TSINGHUA SCIENCE AND TECHNOLOGY
ISSNll1007-0214 01/10 pp249–261
DOI: 10.26599/TST.2018.9010069
Volume 24, Number 3, June 2019

Scalability Analysis of Request Scheduling in Cloud Computing

Chao Xue, Chuang Lin*, and Jie Hu

Abstract: Rapid advancement of distributed computing systems enables complex services in remote computing clusters. Massive applications with large-scale and disparate characteristics also create high requirements for computing systems. Cloud computing provides a series of novel approaches to meet new trends and demands. However, some scalability issues have to be addressed in the request scheduling process and few studies have been conducted to solve these problems. Thus, this study investigates the scalability of the request scheduling process in cloud computing. We provide a theoretical definition of the scalability of this process. By modeling the scheduling server as a stochastic preemptive priority queue, we conduct a comprehensive theoretical and numerical analysis of the scalability metric under different structures and various environment configurations. The comparison and conclusion are expected to shed light on the future design and deployment of the request scheduling process in cloud computing.

Key words: cloud computing; request scheduling; scalability; model evaluation

1 Introduction

With the rapid advancement of distributed systems and the demands on computing ability, placing computing tasks in remote computing clusters has become a trend[1]. The traditional general-purpose stand-alone computing paradigm is shifting toward a heterogeneous, transparent, and scalable one[2]. The development of the definition of cloud computing and its industry structure has transformed the service paradigm[3]. Although SaaS, PaaS, and IaaS technologies make applications convenient, an important issue in cloud computing is how to achieve uneven conditioning services or users and enable users to select their desired service level via devices or platforms available to them[4]. Focusing on the capability of the cloud computing platform to ensure proper evaluation and improvement is necessary.

Scalability is one of the foremost issues that designers should take into consideration seriously in cloud computing[5–7]. Scalability is used to determine whether a cloud system can handle a large number of application requests simultaneously[8]. Hwang et al.[9] emphasized the importance of IaaS scalability in cloud computing. Chen et al.[10] discussed the scalability issues in video streaming technique in a multi-cloud environment. Anandhi and Chitra[11] provided the scalability analysis of the consistency attribute in a cloud database. Cáceres et al.[12] and Gao et al.[13] conducted service and SaaS scalability analyses, respectively. Tian et al.[14] also considered energy consumption as a bottleneck metric to evaluate the performance scalability of cloud computing via a stochastic service decision net model. However, no effective uniform formal model exists to consider the scalability challenge in previous works. In the present study, we address the scalability issues from the perspective of the request scheduling process in cloud computing.

We propose a stochastic preemptive priority queue model that builds the scalability evaluation system for

• Chao Xue, Chuang Lin, and Jie Hu are with Tsinghua National Laboratory for Information Science and Technology (TNList) & Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China. E-mail: xuec11@mails.tsinghua.edu.cn; chlin@tsinghua.edu.cn; j-hu11@mails.tsinghua.edu.cn.
* To whom correspondence should be addressed.
Manuscript received: 2017-04-20; revised: 2017-06-15; accepted: 2017-06-16

the request scheduling in cloud computing and conducts an essential theoretical derivation and numerical analysis on it. We discuss the scalability in different structures and environment configurations. We also provide valid conclusions that may be helpful in the design and optimization of the request scheduling process in cloud computing.

The rest of this paper is organized as follows. Section 2 introduces the service architecture and the related work on scalability issues in typical distributed systems. The scalability of request scheduling process is also discussed. The formulation of our scalability evaluation model is proposed in Section 3. Section 4 discusses the scalability in three fundamental structures based on our scalability evaluation model. Section 5 provides a numerical analysis of the variation of scalability in different configurations and environments. Scalability comparison of different structures is also performed based on our understanding. Conclusion and future work are presented in Section 6.

2 Background and Preliminary

2.1 Request scheduling in cloud computing

As cloud computing environments need to scale to a large number of users and tasks, designing a scheduling mechanism that can efficiently distribute the tasks and resources becomes a key point for research[15]. In cloud computing, clients generate their requests for various resources such as computing and storage access. The request type of a given device depends on its access mode and may differ from the requests of other devices. The request type may be broadcast in a wireless environment or be unicast/multicast in a configured LAN. For convenience of analysis, we focus on the unicast request mode with the view that a broadcast/multicast can be virtually equivalent to a group of unicast requests from some isomorphic virtual devices. A client may be managed by one or more servers in the request scheduling layer to describe different environments. For example, a given broadcast request may be regarded as this device is managed by all the servers in the request schedule layer and always delivers the requests to them simultaneously. Considering diversity link delay to different servers and other environmental factors, we assume that this broadcast request can also be equivalent to a group of devices managed by only one server.

Virtual servers in the cloud or cluster deliver responses to clients, providing intensive computing service and virtual resource management. After request scheduling, clients obtain relative optimal virtual servers to fetch computing and storage services according to their need. At the same time, every virtual server updates its status to request a scheduling layer. When a status report is missed for a given time, the request scheduling system regards this condition as a passive timeout report, which implies failure of a unique virtual server. The virtual server will trigger live migration according to the virtual management requirement or as proposed by the request scheduling layer. The former usually occurs due to load and power management issues. However, the latter is supervised by the request scheduling layer to reconstruct the distribution of the resource to fit the clients demand or a higher performance.

2.2 Related scalability issues

Scalability is always an essential topic in every large-scale distributed service system to evaluate the relationship between cost and efficiency. Many studies have been conducted on this topic for each unique system, but few of them addressed the general definition of scalability itself. Armbrust et al.[16] divided scalability into scale-up elasticity and scale-down elasticity; the former is not a cost optimization but an operational requirement that integrates large-scale computing, storage, and network resource to achieve user demands[17], and the latter allows the steady-state expenditure to closely match the steady-state workload usually in small- and medium-sized enterprises and organizations that have limited computing resources[18]. Eager et al.[19] provided the scalability of multiprocessor systems defined by *efficiency* based on *speedup*. Execution time on a system with size n and k processors is expressed by $\text{Time}(k,n)$, and the corresponding speedup and efficiency is defined by $\text{Speedup}(k,n)=\frac{\text{Time}(1,n)}{\text{Time}(k,n)}$ and $\text{Efficiency}(k)=\frac{\text{Speedup}(k,n)}{k}$, respectively. Jogalekar and Woodside[20] evaluated the scalability of distributed computing systems and defined $f(k)$ as the quality of service for requests, $\lambda(k)$ as the throughput, and $C(k)$ as the cost of the entire distributed system where k represents the system scale. Based on the productivity defined by $F(k)=\frac{\lambda(k)\cdot f(k)}{C(k)}$, the relative scalability metric $\Psi(k_1,k_2)=F(k_1)/F(k_2)$, which can be written as $\Psi(k_1)$, as k_2 is always a given value. Hu et al.[21] presented an integrated scalability

model based on previous studies on the control plane of software defined network and evaluated the average distance between controllers.

