TRUST: Real-Time Request Updating with Elastic Resource Provisioning in Clouds

Jingzhou Wang[1,2] Gongming Zhao[1,2] Hongli Xu[1,2] Yangming Zhao[1,2] Xuwei Yang[3] He Huang[4]
[1]School of Computer Science and Technology, University of Science and Technology of China
[2]Suzhou Institute for Advanced Research, University of Science and Technology of China
[3]Huawei Technologies Co., Ltd
[4]School of Computer Science and Technology, Soochow University

***Abstract*—In a commercial cloud, service providers (*e.g.*, video streaming service provider) rent resources from cloud vendors (*e.g.*, Google Cloud Platform) and provide services to cloud users, making a profit from the price gap. Cloud users acquire services by forwarding their requests to corresponding servers. In practice, as a common scenario, traffic dynamics will cause server overload or load-unbalancing. Existing works mainly deal with the problem by two methods: *elastic resource provisioning* and *request updating*. Elastic resource provisioning is a fast and agile solution but may cost too much since service providers need to buy extra resources from cloud vendors. Though request updating is a free solution, it will cause a significant delay, resulting in a bad users' QoS. In this paper, we present a new scheme, called real-time request updating with elastic resource provisioning (TRUST), to help service providers pay less cost with users' QoS guarantee in clouds. In addition, we propose an efficient algorithm for TRUST with a bounded approximation factor based on randomized rounding. Both small-scale experiment results and large-scale simulation results show the superior performance of our proposed algorithm compared with state-of-the-art benchmarks.**

Index Terms*—Cloud Computing, Elasticity, Request Updating, Resource Provisioning

I. Introduction

Driven by the rapid growth of the demand for flexible and efficient computation power, cloud computing has gained much attention from both industry and academia [1]. Comparing with building IT infrastructure, cloud computing liberates us from cumbersome tasks (*e.g.*, managing and maintaining devices). Consequently, cloud computing has gained enormous economic earnings in recent years thanks to its convenience and efficiency, and an increasing number of enterprises/individuals are outsourcing their services/workloads to clouds [2].

In practice, there are mainly three roles in clouds: cloud vendors, service providers and cloud users [3]. Cloud vendors (*e.g.*, Amazon Web Services and Google Cloud Platform) are responsible for building and maintaining physical servers. They lease a certain amount of VMs/containers to each service provider. Then, service providers can implement their services like security [4], storage [5], auditing [6], and so on, also called the *Network Function* (NF) instances. Cloud users can purchase the services according to their requirements, so their requests can be scheduled to the corresponding NFs. For instance, a VPN service provider rents several VMs from a cloud vendor to implement a VPN function. Then, cloud users can buy the VPN service from the service provider and acquire services by forwarding their traffic to the corresponding VPNs.

In a large-scale multi-tenant cloud, it is evident that the amount of traffic is tremendous and traffic dynamics is a long-standing problem [2], causing load-unbalancing and even overload among NFs, which will severely affect service availability and execution efficiency [7]. Consequently, it will decrease users' QoS [8]. To deal with traffic dynamics in clouds and ameliorate the QoS, request updating has been widely adopted, which means transferring the requests to another available NF. Existing works usually design the request updating schemes for NF load-balancing [9] [10] or minimizing the makespan [11] [12]. For example, the authors in [9] present a distributed request scheduling mechanism so as to achieve load balancing among servers in data centers by fairly distributing the traffic from the edge switches. The work [12] tries to find an optimal solution for less cloud resource cost under the deadline constraint via the immune-based particle swarm optimization algorithm [13].

Although plenty of works have designed reasonable request updating schemes to handle traffic dynamics, there exist two fundamental disadvantages in request updating. Firstly, the delay of updating requests may severely degrade users' QoS [14]. Commercial clouds usually adopt the architecture of centralized control [15] and the clouds are managed by the controller. The update delay for requests updating may become the system bottleneck [16], due to the limited processing capacity of the controller. For instance, it takes at least 0.5ms for the controller to update a flow entry on a switch [17]. Assume that more than 80% of requests/flows last less than 10s [18] and millions of requests are injected into the cloud every minute [19]. The update delay may affect a considerable number of requests. That is because during the delay time-span, many requests have terminated and many new requests have arrived [20]. Secondly, since NFs need to record the states of processed requests, there exists request state consistency issue while transferring requests. Request updating will bring extra delay and overhead to maintain the requests' state consistency on NFs [21]. For example, after request updating, if a user's requests are scheduled to an NF without maintaining the requests' state, it will incur request state inconsistency and wrong operations. Because of these two disadvantages, request updating is hard to assure users' QoS in large-scale clouds, and alternative solutions as supplementary methods are urgently needed.

By enabling the virtualization technology, elastic resource provisioning has become a new trend to deal with traffic dynamics in clouds [22] [23] [24] [25] [26]. Elastic resource provisioning means that the system can add or remove resources (such as CPU cores, memory, VM or container instances) to adapt to load variation in a real-time manner [26]. In practice, the cloud vendors usually would offer several suitable configuration types associated with fixed resource combinations [23] and each NF will choose one of the resource configuration types to serve requests from cloud users. For instance, in Google Cloud Platform (GCP) [27], the price for a virtual machine with 4 CPU cores and 15 GB memory is \$97 per month and the price for a VM with 8 CPU cores and 30 GB memory is \$197 per month. The previous works about elastic resource provisioning mainly focus on how to improve network performance [28], increase resource capacity [29] or save the energy [30]. For instance, the authors in [28] provide automatic deployment and proactive scaling of multiple simultaneous web applications methods to improve the infrastructure performance, which has been deployed in Amazon Elastic Compute Cloud.

The idea of elastic resource provisioning holds an excellent promise of solving the problem of traffic dynamics. Compared with request updating solutions, resizing a VM/container only takes tens of milliseconds [23] [31] and there is no need to worry about the problem of request state consistency. Accordingly, users' QoS can be guaranteed. However, it will increase the cost of service providers since they should pay more to cloud vendors for the extra resources. In contrast, with the request updating method, service providers do not have to pay the extra money, but users' QoS may be affected, due to the update delay and the requirement of request state consistency. Hence, we find the two approaches can be complementary to each other to help service providers pay less cost with users' QoS guarantee.

In this paper, we propose real-time request updating with elastic resource provisioning (TRUST) when facing traffic dynamics. Specifically, since request updating would increase the update delay and decrease users' QoS, we try to finish the updating operation under the time threshold T, which will be determined by users' QoS demand. For instance, in a cellular communication network, the authors in [32] show that when the delivery delay is below 150ms, the quality of propagation can be still guaranteed. Meanwhile, we try to minimize the infrastructure cost of purchasing cloud resources. In a nutshell, TRUST can enlighten a way that helps service providers to spend less money with users' QoS guarantee. The main contributions of this paper can be summarized as follows:

1) We comprehensively analyze the current methods to deal with traffic dynamics in clouds and show the advantages and weaknesses of request updating and elastic resource provisioning.
2) We give the formulation of real-time request updating with elastic resource provisioning (TRUST), which can assure users' QoS and save money for service providers. To our best knowledge, this is the first work that takes advantages of both elastic resource provisioning and request updating to handle traffic dynamics.
3) We show the problem complexity of TRUST and present a randomized rounding based algorithm. The performance analysis shows that our algorithm can achieve the optimal value with a high probability.
4) We conduct small-scale testbed experiments and large-scale simulations using real-world topologies and datasets to show that the proposed algorithm can achieve superior performance compared with the state-of-the-art solutions.

The rest of this paper is organized as follows. Section II introduces the preliminaries of our work, including the commercial cloud model, a motivating example and the problem formulation. Section III presents our proposed algorithm based on randomized rounding. The simulation and testbed evaluation results are presented in Section IV. We conclude this paper in Section V.

