Deadline and Reliability Aware Multiserver Configuration Optimization for Maximizing Profit

Tian Wang, *Student Member, IEEE*, Junlong Zhou, *Member, IEEE*, Liying Li, *Student Member, IEEE*, Gongxuan Zhang, *Senior Member, IEEE*, Keqin Li, *Fellow, IEEE*, and Xiaobo Sharon Hu, *Fellow, IEEE*

Abstract—Maximizing profit is a key goal for cloud service providers in the modern cloud business market. Service revenue and business cost are two major factors in determining profit and highly depend on multiserver configuration. Understanding the relationship between multiserver configuration and profit is important to service providers. Although existing articles have explored this issue, few of them consider deadline miss rate and soft error reliability of cloud services in multiserver configuration for profit maximization. Since deadline misses violate cloud services' real-time requirements and soft error prevents successful processing of cloud services, it is necessary to consider the impact of deadline miss rate and soft error reliability on service providers' profits when configuring the multiserver. This article introduces a deadline miss rate and soft error reliability aware multiserver configuration scheme for maximizing cloud service providers' profit. Specifically, we derive the deadline miss rate considering the heterogeneity of cloud service requests, and propose an analytical method to compute the soft error reliability of multiserver systems. Based on the new deadline miss rate and soft error reliability models, we formulate the multiserver configuration optimization problem and introduce an augmented Lagrange multiplier-based iterative method to find the optimal multiserver configuration. Extensive experiments evaluate the efficacy of the proposed multiserver configuration approach. Compared with the two state-of-the-art methods, the profit gained by our scheme can be up to 11.92% higher.

Index Terms—Cloud service, deadline miss rate, multiserver configuration, profit maximization, soft error reliability

1 Introduction

Cloud computing is a successful commercial paradigm that delivers user services over communication networks and virtualizes infrastructure resources into ordinary commodities in a pay-as-you-go manner [1], [2], [3], [4], [5]. With the rapid development of virtualization technology and the increasing scale of infrastructure deployment, cloud service providers such as Amazon [7], Google [8] and Microsoft Azure [9] are gaining popularity. In the cloud business market, service providers would naturally pursue the goal of maximum profit. Meanwhile, for users, they expect high-quality cloud services in terms of *timeliness* and *correctness*. Therefore, how to configure a cloud infrastructure platform based on the features of cloud service requests for maximizing profit and ensuring the timeliness and correctness of service request processing is of utmost importance.

Timeliness of service refers to satisfying the real-time requirements of service requests such as deadlines. Different from hard real-time constraints found in cyber-physical systems such as automotive and airplane controllers [10], many common service requests in the Cloud [11], such as video/music-on-demand, cloud gaming, and scientific computing, are associated with soft real-time requirements. For such service requests in the Cloud, their timeliness is not stringent and occasional deadline misses are acceptable but would decrease the quality of service. Service requests in the Cloud may experience transient faults [12], which may prevent successful processing of service requests. In summary, only when service requests are successfully processed within the deadline can bring profit to the service provider.

Resource scheduling (e.g., multiserver configuration) in cloud computing plays a key role in timely and successful completion of service requests. Resources scheduling can be divided into two broad categories based on the type of information on which the scheduling decisions are made [13]: *average-based scheduling* [1], [2], [3], [4], [5] and *instantaneous-based scheduling* [14], [15]. Average-based scheduling uses

- *Tian Wang and Gongxuan Zhang are with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, Jiangsu 210094, China. E-mail: {eclipse_wt, gongxuan}@njust.edu.cn.*
- *Junlong Zhou is with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, Jiangsu 210094, China, also with State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China, and also with National Trusted Embedded Software Engineering Technology Research Center, East China Normal University, Shanghai 200062, China. E-mail: jlzhou@njust.edu.cn.*
- *Liying Li is with the Department of Computer Science and Technology, East China Normal University, Shanghai 200062, China. E-mail: 52194506009@stu.ecnu.edu.cn.*
- *Keqin Li is with the Department of Computer Science, State University of New York, New York, NY 12561 USA. E-mail: lik@newpaltz.edu.*
- *Xiaobo Sharon Hu is with the Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556 USA. E-mail: shu@nd.edu.*

Manuscript received 22 June 2021; revised 1 March 2022; accepted 9 April 2022. Date of publication 26 April 2022; date of current version 26 July 2022. This work was supported in part by the National Natural Science Foundation of China under Grants 62172224 and 61802185, in part by the China Postdoctoral Science Foundation under Grants BX2021128, 2021T140327, and 2020M680068, in part by the Postdoctoral Science Foundation of Jiangsu Province under Grant 2021K066A, in part by the Open Research Fund of the State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences under Grant CARCHA202105, in part by the Future Network Scientific Research Fund Project under Grant FNSRFP-2021-YB-6, and in part by the Open Research Fund of National Trusted Embedded Software Engineering Technology Research Center (East China Normal University). (Corresponding authors: Junlong Zhou and Gongxuan Zhang.) Recommended for acceptance by O. Ozkasap. Digital Object Identifier no. 10.1109/TPDS.2022.3170305

averaged metrics (e.g., the average number of arrival service requests and the average processing capabilities of servers) to guide resource management, which is also called static scheduling. On the other hand, instantaneous-based scheduling guides resource management by instantaneous metrics (e.g., the execution time of current processing requests and current residual processing capabilities of each server), which is also called dynamic scheduling. Typically, instantaneous-based scheduling outperforms average-based scheduling. Nevertheless, obtaining real-time system information [13] to schedule each service request induces high time and energy overheads [13]. This becomes more severe when the cloud servers are geo-distributed, in which the latency for retrieving and transmitting system information may be overwhelming, or when the recorded information at servers is not always available. Furthermore, in cloud data centers, an easy-to-maintain and large-scale inexpensive network usually tends to deliver service requests in a first-come-first-serve (FCFS) order [16]. Therefore, this work explores average-based scheduling to optimize the multiserver configuration for intermittent cloud service requests with soft real-time requirements.

The multiserver configuration problem for maximizing cloud service providers' profit has been studied in [1], [2], [3], [4], [5] which assume that cloud service requests have the same maximum waiting time. However, the cloud service requests' maximum waiting time is not always the same due to the heterogeneity of cloud service requests [17]. In addition, none of the existing papers considers deadline miss rate and soft error reliability simultaneously which do affect the profit of service providers. In this paper we explore an optimal multiserver configuration that maximizes the profit of cloud service providers with deadline miss rate and soft error reliability consideration. Our main contributions are summarized as follows.

- We derive the cloud service requests' deadline miss rate based on the probability distribution of cloud service requests' slack, and model the soft error reliability of processing service requests on the multiserver system.
- Based on our deadline miss rate and soft error reliability models, we formulate the profit maximization problem as a constrained optimization problem and then convert it into an unconstrained optimization problem. The conversion is achieved by utilizing the augmented Lagrange multiplier (ALM).
- We design an iterative algorithm to find the optimal solution to the converted unconstrained problem (i.e., the profit maximization problem), which is considered as the solution to the constrained optimization problem if the solution error is less than a sufficiently small threshold.
- Comprehensive simulation experiments are carried out to evaluate the proposed scheme by i) observing the relationship between the profit and the multiserver configuration under varying setups, and ii) comparing the performance of our scheme with that of two state-of-the-art (SOTA) methods in maximizing profit.

Results show that our scheme outperforms the two SOTA methods by as much as 11.92% profit increase.

The remainder of this paper is organized as follows. Section 2 reviews the related work on profit maximization. Section 3 introduces the cloud business market architecture, cloud service model, cost model, and service charge model considered in this paper. In Section 4, we present our deadline miss rate and soft error reliability models as well as formulate the profit maximization problem. Section 5 details our proposed multiserver configuration optimization scheme. In Section 6, we evaluate our scheme through three sets of experiments. Finally, Section 7 concludes the paper.

2 Related Work

Since a cloud service provider's profit is related to the monetary cost of processing cloud customers' service requests and the revenue earned by providing customers cloud services, increasing services' revenue and reducing operation costs are both helpful in increasing the profit.

Pricing is a common way to increase service revenue. It can be broadly classified into two categories: static and dynamic pricing. Unlike static pricing schemes that the service price is fixed, dynamic pricing schemes allow flexible adjustment of service price based on the customer demand and hence have been widely used by cloud service providers. For example, a service level agreement-based dynamic service pricing method (i.e, TSPM) is proposed in [5] to improve the service provider's revenue. To guarantee the revenue of service providers, Mei *et al.* [1] design a hybrid server renting strategy containing a long-term and a short-term renting method that are both developed based on an $\mathrm{M}/\mathrm{M}/m + D$ queuing theory. Considering the interaction between cloud customers and service providers, customer perceived value is introduced to design a dynamic pricing model which is further used to increase service revenue [3]. Inspired by [3], Wang *et al.* [4] adopt customer perceived value and service satisfaction to propose a dynamic strategy and present a simulated annealing based approach which estimates the demand of customers for cloud services and then sets the multiserver configuration for earning more revenue. A feedback control technique is also developed in [4] to mitigate the risk induced by inaccurate service demand prediction.

Reducing operation cost for profit maximization has been explored recently. Ye *et al.* [18] present an autonomous resource management algorithm to select the shared resources or dedicated resources. The algorithm can reduce servers' energy consumption and hence decrease electricity bill by using the least number of servers to meet the service-level requirements. Mazzucco *et al.* [19] propose a strategy that maximizes the service provider's profit by estimating user demand and system behavior as well as dynamically reducing the number of activated servers. Beloglazov *et al.* [20] develop a resource allocation heuristic for energy consumption reduction by increasing the servers' utilization. An optimal power and workload assignment scheme is designed in [21] for saving energy of heterogeneous servers.

Although existing works are effective for profit improvement, few of them consider the adverse impact of soft errors on processing of service requests. In a cloud computing system, soft errors are unacceptable since they propagate through the system and induce corrupted outputs or even system crashes [22], [23]. In addition, most

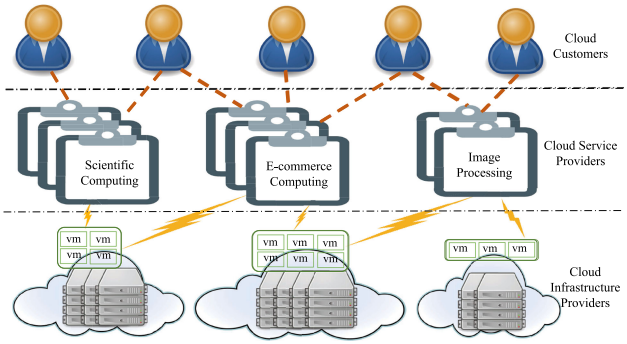


Fig. 1. Three-tier architecture of the cloud business market.

existing approaches assume that cloud service requests have the same maximum waiting time when configuring a multiserver. Under this assumption, the heterogeneity in the timing requirements of cloud service requests is ignored [17], which may lead to a high deadline miss rate or inappropriate server configuration and thus in turn decrease the profit [39]. To fill the gap in existing literature, this paper proposes a deadline miss rate and soft error reliability aware multiserver configuration scheme to maximize cloud service providers' profit while considering the heterogeneity of service requests as well as the impact of transient faults.

3 Model and Problem Formulation

This section describes the architecture of the cloud business market and related models adopted in our work.