2.3 Scalability of request schedule layer

Similar to Arpacioglu and Zygmunt[22], we analyze the parameters, environment, and metrics of scalability on the request scheduling process in cloud computing. The issue is whether the defined metrics can be stable or better as the parameters vary under a given environment. The environment is supposed to be fixed. The parameters in the request scheduling process include request characteristics, complexity of the request scheduling system, and capability of the server to deal with request scheduling.

Request character contains the number of clients, average request load, burstiness, and communication technique such as unicast, multicast, and broadcast. Complexity refers to scale, distribution, and synchronism. In this study, we mainly discuss the scalability of the request scheduling process with the variation of the clients' number, amount, and network structure of the servers.

The request scheduling process is the basic function of cloud computing. We use a metric vector $m(\boldsymbol{p})$ to express the scalability metrics with the parameter vector $\boldsymbol{p}$. The scalability is described by a comprehensive expression $\Phi(m(\boldsymbol{p}))$ that combines various metrics.

Hu et al.[21] and Woodside[23] followed the evaluation philosophy mentioned. We fit this approach into the evaluation of the request scheduling process. Furthermore, we bring synchronization distance into the consideration of complexity. The scalability semantic of the request scheduling layer is defined as follows.

Definition 1 The request scheduling layer of a cloud computing system is scalable in a given environment, if the comprehensive expression $\Phi(m(\boldsymbol{p}))$ is approximately the same as the change of parameter vector $\boldsymbol{p}$.

3 Problem Formulation

3.1 System model

The entire system is described by a stochastic model that contains four components (clients, request scheduling network, virtual servers, and network OS) included in a cloud environment, as explained in the following.

3.1.1 Service request

Let T_c^{ij} be the time interval of two consecutive independent complete operation requests from client i to the target scheduling server j. Our stochastic model assumes that this random variable is independent and subject to exponential distribution with average value λ_c.

3.1.2 Request scheduling

We assume that the time interval random variable T_{vs}^{j}, which is the time between two consecutive status update reports to servers in a request scheduling layer, is subject to exponential distribution with an average value of λ_{vs}.

The details of this process are illustrated in Fig. 1. ① A client sends a request for a unique service. ② Servers in the request scheduling layer receive this request and make the decision. ③ Clients generate real service requests. ④ A target virtual server serves the client with corresponding resource response ⑤ and service response ⑥. A service request represents a whole that may contain many parts and trigger requests only once. Thus, the time interval between the two scheduling request to the request scheduling layer is also T_c^{ij} from the client i to virtual server j and subject to Poisson distribution.

3.1.3 Request scheduling layer

We model this layer on a node with sufficient memory and I/O ability. Parameters are process rate μ_{b1}; update rate μ_{b2}; node set V with amount v numbered by $\{0, 1, 2, ..., v-1\}$; and edge set E with amount e. Both the process time and update time of the CPU of a given server are assumbed as exponential distribution. The scale of the status table stored on each server in the virtual server layer is q. We define the average complexity to work out a response to the client as $\phi_1(V, E, q)$. Therefore the average execution time is $\phi_1(V, E, q)/\mu_{b1}$. We define the average complexity

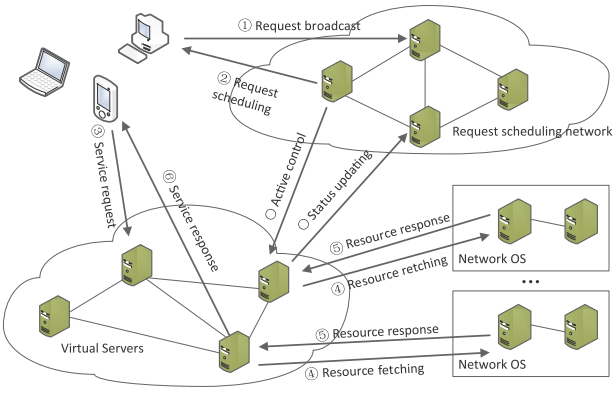

Fig. 1 Request scheduling process of cloud computing.

to update a table entry as $\phi_2(V, E, q)$. Therefore, the average update time is $\phi_2(V, E, q)/\mu_{b2}$. A status table maintained by the request schedule server can be part of the servers it manages or the whole servers in the virtual server network. This condition causes the differences in the request scheduling process. Let T_b^k be the response time of request schedule server k, Q_b^k be the queuing time of request scheduling server k with respect to the scheduling request, and T_{sync}^k be the duration of synchronization process handling the status update request. Then, we have the following two kinds of response time. The status update request has a higher priority. All status update requests should be done before the scheduling requests' start.

(1) If the server maintains a status table covering the whole virtual server layer network, i.e., q is the whole table, then the server will make the request scheduling decision after a synchronization process; thus, it deals not only with the status update requests from the virtual servers it manages but also with requests from other servers.

$$T_b^k = Q_b^k + \frac{\phi_1(V, E, q)}{\mu_b} + T_{sync}^k \tag{1}$$

(2) If the server maintains a status table covering the part of servers managed by it only, i.e., q is the partial table, then the server will make the request scheduling decision locally. If the virtual server requested is not in its management, then the server will hand over the request to the neighbor request scheduling server for execution. The synchronization process only contains the status update request from the virtual server it manages.

$$T_b^k = Q_b^k + \frac{\phi_1(V, E, q)}{\mu_b} + T_{sync}^k + T_b^{k'} \tag{2}$$

For a request, the request scheduling server should find the most appropriate one to serve the client and only needs to find the best one. Therefore, the complexity to finish this job is $O(q)$. To illustrate our computation and comparison, our model uses $\phi_1(V, E, q) = \alpha \cdot q$ where α represents a constant. Correspondingly, hash can be applied to the status update process to make the complexity $O(1)$. Thus, we have $\phi_2(V, E, q) = \alpha$ proportionable with $\phi_1(V, E, q)$. According to the conclusion that the sum of independent Poisson distributed random variables is also subject to Poisson distribution, the aggregated arrival of requests to a given server in the request scheduling layer is subject to the Poisson distribution. We model each server in this layer as a preemptive priority queue. The priority is intuitive because if we have done all the status updates, the accuracy and effectiveness of the request scheduling improve. The $O(1)$ complexity of the status updating operation means that the priority duration is not extremely long.

The topology implementation of the request schedule layer has three typical structures: centralized, decentralized, and hierarchical, as illustrated respectively in Fig. 2. The centralized structure contains only one server in the request scheduling layer. The decentralized structure has multiple servers with peer-to-peer relationship. The hierarchical structure contains different types of servers at different management levels logically. Calculations of the aggregated arrival rates of scheduling requests on different structures differ from each other and are according to the server management relationship. The detailed computation is presented in Section 4.

3.2 Scalability definition

We use a comprehensive metric to provide the scalability evaluation. This metric inherits the productive base and introduces synchronization. We also use throughput, average response time, and complexity to evaluate this metric. Number of clients is the most important variable. The scalability definition in this work is as follows.

Definition 2 Scalability metric of request scheduling layer with topology (V, E) is defined as

$$\Phi(N_1, N_2) = \frac{\lambda(N_1)}{\lambda(N_2)} \cdot \frac{T(N_2)}{T(N_1)} \cdot \frac{C(N_2)}{C(N_1)} \tag{3}$$

The scalability definition is a relative ratio form metric, which represents the scalability of request scheduling layer with a given structure when the total number of the clients scales from N_2 to N_1. $\lambda(N)$ represents the throughput, which is the average number of scheduling requests completed in a unit time. $T(N)$ represents the average response time for each request.

