReduce workflows

The main purpose of our work is to optimize the performance of MapReduce workflows. Specifically, we aim to minimize the total task execution time as well as the time span of a specific MapReduce workflow.

$$Min \quad T_{total} = T_{total}^{map} + T_{total}^{reduce} \quad \forall i \in [1, m] \quad \forall j \in [1, n_i]$$

$$Min \quad T_{span} = Max\left(t_{ij}^{map} + t_{ij}^{reduce}\right) \quad \forall i \in [1, m] \quad \forall j \in [1, n_i]$$

Combining (3) and (7), (2) and (4), we get the expressions of T_{total} and T_{span} respectively:

$$\begin{aligned} T_{total} &= T_{total}^{map} + T_{total}^{reduce} \\ &= \frac{NS}{B} + \frac{S(N-\gamma N_a)(N-N_a)}{kBN\gamma} + \frac{N^2 S}{\sum_{i=1}^{m}\sum_{j=1}^{n_i}\mu_{ij}[N-(1-\gamma)N_a]} \\ &\quad + \frac{[(2k+1)N-(k+1)N_a]S_r n_r}{kB} - \frac{(N-2N_a)S_r n_r}{B(n_r+k-1)} \end{aligned} \tag{8}$$

$$\begin{aligned} T_{span} &= Max\left(t_{ij}^{map} + t_{ij}^{reduce}\right) \\ &= \frac{S+3S_r}{B} + S/\sum_{i=1}^{m}\sum_{j=1}^{n_i}\mu_{ij} \\ &\quad + Max\left\{\left(\sum_{u,v}^{V_{uv}\in V(A_{ij})} s_{uv} - \sum_{v=1}^{n_{l(A_{ij})}} s_{uv}\right)\Big/B\right\} \end{aligned} \tag{9}$$

Next, we gradually determine the values of the optimization factors in TOMON by solving this optimization problem.

4.2. Optimal number of communication agents

The optimal number of communication agents deployed in a specific cloud platform is determined by calculating the minimum value of (8):

$$\begin{cases} \frac{\partial T_{total}}{\partial k} = 0, \quad \frac{\partial T_{total}}{\partial N_a} = 0 \\ \frac{\partial^2 T_{total}}{\partial k^2} \cdot \frac{\partial^2 T_{total}}{\partial {N_a}^2} - \left(\frac{\partial^2 T_{total}}{\partial k \partial N_a} \right)^2 > 0 \end{cases} \Rightarrow k_{optimal} = \frac{NS n_r \mu_0 \gamma}{|BS(1-\gamma) - (n_r+1) NS_r \mu_0 \gamma|}$$
$$N_a^{optimal} = N + \frac{(k+1) NS_r n_r}{2S} + \frac{kNS_r n_r}{2S(n_r+k-1)} - \frac{kB(1-\gamma)}{2\mu_0 \gamma} \quad (10)$$

Thus, the optimal number of communication agents should be $\lceil k_{optimal} \rceil$.

Note it is possible that value of $k_{optimal}$ exceeds m when $S \gg S_r n_r$, and then the optimal agent number is N, which indicates the scenario that the data transmission overhead is much more than the data processing overhead as discussed in [31], and the data transmission time may dominate the total execution time. Thus, the topology optimization mechanism should deploy as many agents as possible to maximize the platform's communication performance.

4.3. Optimal placement of each communication agent

The optimal placement of each communication agent is also determined by minimizing the value of (8). Once the number of the communication agents (k) is determined, to minimize the total execution latency, the number of VMs co-located with the k agents should be as close to $N_a^{optimal}$ as possible. Therefore, the optimal locations of the k agents $L = \{l(1), l(2), \ldots, l(k)\}$ should satisfy:

$$OPT(L) = \{l(1)\ldots l(k) \in [1, m] | Min|n_{l(1)} + n_{l(2)} + \ldots + n_{l(k)} - N_a^{optimal}|\}$$

The placement optimization problem can be viewed as an extension of the classical 0-1 *Knapsack* problem [32]. We design a dynamic programming algorithm to determine the optimal location of each communication agent, as shown in Algorithm 1.

The key recursive function of our algorithm is defined as follows:

$$S(m, k, N_a) = \begin{cases} S(m-1, k, N_a) \quad if \quad |S(m-1, k, N_a) - N_a| \\ \quad \le |S(m-1, k-1, N_a - n_i) + n_m - N_a| \\ S(m-1, k-1, N_a - n_m) + n_m \quad else \end{cases}$$

Suppose $S(m, k, N_a)$ is the value (amount of VMs) of the optimal solution to the problem "Select k servers among all the m servers, to make the amount of VMs allocated on the k servers as close to N_a as possible", then the value of $S(m, k, N_a)$ can be calcutated by solving two sub-problems:

1. If server m is not included in the optimal location set of all the communication agents (Set L), then the k servers are selected from the first $m-1$ servers (server 1 ~ server $m-1$). In this case, the solution to the problem "Select k servers among all the m servers, to make the amount of VMs allocated on the k servers as close to N_a as possible" is the same as the solution to the sub-problem "Select k servers from server 1~server $m-1$, to make the amount of VMs allocated on the k servers as close to N_a as possible". The value of $S(m, k, N_a)$ in this case can be calculated as: $S(m, k, N_a) = S(m-1, k, N_a)$.
2. If server m is included in the optimal location set of all the communication agents (Set L), then $k-1$ servers are selected from the first $m-1$ servers (server 1 ~ server $m-1$). In this case, the solution to the problem "Select k servers from all the m servers, to make the amount of VMs allocated on the k servers as close to N_a as possible" is the sum of the solution to the sub-problem "Select $k-1$ servers among the first $m-1$ servers, to make the amount of VMs allocated on the $k-1$ servers as close to N_a-n_m as possible" and the amount of VMs allocated on server m (n_m). The value of $S(m,k,N_a)$ in this case can be calculated as: $S(m, k, N_a) = S(m-1, k-1, N_a - n_m) + n_m$.

