Journal of Systems Engineering and Electronics Vol. 21, No. 1, March 2014, pp.72–80
Available online at www.jseepub.com

Workload-Aware Request Routing in Cloud Data Center Using Software-Defined Networking

Haitao Yuan[1,*], Jing Bi[2] and Bo Hu Li[1]

1. School of Automation Science and Electrical Engineering, Beihang University, Beijing 100083, P. R. China;
2. Department of Automation, Tsinghua University, Beijing 100084, P. R. China

Abstract: Large latency of applications will bring revenue loss to cloud infrastructure providers in cloud data center. The existing controllers of software-defined networking architecture can fetch and process traffic information in network. Therefore, the controllers can only optimize the network latency of applications. However, the serving latency of applications is also an important factor in delivered user-experience for arrival requests. Unintelligent request routing will cause large serving latency if arrival requests are allocated to overloaded virtual machines. To deal with the request routing problem, this work proposes the workload-aware software-defined networking controller architecture. Then, request routing algorithms are proposed to minimize the total round trip time for every type of requests by considering the congestion in network and the workload in VMs. This work finally provides the evaluation of the proposed algorithms on a simulated prototype. The simulation results show that the proposed methodology is efficient using comparison with the existing approaches.

Keywords: cloud data center, software-defined networking, request routing, resource allocation, network latency optimization.

DOI: 10.3969/j.issn.1004-4132.2010.01.013

1. Introduction

Nowadays, cloud data center (CDC) supports multiple different types of applications concurrently and provides services to many application users by sharing the IT resources [1-2]. Furthermore, virtualization mechanisms have been commonly applied to realize the efficient use of physical resources in the CDC [3]. With virtualization mechanisms, the CDC can support multiple applications in virtualized CDC infrastructure. Physical servers are pooled and separated into many isolated virtual machines (VMs) in CDC infrastructure. Each VM in underlying virtualized computing pool only deploys one single type of application. The arrival requests admitted by the CDC need to traverse network in the CDC and to be served by a specific VM in the computing pool in the CDC. Consequently the total round trip time (RTT) of requests consists of the network latency in network and the serving latency in VMs. It has been shown that the latency of applications in the CDC has a significant impact on user-experience and will bring revenue loss to cloud infrastructure providers. For example, authors in [4] show that both Amazon and Google experienced loss of sales and workload under longer application latency. What's more, a half-second latency will result a 20% loss of traffic in Google, and a tenth of a second latency will bring a loss of one percent of Amazons sales.

Recently, the newly emerging software-defined networking (SDN) can provide centralized control of the network by programmable OpenFlow-enabled network devices including SDN controller and forwarding elements (switches or routers) [5]. SDN realizes the separation of control plane and data plane and provides centralized and globally optimized routing decision for the network in the CDC. The control and management plane in traditional network elements is moved to the remote SDN controller. The OpenFlow-enabled network element only needs to contain the data plane in SDN and to provide public OpenFlow APIs to the remote SDN controller. The OpenFlow is the standard protocol for information exchange between the controller plane and data plane [6]. The SDN controller needs to control the network and to provide the functions including forwarding, VPN, security, bandwidth allocation, QoS, network virtualization and load balancing.

More specifically, the SDN controller periodically queries real-time network statistics including network link usage and network element state. Furthermore, SDN makes it easy to scale and manage the CDC network for meeting application needs in real time. Therefore, SDN greatly realizes better network management, higher availability and utilization by high-performance, fine-grained traffic engineering for different applications. Google has announced [7] that it is applying the SDN to interconnect multiple data centers because of the flexibility and efficiency in realizing traffic engineering. It expects that the SDN architecture will bring higher network utilization and re-

duced loss.

There is many open challenges involved in the existing SDN architecture. The existing controllers only query traffic information in network, thus they can only control OpenFlow-enabled switches and optimize the network latency. However, the serving latency in VMs also plays an important role in user-experienced latency. Large serving latency may occur if the arrival requests are allocated to an already overloaded VM. Besides, the routing mechanism in the existing controller such as Floodlight [8] only selects the shortest routing path to the destination VM. This may cause large RTT due to the network congestion or large serving delay in VMs. What's more, each application in CDC is deployed on multiple homogeneous or heterogeneous VMs for the sake of scalability or stability [1,9]. Therefore, there are multiple corresponding VMs which can serve the requests of each application in CDC.

In order to deal with this problem, this paper presents the workload-aware SDN controller architecture. In the architecture, the workload-aware SDN controller consisting of the traditional SDN controller and the cloud controller. The SDN controller can periodically update information of links in network while the cloud controller can query the workload in VMs. Based on the architecture, this paper proposes the workload-aware request routing algorithms (WARRA) to minimize the RTT for every type of requests (CPU-intensive or network-intensive) by considering the congestion in network and the workload in VMs. The proposed algorithms can determine the optimal combination of routing path and the destination VM to minimize the total RTT for every type of application.

To address the challenge of workload-aware request routing in CDC, the work aims to:

(1) Propose a workload-aware SDN controller architecture to consider both the congestion in network and the workload in VMs.

(2) Propose workload-aware request routing algorithms to minimize the total RTT for different types of requests.

(3) Evaluate the proposed algorithms by comparing with existing methods.

The rest of the paper is organized as follows. Section 2 reviews the related work. Section 3 provides the workload-aware SDN controller architecture, and formulates the workload-aware request routing problem. Section 4 describes the proposed workload-aware request routing algorithms in detail. Section 5 evaluates the proposed workload-aware request routing algorithms by trace-driven simulation using the publicly available real-world workload traces. Section 6 concludes the paper.

