Cost-Effective Web Application Replication and Deployment in Multi-Cloud Environment

Tao Shi, Hui Ma, Gang Chen, and Sven Hartmann

Abstract—Multi-cloud is becoming a popular cloud ecosystem because it allows enterprise users to share the workload across multiple cloud service providers to achieve high-quality services with lower operation cost and higher application resilience. In multi-cloud, cloud services are widely distributed at different locations with differentiated prices. Therefore, Web application providers face the challenge to select proper cloud services for application replication and deployment with the aim of minimizing the deployment cost. Meanwhile, the deployed application replicas must satisfy the constraint on request response time to maintain the quality of user experience. To meet the two major requirements, this article studies a new problem of Web application replication and deployment in multi-cloud (WARDMC) that jointly considers both the cost minimization and constraints on average response time, including particularly request processing time and network latency. To address the problem, we develop a new approach named MCApp. MCApp combines iterative mixed integer linear programming with domain-tailored large neighborhood search to optimize both application replicas deployment and user requests dispatching. Extensive experiments using the real-world datasets demonstrate that MCApp significantly outperforms several recently proposed approaches.

Index Terms—Multi-cloud, Web application deployment, service replication, cost optimization, mixed integer linear programming, large neighborhood search.

1 Introduction

According to Statista[1], 48% of enterprises have already moved workloads to the public clouds in 2020. Through diverse virtualization technologies, e.g., server virtualization and storage virtualization, enterprises can access on-demand cloud resources with minimal management efforts. To achieve lower operation cost and higher application resilience, enterprises usually utilize services and resources from multiple cloud providers [1], [2].

In recent years, enterprises are interested in embracing a *multi-cloud* environment [3]. The deployment and management of multiple clouds are usually abstracted by a third-party, i.e., *broker*, which can provide a single entry point to multiple clouds [4], [5]. Supported by the broker and the multi-cloud platform, enterprise applications can be easily deployed at multiple available locations to bring services close to users, which is essential to maintain high quality of user experience [6], [7].

In practice, enterprise applications are often deployed to serve global users [3]. Many enterprise applications, such as the meteorological forecast application, must serve diverse user demands with a suite of *Web applications*, which interact with users through a front-end programmed and browser-based language [8]. To evaluate accurately the performance of the applications with widely distributed user requests, we must consider the *average response time* for all user requests [9], [10], including both the *request processing time* and the *network latency* [3]. This requires us to adopt a queuing model to precisely capture the dynamics involved in the processing of ongoing application requests [11].

In public clouds, three main deployment paradigms, including Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS) [12], are applicable to Web applications. Enterprise applications providers often prefer IaaS in practice in order to achieve high level of control and flexible management goals [8], [13], [14]. In this work, we focus on the IaaS paradigm, i.e., deploying Web applications on a large group of virtual machines (VMs), from the perspective of application providers. Meanwhile service replication is gaining increasing popularity for cloud-hosted application deployment [7], [15]. Therefore, we take Web application *replication* into account.

It is widely known that leading cloud providers usually provide cloud resources, e.g., cloud storage, with region-tailored pricing[2]. Several studies take advantage of the storage pricing differences to minimize the deployment cost of data replicas in clouds [9], [10]. Different from these works, Web application deployment focuses on request processing rather than data delivery, which incurs the additional VM rental cost [3]. Because enterprise application providers usually care more about cost minimization while ensuring acceptable performance of applications [9], [10], we consider the problem of Web application replication and deployment in multi-cloud (WARDMC) as a constrained optimization problem to minimize the total deployment cost subject to the constraints on the response time.

It is challenging to achieve cost minimization without

- *Tao Shi, Hui Ma, and Gang Chen are with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand.*
  E-mail: {tao.shi, hui.ma, aaron.chen}@ecs.vuw.ac.nz
- *Sven Hartmann is with the Department of Informatics, Clausthal University of Technology, Germany.*
  E-mail: sven.hartmann@tu-clausthal.de

Manuscript received x x, x; revised x x, x.

1. https://www.statista.com

2. https://aws.amazon.com/s3/pricing/

significantly affecting the performance of cloud-hosted applications [16], [17]. Concretely, two key issues must be resolved: (1) How should the types and locations of VMs in multi-cloud be selected to serve all user requests so that the total deployment cost is minimized? (2) How should user requests be distributed among application replicas such that the average response time can meet application providers' requirements?

For cloud-hosted Web applications, the request processing time is nonlinearly related to the capacity of deployed VMs and users' demands [18]. Therefore, we apply nonlinear programming [19] to define the WARDMC problem with two optimization decisions: (1) the *replica deployment plan* that determines the deployment of all application replicas on specific VM types in specific multi-cloud data centres, and (2) the *request dispatch plan* that determines how to dispatch user requests among all application replicas. In the remaining of this paper, we use *WARDMC solutions* to refer to the combinations of the two matching plans.