3.1 Architecture of Cloud Business Market

Fig. 1 illustrates a typical three-tier architecture of the cloud business market which is considered in this paper. This architecture contains three entities: 1) cloud infrastructure providers (CIPs), 2) cloud service providers (CSPs), and 3) cloud customers (CCs). As an attractive business model (i.e., cloud computing environment model), this structure has been widely found in the actual cloud service market [7], [8], [9] and commonly adopted in the literature [1], [2], [3], [4]. In this architecture, CIPs own and maintain hardware and software facilities, and CSPs pay for leasing infrastructure resources provided by CIPs. CSP uses the rented infrastructure resources to build up the cloud service platform (i.e., the multiserver system) and prepares cloud services with virtualization techniques, i.e., virtual machines, and then charges CCs for providing cloud services. A CC submits a service request to a CSP, receives the desired result from the CSP with certain service-level agreement and pays for the service based on the amount and quality of the service.

Similar to [1], [2], [3], [4], [5], CSPs are conceptually distinct from CIPs in the sense that CSPs explore the intelligent algorithms to optimize multiserver configuration to process CCs' service requests for the desired results. In other words, CSPs optimize the configuration of the multiserver system in pursuit of maximum cloud service profit, while CIPs simply charge CSPs based on the rented infrastructure resources. In practice, the CSP also cloud be the CIP (who provides IaaS, PaaS, and SaaS simultaneously) and our model can also handle this case by combining the rental cost and utility cost, or a separate party that only provides CCs with PaaS or SaaS services. This attractive three-tier architecture also can be found in the cluster computing system [5] with three parties (i.e., cluster nodes, cluster managers, and consumers) or in the grid computing system [5] with three parties (i.e., resource providers, service providers, and clients).

3.2 Cloud Service Model

Three modes of services are generally provided by CSPs: 1) infrastructure-as-a-service (IaaS), 2) platform-as-a-service (PaaS), and 3) software-as-a-service (SaaS). In our work, we configure the cloud service platform to process customers' cloud service requests. We consider the scenario that infrastructure resources are rented by PaaS/SaaS cloud service provider from the IaaS provider. Common examples of IaaS are Google Compute Engine and Amazon CloudFormation which provision the cloud infrastructure resources through virtual machines [4]. Thanks to the virtualization technology, CSPs are able to focus on providing the cloud service business without spending any efforts in maintaining hardware platforms. In this paper, our goal is to maximize CSPs' profit by optimizing the configuration settings of cloud service platforms. Similar to [1], [2], [3], [4], the cloud service platform used in our work is viewed as a multiserver system. The system architecture can vary [5]. For example, a multiserver system could be a group of blade servers, a set of commodity servers, or a multicore processor [5]. For simplicity, we refer to the blades [25]/processors [26]/cores [27] above collectively as servers.

To formally model the multiserver system, we make the following assumptions. Each multiserver system deploys dedicated software to serve one type of service requests [2], e.g., Fig. 1 shows three multiserver systems for scientific computing, e-commerce computing, and image processing, respectively. A multiserver system is characterized by two basic features: the arrival rate of cloud service requests and the expected number of instructions for processing a cloud service request if no soft errors occur [5]. The adopted multiserver system is equipped with m homogeneous servers with execution speed s. The multiserver configuration m and s are the two basic characteristics of the multiserver system. Although the cloud infrastructure is inherently heterogeneous, this assumption on the multiserver system is still reasonable since these servers usually prefer load-balanced [28]. As in the literature [1], [2], [3], [4], [5], [13], a multiserver system is viewed as an $M/M/m$ queuing system. The arrival of service requests in the multiserver system follows a Poisson distribution with an arrival rate denoted by λ. Cloud service requests follow the FCFS policy, i.e., they wait in a queue with infinite storage capacity.

The resource for processing a service request from a cloud customer is quantified by the number of instructions r. The number of instructions is assumed to follow the exponential distribution and its mean is denoted by $\overline{r}$. Therefore, the service request processing time t can be calculated by $t = r/s$, and also follows the exponential distribution. The mean of t is represented by $\overline{t}$, which is calculated by $\overline{t} = \overline{r}/s$. The server service rate is represented by μ, which is the average number of service requests that can be processed at the server running speed s in a unit of time. The system utilization is then calculated as

$$\rho = \frac{\lambda}{m\mu} = \frac{\lambda \overline{r}}{ms}. \tag{1}$$

Let P_k be the probability that k requests are waiting or processing. Similar to [29], we have

$$P_k = \begin{cases} P_0 \frac{(m\rho)^k}{k!}, & k \leqslant m \\ P_0 \frac{m^m \rho^k}{m!}, & k > m \end{cases}, \tag{2}$$

where P_0 represents the probability that there is no cloud service request in the system and is calculated by

$$P_0 = \left(\sum_{k=0}^{m-1} \frac{(m\rho)^k}{k!} + \frac{(m\rho)^m}{m!} \cdot \frac{1}{1-\rho} \right)^{-1}. \tag{3}$$

Let $PW(x)$ denote the probability that a service request's waiting time is less than x (i.e., the maximum waiting time), which is in fact the Cumulative Distribution Function (CDF) of x. According to the queuing theory [29], $PW(x)$ can be derived by using

$$PW(x) = 1 - \frac{P_m}{1-\rho} e^{-m\mu(1-\rho)x}, \tag{4}$$

where P_m can be derived based on Eq. (2) and is expressed as

$$P_m = \frac{(m\rho)^m}{m!} \left(\sum_{k=0}^{m-1} \frac{(m\rho)^k}{k!} + \frac{(m\rho)^m}{m!(1-\rho)} \right)^{-1}. \tag{5}$$

3.3 Cost Model

CSPs need to pay for the cloud infrastructure resources to CIPs. The cost of a CSP can be divided into two parts: 1) the cloud infrastructure renting cost, and 2) the utility cost [1], [2], [3], [4]. The CSP rents servers from a cloud infrastructure provider and pays the corresponding rent. The utility cost is essentially the cost of energy consumption and is determined by the number and speed of the m-server system [5], [6]. Assuming that the renting price of a server per unit time is γ, then the renting cost of a m-server system per unit time is

$$Cost_{\mathrm{r}}(m) = m\gamma. \tag{6}$$

The energy cost of a multiserver system is obtained by the product of energy price and system energy consumption. Although the energy consumption of a multiserver may come from various sources such as processors, network components, and cooling systems, the processor energy consumption dominates the energy consumption of the entire multiserver system [30]. This is because processors dominate system-wide power consumption [30] and the system-wide power consumption is in proportion to the processors' power consumption [31]. Therefore, we focus on the energy consumption of processors in this paper. The energy consumption of a CMOS processor consists of dynamic and static energy consumption. The dynamic power is calculated as $N_{\mathrm{sw}} C_{\mathrm{L}} f^3$ [5], [32], where N_{sw} denotes the average gate switching factor at each clock cycle of the CMOS processor, C_{L} represents the processor loading capacitance, and f is the clock frequency. The server speed s is linearly proportional to the clock frequency of its processor f, i.e., $s \propto f$. In this context, for simplicity, we rewrite the dynamic power of a server as ξs^3 where ξ is a constant related to the hardware. Since we assume that the arrival of service requests follows a Poisson distribution, a multiserver system is not always fully utilized. Given the multiserver's utilization ρ, the dynamic energy consumption of the system with m servers and running speed s during a unit time can be calculated as $m\rho\xi s^3$. Let E^* and δ be the static energy consumption of a server during a unit time and the energy price, respectively. The multiserver system's energy cost per unit time can be calculated by

$$\begin{aligned} Cost_{\mathrm{ut}}(m, s) &= \delta(m\rho\xi s^3 + mE^*) \\ &= \delta m(\rho\xi s^3 + E^*). \end{aligned} \tag{7}$$

Therefore, we obtain the total cost of the service provider per unit time, denoted by $Cost_{\mathrm{tot}}$, as

$$\begin{aligned} Cost_{\mathrm{tot}} &= Cost_{\mathrm{r}} + Cost_{\mathrm{ut}} \\ &= m\gamma + \delta m(\rho\xi s^3 + E^*) \\ &= m\big(\gamma + \delta(\rho\xi s^3 + E^*)\big). \end{aligned} \tag{8}$$

Note that we focus on the homogeneous multiserver system and hence do not consider the server heterogeneity in the configuration model for the multiserver system. But this model can still be extended to the heterogeneous multiserver system by adopting the scheme presented in [2], [28].

3.4 Service Charge Model

To ensure customer satisfaction, negotiation on the service price and the service quality between the service provider and customers is needed. Without loss of generality, we adopt a popular service charge model [1], [2] to describe the negotiation. It is also a service-level-agreement, which is reflected by the waiting time of requests. The service charge of the cloud service for CCs is related to the amount of the service requests and the service-level-agreement. Specifically, the service charge model is developed based on the service request processing requirement r and waiting time ϖ and it is expressed as

$$SC(r) = \begin{cases} ar, & 0 \leqslant \varpi \leqslant x \\ 0, & \varpi > x \end{cases}. \tag{9}$$

a is a constant that represents the service charge per unit service request. x denotes the maximum waiting time in a waiting queue that a service request can tolerate. The expected charge to service requests during a unit time when all the requests are timely served, is then derived as [1]

$$C = \lambda a \overline{r}, \tag{10}$$

where λ and $\overline{r}$ are the cloud service request arrival rate and the expected number of instructions of the cloud service request without suffering any soft errors, respectively.

3.5 The Profit Maximization Problem

The CSP's profit per unit time is defined as the difference between the service revenue gained from providing cloud services to customers and the monetary cost of using a multiserver system and is expressed as

$$Profit = Rev_{\text{dmr}}^{\text{ser}} - Cost_{\text{tot}}, \tag{11}$$

where $Rev_{\text{dmr}}^{\text{ser}}$ is the service revenue earned by the portion of the service requests that are successfully processed and do not miss their deadlines. The details on the derivation of $Rev_{\text{dmr}}^{\text{ser}}$ is presented in Section 4. $Cost_{\text{tot}}$ is the total cost given in Eq. (8).

The goal of this work is to decide the number of servers and the servers' speed for maximizing the profit. To this end, we formulate the profit maximization problem as

$$\text{Max} : Profit(m, s) \tag{12}$$

$$\text{s.t.} : \rho < 1, \tag{13}$$

$$s_l \leq s \leq s_u, \tag{14}$$

$$m_l \leq m \leq m_u. \tag{15}$$

Eq. (13) indicates the utilization constraint that the multiserver utilization ρ must be lower than 1, in order to avoid server crashes [33]. Eqs. (14) and (15) represent the constraints on sever speed and server size, respectively. s_l and s_u are the lower and upper bound on the server speed, respectively. m_l and m_u denote the lower and upper bound on the number of servers, respectively.

4 Deadline Miss Rate and Reliability

This section presents our method to derive the deadline miss rate of cloud service requests, the soft error reliability for average-based scheduling, and the service revenue considering both deadline miss rate and soft error reliability.

4.1 Deadline Miss Rate of Service Requests

As aforementioned, we focus on the service requests with soft real-time requirements, i.e., occasional deadline misses can be tolerated by service requests [13]. The real-time requirement of such service requests is characterized by two commonly used metrics, i.e., slack [11] and deadline miss rate [34]. Slack is defined as the maximum waiting time (i.e., x in Eqs. (4) and (9)) that can be tolerated by a service request before being processed. In other words, for a newly arrived cloud service request in the waiting queue of the multiserver system, if the waiting time of this request is less than its slack, the deadline requirement of this service request would be met. The deadline miss rate is defined as the ratio between the number of cloud service requests that miss their corresponding deadlines and the total number of cloud service requests to be served. The smaller the deadline miss rate, the better the system performance as well as the more profit gained by the service provider.