Which sub-problem will be entered during the recursive procedure is determined by which value is closer to N_a, $S(m-1, k, N_a)$ or $S(m-1, k-1, N_a - n_m) + n_m$.

Consider a more general case, the value of the solution to the sub-problem "Select j servers from the first i servers, to make the amount of VMs allocated on the j servers as close to q as possible" ($S(i, j, q)$) can be calculated as:

$$S(i, j, q) = \begin{cases} S(i-1, j, q) \quad if \quad |S(i-1, j, q) - q| \\ \quad \le |S(i-1, j-1, q - n_i) + n_i - q| \\ S(i-1, j-1, q-n_i) + n_i \quad else \end{cases}$$

The terminations of the recursive procedure (calculating the value of $S(i, j, q)$) include the following three cases:

1. If the number of the undeployed communication agents is equal to the number of the candidate servers ($i = j$), then all the remaining candidate servers are included in the optimal agent location set (Set L):
   $$S(i, i, 0) = \ldots = S(i, i, N_a) = n_1 + n_2 + \ldots + n_i \quad \forall 1 \le i \le k$$
2. Suppose the number of VMs located on each server is sequenced in ascending order, thus $n_1 \le n_2 \le \ldots \le n_m$. If the target value ($q$) of the current function ($S(i, j, q)$) is less than n_1, to make the value of $S(i, j, q)$ as close to n_1 as possible, the optimal deployment mechanism should minimize the value of $S(i, j, q)$. In this case, all the j communication agents should be deployed on the first j servers (server 1 ~ server j). Hence we get:
   $$S(i, j, 0) = S(i, j, 1)\ldots = S(i, j, n_1) = n_1 + n_2 + \ldots + n_j$$
3. If there exists only 1 undeployed communication agent, we can determine the location of this agent directly, according to the number of VMs located on the current candidate servers. Thus:
   $$S(i, 1, j) = Min\{|n_1 - j|, |n_2 - j|, \ldots, |n_i - j|\}$$

The optimal agent placement algorithm will repeat the recursive procedure, until it reaches one of the three terminations. More specifically, the algorithm is composed of two parts: the first part (steps 3–6) initializes the values of *Knapsack* solutions in some specific scenarios, which are the terminations of the recursion; the second part (step 7) is the recursive procedure, calculating the solution S (m, k, N_a). The algorithm outputs the optimal location set of all the communication agents (Set L).

4.4. Optimal matching between VMs and communication agents

The optimal matching between VMs and communication agents is determined by minimizing the time span of a MapReduce workflow (T_{span}). According to (9), the time span of a MapReduce workflow is determined by the cross-server data transmission latency of the slowest communication agent in the map phase:

$$Max \left\{ \left(\sum_{u,v}^{V_{uv} \in V(A_{ij})} s_{uv} - \sum_{v=1}^{n_{l(A_{ij})}} s_{uv} \right) \middle/ B \right\}$$

Once the number and locations of the communication agents are determined, the total cross-server traffic is fixed. Thus, to minimize the maximum transmission overhead of all the agents, the optimal matching strategy should assign the total cross-server traffic equally to the k agents.

Finding the absolute optimal solution to the uniform distribution of cross-server traffic is an NP-hard problem. As is shown in Algorithm 2, we propose a greedy algorithm to solve this problem and provide approximate optimal matching between VMs and agents. The general idea of this algorithm is to assign the VM with the maximum transmission volume to the agent with the lightest transmission load.

Next we prove that our algorithm is 1.5-approximation for the average case.

Suppose the time span of a cloud-based MapReduce application under the absolute optimal matching mechanism is T_{OPT}, and the time span of the ith VM is t_i. Since the optimal matching mechanism is no better than the absolute even distribution of transmission volume (the total transmission volume cannot be evenly assigned to the k communication agents all the time, due to the unequal sizes of the data chunks), hence we get:

$$T_{OPT} \geq \sum_{i=1}^{h} t_i / k \tag{11}$$

Each communication agent is mapped with at least one VM, hence the time span of the whole MapReduce application is longer than or equal to the longest time span of all the h $(h = N - N_a)$ VMs:

$$T_{OPT} \geq Max\{t_1, t_2, ..., t_h\}$$

Suppose the h VMs are sequenced in descending order according to their transmission volume, thus: $s_1 \geq s_2 \geq ... \geq s_h$. The transmission time span of a VM is correlated to its transmission volume, thus: $t_1 \geq t_2 \geq ... \geq t_h$.

Normally, the amount of VMs is much larger than the amount of communication agents in a virtual cluster. According to the *Pigeon Hole Principle*, at least two VMs are matched to the same communication agent. Suppose the fth and gth VM are matched to the same agent, then we can calculate another lower bound of T_{OPT} as:

$$T_{OPT} \geq t_f + t_g \geq 2t_h \tag{12}$$

Suppose the time span of the cloud-based MapReduce application under TOMON mechanism is T_{TOM}. In the "Agent-VM" matching phase of TOMON, the VM with the heaviest transmission volume will be scehduled to take precedence over any other VM. Since the h VMs are sequenced in descending order according to their transmission volume, they will be scheduled sequentially from V_1 to V_h.

In the average case, the transmission volume of each VM is relatively fixed (the configurations of all the VMs are the same); and then the overall time span is determined by the communication agent which handles the most VMs, i.e. the communication agent of V_h. Suppose V_h is mapped to A_p under TOMON, then the transmission time span of A_p can be viewed as the overall time span of the MapReduce application (T_{TOM}).