2. Related Work

This section presents an overview of related papers in this research area, and compares the proposed request routing with the existing works.

2.1. Traffic engineering

There have been some existing researches on the traffic engineering in software-defined networking [10-12]. Authors in [10] evaluate the effectiveness of traffic engineering in an existing network where SDNs are incrementally introduced. In particular, they show the way to leverage the controller to significantly improve network utilization and reduce average delays and packet losses. The work solves the traffic engineering in an hybrid network consisting of traditional switches and OpenFlow-enabled SDN switches. In contrast to the work, we consider the request routing problem in the CDC where all switches support the OpenFlow protocol and can be controlled by the centralized SDN controller. Authors in [11] apply a measurement-based approach to schedule flows through both the shortest and non-shortest paths according to the temporal utilization in network. They also propose a flow scheduling algorithm which can allocate spare data center network capacity to improve the performance of heavily overloaded links. Authors in [12] present the effectiveness of applying traffic-aware placement of VMs to enhance the scalability of network. The analysis of the characteristics of traffic patterns among VMs can produce the optimal allocation of VMs on host machines. However, the aforementioned methods only consider the link utilization or congestion in network. For a given type of requests, the workload in the destination VM also play an important role in user-experienced latency.

2.2. Load-balancing problem

Several recent papers have proposed different methods to tackle the load-balancing problem in data center networks using the concept of controller in SDN [13-16]. Authors in [13] propose that the network performance can suffer in the presence of inconsistent global network view. Uncoordinated changes to the physical network state may generate routing loops, sub-optimal load-balancing, and other undesired application-specific actions. In order to solve the problem, the authors present a arrival-based load balancer control mechanism. The objective of the load balancer is to minimize the maximum link utilization (MLU) in the selected network. However, the minimization of MLU can not guarantee the lowest latency for a giv-

en type of requests. Authors in [14] present algorithms which compute specific wildcard rules for the given traffic distribution. The algorithms can automatically adapt to the load-balancing changes without destroying the existing connections. The main goal is to split workload over the server replicas using wildcard rules. Authors in [15] propose a scheme that brings no communication overhead between servers and users based on job arrival. This scheme can remove the scheduling overhead brought by the critical path of each job. However, they do not distinguish the types of jobs and treat every job equally. The high-priority jobs do not have the privilege to traverse the network. Authors in [16] propose methods to realize the load-balancing between requests in unstructured networks. This work proposes an OpenFlow-based server system to realize the load-balancing for web traffic. The system can effectively reduce response time of web services in unstructured networks built with cheap commodity hardware. This approach can specify the destination server after considering the workload of all servers. However, this work treats every request equally. However, the performance requirements of the arrival requests are different.

In contrast to the previous work, this paper proposes the workload-aware request routing mechanism. The routing mechanism considers both the congestion in network and the workload in the destination VM. In addition, the routing mechanism distinguishes the types of requests and always tries to ensure the performance of high-priority requests.

3. Scenario and performance metrics

This section firstly presents the workload-aware software-defined networking controller architecture. Then, it presents the formulation of workload-aware request routing problem in CDC.

3.1. Architecture overview

The overall response time of requests includes the time spent in network and VMs. The consideration of both the congestion in network and the workload in VMs can minimize the overall response time of requests by intelligently choosing the routing path and the destination VM. In typical SDN architecture, the traditional controller only fetches and processes traffic information in the network, thus the controller only manages and schedules the forwarding elements (OpenFlow-enabled switches) in the network. The traditional controller only can optimize the network latency, i.e., time spent in the network. However, the serving latency, i.e., time spent in the VM, is also important in user-experience. A new arrival request may experience large serving latency if it is unintelligently allocated to an overloaded VM. The proposed workload-aware SDN controller architecture, which is illustrated in Fig. 1, consists of two major component: traditional SDN controller and cloud controller. The former component periodically probes the traffic information (network topology and the utilization of each link) in network and makes decision about selection of routing path for requests. The latter component periodically reports the information about VMs including utilization, the remaining requests in each VM, etc. to the SDN controller. The combination of the two components builds the proposed workload-aware SDN controller.

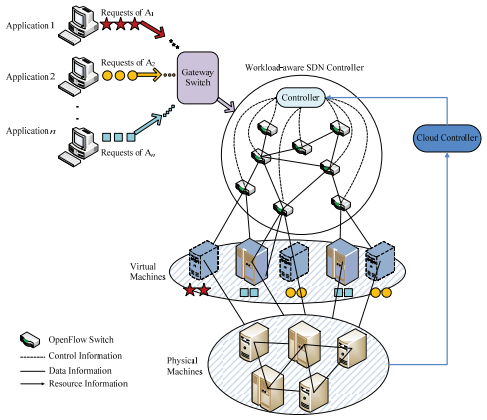

Fig. 1 Workload-aware SDN controller architecture.

Based on the architecture, the proposed workload-aware request routing mechanism aims to minimize the RTT for different types of requests by considering the congestion in network and the workload in VMs. The optimal request routing can be achieved by considering many essential factors including the request arrival rates, the types of different requests, etc.