The nonlinear nature of the WARDMC problem renders several integer linear programming (ILP)-based approaches in [20], [21] inapplicable. Furthermore, recently developed meta-heuristic approaches such as GA-ARP [7] and HMOHM [8] may not be able to solve the WARDMC problem effectively. For example, GA-ARP applies a heuristic to dispatch all application requests to the closest application replicas, which may result in deployed VMs under-utilized. In this paper, we propose a new approach for Multi-Cloud Application deployment, named MCApp, to simultaneously optimize the replica deployment plan and the request dispatch plan. MCApp first transfers the WARDMC problem to a series of mixed integer linear programming (MILP) problems by bounding the utilization rates of all deployed VMs. Then a two-stage optimization framework is designed to effectively optimize WARDMC solutions. An overview of MCApp is shown in Fig. 1.

Besides the upper bound on VMs' utilization rate, we must also consider the upper bound on average response time that helps to find WARDMC solutions with lower total deployment cost. Accordingly, we propose a MILP-based algorithm to obtain a high-quality base solution. Following a novel *double iterative mechanism*, the algorithm adaptively updates the upper bounds on both the VMs' utilization rate and average response time to improve the performance of the base solution. Moreover, we theoretically analyse the performance of the base solution in terms of worst-case ratio.

Note that setting the upper bound on the VMs' utilization rate helps to reduce the solution space, since it rules out the chance of further reducing the total deployment cost by increasing the utilization rates of some application replicas. To address this limitation, we develop a large neighborhood search (LNS)-based algorithm to further improve the base solution. To build an effective LNS process, a new *destroy heuristic* is proposed to remove multiple application replicas from the replica deployment plan. Then a problem-specific *repair heuristic* is designed to effectively deploy new application replicas. Besides, we propose a *delay-oriented heuristic* to dispatch user requests in order to achieve high performance based on the current replica deployment plan. The main contributions of this paper are summarized as follows:

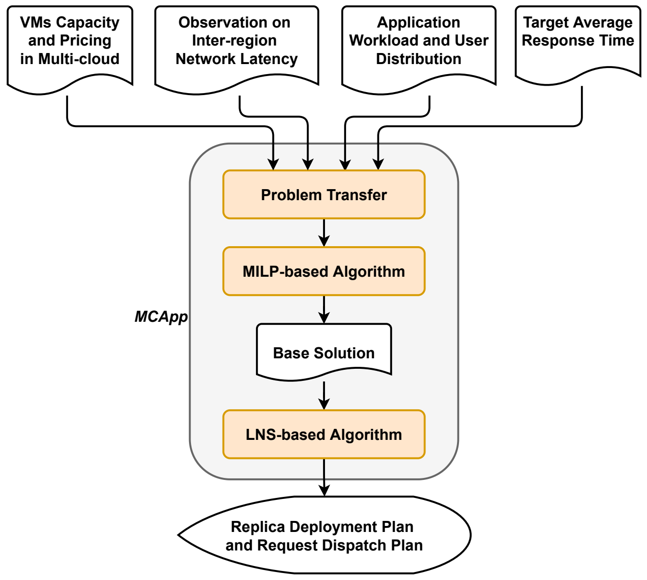

VMs Capacity and Pricing in Multi-cloud
Observation on Inter-region Network Latency
Application Workload and User Distribution
Target Average Response Time
MCApp
Problem Transfer
MILP-based Algorithm
Base Solution
LNS-based Algorithm
Replica Deployment Plan and Request Dispatch Plan

Fig. 1. Overview of MCApp with inputs and outputs.

(1) We formalize the WARDMC problem as a constrained optimization problem with the goal to minimize the total deployment cost subject to constraints on average response time. To the best of our knowledge, this is the first study in the literature on the Web application deployment problem with realistic consideration of both cost effectiveness and application replication in multi-cloud environment.

(2) We propose an approach, named MCApp, to solve the WARDMC problem. MCApp first introduces a bound on the utilization rates of the deployed VMs to linearize the WARDMC problem. Then, it creates a hybrid optimization process that combines an iterative MILP-based algorithm and a domain-tailored LNS-based algorithm to simultaneously optimize the replica deployment plan and the request dispatch plan.