Consider the time point when the matching of $V_{h\text{-}1}$ is determined, and A_p is the agent with the lightest transmission load at this time point. Since each VM is assigned to the agent with the lightest transmission load at each step, and V_h is matched to A_p at the hth step; hence A_p is the agent with the lightest transmission load after the $(h-1)$th step. The total transmission timespan of A_p after the $(h-1)$th step is $T_{TOM} - t_h$. Thus we get:

$$k(T_{TOM} - t_h) \leq \sum_{i=1}^{h-1} t_i = \sum_{i=1}^{h} t_i - t_h \tag{13}$$

Subsituting (11) and (12) into (13), we can calculate the upper bound of T_{TOM} as follows:

$$k(T_{TOM} - t_h) \leq \sum_{i=1}^{h} t_i - t_h$$

$$\Rightarrow T_{TOM} \leq \sum_{i=1}^{h} t_i / k + \left(1 - \frac{1}{k}\right) \cdot t_h$$

$$\Rightarrow T_{TOM} \leq T_{OPT} + \left(1 - \frac{1}{k}\right) \cdot \frac{T_{OPT}}{2}$$

$$\Rightarrow T_{TOM} \leq \left(\frac{3}{2} - \frac{1}{2k}\right) \cdot T_{OPT}$$

Therefore, the "Agent-VM" matching policy in TOMON is 1.5-approximation for the average case.

5. Experiment

This section evaluates TOMON in simulation environment as well as real cloud platform. By comparing the performance of the same cloud-based MapReduce application under different virtual network topologies, we validate TOMON mechanism and analyze the impacts of the three optimization factors (agent number, agent placement and matching strategy) on the performance of the MapReduce application.

5.1. Evaluation of TOMON in real cloud platform

In this experiment, we deploy a local cloud computing platform using OpenStack software (release Havana), and construct a MapReduce testbed based on this cloud platform. After that, we generate big data processing jobs to our testbed in accordance with the real job arrival rate shown in [33], and evaluate the performance of the same cloud-based MapReduce application under different virtual network topologies.

Our cloud computing platform is built using 21 physical servers. Some of the servers have high-end configuration of 24 core 2.30 GHz CPU, 96 GB memory, 4T Disk and 4 NICs. The detailed information of each server is illustrated in Table 2.

Table 2
Detailed information of cloud servers.

Server quantity	Server configuration		
	CPU	Memory	Disk
2	24	96 GB	4 TB
4	16	32 GB	150 GB
12	8	8 GB	500 GB
3	4	8 GB	300 GB

As shown in Fig. 5, centralized architecture is used to deploy the physical cluster: one server is used as the cloud controller which installs all the OpenStack Nova services; the other 20 servers are installed with OpenStack Nova-compute service, which act as compute nodes to provide virtual resources. Each compute node may host several VMs, and the controller is responsible for monitoring the status of the VMs allocated on the compute nodes. All the servers are connected to a single switch. Our cloud-based MapReduce testbed platform is deployed using 100 VMs allocated on the 20 compute nodes. The configurations of all the VMs in the cloud-based MapReduce cluster are the same (1VCPU, 1 GB memory, 20 GB disk). We employ WordCount as the MapReduce job to test the performance of our testbed, and apply the document package from Wikipedia as the input data. The sizes of our input datasets range from 10 GB to 100 GB.

Using the Restful APIs (e.g. Create Network, Schedule router to L3 agent, etc.) provided by OpenStack, we implement TOMON mechanism and integrate TOMON into OpenStack Neutron component to automatically construct virtual networks in our physical platform. In Software Code 1, an example of Python code to realize optimal matching strategy in TOMON is presented. The open source of TOMON is in progress.

We generate input data to our cloud-based MapReduce testbed in accordance with the task arrival rate shown in real records [33]. The size of the input data generated at each time point is proportional to the task execution latency in the records. Using OpenStack Neutron, we deploy virtual networks with different topologies on

