Going Fast and Fair: Latency Optimization for Cloud-based Service Chains

Yuchao Zhang[1] *IEEE Student Member*, Ke Xu[2] *IEEE Senior Member*, Haiyang Wang[3] *IEEE Member*,
Qi Li[4] *IEEE Senior Member*, Tong Li[5] *IEEE Student Member*, Xuan Cao[6]
[1]Beijing University of Posts and Telecommunications [2]Tsinghua University
[3]University of Minnesota at Duluth [4]Shenzhen Graduate School of Tsinghua University [5]Huawei [6]Baidu

***Abstract*—State-of-the-art mircoservices are starting to get more and more attention in recent years. A broad spectrum of online interactive applications are now programmed to service chains on cloud, seeking for better system scalability and lower operation cost. Different from the conventional batch jobs, most of these applications consist of multiple stand-alone services that communicate with each other. These step-by-step operations unavoidably introduce higher latency to the delay-sensitive chained services.**

In this paper, we aim at designing an optimization approach to reduce the latency of chained services. Specifically, presenting the measurement and analysis of chained services on Baidu's cloud platform, our real-world trace indicates that these chained services are suffering from significantly high latency because they are mostly handled by different queues on cloud servers for multiple times. Such a unique feature, however, introduces significant challenge to optimize microservice's overall queueing delay. To address this problem, we propose a delay-guaranteed approach to accelerate the overall queueing of chained services while obtaining fairness across all the workloads. Our evaluations on Baidu servers shows that the proposed design can successfully reduce the latency of chained services by 35% with minimal impact on other workloads.

***Index Terms*—Service chains; Interactive applications; Latency optimization; Fairness**

I. Introduction

The rapid growth of service chains is changing the landscape of cloud-based applications. Different stand-alone components are now handled by cloud servers, providing cost efficient and reliable services to Internet users. It is known that the workloads from service chains are more complex than the traditional non-interactive (or batch) workloads [1]: Non-interactive workloads are the workloads that can be processed on only one specific server and do not need interactions with other servers (such as scientific computing and image processing). Being not strictly time-sensitive, these workloads can be scheduled to run anytime as long as they could be finished before a soft deadline. While interactive workloads from service chains are the workloads that have to go through multiple servers to apply different functions (such as business transactional and complex gaming control), and these chained services typically process real-time user requests. But the interactions unavoidably introduce additional latency, making the performance for service chains in urgent need to be ensured.

Work partly done when Yuchao Zhang is interned in Baidu.

We then measure the interactive workloads performance on Baidu's cloud platform. The real-world traces indicate that interactive workloads are really suffering from significantly longer latency than non-interactive workloads. The measured case is shown in Figure 1(a) where *Nuomi* is a group buying application, *Waimai* is a take-out service, and *Alipay* is an online payment platform. When a user clicks an item on Nuomi, the latency is quite short because this query does not require many interactions among services. However, the story will be different when this user orders a take-out and purchase the item. In this case, the request goes through Nuomi, Waimai, and then Alipay. In other words, this interactive workload consists of several highly-dependent operations which have to be processed on different servers separately, like Figure 1(b), there are 6 procedures for interactive workloads and only 2 procedures for non-interactive workloads. It is easy to see that such interactive workloads in chained applications will introduce extra latency to users because these requests will be handled by different services for multiple times.

Unfortunately, we find that most of the existing workload scheduling approaches are aimed to re-schedule [2] and leverage different priorities [3], [4] on individual queues. In other words, these optimizations are made on intermediate servers separately, so the overall latency of interactive workloads is still unpredictable. To better optimize the overall latency of chained services, we apply a latency estimation approach to predict overall latency and try to accelerate the interactive workloads. Furthermore, we design a feedback scheme to ensure workload fairness and avoid remarkable degradation of non-interactive workloads. Our real-world deployments on Baidu indicate that the proposed Delay-Guarantee (D^3G) framework can successfully reduce the latency of interactive applications with minimal affect on other workloads.

The main contributions are summarized as follows:

- We present the measurement and latency analysis of service chains in Baidu networks and disclose the long latency for interactive workloads.
- We design the D^3G algorithm to accelerate interactive workloads in a global manner other than in each independent server, and leverage a latency estimation algorithm and a feedback scheme to ensure fairness.
- We evaluate our methods on servers in Baidu networks, and the extensive experiment results show that D^3G succeeds in accelerating interactive chained applications while ensuring workload fairness.