Referring to [11], [16], we assume the slack of a service request follows a uniform distribution and has a soft real-time requirement

$$S(x) = \begin{cases} 0, & x < S_{\min} \\ \frac{1}{S_{\max} - S_{\min}}, & S_{\min} \leq x \leq S_{\max} \\ 0, & x > S_{\max} \end{cases}. \tag{16}$$

$S_{\min}$ is the minimum slack of a service request, while $S_{\max}$ denotes the maximum value of the slack. According to Eqs. (4), (5), and (16), the deadline miss rate (DMR) of service requests for a multiserver system can be derived as

$$DMR = \frac{m\lambda^m \left(e^{-(\frac{ms}{\overline{r}} - \lambda)S_{\min}} - e^{-(\frac{ms}{\overline{r}} - \lambda)S_{\max}}\right)}{(\frac{s}{\overline{r}})^{m-1}(\frac{ms}{\overline{r}} - \lambda)^2 (S_{\max} - S_{\min})[m! \sum_{k=0}^{m-1} \frac{(m\rho)^k}{k!} + \frac{(m\rho)^m}{1-\rho}]}. \tag{17}$$

To ensure the quality of services, the cloud service requests missing deadlines are free of charge [1], [2]. Under this assumption, the revenue from the cloud services during a unit time when they are all successfully completed is

$$Rev_{\text{dmr}} = \lambda a \overline{r} \cdot (1 - DMR). \tag{18}$$

4.2 Soft Error Reliability for Average-Based Scheduling

There are extensive research efforts [35], [36], [37] that model the soft error reliability for instantaneous-based scheduling. However, there are few works on modeling the soft error reliability for the average-based scheduling. Since the application of our interests is the average-based scheduling of intermittent cloud service requests with soft real-time requirements, we propose a new approach to derive the expected soft error reliability for the average-based scheduling of service requests as follows.

Let λ_{se} be the soft error rate of a server, which is calculated as the average number of soft errors experienced by the server during a unit time. According to [35], [36], [37], soft error rate can be modeled by an exponential distribution as

$$\lambda_{\text{se}}(s) = \lambda_0 e^{d_0 \frac{s_{\max} - s}{s_{\max} - s_{\min}}}. \tag{19}$$

Clearly, when the server is running at the maximum speed, i.e., $s = s_{\max}$, the server system error rate is λ_0. d_0 represents a hardware-related constant used to measure the sensitivity of the error rate to dynamic voltage scaling.

When the service request runs on the server, the reliability of a service request with the existence of soft errors is the probability that the cloud service request is processed without suffering any soft errors. Based on the error rate model given in Eq. (19), it can be formulated as

$$R(m, s, r) = e^{-\lambda_{\text{se}}(s)\frac{r}{ms}}. \tag{20}$$

Like the assumption in [40], in this paper, each service request can tolerate at most one transient fault when run on the server and re-execution is used to improve the reliability of the system due to soft errors. Note that the service request is deemed as failed if itself and its re-execution both suffer a failure. Hence, the soft error reliability of a service request with re-execution can be obtained by

$$R_{\text{re}}(m, s, r) = 1 - (1 - R(m, s, r))^2. \tag{21}$$

Given the above-mentioned models, the following theorem describes how to obtain the expected soft error reliability of a request.

Theorem 4.1. *The expected soft error reliability of a service request, represented by SER satisfies the following*

$$SER = ms \cdot \left(\frac{2}{ms + \lambda_{\text{se}}(s)\overline{r}} - \frac{1}{ms + 2\lambda_{\text{se}}(s)\overline{r}}\right). \tag{22}$$

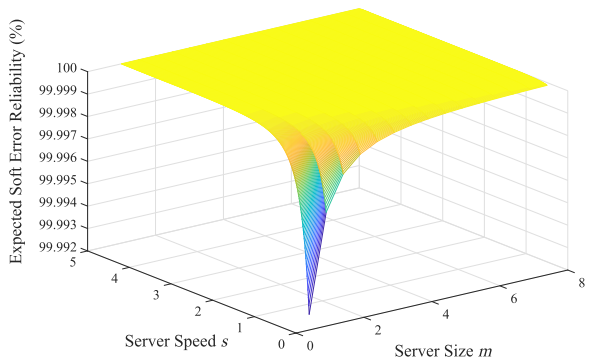


Fig. 2. The expected soft error reliability of a service request processed at varying multiserver settings.

Proof. The number of instructions of a service request r with mean $\overline{r}$ is a random variable that follows the exponential distribution. The probability distribution function of r is $f_r(z) = \frac{1}{\overline{r}} e^{-z/\overline{r}}$. According to Eq. (21) and the probability distribution function f_r, the expected soft error reliability is derived as

$$\begin{aligned}
SER &= R_{\mathrm{re}}(m,s) = \overline{R_{\mathrm{re}}(m,s,r)} \\
&= \int_0^{\infty} f_r(z) \cdot R_{\mathrm{re}}(m,s,z)\, dz \\
&= \int_0^{\infty} \frac{1}{\overline{r}} e^{-z/\overline{r}} \cdot (1 - (1 - e^{-\lambda_{\mathrm{se}}(s)\frac{z}{ms}})^2)\, dz \\
&= \int_0^{\infty} \frac{1}{\overline{r}} e^{-z/\overline{r}}\, dz \\
&- \int_0^{\infty} \frac{1}{\overline{r}} e^{-z/\overline{r}} \cdot (1 - 2e^{-\lambda_{\mathrm{se}}(s)\frac{z}{ms}} + e^{-2\lambda_{\mathrm{se}}(s)\frac{z}{ms}})\, dz \\
&= \int_0^{\infty} \frac{2}{\overline{r}} e^{-z(\frac{1}{\overline{r}} + \frac{\lambda_{\mathrm{se}}(s)}{ms})}\, dz - \int_0^{\infty} \frac{1}{\overline{r}} e^{-z(\frac{1}{\overline{r}} + \frac{2\lambda_{\mathrm{se}}(s)}{ms})}\, dz \\
&= \frac{2}{\overline{r}} \cdot \frac{1}{1/\overline{r} + \frac{\lambda_{\mathrm{se}}(s)}{ms}} - \frac{1}{\overline{r}} \cdot \frac{1}{1/\overline{r} + \frac{2\lambda_{\mathrm{se}}(s)}{ms}} \\
&= ms \cdot \left(\frac{2}{ms + \lambda_{\mathrm{se}}(s)\overline{r}} - \frac{1}{ms + 2\lambda_{\mathrm{se}}(s)\overline{r}} \right).
\end{aligned}$$

□

Given the formula for the expected soft error reliability for service requests, the expected service revenue by serving these requests during a unit time, which use re-execution to improve soft error reliability, is calculated as

$$Rev_{\mathrm{dmr}}^{\mathrm{ser}} = \lambda a \overline{r} \cdot (1 - DMR) \cdot SER, \tag{23}$$

where λ is the cloud service requests' arrival rate and the service price per service request is represented by a.

Fig. 2 plots the expected soft error reliability of a service request under varying multiserver settings. In this figure, we set $s = [0.1, 5.0]$, $m = [1, 8]$ and use the same value of parameters λ_0, d_0 as in [38]. The results clearly show that the expected soft error reliability increases with the growth of server size and speed. Although the expected system reliability due to soft errors can be effectively improved by increasing server size or speed, a higher cost (and hence a lower profit) is inevitable. Thus, it is crucial to consider the impact of soft error reliability on cost when the CSP configures the multiserver to achieve maximum profit.

5 Profit Maximization

Given the problem formulation in Eqs. (11), (12), (13), (14), and (15) and the DMR and SER computation models, this section presents the details of our augmented Lagrange multiplier based approach to determine a multiserver configuration that maximizes the CSP's profit. Specifically, our approach leverages augmented Lagrange multiplier, since augmented Lagrange multiplier is powerful for solving constrained optimization problems. Our approach contains two key steps. First, we convert the profit maximization problem defined in Eqs. (11), (12), (13), (14), and (15) into an unconstrained problem by building an augmented Lagrange function. Then, to obtain the optimal solution to the unconstrained problem, we develop an augmented Lagrange multiplier-based iterative algorithm. The obtained optimal solution to the unconstrained problem is deemed as an optimal solution to the corresponding constrained problem when the solution error is less than a sufficiently small threshold. For ease of reading, Table 1 shows the main variables and descriptions in this paper.

5.1 Augmented Lagrange Function

Similar to [3], the problem of profit maximization for the service provider is first converted into an unconstrained minimization problem by building an augmented Lagrange function. In order to utilize the augmented Lagrange method [3], the profit maximization objective defined in Eq. (12) is converted into a minimization function (i.e., the standard form) as

$$\mathrm{Min} : -Profit. \tag{24}$$

Combining the equations given in Eqs. (8), (10), (11), (17), and (22) with the minimization function, our constrained minimization problem can be rewritten as

$$\begin{aligned}
\mathrm{Min} : O(m,s) &= -Profit \\
&= -(\lambda a \overline{r}(1 - DMR) \cdot SER - Cost_{\mathrm{tot}}) \qquad (25)
\end{aligned}$$

$$\text{s. t.} : g_1(m,s) = 1 - \frac{\lambda \overline{r}}{ms} > 0 \qquad (26)$$

$$g_2(m,s) = m - m_l \geq 0 \qquad (27)$$

$$g_3(m,s) = m_u - m \geq 0 \qquad (28)$$

$$g_4(m,s) = s - s_l \geq 0 \qquad (29)$$

$$g_5(m,s) = s_u - s \geq 0, \qquad (30)$$

where $g_j(m,s)$ $(j = 1,2,3,4,5)$ are the constraints on m and s corresponding to constraints in Eqs. (13), (14), and (15). The above formulation (i.e., Eqs. (25), (26), (27), (28), (29), and (30)) can be converted into an unconstrained minimization problem (i.e., the augmented Lagrange function) as

$$\begin{aligned}
\phi(m,s,\sigma,\mathbf{v},\mathbf{y}) &= O(m,s) - \sum_{j=1}^{5} v_j(g_j(m,s) - y_j^2) \\
&+ \frac{\sigma}{2} \sum_{j=1}^{5} (g_j(m,s) - y_j^2)^2, \qquad (31)
\end{aligned}$$

where σ $(\sigma > 0)$ is a constant denoting the penalty factor of the augmented Lagrange function, $\mathbf{v}$ is the Lagrange

TABLE 1
Description of Main Variables

Variable	Description
m	server size, i.e., the number of servers in a multiserver system
s	server speed, i.e., the execution speed of a server
ρ	system utilization of a multiserver per unit time
P_k	probability that k service requests are waiting or processing in the system
P_0	probability that there is no service request in the system
x	maximum waiting time of a service request can tolerate in the system
$PW(x)$	probability that a service request's waiting time is less than x
λ	number of service requests arrive to the system per unit time
λ_{se}	number of soft errors experienced by the server per unit time
C	expected charge to service requests during a unit time when all the requests are timely served
DMR	ratio between the number of requests that miss their deadlines and the total number of requests
SER	probability of a service request with re-execution which can be successfully executed
$Rev_{\text{dmr}}^{\text{ser}}$	expected service revenue by serving service requests during a unit time
$Cost_{\text{tot}}$	total cost of the service provider per unit time
$\mathbf{v}$	Lagrange multiplier vector, i.e., $\mathbf{v} = \{v_j\}, (j = 1, 2, 3, 4, 5)$
$\mathbf{y}$	auxiliary variable vector, i.e., $\mathbf{y} = \{y_j\}, (j = 1, 2, 3, 4, 5)$
σ	a constant used to constrain the bounds of the augmented Lagrangian function
$g_j(m, s), (j = 1, 2, 3, 4, 5)$	constraints on m and s corresponding to constraints in Eqs. (13), (14), and (15)
$\phi(m, s, \sigma, \mathbf{v}, \mathbf{y})$	augmented Lagrangian function that is converted by Eqs. (25), (26), (27), (28), (29), and (30)
$O(m, s)$	objective function, i.e., Eq. (25)
$F(m, s)$	function used to simplify expressions, i.e., Eq. (36)
D_1, D_2, D_3, D_4	functions used to simplify expressions, which are in the 4th paragraph of Section 5.2.2
$G_j(m, s)$	function used to simplify expressions, which is in the 5th paragraph of Section 5.2.2

multiplier vector defined by the multiplier method, and $\mathbf{y}$ is the vector used to convert inequality constraints into equality constraints. The augmented Lagrange function defined in Eq. (31) is also called the multiplier penalty function.