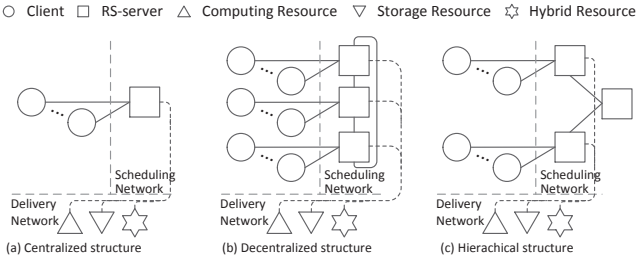

Fig. 2 Three structures of request scheduling layer.

$C(N)$ indicates the complexity to deploy and maintain the servers to confirm the validity of the request scheduling layer.

Variables in Eq. (3) are stochastic mean. The complexity is calculated by $C(N) = v/(d_{sync}+1)$ where d_{sync} indicates the farthest synchronization distance and varies depending on the different structures deployed. Similarly, we use d_{ij} to denote the shortest distance from request scheduling server i to j.

4 Scalability Evaluation

Our work entails evaluating and comparing the scalability of different structures of the request scheduling process in cloud computing. Moreover, we analyze how the changes in environment and structure configurations will influence the scalability metric. In this section, we calculate the components of the scalability metric for different structures. For the centralized structure, the metric can be calculated directly. For the decentralized structure, each request scheduling server has the same role in the scalability metric. For the hierarchical structure, the expression varies according to the function of different request scheduling servers.

4.1 Queue model of one node

We deduce the core computation of our model called "preemptive priority queue" in this subsection. Consider an aggregated scheduling request arriving λ_1 and an aggregated status updating request arriving λ_2. Their corresponding executing rates are noted by $\mu_1 = \mu_{b1} = \mu_b/\phi_1(V,E,q)$ and $\mu_2 = \mu_{b2} = \mu_b/\phi_2(V,E,q)$. As the two arriving rates are subject to Poisson distribution and the two executing rates are subject to exponential distribution, we can represent the queuing process by a 2-dimensional Markov chain. Notably, the priority of the status updating requests means that μ_1 works when and only when $j=0$. Let p_{ij} be the probability when there are i scheduling requests and j status updating requests in the queue. Let $\rho_1 = \lambda_1/\mu_1$ and $\rho_2 = \lambda_2/\mu_2$ denote the virtualized utility of the two virtualized queue. We can calculate the average response time of the aggregated scheduling request and obtaining the following equation.

$$E\{T_b^k\} = \frac{1}{1-\rho_2}\left(\frac{1}{\mu_1} + \frac{\rho_1\mu_2 + \rho_2\mu_1}{\mu_1\mu_2(1-\rho_1-\rho_2)}\right) \quad (4)$$

Applying the matrix geometric method proposed by Neuts[24], Miller[25] provided an iterative solution to the steady-state probabilities of this 2-dimensional Markov chain. Based on the method, we can determine the average response time of the scheduling request by solving the average queue length and using Little's Law.

4.2 Centralized structure

In a centralized structure, the average arrival rate of requests is $\lambda_1^C = N\lambda_c$, and that of updating requests is $\lambda_2^C = M\lambda_{vs}$, where M denotes the total number of the virtual servers. Both types of requests have a Poisson distribution. The processing time of the server in the request scheduling layer is a stochastic variable that is subject to the exponential distribution; it is the reciprocal of the average processing rate. Only one request schedule server exists in this situation, as shown in Fig. 2a, so table q maintained on this server covers all virtual servers, i.e., $q = M$. The farthest synchronization distance is $d_{sync} = 0$. The executing rates are $\mu_1^C = \mu_b/\phi_1(V,E,M) = \mu_b/(\alpha M)$ and $\mu_2^C = \mu_b/\phi_2(V,E,M) = \mu_b/\alpha$. Therefore, the average response time of a scheduling request can be calculated as follows:

$$T_C = \frac{\alpha M}{\mu_b - \alpha M\lambda_{vs}}\left(1 + \frac{\alpha(MN\lambda_c + \lambda_{vs})}{\mu_b - \alpha M(N\lambda_c + \lambda_{vs})}\right) \quad (5)$$

The complexity is $C_C(N) = v/(d_{sync}+1) = 1$.

4.3 Decentralized structure

We assume a decentralized structure to be a symmetric one with isomorphic view from an arbitrary node. This assumption is also our understanding of the decentralized and peer-to-peer concepts. As mentioned in the previous section, the status table maintained by a request scheduling can be of two types. Thus, two strategies exist in the decentralized structure for the request scheduling process, namely, full table as the first and partial as the second. The first one generates a higher demand for storage space. As the number of servers is v, we assume that each server manages its clients equitably with the same number N/v and manages the virtual servers with the same amount M/v.

4.3.1 First strategy

A request scheduling server can process all requests generated by clients under its management. We also have the table scale $q = M$ similar to the centralized structure.

For a given request scheduling server, the average arrival rate of scheduling requests is $\lambda_1^{D_1} = N\lambda_c/v$, and the average arrival rate of the status updating request is $\lambda_2^{D_1} = M\lambda_{vs}/v \cdot v = M\lambda_{vs}$. The reason for the latter is that each update is sent to every server once to maintain the corresponding

table entry. Table q is also the whole one such that $q = M$. The variable d_{sync} depends on a different topology. $\mu_1^{D_1} = \mu_b/\phi_1(V, E, M) = \mu_b/(\alpha M)$, $\mu_2^{D_1} = \mu_b/\phi_2(V, E, M) = \mu_b/\alpha$. Therefore, the average response time of a scheduling request is calculated as follows:

$$T_{D_1} = \frac{\alpha M}{\mu_b - \alpha M \lambda_{vs}} \left(1 + \frac{\alpha(\frac{MN}{v}\lambda_c + \lambda_{vs})}{\mu_b - \alpha M(\frac{N}{v}\lambda_c + \lambda_{vs})}\right) \quad (6)$$

The complexity is $C_{D_1} = v/(d_{sync} + 1) = v$.

4.3.2 Second strategy

A request scheduling server can only process the scheduling requests target to the virtual servers it manages. As mentioned in the previous section, the server will hand over the request to the request scheduling server nearby for follow-up process. We assume that the probability distribution of the destination of a request is uniform in the view of request scheduling servers. That is, a request has an equal probability $1/M \cdot M/v = 1/v$ to make each request scheduling server as a target to work out the virtual server under its management. Considering the symmetry assumption, we can first calculate the response time on every request scheduling server under the aggregate requests, and then probabilistically sum up the response time along the handover trace as the final average response time.