Nowadays, each application is deployed on several homogeneous or heterogeneous VMs for the sake of scalability or stability in the CDC. Therefore requests of each application cannot be served by all VMs in the CDC. For a specific type of requests, there are several corresponding VMs that can serve the requests. Note that the workload-aware SDN controller does not process the requests but decides the routing path and installs flow entries into switches in the routing path to the destination VM. Given a type of requests, the workload-aware SDN controller needs to make two decisions. The first decision determines the potential available VMs which are deployed the corresponding application. The second decision specifies the routing path to the selected destination VM. For each routing path to the destination VM, the workload-aware SDN controller can periodically fetch the link information including the remaining bandwidth and utilization of the link. The optimal combination of routing path and destination VM can minimize the total RTT. However, the routing mechanism in the existing controller such as Floodlight [8] just chooses

the routing path according to the shortest path first (SPF) policy. This simple SPF policy may cause the congestion in network or large serving delay in VMs, thus the unintelligent policy will cause large RTT.

For a given type of requests, the controller needs to specify the destination VM to serve the requests. There may be several reachable routing paths to the destination VM. The RTT of the specific type of requests is the sum of the time spent in network and the time spent in the destination VM. The requests can be divided into several types including the CPU-intensive requests and the network-intensive requests. Compared with the network-intensive requests, the CPU-intensive requests need to take longer time in VMs and shorter time in network.

3.2. Problem formulation

Based on the proposed workload-aware SDN Controller architecture, the section formulates the problem of workload-aware request routing in CDC.

The workload-aware request routing problem is formulated as follows. Given fixed network topology, the request routing considers the congestion in network and the remaining workload in VMs. In order to minimize the RTT, the request routing can determine the optimal combination of the destination VM and the routing path in the network. In this way, the request routing can realize load balancing of both the VMs and the network, and finally minimize the RTT of requests. We summarize the main notations used throughout this paper in Table 1 for clarity.

Table 1 Notations

Notations	Definition
p_f	admissible routing path for request type f
d_f	destination VM for request type f
λ_f	arrival rate of request type f
n_f	number of VMs which can serve request type f
T	number of request types
$reqLevel_f$	request priority of request type f
$Reach_{f,s,d}$	set of reachable paths for request type f from gateway switch s to destination VM d
BW_f	bandwidth requirement of request type f
$AP_{f,s,d}$	set of admissible paths for request type f from gateway switch s to destination VM d
$SP_{f,s,d}$	the shortest path for request type f from gateway switch s to destination VM d
Cap_e	capacity of link e
R_e	remaining capacity of link e
RT_e	set of request types traversing link e
μ_f^d	serving rate of request type f in VM d
n_f^p	number of switches in path p for request type f
s_f	size of request type f
$BW_{p,f}^i$	allocated bandwidth in the ith switch in routing path p for request type f
$QReqs_f^d$	number of queueing requets of request type f in VM d

Assume each VM only supports a single application corresponding to a type of requests. This work sets the priorities of different types to each requests in advance. A specific type of requests can only be served by VMs deployed with the corresponding type of application.

Given a specific topology and requests of type f, the workload-aware SDN controller needs to specify the final destination VM and the routing path to the destination VM, i.e., d_f and p_f.

For given requests of type f, the RTT is the sum of the time spent in network, $T_{Network}^{p_f}$ and the time in a specific VM, $T_{VM}^{d_f}$. For simplicity, it is assumed that the time spent in the network from and to the corresponding VM are the same.

$$RTT_{p_f,d_f} = T_{Network}^{p_f} + T_{VM}^{d_f} \tag{1}$$

The time spent in the network from (to) the corresponding VM is the sum of time spent in each switch in the routing path p_f, i.e., $\sum_{w=1}^{n_f^p} \frac{1}{BW_{p,f}^w - \lambda_f}$. Therefore, $T_{Network}^{p_f}$ can be calculated as follow.

$$T_{Network}^{p_f} = 2\left(\sum_{w=1}^{n_f^p} \frac{1}{BW_{p,f}^w - \lambda_f}\right) \tag{2}$$

The time spent in a specific VM d_f, $T_{VM}^{d_f}$, is the time to serve the remaining requests and the new arrival request in the VM. Here, $QReqs_f^d$ denotes the queueing requests of type f, which wait for execution in the VM d_f. s_f denotes the size of the new arrival request. μ_f^d denotes the serving rate of request type f in VM d.

$$T_{VM}^{d_f} = \frac{QReqs_f^d + s_f}{\mu_f^d} \tag{3}$$

Given requests of type f, there are multiple destination VMs which can serve the requests. For each destination VM, d_f, there are multiple routing paths which connect the gateway switch and d_f. Let p_f denote an admissible routing path for requests of type f. Therefore, given all possible combinations (p_f,d_f), the optimal combination can be achieved by minimizing the RTT. Therefore, the optimization problem (**P1**) can be formulated as follow.

$$RTT_{p_f,d_f} = 2(\sum_{i=1}^{n_f^p} \frac{1}{BW_{p,f}^i - \lambda_f}) + \frac{QReqs_f^d + s_f}{\mu_f^d}$$

s.t.

$$\sum_{f \in RT_e} BW_f \leqslant Cap_e \tag{4}$$

$$\lambda_f \leqslant BW_{p,f}^w \tag{5}$$

The constraint (4) shows that for each $e \in p_f$, $p_f \in AP_{f,s,d}$, the sum of the total bandwidth requirement of the newly assigned requests which can traverse the link e must be less

than the capacity of link e, Cap_e. Here, $AP_{f,s,d}$ denotes the set of admissible paths for requests of type f from gateway switch s to the destination VM d. The constraint (5) shows that in order to ensure the response time of requests, it is assumed that admission policies for requests have been adopted to control the admitted request arrival rate of each switch in the routing path.