5.2 Optimal Multiserver Configuration for Profit Maximization

For obtaining the optimal solution (i.e., m and s) of Eq. (31), we need to transform $\phi(m, s, \sigma, \mathbf{v}, \mathbf{y})$ to $\phi(m, s)$ by removing σ, $\mathbf{v}$, and $\mathbf{y}$. If we could remove them, we can find the optimal solution by calculating the partial derivatives of $\phi(m, s)$ with respect to m and s. Since σ is a constant and the Lagrange multiplier vector $\mathbf{v}$ can be readily derived during the process of solving the augmented Lagrange function (i.e., Eq. (31)), below we first describe how we remove $\mathbf{y}$ in $\phi(m, s, \sigma, \mathbf{v}, \mathbf{y})$ and then detail the partial derivatives of $\phi(m, s)$ with respect to m and s. Note that the server speed cloud be taken from a predefined set if not considering overclocking, in this case, our model (Eqs. (12), (14), and (15)) is still applicable, which only needs to modify the Eq. (14). To the solution, for a limited and discrete server speed set, the solution can traverse to find the optimal server speed, in other words, the solution can treat s as a known variable (i.e., $\phi(m, s)$ changes to $\phi(m)$), then it can derive the optimal server size m under an arbitrary server speed s.

5.2.1 Transform $\phi(m, s, \sigma, \mathbf{v}, \mathbf{y})$ to $\phi(m, s)$

$\mathbf{y}$ is an auxiliary matrix used to transform inequality constraints of Eqs. (27), (28), (29), and (30) into equality constraints in order to build the augmented Lagrange function. The main idea of removing $\mathbf{y}$ is to express $\mathbf{y}$ as a function of m, s, σ, and $\mathbf{v}$ when Eq. (31) reaches its minimum. The algebraic transformation for removing $\mathbf{y}$ proceeds as follows.

We first use the method of completing the square for deriving the minimum value of Eq. (31) with respect to $\mathbf{y}$, i.e.,

$$\begin{aligned}\phi(m, s, \sigma, \mathbf{v}, \mathbf{y}) &= O(m, s) \\ &+ \sum_{j=1}^{5}\Big[-v_j(g_j(m, s) - y_j^2) + \frac{\sigma}{2}(g_j(m, s) - y_j^2)^2\Big] = O(m, s) \\ &+ \sum_{j=1}^{5}\left[\frac{\sigma}{2}(y_j^2 - \frac{1}{\sigma}(\sigma g_j(m, s) - v_j))^2 - \frac{v_j^2}{2\sigma}\right]. \end{aligned} \tag{32}$$

Clearly, the augmented Lagrange function ϕ reaches its minimum value only when $(y_j^2 - \frac{1}{\sigma}(\sigma g_j(m, s) - v_j))^2$ takes the minimum value, i.e., y_j meets the condition that

$$\begin{cases} y_j^2 = \frac{1}{\sigma}(\sigma g_j(m, s) - v_j), & \sigma g_j(m, s) - v_j \geq 0 \\ y_j^2 = 0, & \sigma g_j(m, s) - v_j < 0 \end{cases}. \tag{33}$$

From Eq. (33), we can deduce that

$$y_j^2 = \frac{1}{\sigma}\max(0, \sigma g_j(m, s) - v_j), j = 1, 2, 3, 4, 5. \tag{34}$$

We can simplify the augmented Lagrange function by substituting Eq. (34) into Eq. (31). Finally, $\mathbf{y}$ is successfully removed and we get

$$\begin{aligned}\phi(m, s) &= O(m, s) \\ &+ \frac{1}{2\sigma}\sum_{j=1}^{5}\Big[\big(\max(0, v_j - \sigma g_j(m, s))\big)^2 - v_j^2\Big]. \end{aligned} \tag{35}$$

5.2.2 Compute the Partial Derivatives of $\phi(m, s)$

To get the minimum value of $\phi(m, s)$, we need to compute the partial derivatives of $\phi(m, s)$ with respect to m and s.

Below, we detail the computation of these derivatives. For better readability, we let $F(m,s)$ represent the second term in Eq. (35), i.e.,

$$F(m,s)=\frac{1}{2\sigma}\sum_{j=1}^{5}[(\max(0,v_j-\sigma g_j(m,s)))^2-v_j^2]. \tag{36}$$

We can easily derive the partial derivatives of $\phi(m,s)$ with respect to m and s once the corresponding partial derivatives of $O(m,s)$ and $F(m,s)$ are obtained.

The partial derivatives of $O(m,s)$ with respect to variables m and s are calculated as

$$\begin{cases}\frac{\partial O}{\partial m}= & -(\lambda a\overline{r}\cdot\frac{\partial(1-DMR)}{\partial m}\cdot SER\\ & +\ \lambda a\overline{r}\cdot(1-DMR)\cdot\frac{\partial(SER)}{\partial m}-\frac{\partial Cost_{\rm tot}}{\partial m}),\\ \frac{\partial O}{\partial s}= & -(\lambda a\overline{r}\cdot\frac{\partial(1-DMR)}{\partial s}\cdot SER\\ & +\ \lambda a\overline{r}\cdot(1-DMR)\cdot\frac{\partial(SER)}{\partial s}-\frac{\partial Cost_{\rm tot}}{\partial s}),\end{cases} \tag{37}$$

where DMR, SER, and $Cost_{\rm tot}$ are given in Eq. (17), Eq. (22), and Eq. (8), respectively. Clearly, the key to computing the partial derivatives of Eq. (37) concerning m and s is to derive $\frac{\partial(DMR)}{\partial m}$, $\frac{\partial(DMR)}{\partial s}$, $\frac{\partial(SER)}{\partial m}$, $\frac{\partial(SER)}{\partial s}$, $\frac{\partial Cost_{\rm tot}}{\partial m}$, and $\frac{\partial Cost_{\rm tot}}{\partial s}$. To this end, we first calculate DMR's partial derivatives with respect to m and s, and then the corresponding partial derivatives of SER and $Cost_{\rm tot}$, respectively.

By applying Taylor series expansions $\sum_{k=0}^{m-1}\frac{(m\rho)^k}{k!}\approx e^{m\rho}$ and $m!\approx\sqrt{2\pi m}(\frac{m}{e})^m$, DMR in Eq. (17) can be re-written as

$$DMR=\frac{1}{S_{\max}-S_{\min}}\cdot\frac{m\lambda^m(e^{-(\frac{ms}{\overline{r}}-\lambda)S_{\min}}-e^{-(\frac{ms}{\overline{r}}-\lambda)S_{\max}})}{(\frac{s}{\overline{r}})^{m-1}(\frac{ms}{\overline{r}}-\lambda)^2[\sqrt{2\pi m}(\frac{m}{e})^m e^{m\rho}+\frac{(m\rho)^m}{1-\rho}]}. \tag{38}$$

Let $D_1=e^{-(\frac{ms}{\overline{r}}-\lambda)S_{\min}}-e^{-(\frac{ms}{\overline{r}}-\lambda)S_{\max}}$, $D_2=(\frac{s}{\overline{r}})^{m-1}$, $D_3=(\frac{ms}{\overline{r}}-\lambda)^2$, $D_4=\sqrt{2\pi}(\frac{m}{e})^m e^{m\rho}+\frac{(m\rho)^m}{1-\rho}$, then the partial derivatives of DMR with respect to m and s can be obtained as

$$\frac{\partial(DMR)}{\partial m}=\frac{1}{(S_{\max}-S_{\min})(D_2D_3D_4)^2}\cdot\left(((\lambda^m+m\lambda^m\ln\lambda)D_1+m\lambda^m\frac{\partial D_1}{\partial m})D_2D_3D_4-(m\lambda^mD_1)\left(\frac{\partial D_2}{\partial m}D_3D_4+D_2\frac{\partial D_3}{\partial m}D_4+D_2D_3\frac{\partial D_4}{\partial m}\right)\right), \tag{39}$$

$$\frac{\partial(DMR)}{\partial s}=\frac{m\lambda^m}{(S_{\max}-S_{\min})(D_2D_3D_4)^2}\cdot\left(\frac{\partial D_1}{\partial s}D_2D_3D_4-D_1\frac{\partial D_2}{\partial s}D_3D_4-D_1D_2\frac{\partial D_3}{\partial s}D_4-D_1D_2D_3\frac{\partial D_4}{\partial s}\right), \tag{40}$$

where

$$\frac{\partial D_1}{\partial m}=\frac{s}{\overline{r}}\left(-S_{\min}\cdot e^{-(\frac{ms}{\overline{r}}-\lambda)S_{\min}}+S_{\max}\cdot e^{-(\frac{ms}{\overline{r}}-\lambda)S_{\max}}\right),$$
$$\frac{\partial D_1}{\partial s}=\frac{m}{\overline{r}}\left(-S_{\min}\cdot e^{-(\frac{ms}{\overline{r}}-\lambda)S_{\min}}+S_{\max}\cdot e^{-(\frac{ms}{\overline{r}}-\lambda)S_{\max}}\right),$$
$$\frac{\partial D_2}{\partial m}=\left(\frac{s}{\overline{r}}\right)^{m-1}\ln\left(\frac{s}{\overline{r}}\right),$$
$$\frac{\partial D_2}{\partial s}=\frac{m-1}{\overline{r}}\left(\frac{s}{\overline{r}}\right)^{m-2},$$
$$\frac{\partial D_3}{\partial m}=\frac{2s}{\overline{r}}\left(\frac{ms}{\overline{r}}-\lambda\right),$$
$$\frac{\partial D_3}{\partial s}=\frac{2m}{\overline{r}}\left(\frac{ms}{\overline{r}}-\lambda\right),$$
$$\frac{\partial D_4}{\partial m}=\left(\frac{m}{e}\right)^m e^{m\rho}\left(\frac{\pi}{\sqrt{2\pi m}}+\sqrt{2\pi m}(\ln\frac{m}{e}+1)\right)+\frac{(m\rho)^m}{(1-\rho)^2}\left(\ln(m\rho)(1-\rho)+\frac{\lambda\overline{r}}{m^2s}\right),$$
$$\frac{\partial D_4}{\partial s}=-\sqrt{2\pi m}\left(\frac{m}{e}\right)^m e^{m\rho}\left(\frac{\lambda\overline{r}}{s^2}\right)-\frac{(m\rho)^m(1-\rho)-(m\rho)^m(\frac{\lambda\overline{r}}{s^2m})}{(1-\rho)^2}.$$