A request scheduling server i receives the scheduling request from the clients it manages and the request from the request scheduling server j with a shortest distance d_{ij}. The equivalent arrival rate from request scheduling server $j \neq i$ to request scheduling server i can be calculated by the probability expression $(1 - 1/v)N\lambda_c/v$. Thus, we can calculate the average arrival rate of the aggregate scheduling request on request scheduling server i as follows:

$$\lambda_1^{D_2^i} = \left(1 + \sum_{j \neq i}\left(1 - \frac{1}{v}\right)\right) \cdot \frac{N\lambda_c}{v} = \left(1 - \frac{1}{v} + \frac{1}{v^2}\right) N\lambda_c \quad (7)$$

It is also the aggregate scheduling request arrival rate of the other request scheduling servers for symmetric reason, i.e., $\forall i \in V, \lambda_1^{D_2} = \lambda_1^{D_2^i}$. If the decentralized structure is not symmetric, we can also calculate the aggregate arrival rate node by node, but a uniform expression may not exist. While the aggregate status updating request arriving rate is $\lambda_2^{D_2} = M\lambda_{vs}/v$. The table q is the partial one that $q = M/v$. $\mu_1^{D_2} = \mu_b v/(\alpha M)$, $\mu_2^{D_2} = \mu_b/\alpha$. Response time is as follows:

$$T_b^{D_2} = \frac{\alpha M}{v\mu_b - \alpha M\lambda_{vs}} \left(1 + \frac{\alpha(M\lambda_1^{D_2} + v\lambda_{vs})}{v\mu_b - \alpha M(\lambda_1^{D_2} + \lambda_{vs})}\right) \quad (8)$$

If a scheduling request generated by i has a proper target under j's management, the response time becomes $(d_{ij} + 1) \cdot T_b^{D_2}(N)$ with a probability $1/v$. Knowing the distribution of target virtual server is a uniform one, i.e., $1/M \cdot M/v = 1/v$. We can calculate the average response time of a scheduling request in this strategy based on Eqs. (7) and (8) as follows:

$$T_{D_2} = \sum_{j=0}^{v-1} \frac{d_{0j} + 1}{v} \cdot T_b^{D_2} \quad (9)$$

4.4 Hierarchical structure

In hierarchical structure, request scheduling servers play different roles. They can manage both request scheduling servers and virtual servers or only manage virtual servers as previous structures. In our context, it means "maintain the status table". We divide request schduling in two types strictly in this structure named by the layer number k and expressed by L_k. L_0 nodes locate at the bottom of the structure and only maintain the status of servers directly under their management; the status table becomes $q(L_0)$. Similarly, when $k > 0$, an L_k request scheduling server manages several L_{k-1} virtual servers and shares their status table $q(L_k) = \bigcup_{\in} q(L_{k-1})$ where $\in$ indicates the management relation. For analysis convenience, we assume that the total number of L_{k-1} request scheduling servers under the management of an L_k request scheduling server is the same denoted by m_k when $k > 0$, i.e., $q(L_k) = m_k \cdot q(L_{k-1})$ when $k > 0$. Let h be the height of the hierarchical structure and there is only one virtual server in L_h, then we have the total number in L_k, $n(L_k) = \prod_{l=k+1}^{h} m_l$. Intuitively, $n(L_0) = \prod_{l=1}^{h} m_l$, the number of virtual servers managed by each L_0 request scheduling server is $m_0 = M/n(L_0) = M/(\prod_{l=1}^{h} m_l)$. We also have the basic equation on the total number of request scheduling servers as follows:

$$v = \sum_{i=0}^{h} n(L_i) = \sum_{i=0}^{h} \prod_{j=i+1}^{h} m_j \quad (10)$$

For an L_0 request scheduling server, the status table is $q(L_0) = m_0 = M/(\prod_{l=1}^{h} m_l)$. The aggregate arrival rate of scheduling requests from clients it manages is $\lambda_1^{H_0} = N\lambda_c/n(L_0) = N\lambda_c/(\prod_{l=1}^{h} m_l)$.

The aggregate arrival rate of status updating requests from virtual servers it manages is $\lambda_2^{H_0} = m_0\lambda_{vs} = M\lambda_{vs}/(\prod_{l=1}^{h} m_l)$. The average execution rates are $\mu_1^{H_0} = \mu_b/\phi_1(V,E,q(L_0)) = \mu_b\prod_{l=1}^{h} m_l/(\alpha M)$ and $\mu_2^{H_0} = \mu_b/\phi_2(V,E,q(L_0)) = \mu_b/\alpha$, respectively. Then we can calculate the response time on each L_0 request scheduling server by Eq. (4).

Thus, for an L_k request scheduling server where $k > 0$, only when a scheduling request is missed in all layers $L_{<k}$ can L_k finally arrive layer by layer. Treating $1/n(L_l) = 0$ when $l < 0$, the aggregate arrival rate of scheduling requests is as follows:

$$\lambda_1^{H_k} = \left(1 - \frac{1}{m_k}\right) \cdot \frac{N}{n_k} \cdot \lambda_c \tag{11}$$

A status updating request will always be delivered to the upper layer if the management relationship exists. Thus, the aggregate rate of updating requests is $\lambda_2^{H_k} = M\lambda_{vs}/n(L_k)$. The average execution rate is $\mu_1^{H_k} = \mu_b/\phi_1(V,E,q(L_k)) = \mu_b\prod_{l=k+1}^{h} m_l/(\alpha M)$ and $\mu_2^{H_k} = \mu_b/\phi_2(V,E,q(L_k)) = \mu_b/\alpha$, which is the same form as the update rate in L_0. Then, Eq. (4) can be used to calculate the response time on each L_k request scheduling server. Overall, denoting $n(L_k)$ as n_k, we obtain $T_b^{H_k}(N)$ as follows:

$$T_b^{H_k}(N) = \frac{\alpha M}{n_k\mu_b - \alpha M\lambda_{vs}}\left(1 + \frac{\alpha(M\lambda_1^{H_k} + n_k\lambda_{vs})}{n_k\mu_b - \alpha M(\lambda_1^{H_k} + \lambda_{vs})}\right) \tag{12}$$

The destination decides which layer a scheduling request needs to climb to. The probability that a scheduling request will reach L_k is $1/n_k - 1/n_{k-1}$. Response time to reach L_k is $\sum_{l=0}^{k} T_b^{H_k}$. Thus, the average response time of the scheduling requests in hierarchical structure is as follows:

$$T_H = \sum_{k=0}^{h}\left(\frac{1}{n_k} - \frac{1}{n_{k-1}}\right)\sum_{l=0}^{k} T_b^{H_k} = \sum_{k=0}^{h}\left(\frac{1}{n_h} - \frac{1}{n_{k-1}}\right)T_b^{H_k} \tag{13}$$

5 Numerical Analysis

In this section, we numerically analyze the scalability under different structures and provide insights into the parameters that may influence the scalability. We should first clarify the specific structures and parameters. α as a coefficient, M as the scale of the virtual server layer, and d_{sync} as the synchronization characteristic vary in different situations; they are discussed in a separate subsection. We set the parameter $\lambda_c = 50\,\mathrm{s}^{-1}$ as the same in a datacenter. Similarly, we set $\lambda_{vs} = 5\,\mathrm{s}^{-1}$ with the understanding that the status update can be slower than the scheduling request. The processing rate or the computing ability of a request scheduling server is $\mu_b = 4\times10^9\,\mathrm{s}^{-1}$.