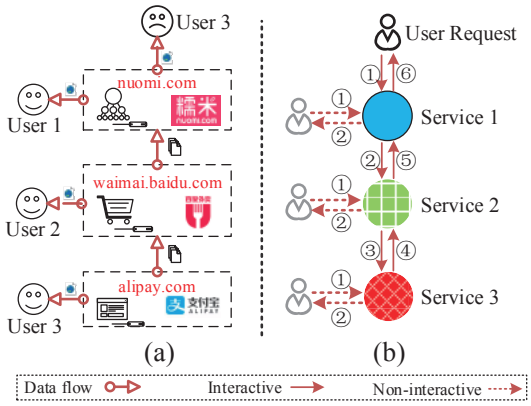


Fig. 1: The processes of interactive and non-interactive workloads.

II. Background

As more applications are deployed on clouds for better system scalability and lower operation cost, service chains are developing quickly. Many studies have shown that the latency is particularly problematic when interaction latency occurs together with network delays [5]. In this section, we'll first introduce the related work about service chains and interactive applications, and then we'll present the measurements from real-world trace, which motivates this paper.

A. Related Work

To minimize the latency on cloud-based applications, many researchers focused on minimizing latency, which do advance the state-of-art.

We classify these literatures into two categories. First, some literatures focused on network and processing latency. For example, Webb et.al. [6] proposed a nearest server assignment to reduce the client-server latency. Vik explored the spanning tree problems in a distributed interactive application system for latency reduction in [7]. Moreover, [8] and [9] introduced game theory into this topic and modeled the latency problem in datacenter as a bargaining game. Second, some researches aim at reducing service latency. Web service is the most related, which is an effective mechanism for data and service integration on the Web. Some studies succeeded in dissecting latency contributors [10], showing that back-office traffic accounts for a significant fraction of Web transactions latency in terms of requests and responses [11].

The above studies handle different components of overall latency, but they ignore the effects brought by service chains, for example, the case in Figure 1. When service 1 receives an interactive request (TCP:0) which cannot be served in service 1 (i.e., this request requires data from services 2 and 3), then service 1 will make a new TCP connection (TCP:1) to service 2, and the origin connection (TCP:0) would be hang up. In the same way, service 2 will make a new TCP connection (TCP:2) to service 3. When service 2 receives the corresponding results from service 3, the TCP:2 connection would be released and TCP:1 connection recovers. After service 2 sends the results back to service 1, TCP:1 connection would be released and the origin TCP:0 connection recovers. Thus, the interactive workloads are suffering from longer latency due to the interactions among multiple intermediate servers. Many researchers investigated the latency for these applications, and their researches showed that although these applications are quite delay-sensitive, service performance is greatly affected by interactions. To address this problem, some studies suggested that the interactions of different services should be further dissected to better understand the performance implications [11], and some researchers have already began to pay attention to interaction latency [12].

Our study explores the potential to reduce the response time for service chains and guarantee the non-interactive workloads simultaneously. In particular, we accelerate the interactive workloads by building a new dedicated queue and try to adjust resource allocation among different queues. By leveraging a feedback scheme, we can bound the influence on non-interactive workloads. We'll describe the algorithm in Section III in detail after introducing our motivation in Section II-B.

B. Measurement and Motivation

In this section, we conduct some measurements in Baidu networks and disclose the long latency of chained services. Aiming at accelerating interactive workloads while not affecting non-interactive workloads (to ensure fairness), we then motivate this paper.

As the largest Chinese searching engine, Baidu has dozens of applications deployed in its networks. These applications cover every corner of people's life, and they can further cooperate with each other to provide more comprehensive functions (as shown in the introduction section).

To evaluate the performance of these services, we measured the workload latency from one server cluster at Baidu. In particular, we monitor all the workloads, record the response time of service calls, and then calculate the average latency per minute for both interactive and non-interactive workloads by analyzing the trace log. We grab the log of these two kinds of workloads from 0:00 to 24:00 on April 7th 2016 and draw the statistical figures in Figure 2. The x-coordinate denotes the time in one day while the y-coordinate denotes the service latency in ms. From these results, we come to the following conclusions:

1) The average latency for non-interactive workloads is about 60 ms to 70 ms, while that for the interactive workloads is nearly 500 ms, i.e., the interactive workloads are suffering from 7 times longer latency compared to the non-interactive workloads.