The partial derivatives of SER with respect to m and s are

$$\frac{\partial(SER)}{\partial m}=\frac{2s}{ms+\lambda_{se}(s)\overline{r}}\cdot(1-\frac{ms}{ms+\lambda_{se}(s)\overline{r}})-\frac{s}{ms+2\lambda_{se}(s)\overline{r}}\cdot\left(1-\frac{ms}{ms+2\lambda_{se}(s)\overline{r}}\right), \tag{41}$$

$$\frac{\partial(SER)}{\partial s}=\frac{m}{ms+\lambda_{se}(s)\overline{r}}\cdot\left(2-\frac{ms+\overline{r}s\frac{\partial\lambda_{se}(s)}{\partial s}}{ms+\lambda_{se}(s)\overline{r}}\right)-\frac{m}{ms+2\lambda_{se}(s)\overline{r}}\cdot\left(1-\frac{ms+2\overline{r}s\frac{\partial\lambda_{se}(s)}{\partial s}}{ms+2\lambda_{se}(s)\overline{r}}\right), \tag{42}$$

where $\frac{\partial\lambda_{se}(s)}{\partial s}$ is expressed as

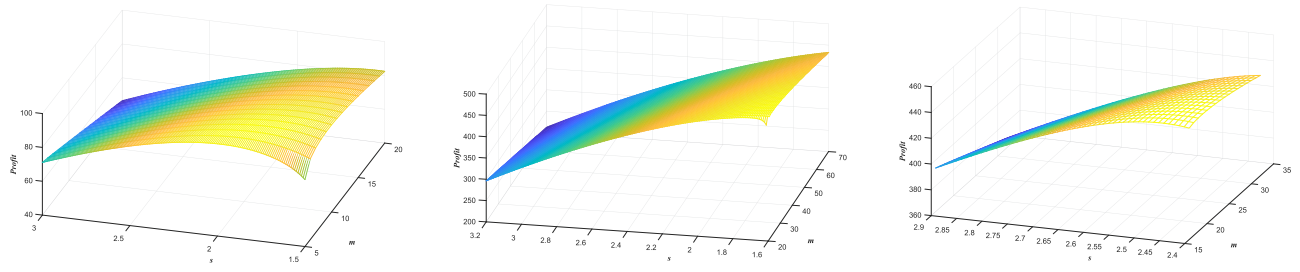
$$\frac{\partial\lambda_{se}(s)}{\partial s}=-\lambda_0e^{d_0\frac{s_{\max}-s}{s_{\max}-s_{\min}}}\left(\frac{d_0}{s_{\max}-s_{\min}}\right).$$

The corresponding partial derivatives of $Cost_{\rm tot}$ are

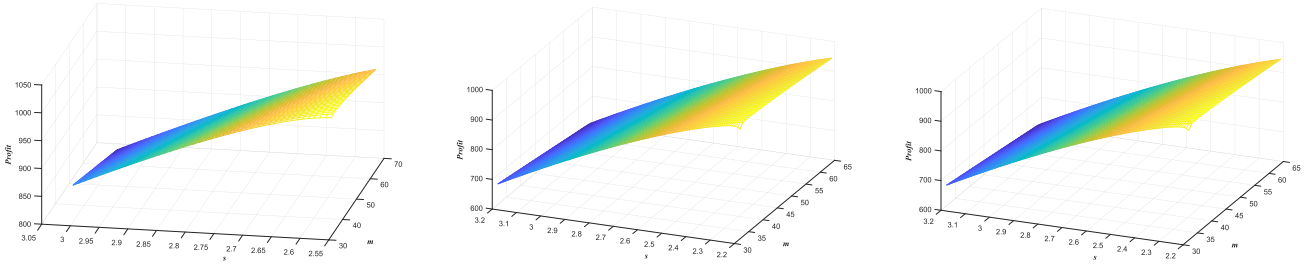
$$\frac{\partial Cost_{\rm tot}}{\partial m}=\delta E^*+\gamma, \tag{43}$$
$$\frac{\partial Cost_{\rm tot}}{\partial s}=2\delta\xi\lambda\overline{r}s. \tag{44}$$

Let $G_j(m,s)=(\max(0,v_j-\sigma g_j(m,s)))^2-v_j^2,\ j=1,2,3,4,5$, then we have

$$F(m,s)=\frac{1}{2\sigma}\sum_{j=1}^{5}G_j(m,s). \tag{45}$$



(a) $\lambda = 7$, $s = 1.5 \rightarrow 3.0$, $m = 5 \rightarrow 20$. (b) $\lambda = 32$, $s = 1.6 \rightarrow 3.2$, $M = 20 \rightarrow 70$. (c) $\lambda = 35$, $s = 2.4 \rightarrow 2.9$, $M = 16 \rightarrow 32$.



(d) $\lambda = 80$, $s = 2.55 \rightarrow 3$, $M = 32 \rightarrow 64$. (e) $\lambda = 38$, $s = 2.4 \rightarrow 3.2$, $M = 16 \rightarrow 32$. (f) $\lambda = 38$, $s = 2.2 \rightarrow 3.2$, $M = 32 \rightarrow 64$.

Fig. 3. The values of $Profit(m, s)$ under varying configurations.

The partial derivatives of $F(m, s)$ with respect to variables m and s are

$$\begin{cases} \frac{\partial F}{\partial m} = & \frac{1}{2\sigma}\sum_{j=1}^{5}\frac{\partial G_j}{\partial m} \\ \frac{\partial F}{\partial s} = & \frac{1}{2\sigma}\sum_{j=1}^{5}\frac{\partial G_j}{\partial s} \end{cases}, \tag{46}$$

where

$$\frac{\partial G_j}{\partial m} = \begin{cases} 0, v_j - \sigma g_j(m,s)) \leq 0 \\ -2\sigma(v_j - \sigma g_j(m,s))\frac{\partial g_i(m,s)}{\partial m}, v_j - \sigma g_j(m,s)) > 0 \end{cases}, \tag{47}$$

$$\frac{\partial G_j}{\partial s} = \begin{cases} 0, v_j - \sigma g_j(m,s)) \leq 0 \\ -2\sigma(v_j - \sigma g_j(m,s))\frac{\partial g_i(m,s)}{\partial s}, v_j - \sigma g_j(m,s)) > 0 \end{cases}. \tag{48}$$

The partial derivatives of $g_j(m, s)$ in Eqs. (47) and (48) can be easily derived from Eqs. (26), (27), (28), (29), and (30).

Once the partial derivatives with respect to m and s are derived, the optimal solution for the profit maximization problem can be obtained by setting these partial derivatives to 0. Obtaining the optimal solution for the formulated problem analytically requires the problem convex, m and s are continuous variables and also needs to introduce a closed-form expression for approximating the function (i.e., Eq. (12)) [1], [2], [4], [5]. However, since the server size m (or the server speed s) of an actual multiserver system is a discrete variable, the error caused by the closed-form approximation is not negligible, thus it is not feasible to prove the convergence of the our optimal problem analytically. Although we cannot prove the convergence of the formulated problem analytically, we can still perform extensive experiments to plot $Profit(m, s)$ for different configurations to check whether the optimal solution of $Profit(m, s)$ exists, which is similar to [1], [4]. Below we show the values of $Profit(m, s)$ under six different multiserver configurations. All the experiments of $Profit(m, s)$ indicate that there must be an optimal point for the profit maximization problem. The experimental results here are not comprehensive and are only meant to demonstrate the existence of an optimal point. Based on the observation, an iterative algorithm is utilized to find the numerical optimal solution.

5.3 ALM-Based Iterative Method

Since the analytical solution of m and s cannot be directly calculated through the partial derivatives in Section 5.2, we introduce an ALM-based iterative (ALMI) method which can efficiently find numerical solutions. During each iteration of ALMI, it determines a numerical solution and verifies whether the solution satisfies the constraints. If the constraints are not met, the Lagrangian multipliers are updated and the process repeats until the optimal solution is found. The pseudo-code of the proposed method ALMI is given in Algorithm 1.

Algorithm 1 takes the following inputs: threshold L on the number of iterations, penalty factor σ, update rate of penalty factor η (to avoid the solution converging too slow or not converging), threshold ε on the error between the solutions of the unconstrained problem in Eq. (35) and that of the constrained problem in Eqs. (25), (26), (27), (28), (29), and (30), and threshold β on solution's convergence speed.

Algorithm 1 first generates an initial solution $(m^{(0)}, s^{(0)})$ randomly, sets the initial value of multiplier vector $v^{(1)}$ to the all-one vector, and initializes the counter for iteration l to 1 (Lines 1-3). It then searches for the optimal solution iteratively (Lines 4-17). In each iteration, Algorithm 1 attempts to find an optimal solution, represented by $(m^{(l)}, s^{(l)})$, for the unconstrained minimization problem given in Eq. (35) using our *ALF-Solver* (Line 5). The details on *ALF-Solver* are provided in Algorithm 2. After that, Algorithm 1 exploits the optimal solution obtained in the previous iteration to address the unconstrained problem. If the error between the solution to the unconstrained minimization problem given in Eq. (35) and the solution to the constrained minimization problem given in Eqs. (25), (26), (27), (28), (29), and (30) is sufficiently small, i.e., $\|g_j(m^{(l)}, s^{(l)}) - y_j^2\| < \varepsilon$ (line 6), then the Lagrange multiplier vector v converges, and the solution found by *ALF-Solver* can be treated as the optimal solution to

the constrained minimization problem (Lines 7-8). In this case, the optimal multiserver configuration is found and the corresponding maximum profit can be calculated using Eqs. (11), (37), (39), (40), (41), (42), (43), (44), (46), (47), and (48) (Lines 9-12). Thus, the iteration is terminated and the algorithm returns the optimal profit $Profit^*$ and the optimal configuration (m^*, s^*) (Line 18). Otherwise, the solution derived by *ALF-Solver* cannot be deemed as the optimal solution to the constrained minimization problem. Further, if this solution meets the condition $\|g_j(m^{(l)}, s^{(l)}) - y_j^2\|/\|g_j(m^{(l-1)}, s^{(l-1)}) - y_j^2\| \geq \beta$ (Line 13), indicating that the convergence speed of the solution found by **ALF-Solver** is slow (Line 14), Algorithm 1 then modifies the value of penalty factor σ to speed up the convergence speed by setting $\sigma = \eta\sigma$ (Line 15). The Lagrange multiplier and iteration counter are updated by $v_j^{(l+1)} = \max(0, v_j^{(l)} - \sigma g_j(m^{(l)}, s^{(l)}))$, $j = 1, 2, 3, 4, 5$ (Lines 16-17). The iteration stops if the predefined number of iteration is reached and Algorithm 1 finally returns the best profit and configuration $[Profit^*, m^*, s^*]$ found (Line 18).