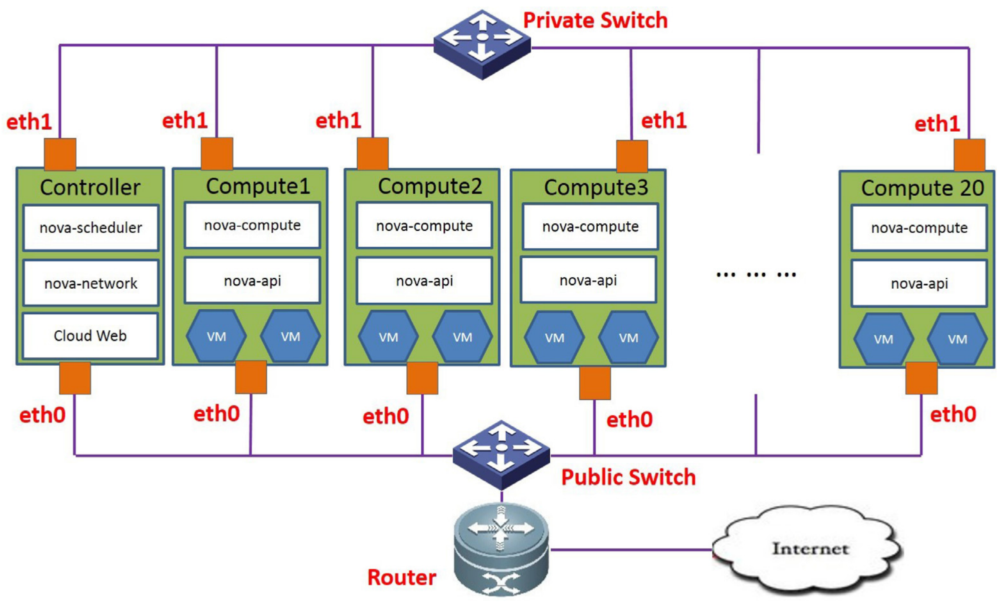


Fig. 5. Centralized physical architecture of the cloud computing platform.

```python
def LoadBalancing(restVM, agentNumber):
    restVM = sorted(restVM, key = lambda d:d[1], reverse = True)
    restVMNumber = len(restVM)
    agentLoad = [0 for i in range(agentNumber)]
    mapping = []

    minAgentId = 0
    for i in range(restVMNumber):
        agentLoad[minAgentId] += restVM[i][1]
        mapping.append([restVM[i][0], minAgentId+1])
        minAgentId = 0
        for j in range(agentNumber):
            if agentLoad[j] < agentLoad[minAgentId]:
                minAgentId = j
    mapping = sorted(mapping, key = lambda d:d[0])
    return mapping
```

Software Code 1. Example of Python code used to realize optimal matching strategy in TOMON.

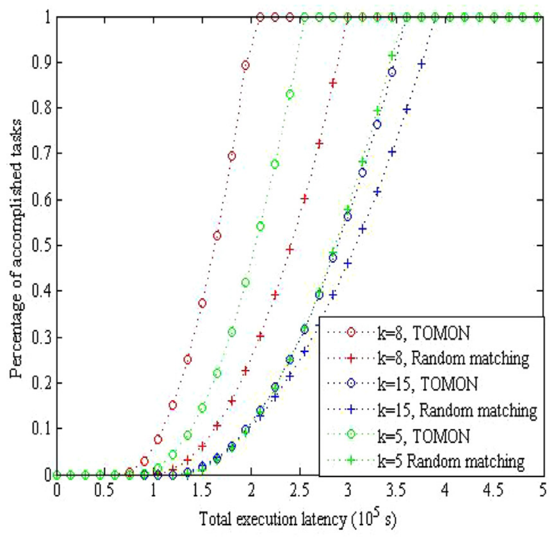
the same physical cluster. Average size of the input data generated during 1 h is about 200 GB, and we evaluate the steady-state performance of our testbed platform under this workload.

Fig. 6A shows the performance of a cloud-based MapReduce application under different virtual network topologies. The input data is the same in each scenario. We change the amount of agents deployed in our platform, and once the agent number is determined, two different virtual networks are deployed for performance evaluation: one virtual network is deployed using TOMON (both optimal placement and matching strategies are used), the other is deployed using optimal placement strategy only (VMs are randomly assigned to agents).

Applying TOMON to our cloud-based MapReduce testbed, we theoretically calculate the optimal number of communication agents to be deployed in our platform, which is $[k_{optimal}] = 8$. From Fig. 6A, we can see that the cloud-based MapReduce cluster attains optimal performance when $k = 8$ using TOMON, which is consistent with the theoretical modeling result. Increasing or decreasing the number of agents ($k = 15$ and $k = 5$) will improve the platform's data transmission rate or data processing rate respectively; however, it will damage the right balance between the data transmission and data processing rate of a virtual cluster, and consequently the overall performance of the cloud-based MapReduce application is not optimal as shown in Fig. 6A. Moreover, the testbed platform attains better performance (up to 40% speedup in the best case) in almost all the scenarios when the virtual network is built using TOMON, since our optimal matching strategy minimizes the overall cross-server communications as well as the time span of a cloud-based MapReduce application. The experiment results validate both the optimal agent number and optimal matching strategy of TOMON.

To further evaluate the two factors (optimal agent number and optimal matching strategy) individually, we analyze the performance of our MapReduce testbed with different input data sizes. Four virtual networks are deployed in our physical platform using different mechanisms: (a) single-host virtual network; (b) multi-host virtual network constructed using five agents and optimal matching strategy; (c) multi-host virtual network constructed using optimal number of agents and random matching strategy; (d) multi-host virtual network constructed using TOMON. The size of the input data increases from 50 GB to 500 GB, and the total execution latency of each MapReduce application is shown in Fig. 6B.

We can see from the figure that multi-host deployment mechanism improves the perfomormance of the virtual cluster, especially when the input data size is large. The virtual MapReduce cluster attains around 150%–300% performance improvement under the multi-host deployment compared with the single-host deployment. Comparing the perfomance changing trends of the MapRe-



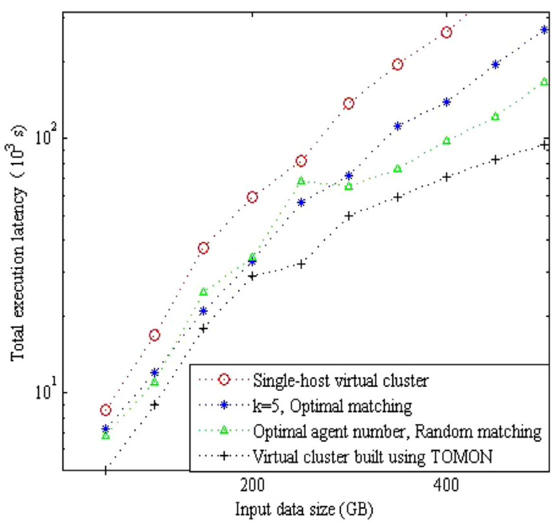


Fig. 6. Performance of real MapReduce applications in different virtual network topologies.

duce clusters under mechanisms (b) and (c), we find that, when the input data size is small (less than 200 GB), the impact of matching strategy on the overall performance is more obvious compared with agent number. However, as the size of the input data becomes larger (over 300 GB), the agent number gradually becomes the dominant factor that affects the overall performance of a cloud-based MapReduce cluster; thus, the perfomance of the MapReduce cluster under mechanism (c) is better than mechanism (b) as shown in Fig. 6B.