2) Even when the network is not congested (e.g., during the midnight), the interactive workload latency is still much longer than the non-interactive workload latency.

3) When there is a slight burst, for example, 11:00 am or 16:00 pm, the performance for interactive workloads is influenced obviously, making the latency even higher.

As we analyzed before, non-interactive workloads can be completed in just one instance while the interactive workloads

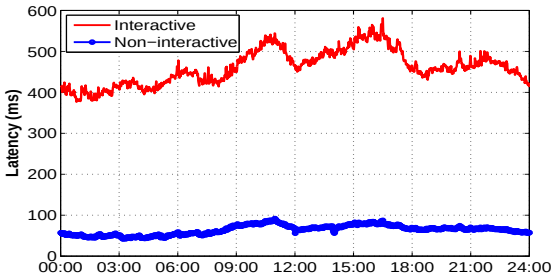


Fig. 2: The workload latency in Baidu networks.

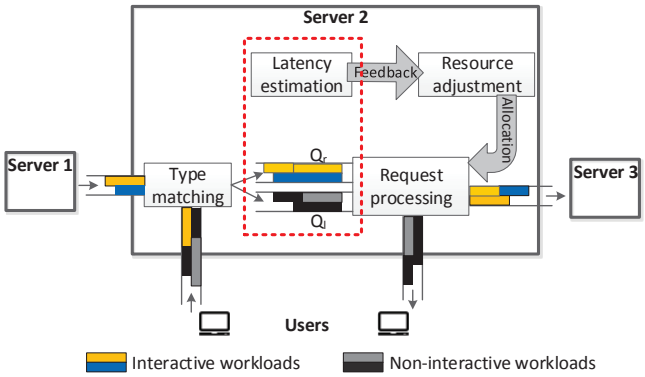


Fig. 3: The framework of D^{3}G scheme.

have to go through different servers one after another. To optimize the performance of delay-sensitive interactive workloads, we should accelerate the processing of these workloads, such as assigning higher priorities or allocating more resources. However, improving interactive workloads will unavoidably affect non-interactive workloads because they are sharing the same infrastructures. So a fair optimization scheme should have the following two characteristics: 1) Reduce the latency for delay-sensitive interactive workloads; 2) Ensure the fairness across all workloads (not to degrade non-interactive workloads severely).

III. Dynamic Differentiated Service with Delay-Guarantee

In this section, we study the essence of the latency gap in Section III-A before introducing the design philosophy of our approach in Section III-B. According to the philosophy, we design an algorithm called the Dynamic Differentiated service with Delay-Guarantee (D^{3}G), which reduces the latency of service chains while ensuring workload fairness.

A. The components of latency

As interactive applications consist of basic functions applied on different servers, those workloads should go through multiple servers in a specific order so that the required functions are applied step by step. Therefore, the interactive workloads will be queued in servers for several times while the non-interactive workloads will be queued for just once.

To be specific, we analyze the latency for interactive $R_{i,j}$ and non-interactive $R_{i,i}$ workloads: As interactive workloads travel across multiple servers and are queued in each one, the final latency is the sum of the queuing and serving time on each server. Besides, the transfer time among different servers also contributes to the latency. While non-interactive workloads only go through one particular server with only once queuing and serving time. Thus, the overall latency of interactive workloads is much longer than that of non-interactive workloads (as shown in Figure 2).

B. Design Philosophy

As users patience is limited and they would abandon the system once the latency exceeds their patience, the interactive workload latency is essential for these delay-sensitive applications. This limited patience can be formulated as an exponential function [13]. Considering that one exponential function is not precise enough to model user patience, we use a weighted sum of exponential functions to calculate user patience in D^{3}G algorithm. With this expected patience, the system would have a leaving rate, rephrased as follows: When the overall latency exceeds user patience, users will abandon the system, and this leads to an abandoning rate of waiting queues.

To ensure constant service and prevent users from abandoning system, interactive workloads should be scheduled within user tolerance. To do so, we do re-scheduling and resource adjustment in this work. Specifically, we separate interactive workloads from non-interactive ones and make them pending in different queues. We leverage two queues in each server. Q_l represents the queue for non-interactive requests, and Q_r represents the queue for interactive requests. The two queues share the infrastructure and resources in the same server. The difficulty in accelerating interactive workloads is that allocating more resources to Q_r will unavoidably affect the process of Q_l. Thus, how to share the resources on one server among different workloads becomes a key concern.