The decentralized structure includes three symmetric conditions illustrated in Fig. 3. They have different $\{d_{ij}\}$ sets. For computation convenience, we set v as an even number. Then, we have the following equation for $j = 1,\ldots,v-1$.

$$\begin{cases} d_{0j}^a = \min\{j, v-j\}, \\ d_{0j}^b = 1, \\ d_{0j}^c = \min\{j, v-j, 1+|j-v/2|\} \end{cases} \tag{14}$$

5.1 Service capacity

A server can not serve infinite clients, thus, the steady state in our stochastic model needs preconditions. For a given request scheduling client, $1-\rho_1-\rho_2 > 0$ must be satisfied, otherwise it will never reach the steady state and the response time of the scheduling request will be infinite. Considering this constraint, we can first analyze the service capacity in our stochastic model. We can determine the inequations in the centralized, decentralized, and hierarchical structures correspondingly as follows:

$$N_C < \frac{\mu_b - \alpha M\lambda_{vs}}{\alpha M\lambda_c} \tag{15}$$

$$N_{D_1} < \frac{\mu_b - \alpha M\lambda_{vs}}{\alpha M\lambda_c} \cdot v \tag{16}$$

$$N_{D_2} < \frac{\mu_b - \alpha M\lambda_{vs}/v}{\alpha M\lambda_c(1 - 1/v + 1/v^2)} \cdot v \tag{17}$$

$$N_H < \frac{n_k(n_k\mu_b - \alpha M\lambda_{vs})}{\alpha M\lambda_c\prod_{l=0}^{k-1}(1 - 1/n_l + 1/n_{l-1})} \tag{18}$$

By changing the total number v, we can obtain the N-curve in Fig. 4. We set $\alpha = 100$ and $m = 4$. The centralized structure has the lowest service capacity because only one server is in use. The two decentralized structures have nearly the same service capacity. In

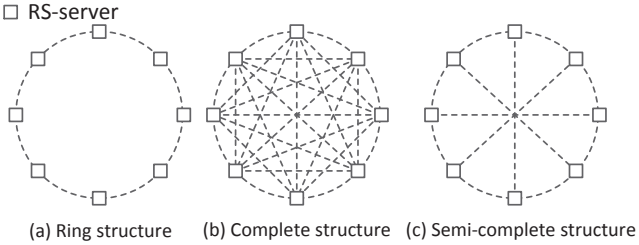

Fig. 3 Three conditions of decentralized structure.

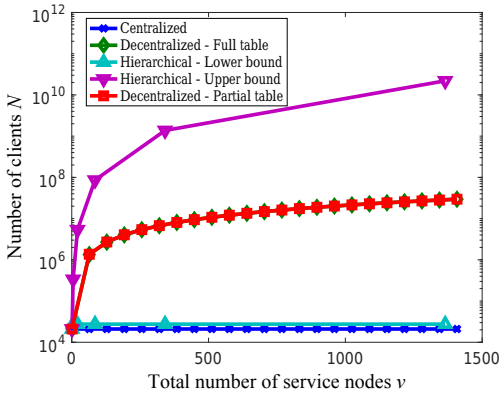

Fig. 4 Service capacity of different structures.

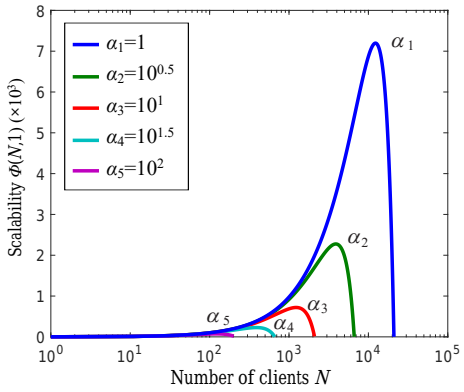

Fig. 5 Scalability metric $\Phi(N, 1)$ as a function of the number of clients N for various task size, α, values under centralized structure ($\Phi_c - \alpha$).

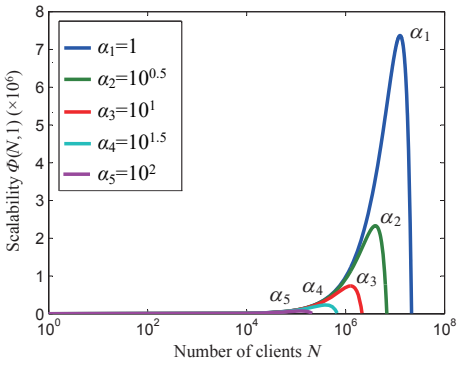

Fig. 6 Scalability metric $\Phi(N, 1)$ as a function of the number of clients N for various task size, α, values under decentralized structure with the first strategy ($\Phi_{d1}-\alpha$).

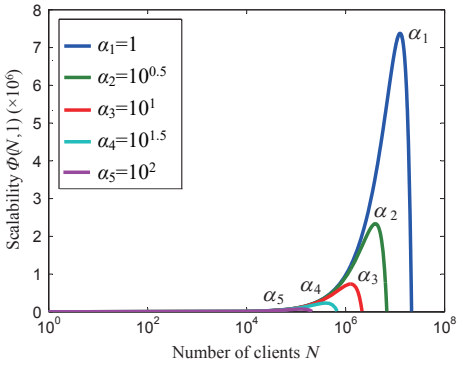

Fig. 7 Scalability metric $\Phi(N, 1)$ as a function of the number of clients N for various task size, α, values under decentralized structure with the second strategy ($\Phi_{d2}-\alpha$).

fact, they are different; the partial table structure has a slightly stronger service capacity than the whole table structure as its denominator is smaller while the numerator is larger in Eq. (17) than in Eq. (16). The lower bound of the hierarchical structure is close to the service capacity of the centralized structure. This result may be due to two reasons. First, our hierarchical structure deploys only one request scheduling server at the top and the destination distribution is uniform. Thus, whatever the lower structure deploys, $(1-1/m_h)$ of total requests will reach the top, thereby causing a bottleneck. Second, the total number of servers not only includes the servers manage virtual servers, which may cause a horizontal movement to the left in a $(m_0^h - 1)/(m_0 - 1)$ scale. At the same time, the upper bound of the hierarchical structure is larger than that of the decentralized one, as they are derived from the lower level and only manage requests from fewer clients. Notably, this approach only considers capacity without regard to the response time. As the $1 - \rho_1 - \rho_2$ condition is in the denominator, the value of the equation becomes infinite if the number of clients approaches the service capacity limit.

5.2 Parameter analysis

In this subsection, numerical analysis is conducted on the influence of some parameters on the scalability.