This phenomenon is caused by the time distribution of different MapReduce operations. When the size of the input data is small, the data loading operation (the first sub-phase in the map phase, as described in Section 3.2) from local storage systems to the cloud-based HDFS (Hadoop Distributed File System) does not take so much time, and the data transmission latency of a cloud-based MapReduce application is mainly determined by the shuffling operations [3]. In this scenario, the optimal deployment mechanism should balance the load of each agent first, to minimize the operation time span. Thus, the optimal matching strategy improves the overall performance of the MapReduce testbed more significantly. However, when the size of the input data is large enough, compared with the data transmission time spent in the shuffling operations, the time used in loading data into the cloud-based HDFS is much longer, and the optimal deployment mechanism should increase the number of the communication agents to minimize the data loading latency first. Therefore, the agent number becomes the dominant factor in this scenario.

5.2. Evaluation of TOMON in simulation environment

To make our topology optimization mechanism more convincible, we further validate our agent placement policy in simulation environment. We simulate the scenario where MapReduce applications are executed on a cloud computing platform composed of 100 servers. The tasks simulated here are also WordCount. 1500 VMs with the same configuration are allocated on the 100 servers to deploy the cloud-based MapReduce cluster. The maximum number of VMs can be co-located on a large-type server is 100, on a median-type server is 30, and on a small-type server is 10. Among the 100 servers, 15 are large-type servers, 35 are median-type servers, 50 are small-type servers. The load of each server is set randomly between 0.25 and 0.75, and the performance degradation trend of the co-located VMs is simulated using formula (1). Physical architecture of this simulated cloud platform is also centralized. We perform the virtual network construction procedures (e.g. deploy L3 agents, deploy ovs plugins, construct virtual networks, etc.) based on OpenStack Neutron in our simulator, and record the time used to construct a virtual network using TOMON for scalability analysis. This paper concentrates on the evaluation of network deployment mechanisms; thus, our simulator only performs the operating procedures of the Neutron component. The other OpenStack components (e.g. Nova, Glance, etc.) are not reconstructed in our simulator. We assume that the VMs have already been allocated on the physical servers based on OpenStack Nova and Glance components, and we focus on analyzing the virtual network deployment procedures based on OpenStack Neutron. In order to evaluate the optimal agent placement mechanism in TOMON, we establish four virtual networks using different agent placement mechanisms: (a) random deployment of agents; (b) deploy communication agents on the servers with the maximum number of VMs: greedy policy that takes the data transmission latency into first consideration; (c) deploy communication agents on the servers with the minimum number of VMs: another kind of greedy policy, in which data processing latency tends to dominate all the other factors; (d) optimal agent placement policy in TOMON. By comparing the performance of the platform under mechanisms (b), (c) and (d), we further evaluate whether the optimal agent placement strategy in TOMON has optimally balanced the effects of the two dominant but competing factors: data processing rate and data transmission latency.

Fig. 7A shows the total execution latencies of the same cloud-based MapReduce application under the four different agent placement policies. Since both the data transmission latency and data processing latency are considered and optimized, the testbed platform attains better performance (up to 36% performance improvement when $k = [k_{optimal}] = 61$) using TOMON. We also find from the figure that when the amount of deployed agents is small (less than 30), compared with TOMON, the platform gains equal performance using the greedy policy (b). The reason is that the communication performance is the bottleneck performance of the platform when the deployed communication agents are scarce, and the optimal agent placement mechanism should maximize the communication performance of the platform at first; thus the optimal agent placement strategy in TOMON is the same as that in the greedy deploy-

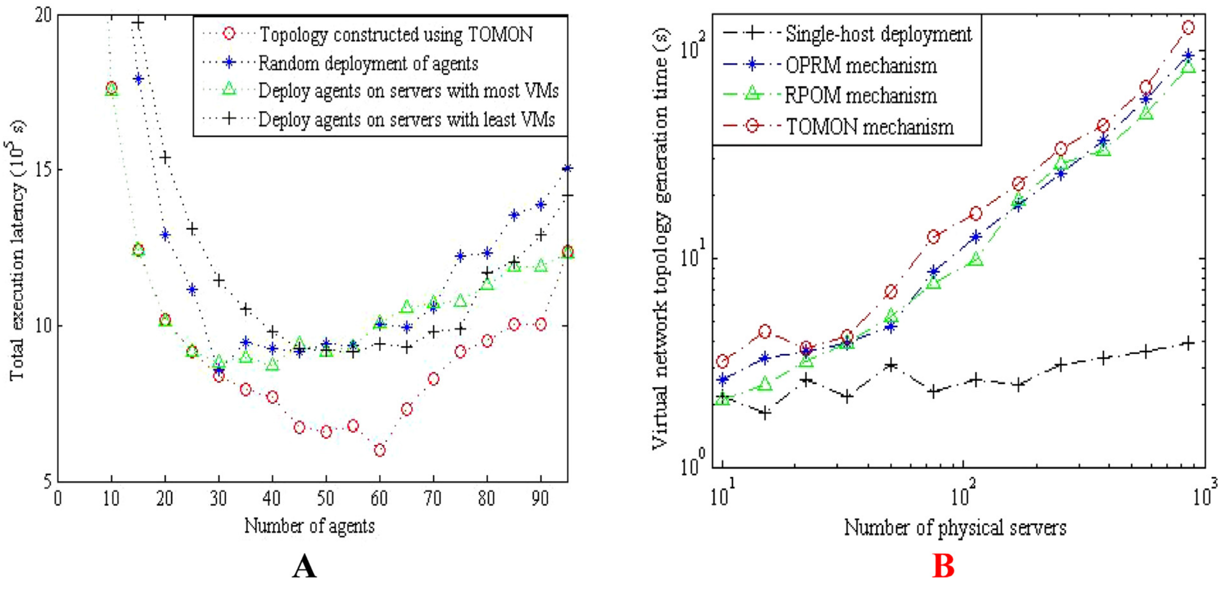


Fig. 7. Performance of the cloud-based MapReduce frameworks in simulation environment.