To address this issue, we design D^{3}G that can adjust resource allocation among different kinds of workloads automatically and in real time. To make D^{3}G more intelligent, we design an estimation algorithm to pre-calculate the processing time on other servers. Furthermore, we also introduce a feedback scheme to reduce negative impact to non-interactive workloads.

C. D^{3}G Framework

As we described in the previous subsection, we separate the interactive workloads from non-interactive workloads and make them queued independently. We design a latency estimation algorithm, and once the estimated latency exceeds user patience, we dynamically adjust the resource allocation among queues according to a feedback scheme. Thus, the interactive workloads will get accelerated in all intermediate servers and finally enjoy the comparable latency as the non-interactive workloads.

The framework of D^{3}G is shown in Fig. 3. For a specific server, it receives interactive workloads from other servers,

and at the same time receives both interactive and non-interactive requests from users. A request type matching scheme will check whether this request should enqueue in Q_r (for interactive workloads) or in Q_l (for non-interactive workloads), with source (s), destination (d), and function (f). For each queue, the latency estimating algorithm pre-calculates the overall latency of these requests. If the estimated latency for interactive workloads exceeds user patience, the resource adjustment module would allocate the more resources to the interactive queue Q_r and update the request processing module. Then the processing speed of interactive workloads is thus promoted. This process executes in real-time automatically.

In the latency estimation algorithm, when a request with $\langle s, d, f \rangle$ enters a queue, we update the queue information and record the arriving time. Once a request begins to be served, we record the beginning time and the queueing time. If this request is an interactive one, it will be transmitted to the next service. If this request is a non-interactive one, we can get the finish time directly, which is calculated by summing the beginning time and the service time. Thus, we can calculate the overall estimated latency.

In the feedback scheme, we formulate the arrival rate of workloads, server's serving rate and user's abandon rate before using the Markov chain to model the two queues. After calculating queue length and the expected waiting time, we equalize workload latencies by adjusting the resource allocation. The detailed adjustment description will be introduced in the next subsection.

Overall, D^{3}G converts the performance optimization problem into a resource allocation problem. By estimating latencies of different network services, D^{3}G mitigates the imbalance by adjusting the allocated resources. The real time estimation algorithm and the intelligent feedback scheme make D^{3}G work efficiently and automatically.

D. Adjusting Resource Allocation

To calculate the resource allocated to interactive workloads and non-interactive workloads, we model the queuing problem in the adjusting scheme. To analyze the arrival and leaving rates of requests, we adopt a Markov model to represent state transmissions of the two queues. By modeling different distributions on service time and abandon rate, we can calculate the latency expectation of various workloads. At last, the feedback scheme could adjust resource allocation to ensure fairness.

The arrival process of requests is a discrete-time random process, and the number of requests in the future is just related to the number at present, i.e., the queue can be formulated by a Markov chain, and the queue length can be calculated. For any specific server: 1) when a request arrives, the queue length increases by 1, 2) when a request abandons the queue or a service is finished, the queue length decreases by 1. Thus, with the arrival rate, service time and abandoning rate, we can model the queues in any particular servers as M/M/1 queues [14] and finally get the queue length. As described before, the service time is in an exponent distribution, and each service is mutually independent with each other, So the waiting time of a request is a convolution. So far, the expectation of waiting time on one server can be calculated.

Recalling that interactive requests will be pending in queues for multiple times in different servers, and non-interactive requests just need to be pending for just once. This may lead to intolerable latency for delay-sensitive workloads. We solve this problem by adjusting resource allocations. With the expected waiting time, we assume the transfer time on server is in a Gaussian distribution [15], and then make the overall latency of the two queues equal to each other. Thus, we can work out the allocation rates to the interactive workloads and non-interactive workloads. With this adjusted allocation, interactive workloads from delay-sensitive applications can enjoy reduced latency that is within user tolerance.

IV. Deployment

We implement D^{3}G in the severs of Baidu networks, and the algorithm is written in C language. The servers use $Linux$ operating system and are configured with $tomcat$ webservers based on $Java$. We choose four servers that are configured with 4G memory, two cores, and 100Mbps public network bandwidth. As to the clients, there are 36 end-hosts and each is configured with an Intel i5 1.7GHz CPU and 2G memory. All these end-hosts keep sending either interactive or non-interactive requests to those servers constantly. The interactive workloads need to be served in each server, and the non-interactive workloads can be processed by just one server.