Algorithm 1. Augmented Lagrange Multiplier Based Iterative (ALMI) Method

Input: Threshold L on the number of iterations, penalty factor σ, update rate of penalty factor, η, threshold ε on the solution error, threshold β on convergence speed.
Output: Optimal solution of server size m^*, server speed s^*, and profit $Profit^*$;
1: Generate an initial solution $(m^{(0)}, s^{(0)})$ randomly;
2: Calculate the initial value of multiplier vector $v^{(1)}$ to the all-one vector;
3: Set $l = 1$;
4: **while** $(l < L)$ **do**
5: $(m^{(l)}, s^{(l)})$ = **ALF-Solver**$(m^{(l-1)}, s^{(l-1)}, v^{(l)}, \sigma)$;
// **ALF-Solver** is called to calculate the optimal solution to the augmented Lagrange function in Eq. (35);
6: **if** $\|g_j(m^{(l)}, s^{(l)}) - y_j^2\| < \varepsilon$ **then**
7: // Ensure the solution to the unconstrained problem in Eq. (35) is an acceptable solution to the constrained problem in Eqs. (25), (26), (27), (28), (29), and (30);
8: // y_j^2 can be calculated by Eq. (34);
9: Calculate $Profit$ using Eq. (11);
10: $Profit^* = Profit$;
11: $(m^*, s^*) = (m^{(l)}, s^{(l)})$;
12: **break**;
13: **else if** $\|g_j(m^{(l)}, s^{(l)}) - y_j^2\|/\|g_j(m^{(l-1)}, s^{(l-1)}) - y_j^2\| \geq \beta$ **then**
14: //The convergence speed of solutions derived by **ALF-Solver** is slow;
15: $\sigma = \eta\sigma$; //Update penalty factor σ to speed up the convergence speed;
16: $v_j^{(l+1)} = \max(0, v_j^{(l)} - \sigma g_j(m^{(l)}, s^{(l)}))$, $j = 1, 2, 3, 4, 5$;
17: $l = l + 1$;
18: **return** $[Profit^*, m^*, s^*]$;

Algorithm 2 summarizes the process of our proposed *ALF-Solver* which uses a gradient descent approach [41] to find the optimal configuration of m and s during each iteration of Algorithm 1. The inputs to Algorithm 2 include the solution $(m^{(l-1)}, s^{(l-1)})$ derived from the $(l-1)$-th iteration and the parameters $v^{(l)}$ used in the l-th iteration. According to [41], we set the maximum iterator number $iter_{num} = 1000$, the iterator error $error = 1e-10$, and the learning rate $alpha = 0.1$ (Line 1). Then, we initialize the iterator counter $i = 1$ and a temporary variable $change = 0$ (Line 2). Following the derivation in Section 5.2, Algorithm 2 computes the partial derivatives of Eq. (31) based on Eqs. (37), (39), (40), (41), (42), (43), (44), and (46), (47), and (48), and puts $v^{(l)}$ into these computed derivatives (Line 3). Then, Algorithm 2 finds the optimal server size m^l and server speed s^l by using a gradient descent process that computes $\partial\phi(m^{(l-1)}, s^{(l-1)}, v^{(l)})/\partial m$ and $\partial\phi(m^{(l-1)}, s^{(l-1)}, v^{(l)})/\partial s$ (Lines 4-8). Once $(m^{(l)}, s^{(l)})$ is found or the iterator counter reaches the maximum iteration, Algorithm 2 returns the solution to Algorithm 1 (Line 9).

Algorithm 2. The Augmented Lagrange Function Solver (**ALF-Solver**)

Input: $m^{(l-1)}, s^{(l-1)}, v^{(l)}$;
Output: $m^{(l)}, s^{(l)}$;
1: Set $iter_{num} = 1000$, $error = 1e-10$, $alpha = 0.1$;
2: Initialize $i = 1$, $change = 0$;
3: Compute the partial derivatives of Eq. (31) based on Eqs. (37), (39), (40), (41), (42), (43), (44), (46), (47), and (48);
4: **while** $i < iter_{num}$ and $change > error$ **do**
5: $m^{(l)} = m^{(l-1)} - alpha \times \frac{\partial\phi(m^{(l-1)}, s^{(l-1)}, v^{(l)})}{\partial m}$;
6: $s^{(l)} = s^{(l-1)} - alpha \times \frac{\partial\phi(m^{(l-1)}, s^{(l-1)}, v^{(l)})}{\partial s}$;
7: $change = |\phi(m^{(l-1)}, s^{(l-1)}, v^{(l)}) - \phi(m^{(l)}, s^{(l)}, v^{(l)})|$;
8: $i = i + 1$;
9: **return** $(m^{(l)}, s^{(l)})$;

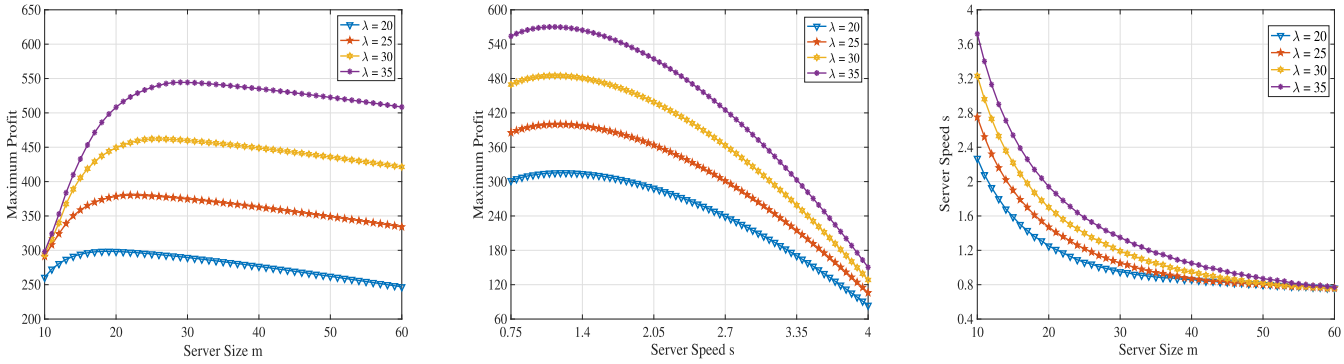
6 Evaluation

Consider the scenario where a CSP leases the infrastructure resources to configure the multiserver system to handle service requests submitted by CCs. Such the multiserver system can be an infrastructure of many forms, e.g., the multiserver system could be the multicore server processor and each server is a single core [27], the traditional server cluster and each server is a normal processor [26], and the blade center and each server is a blade server [25]. To simulate the considered scenario, we assume that the multiserver system is in the form of the multicore server processor, and use advanced multiple processors (e.g., AMD EPYC 7742 processor [43], Intel Platinum 8376 processor [46]) to simulate our multiserver system. Table 1 summarizes the parameters used to estimate the energy consumption cost per unit, infrastructure resource usage cost, and service requests. The key variables like the sever size (m) and speed (s), and the cloud service requests arrival rate (λ) are extracted from real-world systems [43], [44], [45], [46] and existing literature [2], [4]. Besides, the setup of the multiserver configuration not only cover the up-to-date multiprocessors (e.g., the AMD EPYC 7742 processor [43] and the Intel Platinum 8376 processor [46]) but also represent some real-world CSPs (e.g., Huawei Elastic Cloud Server (HECS) [47] where the server size of HECS varies in the range of [1,60] and the maximum server speed of HECS runs at 3.5GHz). All the experiments are implemented on the same Rack Server which is installed with a Linux version of Matlab x64 (version 9.3.0.713579 which contains Simulink, Global Optimization Toolbox, etc.) and equipped with 64GB DDR4 memory and a 2.1GHz Intel Xeon Silver 8-core processor.

In this section, we evaluate the performance of our approach by analyzing the trend of the maximum profit

TABLE 2
Setup of Main Notations Used in the Experiments

Notation	Definition	Value
λ_0	soft error rate while the server at the maximum speed	10^{-5} [38]
d_0	hardware-dependent coefficient related to dynamic voltage scaling	2 [42]
a	service charge per unit quantity of service	18 cents per billion instructions [39]
v	usage fee of a server during a unit of time	1.5 cents per second [1], [2], [4], [5]
ξ	processor dependent coefficient	9.4192 [1], [2], [4], [5]
P^*	static power dissipation	2 $Watts$ per second [2], [4], [5]
δ	energy price paid for a server during a unit of time	0.1 cents per $Watt \times second$ [2], [4], [5]
$S_{\min}$	minimum value of the slack during a unit of time	0.5 s [13]
$S_{\max}$	maximum value of the slack during a unit of time	5 s [13]



(a) Maximum profit vs. server size. (b) Maximum profit vs. server speed. (c) Server speed vs. server size.

Fig. 4. Maximum profit and optimal multiserver configuration under varying service request arrival rates.

with the multiserver configuration. We also compare our approach with two SOTA methods for increasing profit. There sets of simulation experiments are summarized as follows. Specifically, in the first set of experiments, we study the optimal multiserver system configuration (which are the server size m and speed s) as well as the corresponding profits under varying arrival rates of cloud service requests. This set of experiments is to explore how a multiserver configuration affects the CSP's profit under different arrival rates of cloud service requests.

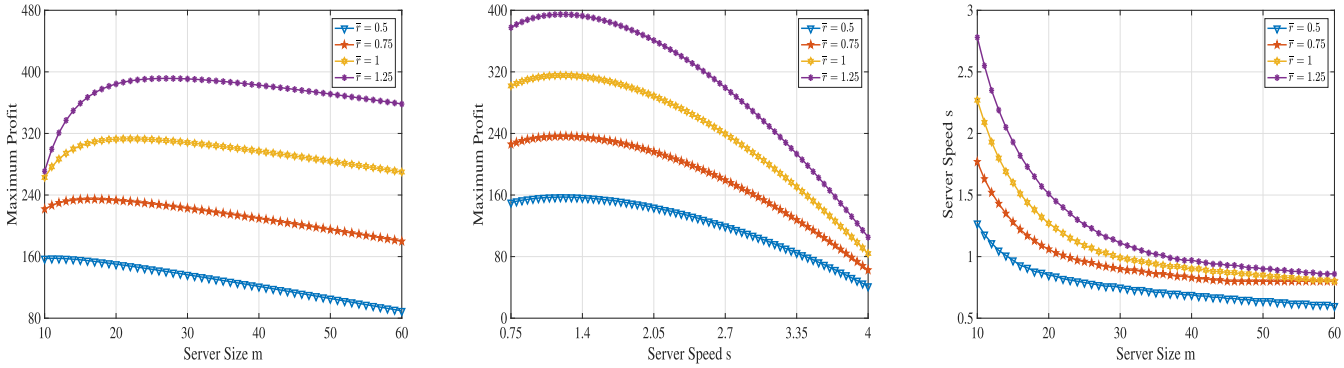
In the second experiment set, we observe the maximum profit of the CSP as well as the corresponding multiserver configuration under varying expected numbers of instructions for service requests. This set of experiments is to investigate how a multiserver configuration affects the profit of the CSP under different expected numbers of instructions for service requests.

In the third experiment set, our proposed multiserver configuration scheme in maximum profit is compared with two SOTA methods. This set of experiments evaluate the effectiveness of the proposed scheme in terms of the profits of the CSP.

6.1 Profit With Varying Service Request Arrival Rates

In this set of experiments, we aim to explore how a multiserver configuration affects the CSP's profit under different cloud service request arrival rates. The expected number of the service request's instructions $\overline{r}$ is set to 1. The server size m ranges from 10 to 60, the server speed s ranges from 0.75 to 4.0, and the value of service request arrival rate λ ranges from {20, 25, 30, 35}. The values of parameters λ_0, d_0, a, v, ξ, P^*, δ, $S_{\min}$ and $S_{\max}$ are consistent with those in Table 2.