(3) To evaluate MCApp, we collect the information regarding VM capacity and pricing offered by the global top three cloud providers, i.e. Amazon, Microsoft, and Alibaba. Based on the collected information, extensive experiments have been conducted to deploy Web application suites with different diversities. Our approach is compared to several state-of-the-art approaches for cloud-hosted Web application deployment, including GA-ARP [7] and HMOHM [8]. The experimental results indicate that MCApp significantly outperforms the existing approaches, achieving up to 35% savings in terms of the total deployment cost.

The remainder of this paper is organised as follows. Section 2 discusses the related work about application deployment and replication as well as data placement and replication in clouds. Section 3 defines the WARDMC problem, including the problem background. Section 4 presents the details of the MCApp approach. Section 5 describes the design of experiments and analyses the evaluation results. Section 6 concludes the paper.

2 Related Work

This section introduces the related works about application deployment and replication in clouds and existing approaches for cloud-hosted data placement and replication problems. The main challenges that need to be addressed in this paper are also highlighted.

2.1 Application Deployment and Replication in Clouds

In recent years, cloud computing is becoming a booming paradigm for flexible and scalable delivery and deployment of enterprise-scale applications [3]. Several research works have studied the Web application deployment problem in clouds [3], [8], [18]. For example, Menzel et al. [8] presented CloudGenius to handle the deployment of Web applications across multiple cloud data centres spanning over geographically distributed network boundaries. A hybrid approach, i.e., HMOHM, combining the analytic hierarchy process (AHP) and the genetic algorithm (GA), was proposed to satisfy conflicting selection criteria, e.g., the lowest latency and cost of services. In the AHP, different selection criteria are aggregated by calculating the sum of the weighted evaluation values. From the perspective of enterprise application providers, application deployment often emphasizes on minimizing monetary cost subject to performance requirements [9]. Therefore, we formulate the WARDMC problem as a cost optimization problem in this paper.

Because Web applications usually involve a potentially large number of user requests and different applications must cater for varied user distributions, we model the processing of ongoing application requests through a queuing model. Although queuing theory has been well studied to analyse the performance of cloud-based systems [22], [23], existing research works on cloud-based application deployment did not pay full attention to its role in performance evaluation based on the application response time [18], including the request processing time and the network latency between users and cloud data centres.

In [3], we studied the multi-cloud service deployment for composite applications, considering the budget impact. Particularly, a hybrid GA-based approach, i.e., H-GA, was proposed to select VMs in multi-cloud for each constituent service with exactly one instance. Note that even under sufficient budget, H-GA may fail to bring the average response time of applications within 200 ms (238-245 ms in our experiments detailed in Subsection 5.5). For many applications, a response time beyond 200 ms will deteriorate the user experience significantly [9].

In practice, deploying application replicas as close to users as possible is widely exercised to maintain low response time [6]. The application replication has attracted significant attention recently [15], [24]. For example, several works have been proposed to replicate applications in multiple locations to ensure the availability of the provisioned services [25], [26], [27]. Some works focused on reducing energy consumption for cloud providers by adjusting the umber of replicas according to the workload, e.g., [28]. From the perspective of application providers, some works also studied the problem of minimizing the deployment cost of replicas [29], [30], [31]. However, these works did not consider the average response time, which seriously affects the quality of experience (QoE) of applications [9], [10].

Considering application replication, both the deployment of application replicas and the dispatching of user requests among application replicas affect the average response time of applications. In this paper, we simultaneously optimize the replica deployment plan and the request dispatch plan. After linearizing the WARDMC problem, we propose an iterative MILP-based algorithm to obtain a high-quality base solution.

As a combination of local search and constraint programming, LNS was proposed by Shaw [32] to take the advantages of both *exploration* and *propagation*. LNS makes moves as local search, but uses a tree-based search with constraint propagation to evaluate the cost and legality of the moves. In [33], LNS has been used to solve the multi-cloud service brokering problem due to its flexibility in designing problem-specific destroy and repair heuristics. In this paper, we develop a domain-tailored LNS-based algorithm to further improve the base solutions obtained by our MILP-based algorithm. The LNS-based algorithm includes newly designed destroy and repair heuristics to minimize the total deployment cost subject to constraints on average response time.

2.2 Data Placement and Replication Problem in Clouds

The cloud-hosted data placement and replication problems have received much attention in the recent literature. ILP has been considered as the dominant method to model these problems [9], [20]. For example, Pyramid was proposed to maximize both the utility- and locality-awareness of replicas for P2P cloud storage systems in [20]. ILP was implemented to find the placement of replicas. To serve the demands on videos, the problem of finding the optimal content deployment and request dispatch strategy was formulated as MILP in [9]. The efficient solutions were achieved by using dual decomposition [34] and linear programming techniques. These ILP problems consistently treat the network latency as the optimization objective. In this paper, we consider the response time of user requests, including both the network latency and the request processing time. For the cloud-hosted Web applications under high workloads, the request processing time is nonlinearly related to the capacity of deployed VMs and users' demands [18]. The nonlinear nature renders these ILP-based approaches inapplicable for the WARDMC problem.