We conduct a series of experiments in the next section: 1. Overall performance. We conduct a series of experiments, measuring the average response time of both interactive and non-interactive workloads under D^{3}G versus the state-of-the-art scheme without D^{3}G. 2. Algorithm dynamism. We test the algorithm performance under a dynamic scenario. 3. System scalability. We evaluate the optimization of D^{3}G under expanding scales.

V. Evaluation

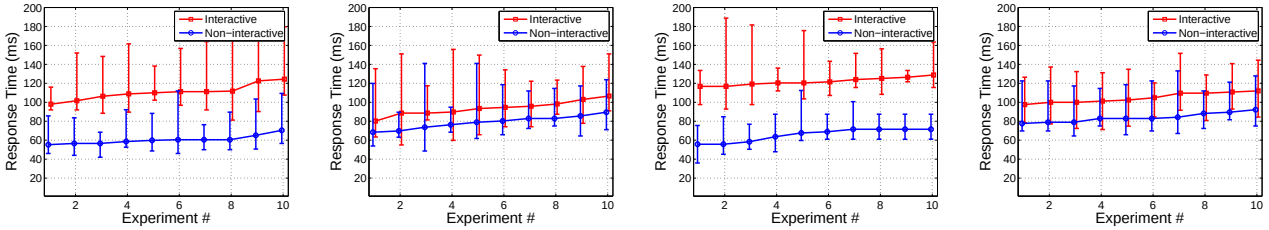
As described in the deployment section, we conduct three groups of experiments to test algorithm efficiency and evaluate average response time and service performance under different network environments. Recall the example in the Motivation Section, the interactive workloads are actually suffering from 7 times longer latency compared to non-interactive workloads.

The experiment results in this section show that D^{3}G significantly reduces latency for time-sensitive workloads. At the same time, non-interactive workloads are not affected seriously and are still enjoying shorter latencies. Besides verifying the effectiveness of D^{3}G algorithm, we also prove the potential practicability for large-scale deployment.

A. Overall Performance

In this subsection, we design several groups of experiments to evaluate the performance of D^{3}G under different network environments. We start the experiment in a 9-to-1 model. In this case, every 9 end-hosts keep sending interactive and non-interactive requests to one server. We record the response time of different workloads under different scenarios.

For small requests, we set the workload length from 100 KB to 200 KB, conduct the experiments for 200 times and



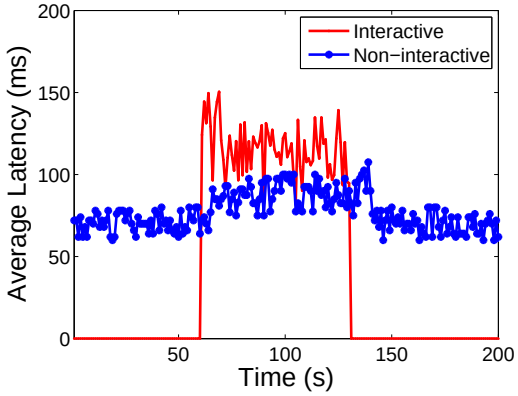
(a) Latency without D^{3}G when requests are (100KB, 200KB].

(b) Latency with D^{3}G when requests are (100KB, 200KB].

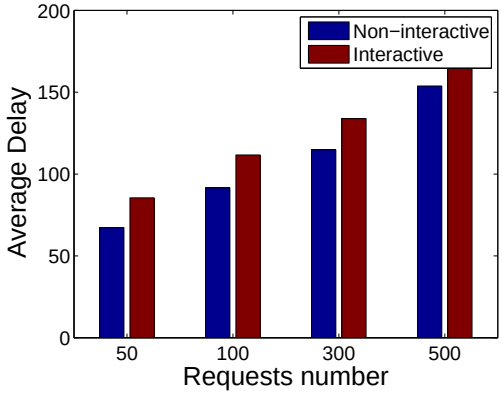
(c) Latency without D^{3}G when requests are 200KB and above.

(d) Latency with D^{3}G when requests are 200KB and above.

Fig. 4: Average response time of both interactive and non-interactive workloads.



(a) Average Delay for dynamic scenario.



(b) Average Delay for different scales.