ment policy (b). However, as the number of the agents increases, the communication performance of the paltform becomes better, and the data processing rate plays an increasingly important role in the overall performance optimization. Without considering the data processing rate, the greedy deployment policy (b) is not applicable in this scenario. On the contrary, TOMON does well all the time since it strikes the right balance between the data transmission and data processing performance, and optimizes bottleneck performance of the virtual cluster in each scenario. The experimental results validate both the optimal agent number and optimal agent placement strategy in TOMON.

Next, we evaluate the scalability of TOMON mechanism using different scales of physical clusters. We establish five virtual networks based on different mechanisms: (1) single-host deployment mechanism; (2) OPRM mechanism: deploy the virtual network based on the **o**ptimal agent placement and random matching strategies; (3) RPOM mechanism: deploy the virtual network based on the random agent placement and optimal matching strategies; (4) TOMON mechanism: both the optimal agent placement and optimal matching strategies are used. Mechanisms (1)–(3) are used as comparison mechanisms to evaluate the time complexity of the optimal agent placement strategy and the optimal matching strategy in TOMON. The scale of the physical cluster increases from 10 servers to 1000 servers.

Fig. 7B shows the time used to generate a virtual network topology under each deployment mechanism. The single-host mechanism takes nearly constant time to generate virtual network topologies, since it does not need to calculate the agent locations and the matching strategies; however, the performance of the single-host virtual network is not acceptable when the size of the input data is large, as shown in Fig. 6B. The topology generation time of OPRM, RPOM and TOMON mechanisms increase with the scale of the physical datacenter, since they need more platform information to determine the virtual network topologies. However, although additional information is required and more calculations are executed in TOMON compared with OPRM and RPOM, the total execution latencies of the three mechanisms (OPRM, RPOM and TOMON) do not show obvious time gaps. The reason is that both optimal placement and optimal matching algorithms in TOMON are with polynomial time complexities as shown in Algorithms 1 and 2, and the additional calculations in TOMON do not increase

Algorithm 1 Optimal placement algorithm of agents.

```
1: Input: $S_r$, $S$, $\mu_0$, $\gamma$, $B$, $m$, $n_r$, $k$, $\{n_1, n_2, ...n_k\}$
2: Output: Optimal locations of communication agents: $L$
           Value of optimal solution $S(m, k, N_a)$
3: Initialize:$L \leftarrow \phi$; calculate $N_a$ according to (10)
4: Initialize: $S(i, i, 0)=...S(i, i, N_a)=n_1+n_2+...+n_i$
5: Initialize: $\forall i \in [1, m]$  $\forall j \in [1, n_i]$
           $S(i, j, 0)=S(i, j, 1)...=S(i, j, n_1)=n_1+n_2+...+n_j$
6: Initialize: $\forall i \in [1, m-k+1]$  $\forall j \in [1, N_a]$
           $S(i, 1, j)=Min\{|n_1-j|, |n_2-j|, ..., |n_i-j|\}$
7: Extended Knapsack recursion:
   for $q:=N_a$ to $n_1$ do
   if $|S(i-1, j, q)-q| \le |S(i-1, j-1, q-n_i)+n_i-q|$ then
         $S(i, j, q)=S(i-1, j, q)$
   else $S(i, j, q)=S(i-1, j-1, q-n_i)+n_i$ $L \leftarrow L\cup\{l(i)\}$
   end for
8: return $L$ and $S(m, k, N_a)$
```

Algorithm 2 Approximate optimal matching algorithm.

```
1: Input: $h$: number of VMs that are not co-located with their agents.
           $(h=N- N_a)$
         $S=\{s_1, s_2, ..., s_h\}$: transmission volume of each VM
2: Output: $M$: matching strategy between the agents $A=\{A_1, A_2, ..., A_k\}$
         and VMs $V=\{V_1, V_2, ..., V_h\}$
3: Initialize:$M \leftarrow \phi$, $load(1)=load(2)=...=load(k)=0$
4: for $i:=1$ to $h$ do
5:    Find max value $s_i$ in $S$
6:    Find min $load(j)$ in $\{load(1), load(2),..., load(k)\}$
7:    $M \leftarrow M\cup\{A_j\leftrightarrow V_i\}$
8:    $load(j)=load(j)+s_i$, $S \leftarrow S-\{s_i\}$
9: end for
10: return $M$
```

the overall time complexity of this mechanism. Thus, TOMON is scalable enough to be used in large-scale data centers.