II. Preliminaries

A. Commercial Cloud Model

We first present the commercial cloud model. A typical commercial cloud mainly consists of three parts: cloud vendors, service providers and cloud users. Cloud vendors (1) host a set of physical machines, (2) construct and maintain the VMs/containers upon the physical servers, and (3) sell or lease resources to service providers and users. Cloud vendors set several configuration types of VMs/containers, each associated with a certain amount of computing resources (*e.g.*, CPU or RAM). Of course, they will charge a reasonable price for the resources service providers use. Service providers will rent resources from cloud vendors such as a VM with 4 core CPUs and 8G RAM to implement their services, *e.g.*, VPN and ELB, also called *Network Function (NF)* instances. Service providers sell these services to cloud users and process the requests from them. Service providers also need to pay the infrastructure prices to cloud vendors, making a profit from the price gap. Obviously, the service providers all wish to spend less money on infrastructure costs and provide services to users with high QoS guarantee.

We use $S = \{s_1, s_2, ...s_{|S|}\}$ to denote the set of physical servers maintained by cloud vendors, and each server s comes with limited resources $R(s)$. Here $R(s)$ can be expanded into a resource vector to represent different types of resources on server s, such as CPU, RAM and bandwidth. Let $C = \{c_1, c_2, ...c_{|C|}\}$ represent the set of configuration types, and each configuration type c comes with resources usage $r(c)$, processing capacity $p(c)$ and infrastructure cost $m(c)$. Note that, $r(c)$ can also be expanded into a resource vector. $N = \{n_1, n_2, ...n_{|N|}\}$ represents the set of NFs and N_s denotes the set of NFs on the physical server $s \in S$. We use $\Gamma = \{\gamma_1, \gamma_2, ...\gamma_{|\Gamma|}\}$ to denote the set of requests generated by cloud users. Each request γ comes with a traffic size of $f(\gamma)$, which can be acquired by collecting flow statistics information on the controller [33].

Type	Configuration	Price	Capacity
A	4 core CPU 15G RAM	$97	50Gbps
B	8 core CPU 30G RAM	$197	100Gbps

TABLE I: Configuration types and prices according to Google Cloud Platform [27]. Configuration type A contains a 4 core CPU and 15G RAM and is capable to handle 50Gbps requests. Configuration type B contains a 8 core CPU and 30G RAM and is capable to handle 100Gbps requests.

Method	NF Type			NF Traffic (Gbps)			Money	Update Delay
	NF_1	NF_2	NF_3	NF_1	NF_2	NF_3		
Original	A	A	A	65	56	27	$291	\
ERP	**B↑**	**B↑**	A	65	56	27	$491	≈ 0
RU	A	A	A	**50↓**	**50↓**	**48↑**	$291	21
TRUST	**B↑**	A	A	65	**50↓**	**33↑**	$391	6

TABLE II: Algorithm Comparison. ERP will upgrade the configuration of NF_1 and NF_2 to type B, which costs $491 and negligible update delay. RU needs to update $1500/600$ requests from NF_1/NF_2 to NF_3, which costs 21 units update delay and $291. TRUST will transfer 600 requests from NF_2 to NF_3, and upgrade the configuration type of NF_1. This costs $391 and 6 units update delay.

B. A Motivating Example

This section presents a motivating example to demonstrate the advantages of TRUST compared with existing solutions.

As shown in Table I, we assume that cloud vendors will provide two kinds of NF configuration types. A video streaming service provider owns three NFs, denoted as NF_1, NF_2 and NF_3, all initialized as type A to provide high quality online videos. NFs with type A are able to deal with 50 Gbps traffic load. Now the service provider is facing traffic dynamics. The current traffic loads of three NFs are 65 Gbps, 56 Gbps and 27 Gbps, respectively, which means overload on both NF_1 and NF_2. Suppose that every user is enjoying the similar video streaming service and one request takes 10 Mbps traffic load on average. Then, we assume that transferring 100 requests from one NF to another will cost one unit time-span. To guarantee the users' QoS, the service provider needs to finish the updating process within 10 units. Otherwise, the delay may upset the users. The performance comparison of three algorithms is shown in Table II.

Existing works mainly deal with traffic dynamics by two methods: Elastic Resource Provisioning (ERP) and Request Updating (RU). ERP (*e.g.*, [26] [34]) will buy extra resources to cope with the burstiness traffic. It is an agile solution and will not cause a decrease in users' QoS, but may be expensive. Specifically, in this example, ERP needs to upgrade the configuration to type B for both NF_1 and NF_2. This way will not involve significant update delay and the problem of request state consistency, but cost the service provider an extra 200 USD per month.

RU (*e.g.*, [35] [36]) mainly transfers the requests from the overloaded NF to another available one to alleviate the traffic dynamics. It is a free solution but may spend much time on request updating. Specifically, RU needs to update $1500/600$ requests from NF_1/NF_2 to NF_3, and will cost 21 units timespan, which means RU cannot well satisfy the QoS demand of users. Moreover, RU may not be able to hold all the traffic under some extreme situations due to the limited capacity of NFs. Under this circumstance, the service provider has to abandon or deny some requests.

In practice, since a small delay may not affect the QoS of users remarkably [37], we try to combine the two methods. As a result, service providers will spend less money and still guarantee the QoS. We introduce TRUST, a brand-new approach that combines these two methods. If TRUST is adopted in this example, we only need to transfer 600 requests from NF_2 to NF_3, and upgrade the configuration type of NF_1. Compared with RU, TRUST can finish the updating process within 10 units timespan and will not cost too much compared with ERP. This example fully demonstrates the advantages of TRUST.

C. Problem Formulation of TRUST

This section describes the problem formualtion of TRUST. In a commercial cloud, the service provider serves the requests from cloud users and processes the requests in NFs. As time goes by, load unbalancing occurs among NFs due to traffic dynamics. At this time, the service provider will choose to change the configuration type of some NFs or update the requests. To help the service provider spend less money on NFs and save time from updating requests, also for achieving a better QoS of users, we should consider the following constraints in coping with traffic dynamics.

1) *Physical Server Resource Constraint:* The total resources used by NFs that are on the same physical server cannot exceed the whole resources of the physical server.

2) *NF Capacity Constraint:* The configuration type that an NF chooses must be able to handle the requests it receives.

3) *Update Delay Constraint:* t_0 represents the update delay of a single request, which is determined by the system hardware performance. For example, it takes about 0.5ms for updating a single request by the test result of [35]. Since the controller will encapsulate and install a flow entry for each updated request, the total update delay can be approximately linear with the number of updated requests [18] [38]. To assure users' QoS, the total update delay cannot exceed the time threshold T, which is determined by user's QoS demand. For instance, in a cellular communication environment, T is set to 150ms [32].

We use the variable $x_c^n \in \{0, 1\}$ to denote whether NF n will choose the configuration type c ($x_c^n = 1$) or not ($x_c^n = 0$). The variable $y_n^\gamma \in \{0, 1\}$ represents whether the request γ will be sent to NF n ($y_n^\gamma = 1$) or not ($y_n^\gamma = 0$). We use a constant $\beta(\gamma, n)$ to denote whether the request γ is assigned to NF n before update ($\beta(\gamma, n) = 0$) or not ($\beta(\gamma, n) = 1$). The formulation is as follows:

$$\min \sum_{n\in N}\sum_{c\in C} m(c)\cdot x_c^n$$

$$S.t. \begin{cases} \sum_{c\in C} x_c^n = 1, & \forall n \in N \\ \sum_{n\in N} y_n^\gamma = 1, & \forall \gamma \in \Gamma \\ \sum_{n\in N_s}\sum_{c\in C} x_c^n \cdot r(c) \le R(s), & \forall s \in S \\ \sum_{\gamma\in\Gamma} y_n^\gamma \cdot f(\gamma) \le \sum_{c\in C} x_c^n \cdot p(c), & \forall n \in N \\ \sum_{\gamma\in\Gamma}\sum_{n\in N} y_n^\gamma \cdot \beta(\gamma,n)\cdot t_0 \le T, & \\ x_c^n, y_n^\gamma \in \{0,1\}, & \forall n, c, \gamma \end{cases} \quad (1)$$

The first set of equations means that each NF will choose a configuration type $c \in C$, and the second set of equations means each request γ will be sent to an NF $n \in N$. The third set of inequalities denotes the physical server resource constraint that all the resources allocated to NFs should not exceed the total resources on the physical server s. The fourth set of inequalities describes that the traffic load on each NF should not exceed its capacity, *i.e.*, the NF capacity constraint. The fifth set of inequalities means that the duration of updating request should not exceed the time threshold T. Our objective function is to minimize the total infrastructure cost, *i.e.*, $\sum_{n\in N}\sum_{c\in C} m(c)\cdot x_c^n$.