Many meta-heuristic algorithms also have been used for the data placement and replication problem in clouds [35], [36]. In [35], two GA-based approaches were proposed to replicate data objects over multiple sites to avert undesired long delays experienced by end-users. To optimize social media data placement and replication in cloud data centres, a GA-based approach was presented to minimise monetary cost while satisfying latency requirements for all users in [36]. However, these approaches focus on the location selection for data replicas and do not explicitly dispatch widely distributed user requests among all application replicas, which is a critical component of the WARDMC problem.

Comparing with the cloud-hosted data placement and replication problems, the WARDMC problem is more challenging due to its nonlinear nature. Moreover, we need to

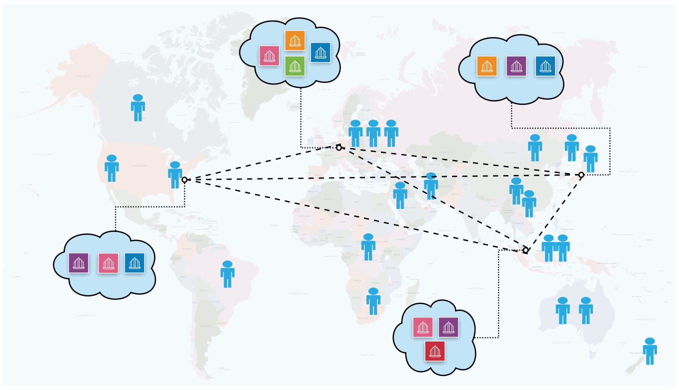


Fig. 2. Web application deployment in multi-cloud based on [9].

design effective mechanisms to optimize the request dispatch plan based on the corresponding replica deployment plan so that the total deployment cost is minimized while satisfying the constraint on average response time.

3 Problem Description

In this section, we first outline the background of multi-cloud environment and Web applications in Subsection 3.1, then formulate the WARDMC problem in Subsection 3.2.

3.1 Multi-cloud and Web Applications

The multi-cloud infrastructure consists of multiple disparate data centres distributed in different geographical locations (see Fig. 2). These data centres are owned by multiple cloud providers. Each data centre provides a large collection of virtualized servers, e.g., VMs. Different data centres are connected over a WAN [9] (dotted lines in Fig. 2). We investigate the scenario of IaaS in this work following [8]. It is widely known that leading cloud providers usually provide VMs with varied pricing subject to regions. There are two major benefits to deploy enterprise applications in multi-cloud. On the one hand, to achieve cost saving, application providers can dynamically distribute workload among different cloud providers to deal with the changes of policies and pricing [4]. On the other hand, multi-cloud can avoid potential service unavailability of the single cloud provider [2].

To satisfy the diverse needs of users, an enterprise application usually consists of a suite of Web applications (coloured squares in Fig. 2). These Web applications can be deployed onto a set of VMs in multi-cloud with the help of a broker. One concrete example is MetService, i.e., the meteorological service of New Zealand[3], which includes eight Web applications for different user groups, i.e., National Forecast, Towns and Cities Forecast, Rural Forecast, Marine Forecast, Mountains and Parks Forecast, Maps and Radar Information, Warnings Information, and Public Information. More industry examples of Web applications deployed in multi-cloud can be found in VMware use cases, e.g., Mercedes-Benz.io [37] and GSL General Hospital and Medical College [38].

3. https://about.metservice.com/our-company/about-us/

TABLE 1
Mathematical notations

Notation	Definition
A	Suite of Web applications
U	Set of regions that the users of all applications span
C	Set of multi-cloud data centres
V	Set of VM types
a_c	Replica of application a in data centre c
$\gamma_{a,u}$	Request rate of application a from user region u
$\delta_{a,v}$	Amount of requests for application a processable by VM type v per time unit
$rc_{c,v}$	Rental cost of VM type v in data centre c
$dt_{u,c}$	Round-trip delay between user region u and data centre c (periodically observed constant)
$\mathcal{R}$	Maximum average response time per request
$x_{a,u,c}$	Independent variables: percentage of application a request rate from user region u to be served in data centre c
$y_{a,c,v}$	Independent variables: to rent an instance of VM type v for application replica a_c (1) or not (0)
$\lambda_{a,c}$	Dependent variables: workload of application replica a_c
$\mu_{a,c}$	Dependent variables: capacity of application replica a_c
$pt_{a,c}$	Dependent variables: average request processing time of application replica a_c
$u_{a,c}$	Dependent variables: utilization rate of the VM when deploying application replica a_c
n	Dependent variable: total number of application replicas

3.2 Formalization of WARDMC Problem

The aim of the WARDMC problem is to select VMs from multi-cloud for Web applications to minimize the total deployment cost (TDC) subject to the performance requirement on average response time (ART). The response time is measured from the moment a user makes an application request to the moment when this user receives the corresponding response, consisting of both request processing time and round-trip delay (RTD) between the user and one application replica. The key notations to be used for problem definition are listed in TABLE 1.