5.2.1 Task size

Task size α is an important parameter that represents the complexity to finish a single operation in our request scheduling process. We also set $M = 4096$. In the decentralized structure, we set $v = 1024$, while in the hierarchical structure, we set $m_k = 4$ and $h = 4$. The influence of α on the scalability metric in different structures is illustrated in Figs. 5–8. In Fig. 5, by fixing α, the scalability metric is increasing with the number of clients increasing at the very beginning, then the scalability metric decreases. This phenomenon leads to a "best" scalability metric value under specific α coordinate with an appropriate N and this point is representative. Curves with similar shape are illustrated in Figs. 6–8 under the 2 decentralized

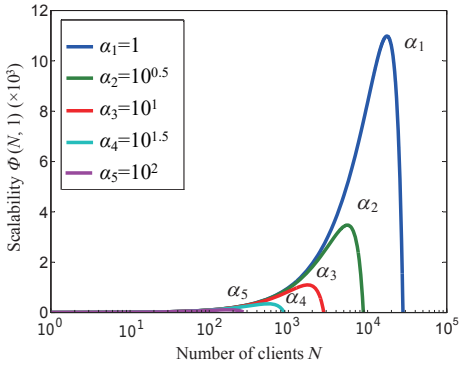

Fig. 8 Scalability metric $\Phi(N, 1)$ as a function of the number of clients N for various task size, α, values under hierarchical structure (Φ_h–α).

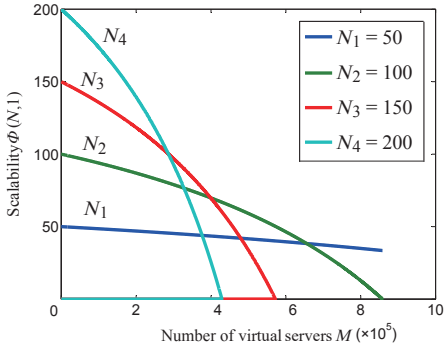

Fig. 9 Scalability metric $\Phi(N, 1)$ as a function of the number of virtual servers M for various numbers of clients, N, under centralized structure (Φ_c–M).

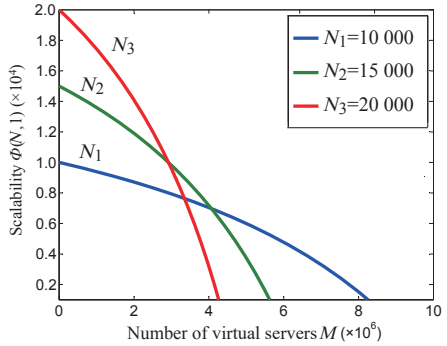

Fig. 10 Scalability metric $\Phi(N,1)$ as a function of the number of virtual servers M for various numbers of clients, N, under decentralized structure (Φ_d–M).

structures and the hierarchical structure. By fixing the number of clients N, we can conclude that the scalability metric is negatively correlated with α in all the three structures. Furthermore, the multiplication of α decreases the scalability metric Φ in order of magnitudes. This conclusion conforms to our intuition. As a complex request always needs more time to be complete, the response time T increases and the scalability Φ decreases. The next conclusion we can obtain from these figures is that the change of α almost does not influence the shape of the scalability curve. Specifically, Figs. 6 and 7 shows that the scalability metric values under two decentralized strategies are almost the same with task size α and the number of clients N. Part of that is because large scale request routing layer configuration $v = 1024$ thus $1 - 1/v + 1/v^2 \approx 1$, and $\mu = 4 \times 10^9\ \mathrm{s}^{-1} \gg \lambda_c > \lambda_{vs}$ thus $\mu_b - \alpha M \lambda_{vs} \approx \mu_b - \alpha M \lambda_{vs}/v$. Furthermore, the scalability metric values are also the same even when the detailed structure is changed from Figs. 3a to 3c. The reason is that the difference $\{d_{ij}\}$ is hidden by the ratio form expression of Φ and in $\Phi(N, 1)$, N and 1 share the same detailed structure.

5.2.2 Resource scale

Resource scale M is the total number of the virtual servers in the virtual server layer, which is below the request scheduling layer. Intuitively, the increase of M increases the performance in the virtual server layer. However, as Figs. 9 and 10 illustrate, a larger M decreases our scalability metric. To a larger N, the influence of M is also larger. The environment contains $v = 1024$. This result can be derived from the equation from the previous theoretical analysis as M and N as always the product relation minus μ_b. From the system itself, the larger M means the larger state information should be stored and maintained, which influences the scalability of the request scheduling servers in request scheduling layer.

5.2.3 Scheduler scale

Similar to M, $v = |V|$ is the total number of request scheduling servers. By configuring $M = 4096$, $h = 6$, $m_k = 4$, and $i = 1, \ldots, 4$, we obtain the relationship between v and the scalability in Figs. 11 and 12 for the decentralized and hierarchical structures. The centralized structure is a naive situation where $v \equiv 1$. In both figures, we can see that the scalability increases with the increase of v but approaches an upper bound. The reason is that the number of clients N is fixed and when v is large enough, the response time decreases to a lower bound while the throughput does not increase. A larger N follows the scalability curves in Fig. 5 that increases the scalability before a special value and then decreases the metric. For example, the scalability metric with $N_5 = 18\,000$ is larger than $N_6 = 20\,000$ one. We also explore the influence of the ratio v/N

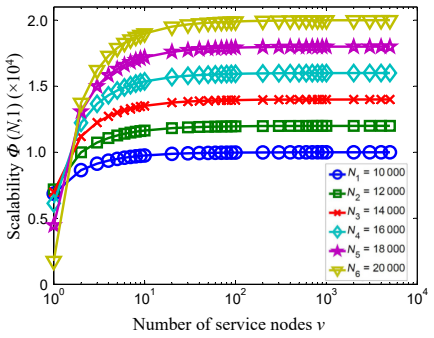

Fig. 11 Scalability metric $\Phi(N,1)$ as a function of the number of service nodes v for various numbers of clients, N, under decentralized structure ($\Phi_d - v$).

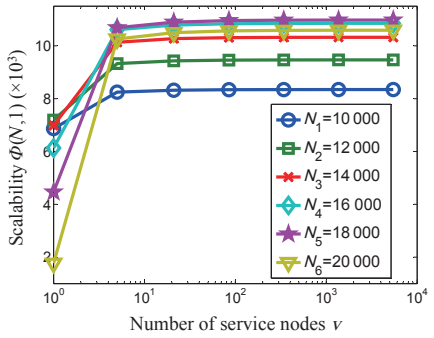

Fig. 12 Scalability metric $\Phi(N,1)$ as a function of the number of service nodes v for various numbers of clients, N, under hierarchical structure (Φ_h–v).

on the scalability as illustrated in Fig. 13 under the decentralized structure with $\alpha = 100$. We can see that the scalability metrics in different N begin to rise and approach to the upper limit almost at the same ratio value. As v follows the increase of N, the scalability will always increase without reaching a bound.

5.2.4 Complexity

We regard the complexity metric $C(N) = v/(d_{sync}+1)$ as the structure's attribute. Here, we determine the influence of synchronization on the response time T in different structures based on the decentralized structure with the second strategy. The distance can be calculated by Eq. (14). We set $M = 4096$, $v = 100$, and $\alpha = 100$ and obtain the response time curves in Fig. 14. In this figure, the response time of the ring structure is the largest and that of the complete structure is the smallest. This result can be obtained intuitively from the expression of the response time. At the same time, the complexity is the largest in the complete structure which suffers a larger management cost.