Theorem 1: TRUST defined in Eq. (1) is an NP-Hard problem.

Proof: We can prove the NP-hardness by showing that the bin-packing problem [39] is a special case of TRUST. Due to limited space, we omit the detailed proof here. ■

III. Algorithm Description

A. Randomized Rounding Algorithm for TRUST

This section presents an approximation algorithm based on randomized rounding [40] for TRUST.

The first step is to relax Eq. (1) into an LP by replacing integer constraints with linear constraints. In this way, we can solve it with a linear program solver (*e.g.*, PuLP [41]) and the solutions are denoted by $\{\widetilde{x}_c^n\}$ and $\{\widetilde{y}_n^\gamma\}$.

In the second step, we identify which configuration type each NF should choose and which NF each request should be scheduled to, *i.e.*, obtain feasible solutions $\{\widehat{x}_c^n\}$ and $\{\widehat{y}_n^\gamma\}$. Then, we round $\widehat{x}_c^n$ to 1 with probability $\widetilde{x}_c^n$ and round $\widehat{y}_n^\gamma$ to 1 with probability $\widetilde{y}_n^\gamma$. We should note that the choice is done in an exclusive manner, *i.e.*, for each n, exactly one of $\{\widetilde{x}_c^n\}$ is set to one; the rest are set to zero. And for each γ, exactly one of $\{\widetilde{y}_n^\gamma\}$ is set to one; the rest are set to zero. According to rounded $\{\widehat{x}_c^n\}$ and $\{\widehat{y}_n^\gamma\}$, we can derive a joint elastic resource provisioning and request updating scheme. The formal algorithm is shown in Algorithm 1.

B. Performance Analysis

Before we analyze algorithm performance, we introduce two famous lemmas for approximation performance analysis.

Lemma 2: (Chernoff Bound) Given n independent variables: $x_1, x_2, ..., x_n$, where $\forall x_i \in [0,1]$. Let $\mu = \mathbb{E}[\sum_{i=1}^n x_i]$. Then, we have $\mathbf{Pr}[\sum_{i=1}^n x_i \ge (1+\epsilon)\mu] \le e^{\frac{-\epsilon^2\mu}{2+\epsilon}}$ and

Algorithm 1 Randomized Rounding-based Algorithm for TRUST

```
1:  Step 1: Solving the Relaxed Formulation
2:  Construct a linear program by replacing the integral constraints with $x_c^n$ and $y_n^\gamma \in [0,1]$
3:  Obtain the optimal solutions $\{\widetilde{x}_c^n\}$ and $\{\widetilde{y}_n^\gamma\}$
4:  Step 2: Acquire a feasible solution by randomized rounding
5:  for each NF $n \in N$ do
6:      Choose one $\widehat{x}_c^n = 1$ with probability $\widetilde{x}_c^n$ and the rest are set 0
7:  for each request $\gamma \in \Gamma$ do
8:      Choose onr $\widehat{y}_n^r = 1$ with probability $\widetilde{y}_n^\gamma$ and the rest are set 0
9:  Solve Eq. (1) get the new solution $\{\widehat{x}_c^n\}$ and $\{\widehat{y}_n^\gamma\}$.
10: Step 3: Derive a joint elastic resource provisioning and request updating scheme according to $\{\widehat{x}_c^n\}$ and $\{\widehat{y}_n^\gamma\}$
```

$\mathbf{Pr}[\sum_{i=1}^n x_i \le (1-\epsilon)\mu] \le e^{\frac{-\epsilon^2\mu}{2}}$ where ϵ is an arbitrarily positive value.

Lemma 3: (Union Bound) Given an accountable set of n events: $A_1, A_2, ...A_n$, each event A_i happens with probability $\mathbf{Pr}(A_i)$. Then, $\mathbf{Pr}(A_1 \cup A_2 \cup ... \cup A_n) \le \sum_{i=1}^n \mathbf{Pr}(A_i)$.

Theorem 4: The proposed algorithm can obtain a value of infrastructure cost close to the optimal infrastructure cost derived by solving the LP with a high probability.

Proof: We define the infrastructure cost of NF n: $P_n = \sum_{c\in C} x_c^n \cdot m(c)$. Then, the total infrastructure cost, *i.e.*, the objective function can be denoted as: $\sum_{n\in N} P_n$. According to Algorithm 1, we know the infrastructure cost of NF n derived by LP, *i.e.*, the optimal value is $\sum_{n\in N}\widetilde{P}_n = \sum_{n\in N}\sum_{c\in C}\widetilde{x}_c^n \cdot m(c)$. After the rounding procedure of algorithm, we acquire the feasible solution $\widehat{x}_c^n$. Thus, we can determine the infrastructure cost derived by algorithm, *i.e.*, $\widehat{P}_n$:

$$\widehat{P}_n = \begin{cases} m(c), & with\ probability\ \widetilde{x}_c^n \\ 0, & otherwise \end{cases} \quad (2)$$

So we have $\mathbb{E}[\sum_{n\in N}\widehat{P}_n] = \sum_{n\in N}\sum_{c\in C}\widetilde{x}_c^n \cdot m(c) = \sum_{n\in N}\widetilde{P}_n$. Then, we define the highest price of configuration type is p_{max}. So we can say $\mathbb{E}[\frac{\sum_{n\in N}\widehat{P}_n}{|N|\cdot p_{max}}] \in [0,1]$, where $|N|$ is the number of NFs. We should note that for different n, $\widehat{P}_n$ is independent from each other. Then, based on Lemma 2, we can conclude that

$$\mathbf{Pr}[\frac{\sum_{n\in N}\widehat{P}_n}{|N|\cdot p_{max}} \ge (1+\epsilon)\mathbb{E}[\frac{\sum_{n\in N}\widehat{P}_n}{|N|\cdot p_{max}}]] \le e^{-\frac{\epsilon^2}{2+\epsilon}\mathbb{E}[\frac{\sum_{n\in N}\widehat{P}_n}{|N|\cdot p_{max}}]}$$
$$\Leftrightarrow \mathbf{Pr}[\sum_{n\in N}\widehat{P}_n \ge (1+\epsilon)\sum_{n\in N}\widetilde{P}_n] \le e^{-\frac{\epsilon^2}{2+\epsilon}\frac{\sum_{n\in N}\widetilde{P}_n}{|N|\cdot p_{max}}} \quad (3)$$

Eq. (3) means that the infrastructure cost derived by algorithm, *i.e.*, $\sum_{n\in N}\widehat{P}_n$ will be more than the value derived by LP, *i.e.*, $\sum_{n\in N}\widetilde{P}_n$ with a very low probability. Thus, we can say the output derived by algorithm, *i.e.*, $\sum_{n\in N}\widehat{P}_n$ is close to the optimal solution of LP, *i.e.*, $\sum_{n\in N}\widetilde{P}_n$ with a high probability. ■

Theorem 5: The algorithm guarantees that the total resources of NFs in server s will not exceed the total resource of server s by a factor of $O(\log|S|)$, where $|S|$ is the number of NFs.

Proof: We use $\mathbb{R}_n$ to denote the resource of NF n and $\mathbb{R}_n=\sum_{c\in C}x_c^n\cdot r(c)$. After solving the LP, $\widetilde{\mathbb{R}}_n=\sum_{c\in C}\widetilde{x}_c^n\cdot r(c)$. Then, we adopt the rounding procedure of $\widetilde{x}_c^n$ to acquire a feasible solution $\widehat{x}_c^n\in\{0,1\}$ and $\widehat{\mathbb{R}}_n$. Specifically, we round $\widehat{x}_c^n$ to 1 with probability $\widetilde{x}_c^n$. Thus, we can determine the capacity of NF n derived by algorithm, *i.e.*, $\widehat{\mathbb{R}}_n$:

$$\widehat{\mathbb{R}}_n=\begin{cases}r(c), & with\ probability\ \widetilde{x}_c^n\\ 0, & otherwise\end{cases}\tag{4}$$

So we have

$$\mathbb{E}[\widehat{\mathbb{R}}_n]=\sum_{c\in C}\widetilde{x}_c^n\cdot r(c)=\widetilde{\mathbb{R}}_n\tag{5}$$

and

$$\mathbb{E}[\sum_{n\in N_s}\widehat{\mathbb{R}}_n]=\sum_{n\in N_s}\sum_{c\in C}\widetilde{x}_c^n\cdot r(c)\le R(s)\tag{6}$$

Then we define a constant α as follows:

$$\alpha=\frac{R_{min}}{r_{max}}\tag{7}$$

where R_{min} denotes the minimal resource among servers and r_{max} denotes the maximum resource among configuration types.