4. Workload-aware request routing algorithms

Based on the proposed workload-aware SDN controller architecture which considers both the congestion in network and the workload in VMs, this section presents the workload-aware request routing algorithms to realize the intelligent request routing. These algorithms run periodically in the workload-aware SDN controller. In order to solve the problem presented in section 3.2, the following several algorithms describe the proposed routing mechanism in this section. Algorithm 1 aims to find the optimal combination of the destination VM and routing path to minimize RTT. Algorithm 2 describes the method to transform $Reach_{f,s,d}$ to $AP_{f,s,d}$. The function $Knapsack$ in the algorithm 2 is described in the algorithm 3.

Algorithm 1 Find the Optimal Combination of VM and Routing Path

Input:
Number of request types: T
Output:
Optimal Combination of VM and Routing Path for Each Request Type f: p_f, d_f

for $f = 1$ to T **do**
  $minimalRTT_f$=$positiveinfinite$;
  for $d = 1$ to n_f **do**
    $minimalRTT_f^d$=$positiveinfinite$;
    for all path p∈$AP_{f,s,d}$ **do**
      $RTT_p = 2\left(\sum_{i=1}^{n_f^p} \frac{1}{BW_{p,f}^i - \lambda_f}\right) + \frac{QReqs_f^d + s_f}{\mu_f^d}$;
      if RTT_p<$minimalRTT_f^d$ **then**
        $minimalRTT_f^d$=RTT_p;
      end if
    end for
    if $minimalRTT_f^d$<$minimalRTT_f$ **then**
      $minimalRTT_f$=$minimalRTT_f^d$;
      d_f=d;
      p_f=p^*; /* Routing path p^* corresponding to $minimalRTT_f$ */
    end if
  end for
end for
return p_f, d_f

The algorithm 3 aims to assign different types of requests to each link e. Different types of requests unfairly share the bandwidth capacity in each link e. The assignment of requests can be formulated as a knapsack problem (KP). The algorithm first orders different types of requests by the $reqLevel$ property in the descending order. Then the algorithm assigns requests to the link e according to the corresponding priority on the condition that the total bandwidth occupied by the assigned requests does not exceed Cap_e. It may be possible that the total bandwidth requirements of requests which can traverse the link e is smaller than Cap_e. In this case, the remaining bandwidth of the link e will be allocated to all types of requests. The amount of bandwidth allocated to each type of requests is proportional to the actual bandwidth requirement of requests. For example, given two types of requests, f_1 and f_2, the actual bandwidth requirements of f_1 and f_2 are 300MBytes/s and 100MBytes/s, respectively. Assume that the capacity of the link e is 500MBytes/s. So the remaining bandwidth, 100MBytes/s, will be allocated to f_1 and f_2 with amount of 75MBytes/s and 25MBytes/s, respectively. Therefore, the final bandwidth of f_1 and f_2 will be 375MBytes/s and 125MBytes/s, respectively.

Algorithm 2 Transform $Reach_{f,s,d}$ to $AP_{f,s,d}$

Input:$Reach_{f,s,d}$
Output:$AP_{f,s,d}$

for all request type f in T **do**
  for all routing path p∈$Reach_{f,s,d}$ **do**
    for all link e in p **do**
      RT_e=$Knapsack(e)$
      if f is not in RT_e **then**
        $Reach_{f,s,d}$=$Reach_{f,s,d}$−$\{p\}$
        $AP_{f,s,d}$=$Reach_{f,s,d}$
      end if
    end for
  end for
end for
return $AP_{f,s,d}$

For each link e, there are multiple types of requests flows, which may traverse the link. All request flows compete with each other for limited bandwidth capacity of the link. The bandwidth requirement of requests of different types is different. The workload-aware SDN controller needs to specify the set of request flows traversing the link to maximize the total priorities of the set without exceeding the bandwidth capacity of the corresponding link e. Let $totalP_e$ denotes the total priorities of request flows which can traverse the link e. So this can be formulated as 0-1 KP, which is a typical combinatorial optimization problem. Here the bandwidth capacity of a link denotes the volume of a knapsack. Each request flow denotes a distinct item that may be put into the knapsack. The bandwidth requirement and the priority of each request flow correspond to the volume and the benefit of the item, respectively. The KP (**P2**) that we intend to solve can be briefly formulated

as follow.

Algorithm 3 Knapsack (Solving 0-1 Knapsack Problem with the Genetic Algorithm)

Input:
e: the link
$elitism$: whether to use elitism or not
$popSize$: number of chromosomes in every population
$chromoSize$: length of every chromosome
$generationSize$: number of generations
$crossRate$: crossover ratio
$mutateRate$: mutation ratio
$volumes$: set of bandwidth requirements of request flows in T
$priorities$: set of priorities of request flows in T
Output:
RT_e for link e