Fig. 5: Performance for different parameters.

calculate the average response time per 20 experiments with upper and lower error bars. Figure 4(a) shows the response time of both interactive and non-interactive requests before deploying D^{3}G, and Figure 4(b) shows the response time after deploying D^{3}G. These figures show that D^{3}G can significantly reduce the response time of interactive workloads (from 110 ms to 95 ms on average), and the latency for non-interactive workloads is not seriously affected (from 60 ms to 75 ms).

For large requests, we set the workload length at least 200 KB. Figure 4(c) shows the results without D^{3}G, from which we can observe that the average latency for interactive workloads is about 120 ms and that for non-interactive workloads is about 75 ms, indicating that interactive requests are suffering from more than 2 times longer latency than non-interactive requests. Figure 4(d) shows the optimized results after deploying D^{3}G, where interactive workload latency is reduced by 33% (to 80 ms) on average with minimal impact on non-interactive workload latency.

From these results, we can conclude that D^{3}G works well in accelerating interactive workloads under various circumstances.

B. Algorithm Dynamism

To evaluate our algorithm in dynamic scenarios, we simulate a dynamic situation to verify the real-time efficiency of D^{3}G.

We send only non-interactive requests in the previous 60 s, and then begin to send interactive requests at the $60th$ s and stop sending at the $130th$ s. Figure 5(a) shows the average delay of this dynamic process. The latency is quite long for a short period of time (from 60 s to 70 s). Then it begins to drop because more resources are allocated to the interactive queue. When interactive workloads stop at the $130th$ s, non-interactive workload latency drops.

From these experiments, we can conclude that D^{3}G accelerate the interactive workloads while not seriously affecting non-interactive workloads.

C. System Scalability

At last, we extend experiment scales and increase concurrency to test algorithm scalability.

We speed up the request sending rate, and Figure 5(b) shows the average latency of various scales. When there are 50 concurrent requests, the average latency is about 65 ms for non-interactive workloads and 80 ms for interactive ones. When the number of concurrent requests increases to 500, the average latencies are about 150 ms and 170 ms, respectively. These results indicate that our algorithm is extensible in large-scale systems. Furthermore, if the interactive workloads are handled by more cloud servers, the latency without D^{3}G will get even higher (like the case in the Motivation Section), and our algorithm optimization will be more obvious.

From the above deployment and evaluations, we can conclude that D^{3}G successfully reduce the interactive workloads latency to a reasonable range with no distinct impact on non-interactive workloads, even in expanding scales. We believe that the crux idea of D^{3}G, reduce latency for interactive workloads from time-sensitive applications, will soon be adopted by nowadays microservices.

VI. Conclusion

For cloud-based service chains, we measure and analyze their performance in Baidu networks, and the results show that these delay-sensitive microservice-like applications are suffering from long latency due to the extra delay from the multiple stand-alone components.

In this paper, we propose a new algorithm called Dynamic Differentiated service for Delay-Guarantee (D^3G), which aims at reducing the overall latency for chained applications while ensuring workload fairness. To this end, we design two queues in servers. One is for interactive requests, and the other is for non-interactive requests. To make the latency within user tolerance, the latency estimation algorithm pre-calculates interaction latency. Furthermore, to guarantee fairness, we introduce a feedback control scheme based on resource allocation to ensure the performance for non-interactive workloads. A wide range of detailed evaluation results demonstrate that D^3G succeeds in accelerating chained services and ensuring workload fairness. As microservice-like application has many obvious advantages such as clean boundary, better system scalability and lower operation cost, it is attracting more and more attentions. We believe that D^3G would further reveal its effectiveness along with the development of service chains.

VII. Acknowledgement

Ke Xu's work was partly supported by the National Natural Foundation of China (61472212) and EU Marie Curie Actions CROWN (FP7-PEOPLE-2013-IRSES-610524). Qi Li's work was supported by the National Natural Foundation of China (61472212 and 61572278), EU Marie Curie Actions CROWN (FP7-PEOPLE-2013-IRSES-610524), the National Key R&D Program of China (2016YFB0800102), and the R&D Program of Shenzhen (JCYJ20170307153259323).

References

[1] Guo Y, Gong Y, Fang Y, et al. Energy and network aware workload management for sustainable data centers with thermal storage[J]. IEEE Transactions on Parallel and Distributed Systems, 2014, 25(8): 2030-2042.

[2] Alizadeh M, Yang S, Sharif M, et al. pfabric: Minimal near-optimal datacenter transport[C]//ACM SIGCOMM Computer Communication Review. ACM, 2013, 43(4): 435-446.