Combining the definition of α and Eq. (6), we have

$$\begin{cases}\frac{\widehat{\mathbb{R}}_n\cdot\alpha}{R(s)}\in[0,1]\\ \mathbb{E}[\sum_{n\in N_s}\frac{\widehat{\mathbb{R}}_n\cdot\alpha}{R(s)}]\le\alpha\end{cases}\tag{8}$$

We should note that for different n, $\widehat{\mathbb{R}}_n$ is independent from each other. Based on Lemma 2, we have

$$\mathbf{Pr}[\sum_{n\in N_s}\frac{\widehat{\mathbb{R}}_n\cdot\alpha}{R(s)}\ge(1+\epsilon)\alpha]\le e^{-\frac{\epsilon^2\cdot\alpha}{2+\epsilon}}\tag{9}$$

Now we assume that

$$\mathbf{Pr}[\sum_{n\in N_s}\frac{\widehat{\mathbb{R}}_n}{R(s)}\ge(1+\epsilon)]\le e^{-\frac{\epsilon^2\cdot\alpha}{2+\epsilon}}\le\frac{1}{|S|^{k+1}}\tag{10}$$

where k is an arbitrary positive integer. We know that $\frac{1}{|S|^{k+1}}\to 0$ when network size grows. The solution to Eq. (10) is

$$\epsilon\ge\frac{(k+1)\log|S|+\sqrt{(k+1)^2\log^2|S|+8(k+1)\alpha\log|S|}}{2\alpha},$$
$$\Rightarrow\epsilon\ge\frac{(k+1)\log|S|}{\alpha}+2,\quad |S|\ge 2\tag{11}$$

By applying Lemma 3, we have

$$\begin{aligned}&\mathbf{Pr}[\bigcup_{s\in S}\sum_{n\in N_s}\frac{\widehat{\mathbb{R}}_n}{R(s)}\ge(1+\epsilon)]\\ \le&\sum_{s\in S}\mathbf{Pr}[\sum_{n\in N_s}\frac{\widehat{\mathbb{R}}_n}{R(s)}\ge(1+\epsilon)]\\ \le&|S|\cdot\frac{1}{|S|^{k+1}}=\frac{1}{|S|^k}\end{aligned}\tag{12}$$

Thus, we can conclude that the approximation factor is $\epsilon+1=\frac{(k+1)\log|S|}{\alpha}+3$. ■

Lemma 6: Suppose that $\widetilde{C}_n$ is the optimal value of the capacity of NF n to the LP while $\widehat{C}_n$ is the value associated with Algorithm 1. $\widehat{C}_n$ will be at most less than $\widetilde{C}_n$ by a factor of $(1-\epsilon)$, where ϵ is an arbitrarily positive value.

Proof: We use C_n to denote the capacity of NF n and $C_n=\sum_{c\in C}x_c^n\cdot p(c)$. After solving the LP, $\widetilde{C}_n=\sum_{c\in C}\widetilde{x}_c^n\cdot p(c)$. Then, we adopt the rounding procedure of $\widetilde{x}_c^n$ to acquire a feasible solution $\widehat{x}_c^n\in\{0,1\}$. Specifically, we round $\widehat{x}_c^n$ to 1 with probability $\widetilde{x}_c^n$. Thus, we can determine that

$$\widehat{C}_n=\begin{cases}p(c), & with\ probability\ \widetilde{x}_c^n\\ 0, & otherwise\end{cases}\tag{13}$$

So we have

$$\mathbb{E}[\widehat{C}_n]=\sum_{c\in C}\widetilde{x}_c^n\cdot p(c)=\widetilde{C}_n\tag{14}$$

Clearly, we also have $\frac{\widehat{C}_n}{c_{max}}$ and $\mathbb{E}[\frac{\widehat{C}_n}{c_{max}}]\in[0,1]$. For different n, $\widehat{C}_n$ is independent from each other. Based on Lemma 2, we have

$$\mathbf{Pr}[\frac{\widehat{C}_n}{c_{max}}\le(1-\epsilon)\mathbb{E}[\frac{\widehat{C}_n}{c_{max}}]]\le e^{-\frac{\epsilon^2}{2}\mathbb{E}[\frac{\widehat{C}_n}{c_{max}}]}\tag{15}$$

Combining Eq. (14), then we have

$$\mathbf{Pr}[\widehat{C}_n\le(1-\epsilon)\widetilde{C}_n]\le e^{-\frac{\epsilon^2}{2}\frac{\widetilde{C}_n}{c_{max}}}\tag{16}$$

Thus, we can conclude that after rounding procedure, Algorithm 1 can derive a throughput, *i.e.*, $\widehat{C}_n$, will not be less than $(1-\epsilon)\widetilde{C}_n$. ■

Theorem 7: Algorithm 1 can achieve the approximation factor of $O(\log|N|)$ for NF capacity, where $|N|$ is the number NFs.

Proof: By Lemma 6, we can know that Algorithm 1 will acquire a value of NF capacity $\widehat{C}_n$ will not be less than $(1-\epsilon)\widetilde{C}_n$. Then, we will show that the total traffic load on each NF will not exceed $\widetilde{C}_n$ by a factor of $O(\log|N|)$.

We use a variable v_n^r to denote the traffic size of request r to NF n.

$$v_n^r=\begin{cases}f(r), & with\ probability\ \widetilde{y}_n^r\\ 0, & otherwise\end{cases}\tag{17}$$

So we have

$$\mathbb{E}[\sum_{r\in\Gamma}v_n^r]=\sum_{r\in\Gamma}f(r)\cdot\widetilde{y}_n^r\le\sum_{c\in C}\widetilde{x}_c^n\cdot p(c)=\widetilde{C}_n\tag{18}$$

We define:

$$\alpha_2=\min\{\frac{c_{min}}{\widetilde{y}_n^r\cdot f(r)_{max}}\}\tag{19}$$

Combining Eq. (18) and Eq. (19), we have

$$\begin{cases}\frac{v_n^r\cdot\alpha_2}{\widetilde{C}_n}\in[0,1]\\ \mathbb{E}[\sum_{r\in\Gamma}\frac{v_n^r\cdot\alpha_2}{\widetilde{C}_n}]\le\alpha_2\end{cases}\tag{20}$$

Note that when n is given, for different r, v_n^r is independent from each other. Thus, by adopting Lemma 2, we have

$$\mathbf{Pr}[\sum_{r\in\Gamma}\frac{v_n^r\alpha_2}{\widetilde{C}_n} \geq (1+\epsilon')\alpha_2] \leq e^{\frac{-\epsilon^2\cdot\alpha_2}{2+\epsilon}} \tag{21}$$

Now we assume that

$$\mathbf{Pr}[\sum_{r\in\Gamma}\frac{v_n^r}{\widetilde{C}_n} \geq (1+\epsilon')] \leq e^{\frac{-\epsilon^2\cdot\alpha_2}{2+\epsilon}} \leq \frac{1}{|N|^2} \tag{22}$$

We know that $\frac{1}{|N|^2} \to 0$ when the network grows. By solving Eq. (22), we have the following result:

$$\epsilon' \geq \frac{\log|N|^2 + \sqrt{\log^2|N|^2 + 8\alpha_2\log|N|^2}}{2\alpha_2}, \quad n \geq 2$$

$$\Rightarrow \epsilon' \geq \frac{2\log|N|}{\alpha_2} + 2, \quad n \geq 2 \tag{23}$$

Then, by applying Lemma 3, we have

$$\begin{aligned}&\mathbf{Pr}[\bigcup_{n\in N}\sum_{r\in\Gamma}\frac{v_n^r}{\widetilde{C}_n} \geq (1+\epsilon')] \\ \leq &\sum_{n\in N}\mathbf{Pr}[\sum_{r\in\Gamma}\frac{v_n^r}{\widetilde{C}_n} \geq (1+\epsilon')] \\ \leq &|N|\cdot\frac{1}{|N|^2} = \frac{1}{|N|}\end{aligned} \tag{24}$$

Combining Lemma 6, we can conclude that the approximation factor is

$$\frac{1+\epsilon'}{1-\epsilon} = \frac{1}{1-\epsilon}(\frac{2\log|N|}{\alpha_2}+3), \tag{25}$$

where ϵ is an arbitrarily positive value. ■

Theorem 8: Our algorithm can guarantee that the update time derived by Algorithm 1 will not exceed the time threshold T by a factor of $O(\log|\Gamma|)$, where $|\Gamma|$ denotes the number of requests.