Figs. 4a and 4b show the trend of the corresponding maximum profit achieved by our proposed scheme when increasing the server size (m) and speed (s) under varying service request arrival rates λ, respectively. From the two figures, we observe that a larger service request arrival rate λ is able to bring higher profit for CSPs. Moreover, we also find that the profit varies with the server size m (see Fig. 4a) and the server running speed s (see Fig. 4b). Specifically, it can be seen from Fig. 4a that when the server size m is small, it is beneficial to lease more servers appropriately. But as the number of rented servers increases and the server size m exceeds a threshold that corresponds to the processing capability required to complete the cloud service requests at rate λ, the profit gradually decreases due to the extra renting and operating costs of excessive servers. As shown in Fig. 4b, when the server speed s falls into a small range (e.g., from 0.75 to 1.2), the service provider's profit increases with the server speed s. If continuously increasing the server speed s, the extra energy cost induced by speedup is larger than the cost savings achieved by reducing the server size, leading to the decrease in the profit of the service provider. From Fig. 4c, we observe that for a given service request arrival rate λ, when the server size m increases, the optimal server running speed s decreases. This is because that there exists an upper bound on the processing capability required by a given cloud service request. If the server size m exceeds a threshold, our proposed strategy will lower server speed s to decrease energy cost.



(a) Maximum profit vs. server size. (b) Maximum profit vs. server speed. (c) Server speed vs. server size.

Fig. 5. Maximum profit and optimal multiserver configuration under varying expected numbers of instructions for requests.

6.2 Profit With Varying Expected Numbers of Instructions

In this set of experiments, we investigate how the configuration of a multiserver system affects the maximum profit of the CSP under varying expected number of instructions $\overline{r}$ for cloud service requests. We set service request arrival rate $\lambda = 20$. The value of server speed s, server size m, expected number of instructions $\overline{r}$ for the cloud service request range from (0.75, 4.0), (10, 60), and {0.5, 0.75, 1, 1.25}, respectively. The values of parameters λ_0, d_0, a, v, ξ, P^*, δ, $S_{\min}$ and $S_{\max}$ are consistent with those in Table 2.

Figs. 5a and 5b show the trend of the maximum profit achieved by our proposed scheme when increasing the server size m (or server speed s) under varying expected number of instructions $\overline{r}$ of cloud service requests. From these two figures, we can observe that given the expected number of instructions, the CSP's profit will gradually rise to the peak and then decline as the server size m (or server speed s) increases further, except for the special curve of $\overline{r} = 0.5$ in Fig. 5a where the proposed strategy can quickly converge to the maximum profit since the expected number of instructions for service requests is too small. The CSP's profit increases when server size m (or the server speed s) is small until m (or s) increases to a threshold, the CSP's profit will decrease. This is because simply increasing the server size or the server speed will possibly reach the point where the processing capacity offered by the CSP exceeds the maximum processing requirements of cloud service requests. At that moment, continuously increasing the server size or the server speed will bring wasted energy consumption and infrastructure resources, thereby reducing the profit of the CSP. Besides, from these two figures, we also observe that the larger the expected number of instructions $\overline{r}$ of cloud service requests, the more profit the CSP can achieve. As can be seen from Fig. 5c, when the server size m increases, our strategy lowers server speed s to reduce energy costs for improving profits as much as possible. This is consistent with the observation in Fig. 4c.

6.3 Performance Comparison

To further evaluate the performance of the proposed multiserver configuration scheme for profit maximization, in this set of experiments, we compare our scheme with a SOTA approach TSPM [5] and a heuristic optimization algorithm SA [4] which is to solve the same problem as ALMI. The two comparison targets, TSPM and SA, are briefly described as follows.

- The multiserver system used in TSPM [5] is modeled as an M/M/m queuing system. It derives the probability density function of the waiting delay for the newly arrived service request and uses a numerical method to calculate the optimal configuration of the multiserver system as well as the maximum service provider profit. The major difference between our ALMI and TSPM is that the latter considers neither the heterogeneity of service requests nor the occurrence of soft errors. In the comparison experiments, we first use TSPM to find the optimal multiserver configuration under the same experimental settings (seen in Table 2) as ALMI and then calculate the maximum profit considering the heterogeneity of service requests and soft errors simultaneously (i.e., substitute the derived m and s by TSPM into Eq. (11)).
- SA [4] is a discrete simulated annealing-based method and also a classic heuristic algorithm. The process of solving the optimal solution by this algorithm is similar to physical annealing and is a probabilistic technique. Since there are no exist SA-based solutions for my optimization problem, we implemented the SA-based approach to solve our proposed deadline miss rate and soft error reliability aware multiserver configuration problem (i.e., Eqs. (25), (26), (27), (28), (29), and (30)). Then, we compare our proposed ALMI scheme with the SA-based approach in terms of profit as well as the mulitserver configuration.

Tables 3 and 4 compare our proposed ALMI with TSPM and SA under the settings of $\overline{r} = 1$ and $\lambda = 20$, respectively. Besides, we set the server speed s range from [0.5,2.5] and the server size m range from [5,45]. The values of parameters λ_0, d_0, a, v, ξ, P^*, δ, $S_{\min}$ and $S_{\max}$ are consistent with those in Table 2.

It can be seen from the tables that ALMI outperforms TSPM and SA in terms of the CSP's profit. Table 3 shows that ALMI can gain 3.62% on average and up to 5.56% higher profit than that of TSPM. In addition, ALMI can achieve 1.39% on average and up to 3.45% higher CSP's profit than that of SA. Although ALMI and SA are solving the same problem using different optimization methods, their results are relatively close on average, this boost is meaningful if in

TABLE 3
Compare the Profit of ALMI, TSPM, and SA Under the Same λ

Application	TSPM			SA			ALMI			Improvement	
λ	m	s	Profit	m	s	Profit	m	s	Profit	$\frac{\text{Profit}_{\text{ALMI}}-\text{Profit}_{\text{TSPM}}}{\text{Profit}_{\text{TSPM}}}$	$\frac{\text{Profit}_{\text{ALMI}}-\text{Profit}_{\text{SA}}}{\text{Profit}_{\text{SA}}}$
5	6.05	1.05	75.24	10	1.34	76.77	7	1.19	79.42	5.56%	3.45%
10	11.67	1.01	156.99	11	1.35	161.65	13	1.12	163.83	4.36%	1.35%
15	17.23	0.99	240.26	16	1.25	247.60	17	1.11	249.44	3.82%	0.74%
20	22.76	0.98	323.60	17	1.53	323.27	24	1.03	335.36	3.63%	3.74%
25	28.27	0.98	409.35	27	1.15	419.52	27	1.13	420.05	2.61%	0.13%
30	33.78	0.97	492.48	33	1.10	506.02	34	1.05	506.77	2.90%	0.15%
35	39.27	0.97	578.54	38	1.10	591.75	40	1.03	592.79	2.46%	0.18%

TABLE 4
Compare the Profit of ALMI, TSPM, and SA Under the Same $\overline{r}$

Application	TSPM			SA			ALMI			Improvement	
$\overline{r}$	m	s	Profit	m	s	Profit	m	s	Profit	$\frac{\text{Profit}_{\text{ALMI}}-\text{Profit}_{\text{TSPM}}}{\text{Profit}_{\text{TSPM}}}$	$\frac{\text{Profit}_{\text{ALMI}}-\text{Profit}_{\text{SA}}}{\text{Profit}_{\text{SA}}}$
0.5	10.84	1.0	158.93	11	1.19	166.15	12	1.11	166.46	4.74%	0.19%
0.75	16.97	0.95	237.01	12	1.49	242.38	16	1.21	249.5	5.27%	2.94%
1.0	22.76	0.98	323.60	17	1.53	323.27	24	1.03	335.36	3.63%	3.74%
1.25	28.49	0.95	403.45	9	1.7	394.6	27	1.18	417.51	3.48%	5.81%
1.5	32.45	1.0	464.86	22	1.7	473.78	32	1.19	501.22	7.82%	5.79%
1.75	29.47	0.95	522.99	24	1.7	549.53	37	1.19	585.34	11.92%	6.52%
2.0	44.50	0.95	563.16	42	2.0	573.88	28	2.0	595.14	5.68%	3.70%

a long time cloud service environment for the CSP. Table 4 the profit comparison between TSPM, SA, and ALMI when the arrival rate λ of cloud service requests is given. The results again validate that our ALMI is superior to TSPM, with 6.08% on average and up to 11.92% more profit. Compared with SA, the improvement in profit achieved by our ALMI is 4.1% on average and is up to 6.52%. Besides, from the two tables, we can seen the expected number of instructions $\overline{r}$ on profit exceeds the effect of the number of service requests λ on profit. ALMI obtains a higher profit than TSPM because ALMI takes into account the impact of deadline miss rate and soft error reliability on the CSP's profit when optimizing the multiserver configuration.

7 Conclusion

This paper studies the problem of finding a multiserver configuration that maximizes the profit of CSPs with the consideration of deadline miss rate and soft error reliability simultaneously. Since no existing work formulates soft error reliability for average-based scheduling to optimize the multiserver configuration, this paper introduces a new analytical approach to derive the deadline miss rate of cloud service requests and the soft error reliability for average-based scheduling. Based on our deadline miss rate and soft error reliability models, we formulate the profit maximization problem as a constrained optimization problem. To solve this problem, we convert it into an unconstrained optimization problem by constructing an augmented Lagrangian function and propose a method based on an augmented Lagrangian multiplier, ALMI, to iteratively improve the quality of the multiserver configuration solution until the optimal solution is derived.

To validate the effectiveness of our multiserver configuration approach, we conduct a series of experiments. We analyze the trend of the profit gained by the proposed method in different configurations as well as service requests. We also compare our ALMI with two SOTA methods. Experimental results show that ALMI can increase profit by up to 11.92% and 6.52% compared with the two SOTA methods, TSPM and SA, in different scenarios, respectively.

References

[1] M. Jing, K. Li, A. Ouyang, and K. Li, "A profit maximization scheme with guaranteed quality of service in cloud computing," *IEEE Trans. Comput.*, vol. 64, no. 120, pp. 3064–3078, Nov. 2015.

[2] K. Li, J. Mei, and K. Li, "A fund-constrained investment scheme for profit maximization in cloud computing," *IEEE Trans. Serv. Comput.*, vol. 11, no. 6, pp. 893–907, Nov. 2018.

[3] P. Cong *et al.*, "Developing user perceived value based pricing models for cloud markets," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 12, pp. 2742–2756, Dec. 2018.

[4] T. Wang, J. Zhou, G. Zhang, T. Wei, and S. Hu, "Customer perceived value- and risk-aware multiserver configuration for profit maximization," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 5, pp. 1074–1088, May 2020.

[5] J. Cao, K. Huang, K. Li, and A. Y. Zomaya, "Optimal multiserver configuration for profit maximization in cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1087–1096, Jun. 2013.

[6] T. Thanakornworakij, R. Nassar, C. Leangsuksun, and M. Paun, "An economic model for maximizing profit of a cloud service provider," in *Proc. 7th Int. Conf. Availability, Rel. Secur.*, 2012, pp. 274–279.

[7] Amazon Web Services, 2019. [Online]. Available: http://aws.amazon.com/

[8] Google App Engine, 2019. [Online]. Available: https://appengine.google.com/

[9] Microsoft Azure, 2019. [Online]. Available: https://azure.microsoft.com/zh-cn/pricing/details/virtual-machine-scale-sets

[10] H. Aydin, V. Devadas, and D. Zhu, "System-level energy management for periodic real-time tasks," in *Proc. IEEE Int. Real-Time Syst. Symp.*, 2006, pp. 313–322.

[11] W. Zhu and B. D. Fleisch, "Performance evaluation of soft real-time scheduling for multicomputer cluster," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2000, pp. 610–617.

[12] F. Wang and V. D. Agrawal, "Soft error considerations for computer web servers," in *Proc. Int. Southeastern Symp. Syst. Theory*, 2010, pp. 269–274.