An application provider delivers a suite of Web applications A for its users distributed in global user regions U. During an epoch, e.g., an hour [39], the request rate $\gamma_{a,u}$ denotes the request frequency for application a $(a \in A)$ from user region u $(u \in U)$.

We consider a set of VM types V and a set of multi-cloud data centre C. If VM type v $(v \in V)$ is available in data centre c $(c \in C)$, we use $rc_{c,v}$ to denote its rental cost per time unit and $\delta_{a,v}$ to measure the maximum amount of requests for application a processable by v per time unit.

Let $x_{a,u,c} \in [0,1]$ denote the percentage of requests from user region u for application a that will be dispatched to the data centre c. We assume that all requests for application a dispatched to data centre c will be served by the only replica of a in c. In the remaining of this paper, we use a_c to refer to the replica of application a in data centre c. The *workload* of a_c can be measured by combining relevant request rates from all user regions:

$$\lambda_{a,c} := \sum_{u \in U} \gamma_{a,u} x_{a,u,c}. \tag{1}$$

Typically the execution platform of a Web application includes the Web server, application server, and database server [40]. Refer to [8], we assume that each application replica is bundled with its own execution platform upon deployed to one instance of the particular VM type v in multi-cloud. For higher flexibility and agility, the application replica with multiple interdependent functionalities, i.e., constituent services, can be deployed as microservices in the same VM instance [41]. We further assume that application replicas apply the eventual consistency model when they access database servers, since the consistency model is commonly used by widespread distributed systems with large numbers of user requests [42], [43]. Let binary variable $y_{a,c,v}$ indicate whether an instance of v is rented for application replica a_c, the *capacity* of a_c depends on the processing speed of the selected VM type by:

$$\mu_{a,c} := \sum_{v \in V} \delta_{a,v} y_{a,c,v}, \tag{2}$$

where $\sum_{v \in V} y_{a,c,v} \leqslant 1$ $(\forall a \in A, \forall c \in C)$. There are two situations for eq. (2). If application a is deployed in data centre c, $\mu_{a,c} = \delta_{a,v}$ where $y_{a,c,v} = 1$. Otherwise, $\mu_{a,c} = 0$.

Referring to [3], the average resource consumption per request over a long sequence of requests for the same application is considered highly stable, while different applications can incur different resource requirements. We follow [44] and model the operation of each individual application replica as an $M/M/1$ queue. According to Little's Law [45], the average request processing time of replica a_c depends on both $x_{a,u,c}$ and $y_{a,c,v}$:

$$pt_{a,c} := \frac{1}{\mu_{a,c} - \lambda_{a,c}} = \frac{1}{\sum_{v \in V} \delta_{a,v} y_{a,c,v} - \sum_{u \in U} \gamma_{a,u} x_{a,u,c}}. \tag{3}$$

With the goal to minimize *TDC* of the Web application suite in multi-cloud, we have two decision variable vectors, i.e. *request dispatch plan* X, which determines how user requests of Web applications are dispatched among all data centres, i.e., $x_{a,u,c}$, and *replica deployment plan* Y, which determines the types of VMs and the corresponding data centres to deploy all application replicas, i.e., $y_{a,c,v}$. Concretely, the WARDMC problem is formulated as follows:

Problem 1.

$$\min \quad TDC = \sum_{a \in A} \sum_{c \in C} \sum_{v \in V} y_{a,c,v} rc_{c,v} \tag{4}$$

subject to:

(a) $y_{a,c,v} \in \{0,1\} \quad \forall a \in A, \forall c \in C, \forall v \in V$

(b) $\sum_{v \in V} y_{a,c,v} \leqslant 1 \quad \forall a \in A, \forall c \in C$

(c) $\dfrac{\sum_{a \in A} \sum_{c \in C} \sum_{u \in U} \gamma_{a,u} x_{a,u,c} (dt_{u,c} + pt_{a,c})}{\mathcal{S}} \leqslant \mathcal{R}$

(d) $pt_{a,c} > 0 \quad \forall a \in A, \forall c \in C$

(e) $0 \leqslant x_{a,u,c} \leqslant 1 \quad \forall a \in A, \forall u \in U, \forall c \in C$