Proof: The proof of Theorem 8 is similar to the proof of Theorem 5. Due to limited space, we omit the proof. ■

Approximation Factor: From our analysis, we can make the following conclusions. First of all, the infrastructure cost derived by algorithm is close to the optimal value derived by solving the LP with a high probability. Secondly, the total capacity of the NFs on the same physical server will not exceed the total resource of the server by a factor of $O(\log|S|)$, where $|S|$ means the number of servers. Thirdly, the total request load on each NF will not exceed the capacity of the NF by a factor of $O(\log|N|)$, where $|N|$ is the number of NFs. Finally, the time threshold for updateing will hardly be violated by a factor of $O(\log|\Gamma|)$, where $|\Gamma|$ denotes the number of requests.

IV. Performance Evaluation

A. Performance Metrics and Benchmarks

Performance Metrics: Since this paper focuses on how to help service providers spend less money and provide services to users with high QoS guarantee, we adopt the following performance metrics in evaluations: (1) the infrastructure cost; (2) the system throughput; (3) the badput ratio [42]; (4) the update delay; (5) the packet loss ratio and (6) the flow completion time (FCT).

During a simulation run, we record each NF's configuration type and calculate the total infrastructure cost that the service provider needs to pay. Then, we measure the total load of all the NFs as the throughput from users that the service provider can serve. Note that not all the traffic from users can be served due to limited capacity. We define the amount of traffic which cannot be served by NFs as badput. We divide badput by the total traffic amount from users as the badput ratio [42]. Finally, we calculate the update delay according to the number of rules that the controller needs to generate. During a system implementation run, we measure the packet loss ratio and the FCT using the command iPerf3.

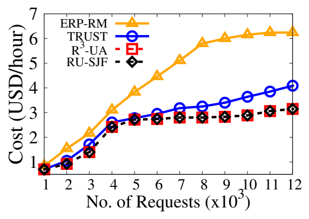
Benchmarks: We compare TRUST with three state-of-the-art benchmarks dealing with traffic dynamics in clouds. The first benchmark is the Elastic Resource Provisioning Reactive Mode (ERP-RM) algorithm [26], which is widely adopted in commercial clouds like Amazon [43], Scalr [44] and Rightscale [45]. ERP-RM sets the capacity threshold for each NF and automatically selects the configuration type with sufficient resources. The second benchmark is the Robustness-aware Real-time Request Updating Algorithm (R^3-UA) [35]. R^3-UA adopts the rounding method to acquire the real-time request updating scheme in order to achieve load-balancing among NFs. This benchmark also has update delay assurance by limiting the number of updating requests. The third benchmark is the Request Updating-Shortest Job First (RU-SJF) algorithm [36], which is highly efficient and also widely adopted in clouds. RU-SJF always chooses the NF with the least burden for the request with the least traffic demand. R^3-UA and RU-SJF are both pure request updating methods and do not involve modifying configuration type. Note that R^3-UA considers the update delay constraint, while RU-SJF does not.

B. Simulation Evaluation

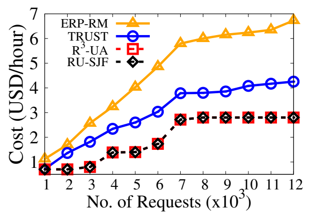
This section presents simulation experiments to evaluate performance of our proposed algorithm and benchmarks.

1) ***Simulation Settings****:* We conduct simulation experiments to compare TRUST with three benchmarks in two practical topologies. The first one is a small-scale NSF network topology, which contains 16 NFs [46]. The second topology is from Google cluster-data [47], which contains 320 NFs. We implement our tests with a set of requests from Google cluster-data [47]. Each request size is set from 500 to 1000 Kbps. The configuration type is set according to Google Cloud Platform [27]. A physical server can provide 100 core CPU and 375 GB RAM for NFs. The configuration types are shown in Table III. According to [35], the average update delay of a single request is set to 0.5ms.

We simulate the traffic dynamics according to [48]. The requests are divided into two parts: primitive requests and newly increased requests. Specifically, there are mainly two kinds of traffic dynamics. The first one is called slight dynamic, where newly increased requests account for 20% of all the requests. The other one is called magnitude dynamic, where newly increased requests account for 50% of all the requests. During traffic dynamic, 20% of NFs will receive newly increased requests and 20% of NFs will reduce about 50% of primitive requests. We conduct simulation experi-

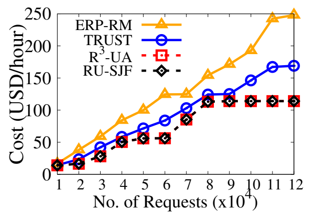


(a) for the slight dynamic

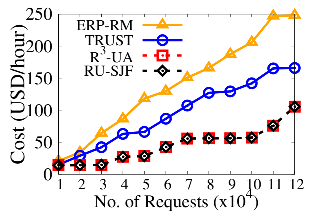


(b) for the magnitude dynamic.

Fig. 1: Cost vs Number of Requests in the small topology

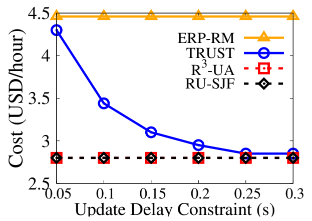


(a) for the slight dynamic

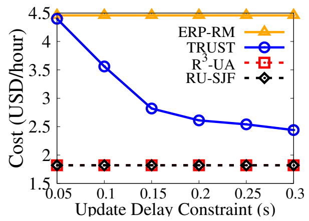


(b) for the magnitude dynamic

Fig. 2: Cost vs Number of Requests in the large topology

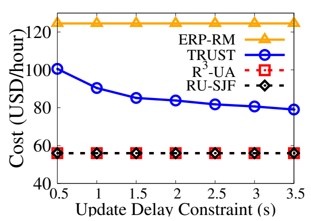


(a) for the slight dynamic

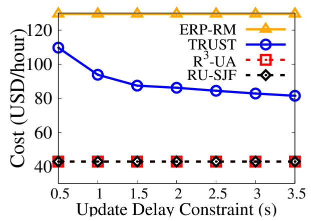


(b) for the magnitude dynamic

Fig. 3: Cost vs. Update Delay Constraint in the small topology



(a) for the slight dynamic



(b) for the magnitude dynamic

Fig. 4: Cost vs. Update Delay Constraint in the large topology

Type	Resource		Price	Capacity
	CPU (cores)	RAM (G)	($/hour)	(Gbps)
1	1	3.75	0.07	1
2	2	7.5	0.14	2
3	4	15	0.28	4
4	8	30	0.57	8
5	16	60	1.13	16
6	32	120	2.26	32
7	64	240	3.40	64

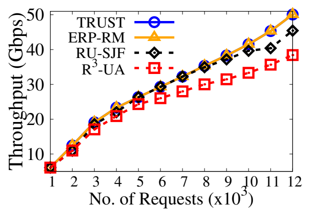
TABLE III: Configuration types according to Google Cloud Platform [27].

ments under the two kinds of traffic dynamics. Besides, we generate $6\times10^3/6\times10^4$ requests for the small/large topology by default. The update delay constraint is set to 0.2/2s for the small/large topology by default.