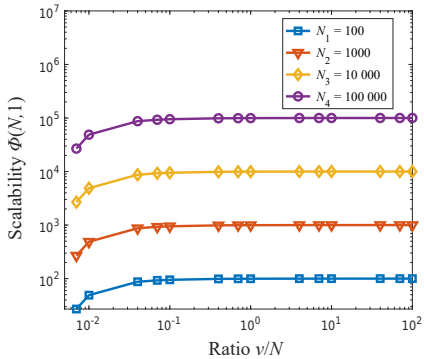

Fig. 13 Scalability metric $\Phi(N,1)$ as a function of the ratio v/N for various numbers of clients, N, under decentralized structure ($\Phi_d - v/N$).

5.2.5 Rate of request and update

The influence of the request and update rate on the scalability metric is illustrated in Figs. 15 and 16. In this case, $v = 342$, $M = 4096$, $N = 10\,000$, $\alpha = 1$, $h = 4$, and $m_k = 4$ for $k = 1, \ldots, 4$. Decentralized structures with different strategies are both strong enough for the increase of the two λs, the hierarchical structure is weaker, and the centralized structure is the weakest or the most sensitive to the variation of the request rate.

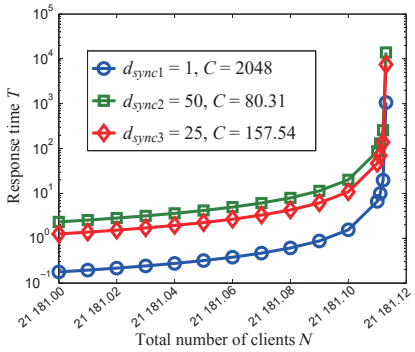

Fig. 14 Response time T as a function of the number of clients N for various farthest synchronization distance, d_{sync}, values under decentralized structure with the second strategy ($T_{d2} - d_{sync}$).

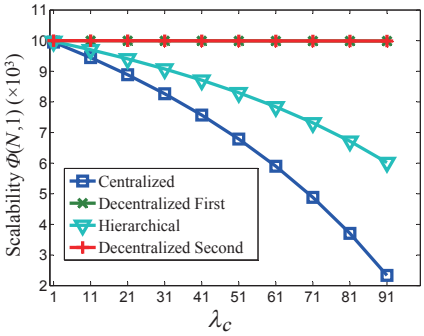

Fig. 15 Scalability metric $\Phi(N,1)$ as a function of the Poisson request input with rate λ_c for various structure configurations under different structures ($\Phi - \lambda_c$).

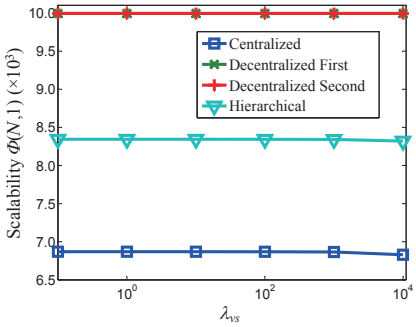

Fig. 16 Scalability metric $\Phi(N,1)$ as a function of the Poisson update input with rate λ_{vs} for various structure configurations under different structures ($\Phi-\lambda_{vs}$).

We can also obtain the conclusion that the scheduling request rate λ_c can influence the system more than the status updating request λ_{vs}.

5.3 Width and height

If servers are used to construct the request scheduling system in the hierarchical structure, we should decide whether the shape should be a lanky one or a short and fat one. The comparison is made in Fig. 17. We set $\alpha = 1$, $M = 4096$, and for computation convenience, we give all m_k the same value m. In this figure, a smaller m will hold the greater scalability when the total number v is the same or nearly the same. The height h has an opposite conclusion with m.

5.4 Crosswise and lengthwise

Scalability means scaling up to meet the increase of requests and the necessity of burst. This task can be acomplished in two ways. One is to increase the computation ability of the computation module which is similar to the evolution history of computers and we call it the lengthwise one. The other one increases the number of the computation modules and we call it the crosswise one. Figure 18 shows that the most efficient way is lengthwise although we know that in real life, increasing the computation ability to keep in step with the necessity and scale is difficult. The decentralized structure with a whole status table in each server is the most scalable structure in the crosswise ones. The hierarchical structure is the weakest as the bottle neck of the sole peak. The centralized structure in this figure contains the computation ability μ_b^C 341 times as the other μ_bs in the other different structures correspondingly. However the rest contains a total number $v = 341$ which is also the 341 times to the centralized structure. Parameters are set as follows: $M = 4096$, $\alpha = 1$, $h = 4$, and $m_k = 4$ for $k = 1, \ldots, h$.

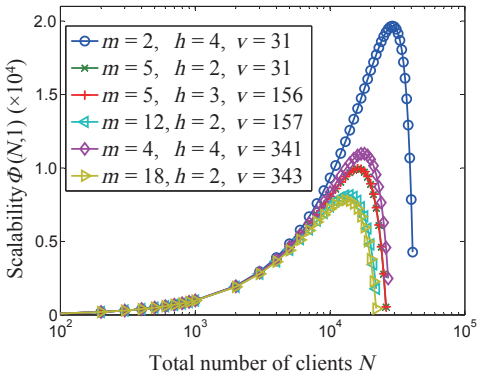

Fig. 17 Scalability metric $\Phi(N,1)$ as a function of the number of clients N for various structure configuration values under hierarchical structure ($\Phi_h-(m,h)$).

6 Conclusion and Future Work

In this paper, we presented a formal scalability definition of the request scheduling process in cloud computing. Considering the deployment request and state update request, we model each request scheduling service node as a priority queue. By solving the 2-dimensional Markov chain of this queue, we present a theoretical evaluation and comparison of the scalability of the request scheduling process in three basic structures. Furthermore, we numerically analyze the influence of some structural parameters to the response time and scalability metric, including the synchronization distance and shape of different structures. Using the configuration of different structural parameters, we finally analyze the scalability of different decentralized topologies and obtain useful conclusions. Our analysis, comparisons, and conclusions can provide a valuable reference for the design and implementation of the request scheduling

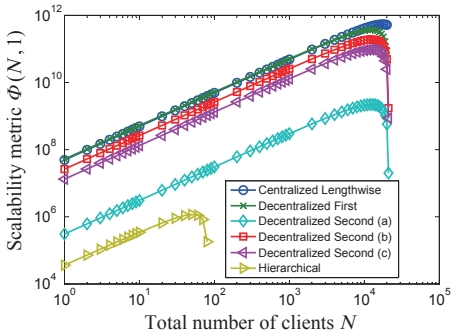

Fig. 18 Scalability metric $\Phi(N,1)$ as a function of the number of clients N for various structure configurations under different structures ($\Phi-$structure).

process in cloud computing.

Some future works could still be explored in structural analysis. A uniform formal description of the optimal structure is the ultimate goal to be achieved. Evaluating the hybrid structure composed of the three structures discussed in this paper is a possible approach. Meanwhile, trustworthiness issue[26] in cloud computing is also an important aspect to be considered in future scalability studies.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (No. 61472199) and Tsinghua University Initiative Scientific Research Program (No. 20121087999).