[13] L. He, S. A. Jarvis, D. P. Spooner, H. Jiang, D. N. Dillenberger, and G. R. Nudd, "Allocating non-real-time and soft real-time jobs in multiclusters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 2, pp. 99–112, Feb. 2006.

[14] M. Sedaghat, E. Wadbro, J. Wilkes, S. D. Luna, O. Seleznjev, and E. Elmroth, "Diehard: Reliable scheduling to survive correlated failures in cloud data centers," in *Proc. Int. Symp. Cluster, Cloud Grid Comput.*, 2016, pp. 52–59.
[15] K. Cao, J. Zhou, G. Xu, T. Wei, and S. Hu, "Exploring renewable-adaptive computation offloading for hierarchical QoS optimization in fog computing," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 39, no. 10, pp. 2095–2108, Oct. 2020.
[16] B. Kao and H. Garcia-Molina, "Scheduling soft real-time jobs over dual non-real-time servers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 7, no. 1, pp. 56–68, Jan. 1996.
[17] D. Cheng, X. Zhou, P. Lama, M. Ji, and C. Jiang, "Energy efficiency aware task assignment with DVFs in heterogeneous hadoop clusters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 1, pp. 70–82, Jan. 2018.
[18] Y. Hu, J. Wong, G. Iszlai, and M. Litoiu, "Resource provisioning for cloud computing," in *Proc. Int. Conf. Center Adv. Stud. Collaborative Res.*, 2009, pp. 101–111.
[19] M. Mazzucco, D. Dyachuk, and R. Deters, "Maximizing cloud providers' revenues via energy aware allocation policies," in *Proc. IEEE Int. Conf. Cloud Comput.*, 2010, pp. 131–138.
[20] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Gener. Comput. Syst.*, vol. 28, no. 5, pp. 755–768, 2012.
[21] J. Cao, K. Li, and I. Stojmenovic, "Optimal power allocation and load distribution for multiple heterogeneous multicore server processors across clouds and data centers," *IEEE Trans. Comput.*, vol. 63, no. 1, pp. 45–58, Jan. 2014.
[22] Y. Gao, S. K. Gupta, Y. Wang, and M. Pedram, "An energy-aware fault tolerant scheduling framework for soft error resilient cloud computing systems," in *Proc. Des. Automat. Test Europe Conf. Exhib.*, 2014, pp. 1–6.
[23] R. Garg and A. K. Singh, "Fault tolerance in grid computing: State of the art and open issues," *J. Comput. Sci. Eng. Surv.*, vol. 2, no. 1, pp. 88–97, 2011.
[24] P. Cong, G. Xu, T. Wei, and K. Li, "A survey of profit optimization techniques for cloud providers," *ACM Comput. Surv.*, vol. 53, no. 2, pp. 1–35, 2020.
[25] K. Li, "Optimal load distribution for multiple heterogeneous blade servers in a cloud computing environment," *J. Grid Comput.*, vol. 11, no. 1, pp. 27–46, 2013.
[26] B. N. Chun and D. E. Culler, "User-centric performance analysis of market-based cluster batch schedulers," in *Proc. IEEE/ACM Int. Symp. CLUSTER Comput. Grid*, 2002, p. 30–30.
[27] K. Li, "Optimal configuration of a multicore server processor for managing the power and performance tradeoff," *J. Supercomput.*, vol. 61, no. 1, pp. 189–214, 2012.
[28] X. Zheng and Y. Cai, "Optimal server provisioning and frequency adjustment in server clusters," in *Proc. Int. Conf. Parallel Process.*, 2010, pp. 504–511.
[29] L. Kleinrock, *Queueing Systems: Computer Applications*, vol. 2. New York, NY, USA: Wiley, 1976.
[30] R. Ge, X. Feng, and K. W. Cameron, "Performance-constrained distributed DVS scheduling for scientific applications on power-aware clusters," in *Proc. Int. Conf. Supercomput.*, 2005, pp. 34–34.
[31] T. Guerout, T. Monteil, G. D. Costa, R. N. Calheiros, R. Buyya, and M. Alexandru, "Energyaware simulation with DVFS," *Simul. Modelling Pract. Theory*, vol. 39, pp. 76–91, 2013.
[32] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low power CMOS digital design," *IEICE Trans. Electron.*, vol. 75, no. 4, pp. 371–382, 1992.
[33] K. Li, C. Liu, K. Li, and A. Y. Zomaya, "A framework of price bidding configurations for resource usage in cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 8, pp. 2168–2181, Aug. 2016.
[34] L. He, S. A. Jarvis, D. P. Spooner, X. Chen, and G. R. Nudd, "Dynamic scheduling of parallel jobs with QoS demands in multi-clusters and grids," in *Proc. IEEE/ACM Int. Conf. Grid Comput.*, pp. 402–409, 2004.
[35] J. Zhou, K. Cao, X. Zhou, M. Chen, T. Wei, and S. Hu, "Throughput-conscious energy allocation and reliability-aware task assignment for renewable powered in-situ server systems," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 41, no. 3, pp. 516–529, Mar. 2022.
[36] J. Zhou, M. Zhang, J. Sun, T. Wang, X. Zhou, and S. Hu, "DRHEFT: Deadline-constrained reliability-aware HEFT algorithm for real-time heterogeneous MPSoC systems," *IEEE Trans. Rel.*, vol. 71, no. 1, pp. 178–189, Mar. 2022.
[37] J. Zhou, L. Li, A. Vajdi, X. Zhou and Z. Wu, "Temperature-constrained reliability optimization of industrial cyber-physical systems using machine learning and feedback contro," *IEEE Trans. Automat. Sci. Eng.*, to be published, doi: 10.1109/TASE.2021.3062408.
[38] G. Aupy, A. Benoit, and Y. Robert, "Energy-aware scheduling under reliability and makespan constraints," in *Proc. Int. Conf. High Perform. Comput.*, 2012, pp. 1–10.
[39] P. Cong, G. Xu, J. Zhou, M. Chen, T. Wei, and M. Qiu, "Personality- and value-aware scheduling of user requests in cloud for profit maximization," *IEEE Trans. Cloud Comput.*, to be published, doi: 10.1109/TCC.2020.3000792.
[40] M. A. Haque, H. Aydin, and D. Zhu, "On reliability management of energy-aware real-time systems through task replication," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 3, pp. 813–825, Mar. 2017.
[41] Gradient Descent, 2022. [Online]. Available: https://en.wikipedia.org/wiki/Gradient_descent
[42] K. V. Vishwanath and N. Nagappan, "Characterizing cloud computing hardware reliability," in *Proc. ACM Symp. Cloud Comput.*, 2010, pp. 193–204.
[43] AMD EPYC7742, 2022. [Online]. Available: https://www.amd.com/zh-hans/products/cpu/amd-epyc-7742
[44] KNL Intel, 2022. [Online]. Available: https://en.wikipedia.org/wiki/Xeon_Phi
[45] NVIDIA V100, 2022. [Online]. Available: https://en.wikipedia.org/wiki/List_of_Nvidia_graphics_processing_units
[46] Intel Platinum 8376H, 2022. [Online]. Available: https://www.intel.cn/content/www/cn/zh/products/sku/204096/intel-xeon-platinum-8376h-processor-38–5m-cache-2-60-ghz/specifications.html
[47] Huawei Cloud, 2022. [Online]. Available: https://www.huaweicloud.com/product/ecs.html

Tian Wang (Student Member, IEEE) is currently working toward the PhD degree with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China. He was a visiting scholar with the University of Notre Dame, Indiana, from 2019 to 2020. His current research interests include the areas of cloud computing, edge computing, and cybersecurity. He has published 14 papers and served as a TPC member for some international conferences. He is the reviewer of numerous journals.

Junlong Zhou (Member, IEEE) received the PhD degree in computer science and technology from East China Normal University, Shanghai, China, in 2017. He was a visiting scholar with the University of Notre Dame, Notre Dame, IN, USA, during 2014-2015. He is currently an associate professor with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China. His research interests include embedded systems, cloud-edge-IoT, and cyber-physical systems. He has been an associate editor for the *Journal of Circuits, Systems, and Computers* and the *IET Cyber-Physical Systems: Theory & Applications*, a subject area editor for the *Journal of Systems Architecture*, and a guest editor for 6 international Journals such as *ACM Transactions on Cyber-Physical Systems*. He received the Best Paper Award from IEEE iThings 2020.

Liying Li (Student Member, IEEE) received the BS degree from the School of Computer Science and Technology, East China Normal University, Shanghai, China, in 2017 and currently working toward the PhD degree with the School of Computer Science and Technology, East China Normal University, Shanghai, China. Her current research interests include data analysis, neural network acceleration, and edge computing. She served as a TPC member for some international conferences and a reviewer of some international journals.

Gongxuan Zhang (Senior Member, IEEE) received the BS degree in electronic computer from Tianjin University in 1983 and the MS and PhD degrees in computer application from the Nanjing University of Science and Technology in 1991 and 2005, respectively. He was a senior visiting Sscholar with the Royal Melbourne Institute of Technology from 2001.9 to 2002.3 and a guest professor with the University of Notre Dame from 2017.7 to 2017.10. Since 1991, he has been with the Nanjing University of Science and Technology, where he is currently a professor with the School of Computer Science and Engineering. His current research interests include web services and distributed system, multicore and parallel processing and distributed computing, and trusted computing system. He has served on the program committees in a couple of conferences. He has more than 100 publications and has led or participated in 40 research projects supported by the National Science Foundation of China, and the Provincial Science Foundation of Jiangsu.

Keqin Li (Fellow, IEEE) is currently a SUNY distinguished professor of computer science with the State University of New York. He is also a national distinguished professor with Hunan University, China. His current research interests include cloud computing, fog computing, and mobile edge computing, energy-efficient computing and communication, embedded systems and cyber-physical systems, heterogeneous computing systems, Big Data computing, high-performance computing, CPU-GPU hybrid, and cooperative computing, computer architectures, and systems, computer networking, machine learning, intelligent and soft computing. He has authored or coauthored more than 840 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He holds over 70 patents announced or authorized by the Chinese National Intellectual Property Administration. He is among the world's top 5 most influential scientists in parallel and distributed computing based on a composite indicator of Scopus citation database. He has chaired many international conferences. He is currently an associate editor for *ACM Computing Surveys* and the *CCF Transactions on High-Performance Computing*. He has served on the editorial boards for *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Services Computing*, and *IEEE Transactions on Sustainable Computing*.

Xiaobo Sharon Hu (Fellow, IEEE) received the BS degree from Tianjin University, Tianjin, China, in 1982, the MS degree from the Polytechnic Institute of New York, New York, NY, USA, in 1984, and the PhD degree from Purdue University, West Lafayette, IN, USA in 1989. She is professor with the Department of Computer Science and Engineering, University of Notre Dame. Her research interests include energy/reliability-aware system design, circuit and architecture design with emerging technologies, real-time embedded systems and hardware-software co-design. She has published more than 400 papers in these areas. She is the editor-in-chief for *ACM Transactions on Design Automation of Electronic Systems*, and also served as associate editor for *IEEE Transactions on CAD*, *IEEE Transactions on VLSI*, *ACM Transactions on Embedded Computing*, etc. She served as the general chair and technical program chair for Design Automation Conference (DAC), IEEE Real-time Systems Symposium, etc. She received the NSF CAREER Award in 1997, and the Best Paper Award from Design Automation Conference in 2001, ACM/IEEE International Symposium on Low Power Electronics and Design in 2018, etc. She was a distinguished lecturer for *IEEE Council of Electronic Design Automation*. She is a fellow of the ACM.