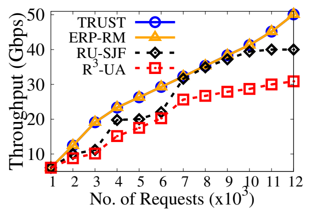
2) ***Performance Comparison:*** We run five groups of experiments to check the effectiveness of our algorithm. The first group of experiments shows the infrastructure costs by varying the number of requests in clouds. The results are shown in Figs. 1-2. We can learn from the figures that the costs of four algorithms increase when the number of requests grows. The figures show that our proposed algorithm always acquires a much lower cost than ERP-RM and a slightly higher cost than R^3-UA and RU-SJF. For example, in Fig. 1(a), given 8×10^3 requests in the cloud, the cost results of four algorithms are 6, 3.4, 2.83 and 2.83 USD/hour, corresponding to ERP-RM, TRUST, R^3-UA and RU-SJF, respectively. TRUST reduces the cost by 43.96% compared with ERP-RM while only increases the cost by 13.79% compared with both R^3-UA and RU-SJF. Since R^3-UA and RU-SJF will not buy any more extra resources when facing traffic dynamics, it is natural that the cost results will be lower than those of ERP-RM and TRUST. As a result, the QoS aspects (*e.g.*, throughput and update delay) will decrease significantly, which will be clarified hereafter. In Fig. 2(a), when there are 9×10^4 requests, the cost results of four algorithms are 171.42, 125.1, 114 and 114 USD/hour, corresponding to ERP-RM, TRUST, R^3-UA and RU-SJF. TRUST reduces the cost by 27% compared with ERP-RM.

The second group of experiments shows the infrastructure costs by varying the update delay constraint and the results are shown in Figs. 3-4. It can be concluded from the figures that, given a larger update delay threshold, TRUST will acquire a solution with less cost. Also, since ERP-RM only executes elastic resource provisioning, and R^3-UA/RU-SJF only executes updating requests, their cost results are steady. For instance, in Fig. 3(a), the cost results of ERP-RM, R^3-UA and RU-SJF are 4.46, 2.8 and 2.8 USD/hour, respectively. Obviously, the cost results of three benchmarks will not be affected by the update delay constraint. When the update delay constraint is set as 0.05s, TRUST is only able to transfer about tens of requests. The cost result of TRUST is nearly the same as that of ERP-RM. However, when given sufficient time, like 0.2s, the cost result of TRUST is only 2.95 USD/hour. TRUST reduces the cost by 33.9% compared with ERP-RM and increases the cost only by about 5.4% compared with both R^3-UA and RU-SJF. The figures also imply that when facing the magnitude dynamics, pure request updating methods will not be able to solve the ultimate problem, *i.e.*, insufficient resources to deal with the sudden increasing number of requests. For instance, in Fig. 4(b), even if TRUST has enough time to update requests, the cost is still a bit higher than those of R^3-UA and RU-SJF. This is because extra resources are necessary, and the consequence of not upgrading the configuration type is to decline redundant requests, resulting in a bad users' QoS.

The third group of experiments show the throuhputs by varying the number of requests in clouds and the results are shown in Figs. 5-6. We can learn from the figures that the throughput results increase as the number of requests grows. By Fig. 5(b), given 12×10^3 requests, TRUST improves throughput by 62.2% and 25.3% compared with R^3-UA and RU-SJF, respectively. In Fig. 6(b), when there are 11×10^4 requests, TRUST improves throughput by 93.2% and 49.3% compared with R^3-UA and RU-SJF, respectively. Since the

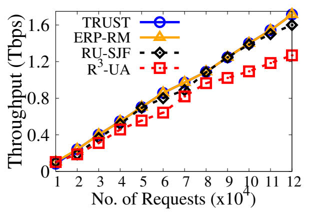


(a) for the slight dynamic

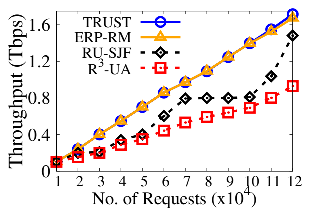


(b) for the magnitude dynamic

Fig. 5: Throughput vs. Number of Requests in the small topology

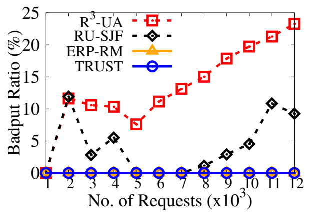


(a) for the slight dynamic

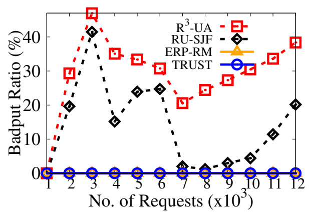


(b) for the magnitude dynamic

Fig. 6: Throughput vs. Number of Requests in the large topology

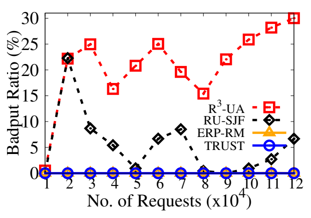


(a) for the slight dynamic

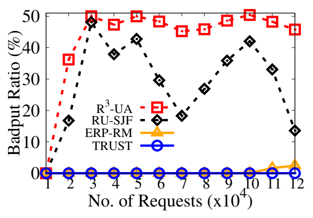


(b) for the magnitude dynamic

Fig. 7: Badput Ratio vs. Number of Requests in the small topology

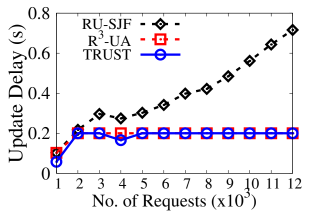


(a) for the slight dynamic

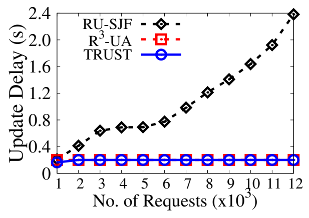


(b) for the magnitude dynamic

Fig. 8: Badput Ratio vs. Number of Requests in the large topology

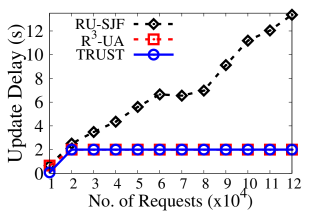


(a) for the slight dynamic

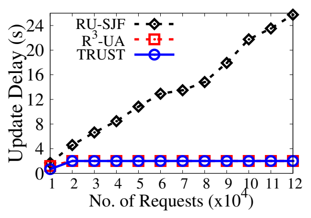


(b) for the magnitude dynamic

Fig. 9: Update Delay vs. Number of Requests in the small topology



(a) for the slight dynamic



(b) for the magnitude dynamic

Fig. 10: Update Delay vs. Number of Requests in the large topology

two methods are only updating requests without buying extra resources, consequently, many requests have to be abandoned due to limited processing capacity on NFs. Note that since ERP-RM and TRUST both tend to buy extra resources when facing dynamics, the throughput results are much better than those of R^3-UA and RU-SJF. However, under some extreme situations, due to the limited resources of physical servers, ERP-RM may not be able to hold all the traffic from users. As comparison, TRUST can balance the load among the servers by updating the requests. As shown in Figs. 6(a)-6(b), when there are 12×10^4 requests under two kinds of traffic dynamics, the throughput results of ERP-RM are a bit lower than those of TRUST.

The fourth set of experiments presents the badput ratio of four algorithms. We can conclude that ERP-RM and TRUST can always achieve the least badput ratio compared with R^3-UA and RU-SJF. We can observe that the function curves of R^3-UA and RU-SJF fluctuate as the number of requests increases. For instance, by Fig. 7(b), the badput ratio of R^3-UA decreases when the number of requests ranges from 3×10^3 to 7×10^3, and increases when the number of requests is between 8×10^3 to 12×10^3, since when the number of requests is ranging from 3×10^3 to 7×10^3, the configuration types of NFs are being upgraded. Consequently, as shown in Fig. 1(b), the cost result of R^3-UA is increasing. In general, TRUST decreases badput by 43% and 28.7% compared with R^3-UA and RU-SJF, respectively, in the large topology.