[3] Dogar F R, Karagiannis T, Ballani H, et al. Decentralized task-aware scheduling for data center networks[C]//ACM SIGCOMM Computer Communication Review. ACM, 2014, 44(4): 431-442.

[4] Zhang Y, Xu K, Wang H, et al. Towards shorter task completion time in datacenter networks[C]//Computing and Communications Conference (IPCCC), 2015 IEEE 34th International Performance. IEEE, 2015: 1-8.

[5] Mauve M, Vogel J, Hilt V, et al. Local-lag and timewarp: providing consistency for replicated continuous applications[J]. IEEE transactions on Multimedia, 2004, 6(1): 47-57.

[6] Webb S D, Soh S, Lau W. Enhanced mirrored servers for network games[C]//Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games. ACM, 2007: 117-122.

[7] Vik K H, Halvorsen P, Griwodz C. Multicast tree diameter for dynamic distributed interactive applications[C]//INFOCOM 2008. The 27th Conference on Computer Communications. IEEE. IEEE, 2008: 1597-1605.

[8] Guo J, Liu F, Zeng D, et al. A cooperative game based allocation for sharing data center networks[C]//INFOCOM, 2013 Proceedings IEEE. IEEE, 2013: 2139-2147.

[9] Xu K, Zhang Y, Shi X, et al. Online combinatorial double auction for mobile cloud computing markets[C]//Performance Computing and Communications Conference (IPCCC), 2014 IEEE International. IEEE, 2014: 1-8.

[10] Zaki Y, Chen J, Ptsch T, et al. Dissecting web latency in ghana[C]//Proceedings of the 2014 Conference on Internet Measurement Conference. ACM, 2014: 241-248.

[11] Pujol E, Richter P, Chandrasekaran B, et al. Back-office web traffic on the internet[C]//Proceedings of the 2014 Conference on Internet Measurement Conference. ACM, 2014: 257-270.

[12] Wang H, Shea R, Ma X, et al. On design and performance of cloud-based distributed interactive applications[C]//Network Protocols (ICNP), 2014 IEEE 22nd International Conference on. IEEE, 2014: 37-46.

[13] Carlstrom J, Rom R. Application-aware admission control and scheduling in web servers[C]//INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE. IEEE, 2002, 2: 506-515.

[14] Mitzenmacher M. The power of two choices in randomized load balancing[J]. IEEE Transactions on Parallel and Distributed Systems, 2001, 12(10): 1094-1104.

[15] Pebesma E, Cornford D, Dubois G, et al. INTAMAP: the design and implementation of an interoperable automated interpolation web service[J]. Computers & Geosciences, 2011, 37(3): 343-352.

Yuchao Zhang received the Bachelor of Science degree in computer science and technology from Jilin University, China in 2012. Then she received her Ph.D. degree in the Department of Computer Science & Technology of Tsinghua University, Beijing, China in 2017. Currently she works in Beijing University of Posts and Telecommunications. Her research interests include cloud computing, large-scale datacenter networks, high-speed network and network function virtualization.

Ke Xu (M'02-SM'09) received his Ph.D. from the Department of Computer Science & Technology of Tsinghua University, Beijing, China, where he serves as a full professor. He has published more than 100 technical papers and holds 20 patents in the research areas of next generation Internet, P2P systems, Internet of Things (IoT), network virtualization and optimization. He is a member of ACM and has guest-edited several special issues in IEEE and Springer Journals. Currently, he is holding visiting professor position at the University of Essex.

Haiyang Wang is an assistant professor in the Department of Computer Science at the University of Minnesota Duluth, USA. His research interests include cloud computing, peer-to-peer networking, social networking, big data and multimedia communications.

Qi Li (M'12) received the B.Sc. and Ph.D. degrees in computer science from Tsinghua University, Beijing, China, in 2003 and 2012, respectively. His research interests include network architecture, protocol design, and system and network security.

Tong Li received his B.S. degree from the Department of Computer Science of Wuhan University, Hubei, China in 2012. Then he received his Ph.D. degree in the Department of Computer Science & Technology of Tsinghua University in 2017. Currently he works in Huawei Company. His research interests include network virtualization and resource management, and network science.

Xuan Cao received his B.S. and Master degree from the Department of Computer Science of Nanjing University of Science and technology in 2008 and 2011, respectively. He joined Beijing Baidu Netcom Science Technology Co., Ltd since then. His research interests include resource management, network virtualization, pattern recognition and intelligent system.