5.3. Importance evaluation of optimization factors in TOMON

Next, we evaluate the importance of the three optimization factors (agent number, agent placement and matching strategy) in TOMON. We deploy four virtual networks in the same data center, using different deployment mechanisms: 1) random selection of all the three optimization factors; 2) optimal agent number, random agent placement and matching strategy; 3) optimal agent number

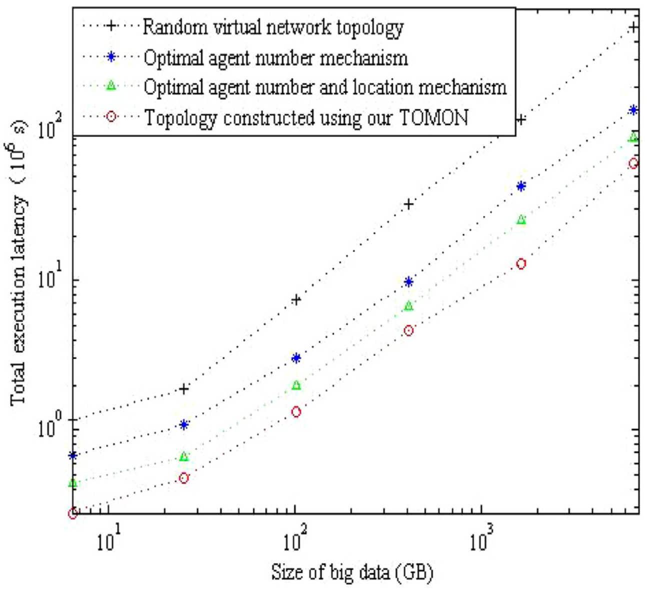


Fig. 8. Importance evaluation of each optimization factor in TOMON.

and placement, random matching strategy; 4) optimal agent number, placement and matching strategy (TOMON). Each following mechanism optimizes one more factor based on its former mechanism.

Fig. 8 exhibits the total service execution latencies of the platform under the four deployment mechanisms and processes the input data with different sizes. We notice that the performance of the platform is improved, whatever additional factor is optimized. And the performance improvement is more remarkable when the size of the input data becomes larger. Thus, larger size of input data makes the impacts of the three optimization factors on the performance of the platform more remarkable. Moreover, the performance improvement is more obvious when the agent number is optimized, which indicates that agent number is the most significant factor in TOMON.

5.4. Applicability analysis of TOMON

At last, we discuss the applicability of TOMON mechanism. TOMON mechanism is implemented based on the multi-host virtual networking model provided by OpenStack Neutron of releases Grizzly and Havana. In the latest OpenStack releases (e.g. Juno and Kilo), multi-host deployment of the communication agents are not supported by Neutron temporarily, since the stability of the multi-host virtual networks cannot be guaranteed. Hence, TOMON cannot be integrated in OpenStack Neutron of the latest releases. However, in order to strengthen the support of OpenStack to communication-intensive applications, multi-host networking model has been realized in OpenStack Nova component of releases Juno and Kilo. Therefore, although our TOMON mechanism cannot be integrated into Neutron in the latest OpenStack releases, it can be implemented based on the multi-host networking model provided by Nova as well. Different from the former implementations, in OpenStack Juno and Kilo release, TOMON is used to determine the optimal deployment of the nova-network plugins but not the communication agents. The optimal number and optimal placement of the nova-network plugins can also been calculated in the same way. Thus, TOMON mechanism can be integrated into Nova component, when using OpenStack releases Juno and Kilo.

6. Conclusion and future work

In this paper, we analyzed the impact of virtual network topologies on the performance of cloud-based big data applications, studied the detailed procedures of cloud-based MapReduce operations in multi-host virtual networks built using OpenStack, formulated the data transmission and data processing overheads of the MapReduce workflows, and put forward TOMON mechanism to optimize the virtual network topologies. TOMON mechanism struck the right balance between the data transmission latency and the data processing rate of a cloud-based MapReduce cluster, and further improve the performance of the cloud-based big data applications compared with other greedy deployment policies. Our work took the first step towards providing optimal deployment mechanism of multi-host virtual networks based on OpenStack Neutron.

In our future work, we plan to improve TOMON mechanism for the optimal deployment of virtual networks on large-scale physical data centers. Firstly, different from the evaluations of TOMON in the centralized physical datacenters shown in this paper, we will further evaluate TOMON mechanism on the large-scale data centers with hierarchical architectures. Secondly, by providing additional performance evaluation mechanisms, our future research will try to evaluate the largest scale of the physical data centers that OpenStack multi-host virtual network can support (When the scale of the physical data center is large enough, the data transmission latency between two physical servers will be long enough, and the virtual network topology may be not the dominant performance factor in this scenario). Finally, based on the experimental results, we will revise TOMON mechanism to accommodate the large-scale data centers with more complex physical architectures. We are also going to explore the properties of the heterogeneous virtual MapReduce clusters, and introduce additional metrics into TOMON to improve this topology optimization mechanism.

Acknowledgment

This work is supported by the Research Fund of Tsinghua-Cisco Joint Labs under Grant No. TCJ2013A004.