The fifth set of experiments shows the update delay of three algorithms. Since ERP-RM does not involve the updating procedure, we only compare TRUST with R^3-UA and RU-SJF. We learn from Figs. 9-10 that the update delay of RU-SJF will significantly increases while TRUST and R^3-UA limit the update delay within a small range. For instance, by Fig. 10(a), when there are 9×10^4 requests, TRUST can reduce update delay by 78.1% compared with RU-SJF. In Fig. 10(b), when there are 10×10^4 requests, TRUST can reduce update delay by 91.5% compared with RU-SJF.

From these simulation results, we can draw some conclusions. Firstly, compared with ERP-RM, TRUST can significantly reduce the cost by $35.5\%/31.3\%$ on average in the small/large topology by Figs. 1-2. At the same time, TRUST only takes a few more seconds of update delay, and it can still satisfy the QoS demand of users. Secondly, compared with R^3-UA, TRUST has a much better performance on throughput and significantly reduces the badput ratio. From Figs. 5-6, we know that TRUST improves the throughput by $44.9\%/88.9\%$ in the small/large topology on average compared with R^3-UA. Then, Figs. 7-8 show that TRUST can reduce the badput ratio by $29.1\%/43\%$ in the small/large topology compared with R^3-UA. Meanwhile, TRUST only increases the cost by about $13.79\%/8.8\%$ in the small/large topology compared with R^3-UA. Finally, compared with RU-SJF, TRUST can greatly reduce the update delay. Figs. 9-10 show that in general, TRUST can reduce $81.8\%/86\%$ update delay compared with RU-SJF in the small/large topology. Also, TRUST can improve the throughput by 15%-44.8% and

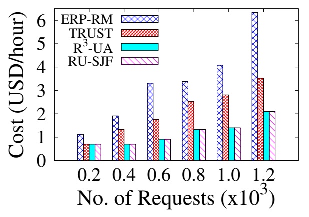


Fig. 11: Cost vs. No. of Flows

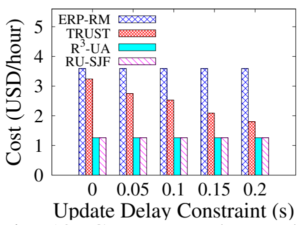


Fig. 12: Cost vs. Update Delay Constraint

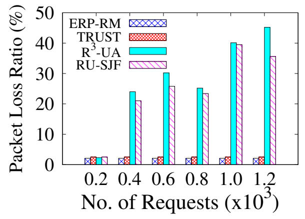


Fig. 13: Packet Loss Ratio vs. No. of Flows

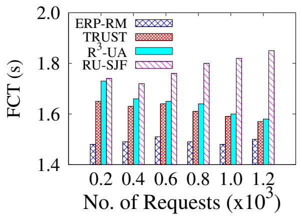


Fig. 14: FCT vs. No. of Flows

reduce the badput ratio by 13.9%-28.7% with cost increased only by 8.8%-13.79% compared with RU-SJF.

C. Testbed Evaluation

1) Testbed Settings: To further evaluate our proposed algorithm, we need a universal platform to realize our small-scale testbed system, which is supposed be similar to the cloud environment deployed in the real-world. Thus, our choice is OpenStack [49], the most advanced and widely used cloud infrastructure software, to implement the system. Specifically, we adopt the latest version of OpenStack called Victoria [49] to estimate our algorithm. By OpenStack's VM service Nova, we can customize different *flavors*, *i.e.*, configuration types, according to Google Cloud Platform [27], which is shown in Table III.

Specifically, our testbed generates 10 instances as NFs, all with Ubuntu 18.04 OS. The NFs are all initialized as type 1 in Table III. We first run SNAT (Static Network Address Translation) on 10 NFs by rewriting the iptables rules to translate the private address into a public address for online requirements. Then, we set 800 requests and 0.1s update delay constraint by default. We implement our tests with a set of requests from Google cluster-data [47]. We mainly consider the magnitude dynamic in our testbed. During traffic dynamic, 20% of NFs will receive newly increased requests, and 20% of NFs will reduce about 50% of primitive requests.

2) Performance Comparison: This set of testbed experiments compares the performance of ERP-RM, TRUST, R^3-UA and RU-SJF and the results are shown in Figs. 11-14. We record each NF's configuration type and calculate the total infrastructure cost and the results are shown in Fig. 11. We observe that TRUST can reduce the cost by 37.8% compared with ERP-RM. Since TRUST takes the advantages of request updating, the purchased resources of TRUST are much fewer than that of ERP-RM.

We demonstrate the costs by varying the update delay constraint in Fig. 12. It can be inferred from the figure that the cost results of TRUST will decrease as the update delay grows. For example, when the update delay is 0, the cost result of TRUST is almost the same as that of ERP-RM since TRUST is unable to transfer any requests. Given enough time, *e.g.*, 0.2s, we find a significant reduction in cost of TRUST. TRUST reduces the cost by 50% compared with ERP-RM.

By Fig. 13, we present the average packet loss ratio by varying the number of requests. Overall, the packet loss ratio results of R^3-UA and RU-SJF increase as the number of requests grows. When the number of requests is 0.8×10^3, the packet loss ratio results of R^3-UA and RU-SJF decrease since the configuration types of NFs are upgraded significantly at that time. When there are 0.6×10^3 requests, TRUST can reduce 91.7% and 90.3% packet loss ratio compared with R^3-UA and RU-SJF, respectively. ERP-RM and TRUST always keep a very low packet loss ratio because both two methods tend to buy enough resources to handle requests.

Fig. 14 shows the average FCT by varying the number of requests. The FCT of a single request mainly depends on its traffic size and link bandwidth. If the request is transferred, the time for updating rules is also included. The FCT results of ERP-RM are always the lowest because ERP-RM does not involve updating the requests. When given 1.2×10^3 requests in the cloud, TRUST can reduce the average FCT by 20.4% compared with RU-SJF and only increase the average FCT by about 4.6% compared with ERP-RM. The reason why TRUST and R^3-UA can achieve a much lower FCT than that of RU-SJF, is that TRUST and R^3-UA can limit the number of updating requests.

From the above experimental results, we can draw some conclusions. Firstly, TRUST reduces the cost by 37.8% and only increases the average FCT by 4.6% compared with ERP-RM. Secondly, compared with R^3-UA and RU-SJF, TRUST can acquire a much better packet loss ratio. TRUST reduces the average packet loss ratio by 91.7% and 90.3% compared with R^3-UA and RU-SJF, respectively, while it only increases the cost by 23.6% compared with both R^3-UA and RU-SJF on average. Finally, TRUST can reduce the average FCT by 20.4% compared with RU-SJF. These experimental results show the high efficiency and cost-saving of TRUST.

V. Conclusion

In this paper, we focus on the problem of real-time request updating with elastic resource provisioning in clouds. To solve the problem, we design an efficient algorithm with bounded approximation factors based on randomzied rounding. Extensive simulation and testbed experiments results show the high efficiency of our proposed algorithm.

Acknowledgement

The corresponding authors of this paper are Gongming Zhao and Hongli Xu. This article was supported in part by the National Science Foundation of China (NSFC) under Grants 61822210, 62102392 and 62132019; and in part by the National Science Foundation of Jiangsu Province under Grant BK20210121; and in part by Anhui Initiative in Quantum Information Technologies under Grant AHY150300.

References

[1] M. M. Sadeeq, N. M. Abdulkareem, S. R. Zeebaree, D. M. Ahmed, A. S. Sami, and R. R. Zebari, "Iot and cloud computing issues, challenges and opportunities: A review," *Qubahan Academic Journal*, vol. 1, no. 2, pp. 1–7, 2021.

[2] I. M. Ibrahim *et al.*, "Task scheduling algorithms in cloud computing: A review," *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, vol. 12, no. 4, pp. 1041–1053, 2021.

[3] 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," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[4] A. Brazhuk, "Threat modeling of cloud systems with ontological security pattern catalog," *International Journal of Open Information Technologies*, vol. 9, no. 5, pp. 36–41, 2021.

[5] C. Wu, V. Sreekanti, and J. M. Hellerstein, "Autoscaling tiered cloud storage in anna," *The VLDB Journal*, vol. 30, no. 1, pp. 25–43, 2021.