1: initialize the data ($volumes$ and $priorities$)
2: initialize the first population by randomly initializing a population of $popSize$ chromosomes, and the length of each chromosome is $chromoSize$
3: **while** the number of generations is smaller than $generationSize$ or the percentage of the chromosomes with the same fitness (total priorities) in the latest population is smaller than 90% **do**
4: rank($popSize$, $chromoSize$); /*rank the chromosomes in the current population*/
5: select($popSize$, $chromoSize$, $elitism$); /*randomly select chromosomes to reproduce new population using the combination of roulette-wheel selection and elitism.*/
6: crossover($popSize$, $chromoSize$, $crossRate$); /*perform crossover by randomly specifying a position and exchanging the subsequences after the position between two chromosomes to create new offspring for the next population*/
7: mutation($popSize$, $chromoSize$, $mutateRate$); /*perform mutation on the chromosomes after crossover*/
8: replace the current population with the new population
9: **end while**
10: RT_e=$\emptyset$
11: R_e=Cap_e
12: **for all** request type f in T **do**
13: **if** the bit corresponding to request flow f in the calculated best chromosome (solution) is 1 **then**
14: **if** $BW_f \leq R_e$ **then**
15: R_e=R_e−BW_f
16: RT_e=RT_e+{f}
17: **else**
18: break;
19: **end if**
20: **end if**
21: **end for**
22: **return** RT_e

For each link e, maximize $totalP_e$, i.e.,

$$max\ totalP_e = \sum_{f=1}^{T} x_f \cdot reqLevel_f$$

s.t.

$$\sum_{f=1}^{T} x_f \cdot BW_f \leqslant Cap_e \tag{6}$$

$$x_f = \begin{cases} 1, & request\ flow\ f\ traverses\ link\ e \\ 0, & otherwise. \end{cases} \tag{7}$$

The constraints (6) and (7) show that the occupied bandwidth of request flows selected to traversing link e can not exceed the bandwidth capacity of the corresponding link. The 0-1 KP is a typical NP-hard problem, traditional algorithms including dynamic programming, branch and bound, backtracking can not effectively solve it. However, the genetic algorithm (GA) proves to be an efficient algorithm in finding approximately optimal solutions for larger NP problems traditionally viewed as computationally infeasible such as the 0-1 KP. GA can search for close-to-optimal solutions to a problem from a set of potential solutions [17]. GA begins with some possible solutions (chromosomes) called population. Then a new population is created from chromosomes of the old population in oder to get a better population. Solutions selected to create new solutions (offspring) are chosen based on their fitness. The more suitable solutions have the larger chances to reproduce. The process is repeated until the termination condition is satisfied. The basic elements of GA include populations of chromosomes, selection based on fitness, crossover to reproduce new offspring, and possible mutation of the new offspring. So, this paper adopts GA to solve the 0-1 KP for each link in the network.

The implementation of GA combines the most used selection method, roulette-wheel and the elitism, which can significantly improve the performance of the roulette-wheel selection. Roulette-wheel is a typical method of supporting fitness-proportionate selection. The elitism is an approach which first directly copies some of the most fittest chromosomes in the old population to the new one, and then generates the rest of the new population. In this way the elitism can preserve the best solutions. Moreover, the implementation adopts binary encoding where each chromosome is coded as a string of bits, 0 or 1. In addition, the implementation uses single point crossover and performs crossover with a certain probability. What's more, mutation is implemented to prevent GAs from falling into a local extreme. The mutation is performed on each bit position of the chromosome with a small probability, 0.1%. The optimization goal of algorithm 3 is to assign different kinds of request flows to every link in the network within each computational period, such that the total RTT is minimized and the total bandwidth requirement for different flows assigned to each link can not exceed the corresponding link bandwidth capacity. Let N denotes the number of

request flows in every link e. The function which initializes the chromosomes in the first generation has a complexity of O(N). The fitness function, crossover function, and mutation function also have complexities of O(N). Thus, the total complexity of the algorithm 3 is O(N).

Note that there is a knapsack problem in each link, but note that every link is independent of each other so that the multiple knapsack problems can be solved in parallel. Authors in [18] have proved that the employment of parallelism can scale the ability of a controller to 20 million flows per second. So, the calculation of routing path will not bring too much overhead in the experiment.

5. Performance evaluation

To realistically evaluate the performance of the proposed workload-aware request routing, a prototype is developed in Java in this section. This work evaluates the proposed workload-aware request routing algorithms by trace-driven simulation using the publicly available real-world workload traces from Google production systems [19].

Fig. 2 shows the dataset which describes the resource requirement of CPU and memory from Google compute cells for over six hours (370 minutes) in the period of a month in May 2011. During the period of time this web site recorded 1,352,804,107 requests. The workload can be predicted accurately as shown in previous works [20], so the simulation simply adopts the measured requests traffic as the request arrival rates. The information including the link utilization in the network, and workload of VMs needs to be updated every 5 minutes. For the clear demonstration, the evaluation focuses on the requests that arrive during the 6 hours period. The arrival rates of requests of four types are illustrated in Fig. 2. Besides, the priorities of four request flows (task type 1, task type 2, task type 3 and task type 4) illustrated in Fig. 2 are set to 1, 2, 3 and 4, respectively. Besides, the priorities of four request flows (task type 1, task type 2, task type 3 and task type 4) illustrated in Fig. 2 are set to 1, 2, 3 and 4, respectively.

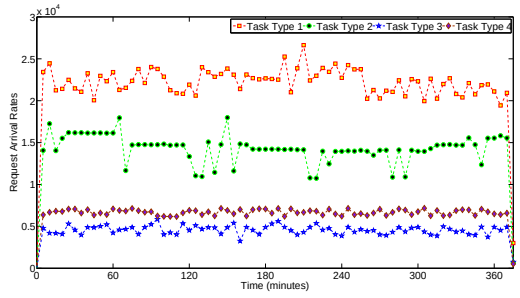

Fig. 2 Google's request arrival rates.