References

[1] BigQuery, https://cloud.google.com/products/bigquery/.
[2] Amazon EMR, http://aws.amazon.com/cn/elasticmapreduce/.
[3] Y. Yuan, H. Wang, D. Wang, J. Liu, On interference-aware provisioning for cloud-based big data processing, IEEE/ACM IWQoS, 2013.
[4] E.E. Schadt, M.D. Linderman, J. Sorenson, L. Lee, G.P. Nolan, Computational solutions to large-scale data management and analysis, Nat. Rev. Genetics 11 (9) (2010) 647–657.
[5] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, ACM Commun. 51 (1) (Jan. 2008) 107–113.
[6] Apache Hadoop, http://hadoop.apache.org.
[7] M. Zaharia, A. Konwinski, A.D. Joseph, R. Katz, I. Stoica, Improving mapreduce performance in heterogeneous environments, in: Proceedings of the OSDI conference, 2008.
[8] H. Chang, M. Kodialam, R.R. Kompella, T.V. Lakshman, M. Lee, S. Mukherjee, Scheduling in mapreduce-like systems for fast completion time, IEEE INFOCOM, 2011.
[9] F. Chen, M. Kodialam, T. Lakshman, Joint scheduling of processing and shuffle phases in MapReduce systems, IEEE INFOCOM, 2012.
[10] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, E. Harris, Reining in the outliers in map-reduce clusters using mantri, USENIX OSDI, 2010.
[11] J. Tan, S. Meng, X. Meng, L. Zhang, Improving reducetask data locality for sequential mapreduce jobs, IEEE INFOCOM, 2013.
[12] M. Zaharia, D. Borthakur, J.S. Sarma, K. Elmeleegy, S. Shenker, I. Stoica, Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling, Eurosysl Conference, 2010.
[13] K. Kambatla, A. Pathak, H. Pucha, Towards optimizing hadoop provisioning in the cloud, *Proceedings of USENIX HotCloud*, 2009.
[14] Y. Geng, S. Chen, Y. Wu, R. Wu, G. Yang, W. Zheng, Location-aware mapreduce in virtual cloud, IEEE International Conference on Parallel Processing (ICPP), 2011.

[15] M. Li, D. Subhraveti, A.R. Butt, A.A. Khasymski, P. Sarkar, Cam: a topology aware minimum cost flow based resource manager for mapreduce applications in the cloud, in: *Proceedings of the 21st ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, 2012.
[16] M. Alicherry, T.V. Lakshman, Optimizing data access latencies in cloud systems by intelligent virtual machine placement, IEEE INFOCOM, 2013.
[17] H. Herodotou, F. Dong, S. Babu, No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics, ACM SOCC, 2011.
[18] F. Tian, K. Chen, Towards optimal resource provisioning for running MapReduce programs in public clouds, IEEE CLOUD, 2011.
[19] J. Lin, A. Bahety, S. Konda, S. Mahindrakar, Low-Latency, High-Throughput Access to Static Global Resources within the Hadoop Framework, University of Maryland, College Park, USA, 2009 *Technical Report HCIL-2009-01*.
[20] OpenStack cloud software, http://www.openstack.org/.
[21] H. Herodotou, F. Dong, S. Babu, No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics, in: Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC), 2011.
[22] Z. Zhang, L. Cherkasova, B.T. Loo, Performance modeling of MapReduce jobs in heterogeneous cloud environments, in: Proceedings of *6th IEEE International Conference on Cloud Computing*, 2013, pp. 839–846.
[23] M. Alicherry, T.V. Lakshman, Network aware resource allocation in distributed clouds, in: Proceedings of *IEEE INFOCOM Conference*, 2012, pp. 963–971.
[24] M. Yu, Y. Yi, J. Rexford, M. Chiang, Rethinking virtual network embedding: substrate support for path splitting and migration, ACM SIGCOMM Comput. Commun. Rev. 38 (2008) 19–29.
[25] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, J. Wang, Virtual network embedding through topology-aware node ranking, ACM SIGCOMM Comput. Commun. Rev. 41 (2011) 39–47.
[26] S. Zhang, Z. Qian, J. Wu, S. Lu, An opportunistic resource sharing and topology-aware mapping framework for virtual networks, in: Proceedings of *IEEE INFOCOM Conference*, 2012, pp. 2408–2416.
[27] J. Lu, J. Turner, Efficient Mapping of Virtual Networks onto a Shared Substrate, Washington University: School of Engineering & Applied Science, 2006.
[28] L. Gong, Y. Wen, Z. Zhu, T. Lee, Toward profit-seeking virtual network embedding algorithm via global resource capacity, in: Proceedings of *IEEE INFOCOM Conference*, 2014, pp. 1–9.
[29] I. Houidi, W. Louati, D. Zeghlache, A distributed virtual network mapping algorithm, in: Proceedings of *IEEE International Conference on Communications (ICC)*, 2008, pp. 5634–5640.
[30] I. Houidi, W. Louati, D. Zeghlache, A distributed and autonomic virtual network mapping framework, in: Proceedings of *4th International Conference on Autonomic and Autonomous Systems (ICAS)*, 2008, pp. 241–247.
[31] C. Gkantsidis, D. Vytiniotis, O. Hodson, D. Narayanan, F. Dinu, A. Rowstron, Rhea: automatic filtering for unstructured cloud storage, USENIX NSDI, 2013.
[32] R. Andonov, V. Poirriez, S. Rajopadhye, Unbounded Knapsack problem: dynamic programming revisited, Eur. J. Oper. Res. 123 (2) (2000) 168–181.
[33] Logs of Real Parallel Workloads: http://www.cs.huji.ac.il/labs/parallel/workload/logs.html.

Cong Xu received his Bachelor degree in software engineering from Beijing University of Posts and Telecommunications, Beijing, P.R.China, in 2010. He is now a doctorate candidate of Tsinghua University. Cong's research interests include network management, cloud computing, etc.

Jiahai Yang received his M.S. and Ph.D. degrees in computer science from Tsinghua University, Beijing, P.R.China, in 1992 and 2003, respectively. He is now a professor of Tsinghua University. Jiahai's research interests include Internet architecture and its protocols, IP routing technology, network measurement, network management, cloud computing, big data, etc.

Kevin Yin started his industrial career in Motorola Corp and Network General in U.S as software engineer and developer manager. He is now the CTO (Chief Technical Officer) for Cisco Research & Development Center in Greater China. Kevin has been working in network architecture design, signaling and protocols development, Cloud Computing and Internet of Things.

Hui Yu received his Bachelor degree in software engineering from Chongqing University, Chongqing, P.R.China, in 2014. He is now a master student of Tsinghua University. Hui's research interests include cloud computing, etc.