(f) $\sum_{c \in C} x_{a,u,c} = 1 \quad \forall a \in A, \forall u \in U$

Constraint (a) indicates whether an instance of VM type v in data centre c is rented for hosting application a. Constraint (b) guarantees that each application replica is deployed to one VM instance. Constraint (c) guarantees that ART of the whole Web application suite is below the acceptable threshold $\mathcal{R}$ set by the application provider. Particularly, $dt_{u,c}$ represents RTD between user region u and data centre c. The denominator at LHS of constraint (c), i.e., $\mathcal{S} = \sum_{a \in A} \sum_{u \in U} \gamma_{a,u}$, is the overall request rate across Web application suite A. Constraint (d) guarantees that $\mu_{a,c} > \lambda_{a,c}$, i.e., the capacity of any application replica must be sufficient to process its workload. Constraints (e) and (f) guarantee that every request for application a from user region u will be processed.

To obtain the optimal solution to Problem 1, we must optimize X and Y jointly. Problem 1 is nonlinear due to constraint (c). In particular, the average request process time $pt_{a,c}$ is determined by the workload $\mu_{a,c}$ and capacity $\lambda_{a,c}$ for application replica a_c. Note that $\mu_{a,c}$ and $\lambda_{a,c}$ also together determine the utilization rate of the deployed VM for a_c. We can linearize Problem 1 by bounding utilization rates of the VMs in order to obtain a base solution to Problem 1. Inevitably, such upper bound reduces the solution space for Problem 1. Hence, the base solution is not necessarily the optimal solution to Problem 1. In view of this, we further design a problem-specific LNS algorithm to improve the base solution. In what follows, we discuss the proposed approach in detail.

4 MCApp – A Hybrid Approach to WARDMC Problem

In this section, we introduce our MCApp approach (see Fig. 1). It first transforms the WARDMC problem to a series of MILP problems in Subsection 4.1. These linearized problems are then solved by using the MILP-based algorithm to obtain the base solution in Subsection 4.2. Finally, the LNS-based algorithm to improve the base solution is introduced in Subsection 4.3.

4.1 Problem Transfer

MCApp linearizes the WARDMC problem making it solvable by off-the-shelf MILP methods. We first determine the utilization rate of the VM for application replica a_c by:

$$u_{a,c} = \frac{\lambda_{a,c}}{\mu_{a,c}}, \tag{5}$$

where $0 \leqslant u_{a,c} < 1$ based on constraint (a), (d), and (e) in Problem 1. According to eq. (5), the average request processing time of a_c in eq. (3) can be rewritten as:

$$pt_{a,c} = \frac{1}{\mu_{a,c} - \lambda_{a,c}} = \frac{1}{\frac{\lambda_{a,c}}{u_{a,c}} - \lambda_{a,c}} = \frac{1}{\lambda_{a,c}(\frac{1}{u_{a,c}} - 1)}$$
$$= \frac{1}{\lambda_{a,c}} \cdot \frac{u_{a,c}}{1 - u_{a,c}}.$$

Therefore, in Constraint (c) of Problem 1:

$$\sum_{u \in U} \gamma_{a,u} x_{a,u,c} pt_{a,c} = \lambda_{a,c} pt_{a,c} = \frac{u_{a,c}}{1 - u_{a,c}}.$$

In the above, $f(u_{a,c}) = \frac{u_{a,c}}{1-u_{a,c}}$ is monotonically increasing with respect to $u_{a,c} \in [0,1)$. Let the constant $\hat{u}_{a,c}$ denote the upper bound on $u_{a,c}$ and $\hat{u} = \max_{a\in A, c\in C}\{\hat{u}_{a,c}\}$, we have:

$$\sum_{u\in U} \gamma_{a,u} x_{a,u,c} pt_{a,c} = \frac{u_{a,c}}{1-u_{a,c}} \leqslant \frac{\hat{u}_{a,c}}{1-\hat{u}_{a,c}} \leqslant \frac{\hat{u}}{1-\hat{u}}. \quad (6)$$

Therefore, Problem 1 can be transferred as follows:

Problem 2.

$$\min \quad TDC = \sum_{a\in A}\sum_{c\in C}\sum_{v\in V} y_{a,c,v} rc_{c,v} \quad (7)$$

subject to:

(a′) $Y \in \mathbb{C}_1$

(b′) $X \in \mathbb{C}_2$

(c′) $\frac{\sum_{a\in A}\sum_{c\in C}(\sum_{u\in U}\gamma_{a,u}x_{a,u,c}dt_{u,c} + \frac{\hat{u}}{1-\hat{u}})}{\mathcal{S}} \leqslant \mathcal{R}$