5.1. Simulation experiment setup

This section evaluates the performance of the proposed workload-aware request routing algorithms based on the typical data center network topology, Fat-tree [21]. Note that the proposed routing algorithms are applicable for any type of data center network topology but the proposed algorithms are only evaluated based on the commonly used pod-4 Fat-tree, as illustrated in Fig. 3. The Fat-tree topology with three-layer topology contains edge tier, aggregation tier, core tier, respectively. The core tier bridges the data center with the Internet. There are multiple corresponding VMs for every type of request flow in Fig. 3. Each VM can only serve a specific type of request flow. So, it is meaningful and important to intelligently route the hybrid request flows to minimize the total RTT for every type of request flow. It is assumed that the link rates between host and switch are 1Gbps. In addition, the link rates between switches are also 1Gbps. The simulation environment consists of 16 hosts and 20 OpenFlow-enabled switches. The switches can collect flow information and report it to the centralized workload-aware SDN controller.

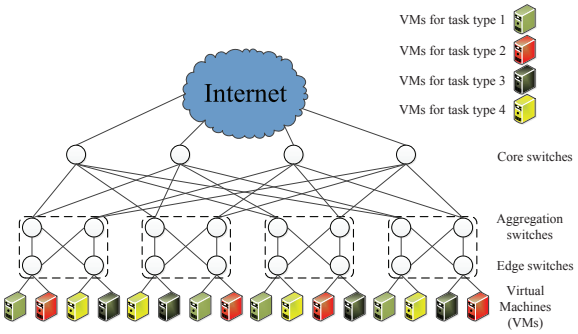

Fig. 3 Fat-tree data center network topology.

The size of large amount of requests in current data center are small ($\leqslant$10KB) [22]. However, the work [23] shows that the mixture of large and small requests is more realistic. Therefore, Table 2 shows the setting of average sizes of four different types of request flows. In addition, the setting of serving rate of request flow f in VM d, $\mu_f^d(f{=}1,2,3,4)$ is also shown in Table 2.

Table 2 Realistic Parameter Setting

f	s_f (KBytes)	μ_f (KBytes per minute)
1	2	3500
2	10	15000
3	50	90000
4	100	180000

5.2. Simulation experiment

This section evaluates the proposed workload-aware request routing algorithms by the comparison with the typical baselines including Open Shortest Path First (OSPF) [24] and random routing [25]. The distance in the OSPF denotes the number of links between the gateway switch

and the destination VM. It's assumed that the OSPF always chooses the leftmost destination VM for a given type of requests. The random routing algorithm differs from the OSPF because the random routing algorithm can select any destination VM other than just the leftmost destination VM for a given type of requests. Therefore, random routing means that the final destination VM is selected randomly from the available corresponding multiple VMs.

Fig. 4, 5, 6 and 7 show the comparison result of the total RTT for requests of type 1, 2, 3 and 4, respectively using WARRA, random routing and OSPF. It can be shown in the four figures that the proposed WARRA performs better than the other two routing algorithms, OSPF and random routing. The OSPF algorithm always chooses the shortest routing path to the destination VM, therefore large RTT will occur if the arrival requests are always scheduled to the shortest routing path. However, the random routing algorithm performs better than the OSPF algorithm because it specifies the routing path randomly and can lower the RTT.

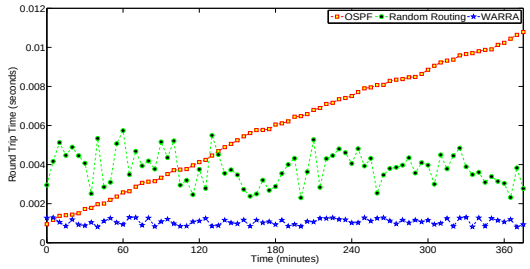

Fig. 4 Comparison result of type 1 requests.

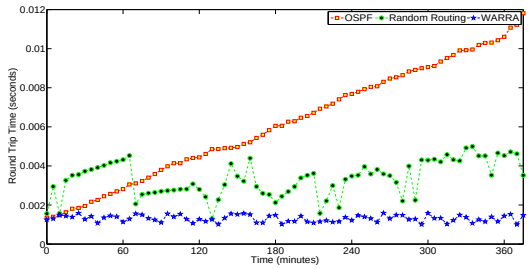

Fig. 5 Comparison result of type 2 requests.

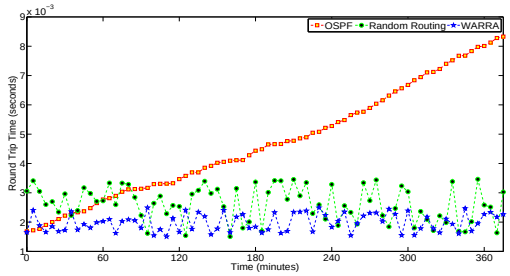

Fig. 6 Comparison result of type 3 requests.

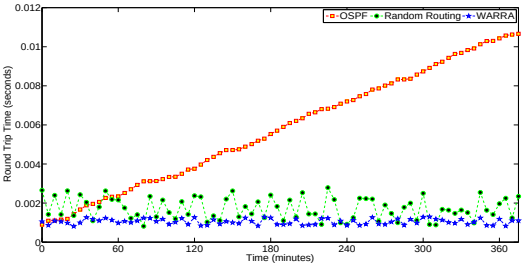

Fig. 7 Comparison result of type 4 requests.