[6] Y. Zhang, G. Lin, H. Gu, F. Zhuang, and G. Wei, "Multi-copy dynamic cloud data auditing model based on imb tree," *Enterprise Information Systems*, vol. 15, no. 2, pp. 248–269, 2021.

[7] Y. Zhou, L. Ruan, L. Xiao, and R. Liu, "A method for load balancing based on software defined network," *Advanced Science and Technology Letters*, vol. 45, pp. 43–48, 2014.

[8] A. Beloglazov and R. Buyya, "Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints," *IEEE transactions on parallel and distributed systems*, vol. 24, no. 7, pp. 1366–1379, 2012.

[9] S. Bharti and K. K. Pattanaik, "Dynamic distributed flow scheduling with load balancing for data center networks," *Procedia Computer Science*, vol. 19, pp. 124–130, 2013.

[10] Z. Guo, Y. Xu, Y.-F. Liu, S. Liu, H. J. Chao, Z.-L. Zhang, and Y. Xia, "Aggreflow: Achieving power efficiency, load balancing, and quality of service in data center networks," *IEEE/ACM Transactions on Networking*, 2020.

[11] M. H. Ho, F. Hnaien, and F. Dugardin, "Electricity cost minimisation for optimal makespan solution in flow shop scheduling under time-of-use tariffs," *International journal of production research*, vol. 59, no. 4, pp. 1041–1067, 2021.

[12] P. Wang, Y. Lei, P. R. Agbedanu, and Z. Zhang, "Makespan-driven workflow scheduling in clouds using immune-based pso algorithm," *IEEE Access*, vol. 8, pp. 29 281–29 290, 2020.

[13] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-international conference on neural networks*, vol. 4. IEEE, 1995, pp. 1942–1948.

[14] T. Hoang, A. A. Yavuz, and J. G. Merchan, "A secure searchable encryption framework for privacy-critical cloud storage services," *IEEE Transactions on Services Computing*, 2019.

[15] A. Greenberg, "Sdn for the cloud," in *Keynote in the 2015 ACM Conference on Special Interest Group on Data Communication*, 2015.

[16] P. Wang, H. Xu, L. Huang, C. Qian, S. Wang, and Y. Sun, "Minimizing controller response time through flow redirecting in sdns," *IEEE/ACM Transactions on Networking*, vol. 26, no. 1, pp. 562–575, 2018.

[17] G. Zhao, L. Huang, Z. Yu, H. Xu, and P. Wang, "On the effect of flow table size and controller capacity on sdn network throughput," in *2017 IEEE International Conference on Communications (ICC)*, pp. 1–6.

[18] H. Xu, Z. Yu, X.-Y. Li, C. Qian, L. Huang, and T. Jung, "Real-time update with joint optimization of route selection and update scheduling for sdns," in *2016 IEEE 24th International Conference on Network Protocols (ICNP)*. IEEE, 2016, pp. 1–10.

[19] K. Kannan and S. Banerjee, "Compact tcam: Flow entry compaction in tcam for power aware sdn," in *International conference on distributed computing and networking*. Springer, 2013, pp. 439–444.

[20] B.-Y. Choi, J. Park, and Z.-L. Zhang, "Adaptive packet sampling for flow volume measurement," 2002.

[21] W. J. A. Silva, "Avoiding inconsistency in openflow stateful applications caused by multiple flow requests," in *2018 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2018, pp. 548–553.

[22] S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimization of resource provisioning cost in cloud computing," *IEEE Transactions on Services Computing*, vol. 5, no. 2, pp. 164–177, 2012.

[23] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh, "A cost-aware elasticity provisioning system for the cloud," in *2011 31st International Conference on Distributed Computing Systems*. IEEE, 2011, pp. 559–570.

[24] M. Nardelli, C. Hochreiner, and S. Schulte, "Elastic provisioning of virtual machines for container deployment," in *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*, 2017, pp. 5–10.

[25] A. da Silva Dias, L. H. V. Nakamura, J. C. Estrella, R. H. C. Santana, and M. J. Santana, "Providing iaas resources automatically through prediction and monitoring approaches," in *2014 IEEE Symposium on Computers and Communications (ISCC)*, 2014, pp. 1–7.

[26] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle, "Elasticity in cloud computing: state of the art and research challenges," *IEEE Transactions on Services Computing*, vol. 11, no. 2, pp. 430–447, 2017.

[27] Google cloud platform. [Online]. Available: https://cloud.google.com/compute/vm-instance-pricing

[28] A. Ashraf, B. Byholm, and I. Porres, "Cramp: Cost-efficient resource allocation for multiple web applications with proactive scaling," in *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*. IEEE, 2012, pp. 581–586.

[29] P. Marshall, K. Keahey, and T. Freeman, "Elastic site: Using clouds to elastically extend site resources," in *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pp. 43–52.

[30] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "Cloudscale: elastic resource scaling for multi-tenant cloud systems," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, 2011, pp. 1–14.

[31] S. Woo, J. Sherry, S. Han, S. Moon, S. Ratnasamy, and S. Shenker, "Elastic scaling of stateful network functions," in *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, 2018, pp. 299–312.

[32] M. Karzand, D. J. Leith, J. Cloud, and M. Medard, "Design of fec for low delay in 5g," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 8, pp. 1783–1793, 2017.

[33] H. Xu, Z. Yu, X.-Y. Li, L. Huang, C. Qian, and T. Jung, "Joint route selection and update scheduling for low-latency update in sdns," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 3073–3087, 2017.

[34] S. M.-K. Gueye, N. De Palma, É. Rutten, A. Tchana, and N. Berthier, "Coordinating self-sizing and self-repair managers for multi-tier systems," *Future Generation Computer Systems*, vol. 35, pp. 14–26, 2014.

[35] H. Tu, G. Zhao, H. Xu, Y. Zhao, and Y. Zhai, "Robustness-aware real-time sfc routing update in multi-tenant clouds," in *2021 IEEE 29nd International Symposium of Quality of Service (IWQoS)*. IEEE, 2021.

[36] M. Nosrati, R. Karimi, and M. Hariri, "Task scheduling algorithms introduction," *World Applied Programming*, vol. 2, no. 6, pp. 394–398, 2017.

[37] W. Yu, L. Musavian, and Q. Ni, "Link-layer capacity of noma under statistical delay qos guarantees," *IEEE Transactions on Communications*, vol. 66, no. 10, pp. 4907–4922, 2018.

[38] X. Wen, B. Yang, Y. Chen, L. E. Li, K. Bu, P. Zheng, Y. Yang, and C. Hu, "Ruletris: Minimizing rule update latency for tcam-based sdn switches," in *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2016, pp. 179–188.

[39] S. Martello and P. Toth, "Bin-packing problem," *Knapsack problems: Algorithms and computer implementations*, pp. 221–245, 1990.

[40] P. Raghavan and C. D. Tompson, "Randomized rounding: a technique for provably good algorithms and algorithmic proofs," *Combinatorica*, vol. 7, no. 4, pp. 365–374, 1987.

[41] Pulp. [Online]. Available: https://pypi.org/project/PuLP/

[42] V. Apte, "" what did i learn in performance analysis last year?" teaching queuing theory for long-term retention," in *Companion of the 2019 ACM/SPEC International Conference on Performance Engineering*, 2019, pp. 71–77.

[43] Amazon. [Online]. Available: https://aws.amazon.com

[44] Scalr. [Online]. Available: https://www.scalr.com

[45] Rightscale. [Online]. Available: https://www.rightscale.com

[46] G. Sun, Z. Chen, H. Yu, X. Du, and M. Guizani, "Online parallelized service function chain orchestration in data center networks," *IEEE Access*, vol. 7, pp. 100 147–100 161, 2019.

[47] Google cluster data. [Online]. Available: https://www.github.com/google/cluster-data

[48] A. Ali-Eldin, O. Seleznjev, S. Sjöstedt-de Luna, J. Tordsson, and E. Elmroth, "Measuring cloud workload burstiness," in *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*. IEEE, 2014, pp. 566–572.

[49] Openstack victoria. [Online]. Available: https://docs.openstack.org/victoria/