(d′) $\hat{u}\mu_{a,c} - \lambda_{a,c} \geqslant 0 \quad \forall a \in A, \forall c \in C$

In constraint (a′), $\mathbb{C}_1$ stands for the set of feasible replica deployment plans Y that satisfy constraints (a), (b) in Problem 1. In constraint (b′), $\mathbb{C}_2$ refers to the set of feasible request dispatch plans X that satisfy constraints (e), (f) in Problem 1. Constraint (c′) is derived from constraints (c) in Problem 1 based on eq. (6). Constraint (d′) guarantees that the utilization rate of the VM instance for application replica a_c never exceed $\hat{u}$. With different $\hat{u}$, Problem 2 can be considered as a series of MILP problems and solved directly using popular MILP methods, e.g., cutting-plane [46] and branch and cut [47], provided by many open source software tools, e.g., Google OR-Tools [48].

4.2 Mixed Integer Linear Programming

Clearly increasing $\hat{u}$ can improve the overall VM utilization rate, thus decrease TDC in Problem 2. Note that if $\hat{u}$ is too large, we cannot find feasible solutions to Problem 2 subject to constraint (c′). One practicable method is to initialize $\hat{u}$ at its maximum, then repeatedly reduce $\hat{u} = \hat{u} - \Delta u$ until a feasible solution can be found. Next, we theoretically analyse the upper bound on $\hat{u}$.

Let $\overline{dt}$ denote the average RTD across all requests for all application replicas. We have:

$$\overline{dt} := \frac{\sum_{a\in A}\sum_{c\in C}\sum_{u\in U}\gamma_{a,u}x_{a,u,c}dt_{u,c}}{\mathcal{S}}. \quad (8)$$

Given request rate $\gamma_{a,u}$, we can easily obtain the lower bound $\mathcal{L}$ of $\overline{dt}$ by:

$$\mathcal{L} = \frac{\sum_{a\in A}\sum_{u\in U}\gamma_{a,u}\min_{c\in C}\{dt_{u,c}\}}{\mathcal{S}}.$$

This lower bound is realized when all user requests are dispatched to the respective data centre with the minimal $dt_{u,c}$.

Algorithm 1 Our MILP-based algorithm in MCApp.

```
Input: $A, U, C, V, \gamma_{a,u}, rc_{c,v}, \delta_{a,c,v}, dt_{u,c}, \mathcal{R}$.
Output: Request dispatch plan $X$ and replica deployment
    plan $Y$.
 1: $\hat{u} \leftarrow \frac{\mathcal{S}(\mathcal{R}-\mathcal{L})}{\mathcal{S}(\mathcal{R}-\mathcal{L})+|A|}$
 2: while MILP methods cannot find a feasible solution to
    Problem 2 with $\hat{u}$ do
 3:     $\hat{u} \leftarrow \hat{u} - \Delta u$
 4: end while
 5: For the feasible solution $X$ and $Y$ to Problem 2, calculate
    $ART$ and $TDC$
 6: $\mathcal{R}' \leftarrow \mathcal{R}$
 7: while $ART \leqslant \mathcal{R}$ and termination rule is not met do
 8:     Gain $X'$ and $Y'$ to Problem 2 with $\mathcal{R}' \leftarrow \mathcal{R}' + \Delta\mathcal{R}$ by
        MILP methods and calculate its $ART'$ and $TDC'$
 9:     if $ART' \leqslant \mathcal{R}$ and $TDC' < TDC$ then
10:         $X, Y \leftarrow X', Y'$
11:         $TDC \leftarrow TDC'$
12:     end if
13:     $ART \leftarrow ART'$
14: end while
15: return $X$ and $Y$
```

Theorem 1. *The maximum of $\hat{u}$ is $\frac{\mathcal{S}(\mathcal{R}-\mathcal{L})}{\mathcal{S}(\mathcal{R}-\mathcal{L})+|A|}$ subject to constraints (c′).*

The proof of the theorem is given in the appendix A.

With $\hat{u}$, ART of any feasible solution to Problem 2 is less than $\mathcal{R}$ in constraint (c′), because $u_{a,c} \leqslant \hat{u}$ for all application replicas. Increasing $\mathcal{R}$ to $\mathcal{R}'$ when solving Problem 2 is helpful to find solutions with lower TDC. However, if $\mathcal{R}'$ is too large, the found solutions cannot be feasible due to violation of constraint (c′), i.e., ART of the solutions exceed $\mathcal{R}$. Another iterative method can be used to gradually increase $\mathcal{R}' = \mathcal{R}' + \Delta\mathcal{R}$ in constraint (c′) until we cannot find feasible solutions.