Fig. 8 shows the variance of average utilization of the corresponding VMs for different types of requests using three routing algorithms including WARRA, random routing and OSPF. The utilization of four VMs for the same type of requests over the period of time are different due to the corresponding routing algorithm. Therefore, this paper calculates and compares the average utilization variances of four VMs for the same type of requests with three routing algorithms. It is shown that compared with the other two routing algorithms, OSPF and random routing, the utilization variations of WARRA are the lowest for every type of requests. This means that the WARRA can optimally choose the routing path and destination VM by considering the congestion in network and the workload in VMs. The random routing algorithm just chooses the routing path and destination VM randomly. Therefore this algorithm performs better than the OSPF in most cases. It is understandable that the OSPF always selects the shortest routing path to schedule the requests. Therefore, the performance is the worst compared with the other routing algorithms including WARRA and random routing.

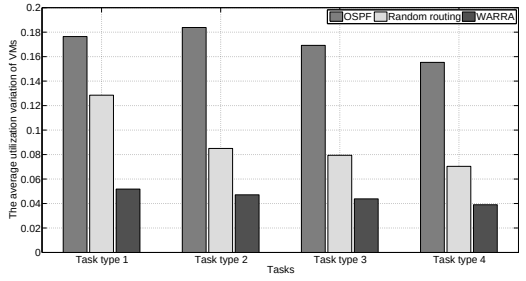

Fig. 8 Variance of average utilization.

Then, Fig. 9 illustrates the average request delays of three routing algorithms. Compared with the baselines including OSPF and random routing, the WARRA decreases the average packet delays of all types of requests. Besides, Fig. 10 shows the packet drop ratios of three routing algorithms. In this figure, the WARRA lowers the packet drop ratio by 0.289% compared with the random routing, and 0.64% compared with the OSPF, respectively. Therefore,

with the consideration of both the network latency of applications and the serving latency in destination VMs, the WARRA performs better than the existing two routing algorithms. Then, the WARRA can minimize the total round trip time for every type of requests by considering the congestion in network and the workload in VMs.

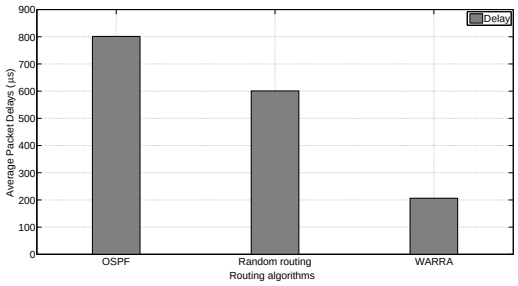

Fig. 9 Average packet delays of WARRA and the baselines.

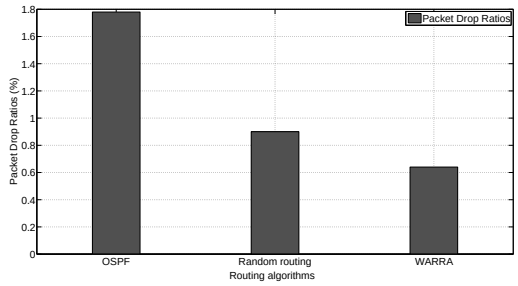

Fig. 10 Packet drop ratios of WARRA and the baselines.

6. Conclusion and future work

The existing controllers in the typical software-defined networking architecture only can optimize the network latency and can not affect the serving latency in virtual machines, so that large serving latency may occur if the arrival requests are allocated to overloaded virtual machines. In order to solve the problem, this paper firstly presents the workload-aware software-defined networking controller architecture. Then, this paper proposes the workload-aware request routing algorithms to minimize the total round trip time for each type of requests by considering the congestion in network and the workload in VMs. Given a type of requests, the proposed algorithms can determine the optimal combination of routing path and destination VM to minimize the corresponding total round trip time. Then, trace-driven simulation is presented using the publicly available real-world workload traces. The simulation result demonstrates that compared with two existing baselines including open shortest path first and random routing, the proposed algorithms can realize less round trip time for every type of requests. In future research, we would like to design a SDN-based request routing algorithms for minimizing round trip time of requests in multiple distributed cloud data centers.

Acknowledgment

This work was supported in part by a grant from the China Postdoctoral Science Foundation (No. 2014M550068).

References

[1] J. Bi, H. T. Yuan, M. Tie, et al. Sla-based optimisation of virtualised resource for multi-tier web applications in cloud data centres. *Enterprise Information Systems*, 2014: 1-25.

[2] M. Armbrust, A. Fox, R. Griffith, et al. A view of cloud computing. *Communications of the ACM*, 2010, 53(4): 50-58.

[3] G. H. Wang, T. E. Ng. The impact of virtualization on network performance of amazon EC2 data center. *Proc. of the 29th Conference on Computer Communications*, 2010: 19.

[4] J. M. Zhu, Z. B. Zheng, M. R. Lyu. DR2: Dynamic Request Routing for Tolerating Latency Variability in Online Cloud Applications. *Proc. of the IEEE Sixth International Conference on Cloud Computing*, 2013: 589-596.

[5] M. Casado, M. J. Freedman, J. Pettit, et al. Ethane: Taking control of the enterprise. *SIGCOMM Computer Communication Review*, 2007, 37(4): 1-12.

[6] N. McKeown, T. Anderson, H. Balakrishnan, et al. Openflow: Enabling innovation in campus networks, *SIGCOMM Computer Communication Review*, 2008, 38(2): 69-74.

[7] C. Y. Hong, S. Kandula, R. Mahajan, et al. Achieving high utilization with software-driven WAN. *Proc. of the ACM SIGCOMM 2013 Conference on SIGCOMM*, 2013: 15-26.