References

[1] S. Akhshabi and C. Dovrolis, The evolution of layered protocol stacks leads to an hourglass-shaped architecture, in *Dynamics on and of Complex Networks*, A. Mukherjee, M. Choudhury, F. Peruani, N. Ganguly, and B. Mitra, eds. New York, NY, USA: Springer, 2013, pp. 55–88.

[2] R. Buyya, C. S. Yeo, and S. Venugopal, Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities, in *Proc. 10th IEEE Int. Conf. High Performance Computing and Communications*, Dalian, China, 2008, pp. 5–13.

[3] B. Hayes, Cloud computing, *Commun. ACM*, vol. 51, no. 7, pp. 9–11, 2008.

[4] Y. Gao, Y. X. Zhang, and Y. Z. Zhou, Performance analysis of virtual disk system for transparent computing, in *Proc. 9th IEEE Int. Conf. Ubiquitous Intelligence and Computing and 9th Int. Conf. Autonomic and Trusted Computing (UIC/ATC)*, Fukuoka, Japan, 2012, pp. 470–477.

[5] T. Banditwattanawong and P. Uthayopas, Improving cloud scalability, economy and responsiveness with client-side cloud cache, in *Proc. 10th IEEE Int. Conf. Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, Krabi, Thailand, 2013, pp. 1–6.

[6] R. K. Chandrahasan, S. Kalaichelvi, S. Priya, and L. Arockiam, Research challenges and security issues in cloud computing, *Int. J. Comput. Intell. Inf. Sec.*, vol. 3, no. 3, pp. 42–48, 2012.

[7] Z. H. Li, Y. Zhang, and Y. H. Liu, Towards a full-stack devops environment (platform-as-a-service) for cloud-hosted applications, *Tsinghua Sci. Technol.*, vol. 22, no. 1, pp. 1–9, 2017.

[8] S. K. Garg, S. Versteeg, and R. Buyya, A framework for ranking of cloud computing services, *Fut. Gener. Comput. Syst.*, vol. 29, no. 4, pp, 1012–1023, 2013.

[9] K. Hwang, X. Y. Bai, Y. Shi, M. Y. Li, W. G. Chen, and Y. W. Wu, Cloud performance modeling with benchmark evaluation of elastic scaling strategies, *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 1, pp. 130–143, 2016.

[10] W. Chen, J. W. Cao, and Y. X. Wan, QoS-aware virtual machine scheduling for video streaming services in multi-cloud, *Tsinghua Sci. Technol.*, vol. 18, no. 3, pp. 308–317, 2013.

[11] R. Anandhi and K. Chitra, A challenge in improving the consistency of transactions in cloud databases-scalability, *Int. J. Comput Appl.*, vol. 52, no. 2, pp. 12–14, 2012.

[12] J. Cáceres, L. M. Vaquero, L. Rodero-Merino, Á. Polo, and J. J. Hierro, Service scalability over the cloud, in *Handbook of Cloud Computing*, B. Furht and A. Escalante, eds. Boston, MA, USA: Springer, 2010, pp. 357–377.

[13] J. Gao, P. Pattabhiraman, X. Y. Bai, and W. Tsai, SaaS performance and scalability evaluation in clouds, in *Proc. IEEE 6th Int. Symp. on Service Oriented System*, Irvine, CA, USA, 2011, pp. 61–71.

[14] Y. Tian, C. Lin, Z. Chen, J. X. Wan, and X. H. Peng, Performance evaluation and dynamic optimization of speed scaling on web servers in cloud computing, *Tsinghua Sci. Technol.*, vol. 18, no. 3, pp. 298–307, 2013.

[15] Z. F. Zhong, K. Chen, X. J. Zhai, and S. G. Zhou, Virtual machine-based task scheduling algorithm in a cloud computing environment, *Tsinghua Sci. Technol.*, vol. 21, no. 6, pp. 660–667, 2016.

[16] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al., A view of cloud computing, *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[17] D. Tao, Z. W. Lin, and B. X. Wang, Load feedback-based resource scheduling and dynamic migration-based data locality for virtual hadoop clusters in openstack-based clouds, *Tsinghua Sci. Technol.*, vol. 22, no. 2, pp. 149–159, 2017.

[18] D. G. Cao, P. D. Liu, W. Cui, Y. H. Zhong, and B. An, Cluster as a service: A resource sharing approach for private cloud, *Tsinghua Sci. Technol.*, vol. 21, no. 6, pp. 610–619, 2016.

[19] D. L. Eager, J. Zahorjan, and E. D. Lazowska, Speedup versus efficiency in parallel systems, *IEEE Trans. Comput.*, vol. 38, no. 3, pp. 408–423, 1989.

[20] P. Jogalekar and M. Woodside, Evaluating the scalability of distributed systems, *IEEE Trans. Parallel Distribut. Syst.*, vol. 11, no. 6, pp. 589–603, 2000.

[21] J. Hu, C. Lin, X. Y. Li, and J. W. Huang, Scalability of control planes for software defined networks: Modeling and evaluation, in *Proc. IEEE 22nd Int. Symp. of Quality of Service*, Hongkong, China, 2014, pp. 147–152.

[22] O. Arpacioglu and H. J. Zygmunt, On the scalability and capacity of planar wireless networks with omnidirectional antennas, *Wireless Communications and Mobile Computing*, vol. 4, no. 3, pp. 263–279, 2004.

[23] C. M. Woodside, Throughput calculation for basic stochastic rendezvous networks, *Performance Eval.*, vol. 9, no. 2, pp. 143–160, 1989.

[24] M. F. Neuts, Markov chains with applications in queueing theory, which have a matrix-geometric invariant probability vector, *Adv. Appl. Probabil.*, vol. 10, no. 1, pp. 185–212, 1978.

[25] D. R. Miller, Computation of steady-state probabilities for M/M/1 priority queues, *Operat. Res.*, vol. 29, no. 5, pp. 945–958, 1981.

[26] C. Lin and C. Xue, Multi-objective evaluation and optimization on trustworthy computing, *Sci. China Inf. Sci.*, doi: 10.1007/s11432-015-0856-7.

Chao Xue received the BEng degree from Tsinghua University in 2011. He is currently working toward the PhD degree in the Department of Computer Science and Technology at Tsinghua University. His research interests are in modeling, simulation and performance analysis of computer systems, and computing paradigms.

Jie Hu received the BS degree from Xi'an Jiaotong University in 2011. He is currently a PhD candidate in the Department of Computer Science and Technology at Tsinghua University. His research interests are in modeling and performance analysis of SDN.

Chuang Lin is a professor of the Department of Computer Science and Technology at Tsinghua University. He received the PhD degree from Tsinghua University in 1994. His current research interests include computer networks, performance evaluation, network security analysis, and Petri Net theory and its applications. He has published more than 300 papers in research journals and IEEE conference proceedings and has published five books. He is a member of the Steering Committee for the International Petri Net Community, a member of ACM Council, a senior member of the IEEE, and the Chinese delegate in TC6 of IFIP. He serves as the associate editor of *IEEE Transactions on Vehicular Technology*, and the area editor of *Computer Networks* and *Journal of Parallel and Distributed Computing*.