[8] D. Erickson. The beacon openflow controller. *In Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, 2013: 13-18.

[9] J. Bi, Z. L. Zhu, R. Tian, et al. Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center. *Proc. of the Third IEEE 2010 International Conference on Cloud Computing*, 2010: 370-377.

[10] S. Agarwal, M. Kodialam, T. V. Lakshman. Traffic engineering in software defined networks. *Proc. of the 32nd IEEE International Conference on Computer Communications*, 2013: 2211-2219.

[11] F. P. Tso, G. Hamilton, R. Weber, et al. Longer is better: exploiting path diversity in data center networks. *Proc. of the IEEE 33rd International Conference on Distributed Computing Systems*, 2013: 430-439.

[12] X. Q. Meng, V. Pappas, L. Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. *Proc. of the 29th IEEE International Conference on Computer Communications*, 2010: 1-9.

[13] S. Schmid, J. Suomela, Exploiting locality in distributed sdn control. *Proc. of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, 2013: 121-126.

[14] R. Wang, D. Butnariu, J. Rexford. Openflow-based server load balancing gone wild. *Proc. of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*. 2011: 12-12.

[15] A. Nahir, A. Orda, D. Raz. Schedule first, manage later: Network-aware load balancing. *Proc. of the 32nd IEEE International Conference on Computer Communications*, 2013: 510-514.

[16] N. Handigol, S. Seetharaman, M. Flajslik, et al. Plug-n-serve: Load-balancing web traffic using openflow. *ACM SIGCOMM Demo*, 2009.

[17] D. E. Goldberg, J. H. Holland. Genetic algorithms and machine learning. *Machine learning*, 1988, 3(2): 95-99.

[18] X. D. Wang, Y. J. Yao, X. R. Wang, et al. Carpo: Correlation-aware power optimization in data center networks. *Proc. of the 32nd IEEE International Conference on Computer Communications*, 2012: 1125-1133.
[19] C. Reiss, J. Wilkes, J. L. Hellerstein. Google cluster-usage traces. *Google Technical Report*, 2011.
[20] J. M. Tirado, D. Higuero, F. Isaila, et al. Reconciling dynamic system sizing and content locality through hierarchical workload forecasting. *Proc. of the 18th IEEE International Conference on Parallel and Distributed Systems*, 2012: 77-84.
[21] M. A.-Fares, A. Loukissas, A. Vahdat. A scalable, commodity data center network architecture. *ACM SIGCOMM Computer Communication Review*, 2008, 38(4):63-74.
[22] T. Benson, A. Akella, D. A. Maltz. Network traffic characteristics of data centers in the wild. *Proc. of the 10th ACM SIGCOMM Conference on Internet Measurement*, 2010: 267-280.
[23] M. A. Adnan, R. Gupta. Path consolidation for dynamic right-sizing of data center networks. *Proc. of the IEEE Sixth International Conference on Cloud Computing*, 2013: 581-588.
[24] S. U. Malik, S. K. Srinivasan, S. U. Khan. Convergence time analysis of open shortest path first routing protocol in Internet scale networks, *Electronics Letters*, 2012, 48(19): 1188-1190.
[25] A. Chatterjee, N. D. Georganas, and P. Verma, Analysis of a packet-switched network with end-to-end congestion control and random routing, *IEEE Transactions on Communications*, 1977, 25(12): 1485-1489.

Biographies

Haitao Yuan was born in 1986. He received his B.S. and M.S. degrees in the College of Software from Northeastern University, China at 2010 and 2012. He received Google Excellence Scholarship 2011. His research interests include software-defined networking, cloud data center network, virtualized resources provisioning and optimization, and mathematical modeling and optimizations.
E-mail: cityu.yuan@gmail.com

Jing Bi was born in 1979. She received her Ph.D. degree in School of Information Science and Engineering from Northeastern University, China at 2011. At present, she is a Postdoctoral Fellow at Department of Automation, Tsinghua University, Beijing, China. From September 2009 to October 2010, she was a Visiting Student supported by IBM Ph.D. Fellowship Award in IBM Research-China. She has published more than 20 research papers in journals and conferences. Her research interests include service computing, service recommendation, cloud computing and Internet data centers.
E-mail: bijing@tsinghua.edu.cn

Bo Hu Li was born in 1938. He graduated from Tsinghua University in 1961 and was a visiting scholar at the University of Michigan in the USA from 1980 to 1982. His research expertise is in the areas of distributed simulation and cloud computing. He was elected a member of the Chinese Academy of Engineering in 2001. He is the president of Chinese Association for System Simulation; a member of the board of directors in the Chinese Society of System Engineering; the vice director of Manufacturing Committee in the Chinese Society of Automation. He was the head of China High Tech & Research Plan 863/CIMS Subject Expert Steering Committee; a member of Expert Steering Committee of Automation Area in the China High Tech & Research Plan 863; the president of Beijing Institute of Computer Application & Simulation Technology; the president of Beijing Simulation Center. He has received two prizes from China Science Convention, one first class scientific advancement award and two second class scientific advancement awards from the State, twelve scientific advancement awards from the Chinese Aerospace and Astronautics Ministry. He served for several international conferences as a member/chairman of international program committee/organization committee. He is a member of Advisory Committee of the international journal "Simulation Theory & Practice" and was a member of the director committee of the Society of International System Simulation.
E-mail: bohu1i@moon.bjnet.edu.cn