Algorithm 1 shows the procedure of the MILP-based algorithm with the adaptive $\hat{u}$ and $\mathcal{R}'$. After initializing $\hat{u}$ (step 1), we iteratively determine an appropriate $\hat{u}$ through step 2-4. After a feasible solution to Problem 2 is found, we calculate ART and TDC in step 5. Next we relax the performance requirement to $\mathcal{R}'$ in an attempt to decrease TDC. The process is repeated until ART is greater than $\mathcal{R}$ or the termination rule (e.g., the maximum optimization time) is met (step 6-14).

Finally, we theoretically analyse the performance of the base solution by Algorithm 1 in terms of worst-case ratio.

For each VM type v in data centre c, we can obtain the unit cost of processing a single request for application a:

$$k_{a,c,v} = \frac{rc_{c,v}}{\delta_{a,v}}. \quad (9)$$

We denote the minimal unit cost for application a across all VM types and data centres as $k_{a.min} = \min_{c\in C, v\in V}\{k_{a,c,v}\}$ and the minimal unit cost among all applications as $\mathcal{K} = \min_{a\in A}\{k_{a.min}\}$.

Next we generate a WARDMC solution subject to all the constraints in Problem 2. Therefore, TDC of this WARDMC solution is the upper bound on TDC of the optimal solution to Problem 2.

Algorithm 2 Our LNS-based algorithm in MCApp.

```
Input: Base solution to Problem 2 $X^\circ$ and $Y^\circ$, $iter_{max}$.
Output: $X_{best}, Y_{best}$
1: $X_{best}, Y_{best} \leftarrow X^\circ, Y^\circ$
2: $i \leftarrow 0$
3: while $i < iter_{max}$ do
4:     $Y \leftarrow destroy(Y_{best})$   /* Algorithm 3
5:     $X, Y \leftarrow repair(Y)$   /* Algorithm 5
6:     if $TDC(X,Y) < TDC(X_{best}, Y_{best})$ then
7:         $X_{best}, Y_{best} \leftarrow X, Y$
8:         $i \leftarrow 0$
9:     else
10:        $i \leftarrow i + 1$
11:    end if
12: end while
13: return $X_{best}, Y_{best}$
```

Concretely, let this WARDMC solution have $\overline{dt} = \mathcal{L}$, i.e., all user requests are served in the data centres with the minimal $dt_{u,c}$. Based on this request dispatch plan, we can obtain the workload of application replica $\lambda'_{a,c}$ and the number of replicas $\mathcal{H}$. In this case, we have $\hat{u}' \leqslant \frac{\mathcal{S}(\mathcal{R}-\mathcal{L})}{\mathcal{S}(\mathcal{R}-\mathcal{L})+\mathcal{H}}$ following the proof of Theorem 1.

Let $\hat{u}' = \frac{\mathcal{S}(\mathcal{R}-\mathcal{L})}{\mathcal{S}(\mathcal{R}-\mathcal{L})+\mathcal{H}}$, we always choose the cheapest VM type to deploy replica a_c subject to $\delta_{a,v'} \geqslant \frac{\lambda'_{a,c}}{\hat{u}'}$. Let $rc'_{a,c}$ denote the rental cost of the cheapest VM for a_c, TDC of this WARDMC solution can be calculated by:

$$\mathcal{C} = \sum_{a \in A} \sum_{c \in C} rc'_{a,c}. \tag{10}$$

Theorem 2. *TDC of the optimal the solution to Problem 2 is at most* $\frac{\mathcal{C}(\mathcal{R}-\mathcal{L})}{\mathcal{K}\mathcal{S}(\mathcal{R}-\mathcal{L})+\mathcal{K}}$ *times of the optimal TDC to Problem 1.*

The proof of the theorem is given in the appendix B.

4.3 Large Neighborhood Search

LNS has been widely used to solve various combinatorial optimization problems with practical importance [49], including multi-cloud service brokering in [33]. In this section, we explain the LNS-based algorithm to improve the base solutions found by Algorithm 1. LNS explores solution space by applying destroy and repair heuristics to the most recently discovered solution in each iteration [32]. Since the best achievable TDC depends on the number, locations, and types of the deployed VMs, we decide to design a LNS algorithm to improve the replica deployment plan Y. Particularly, three problem-specific heuristics are designed for replica replacement, VM selection, and request dispatch, respectively. First, we propose a *destroy heuristic* to remove varied number of application replicas from Y based on a domain-tailored relatedness measurement. Next, we design a *repair heuristic* to effectively transfer the destroyed Y violating constraint (c) to a feasible Y. To facilitate the process of repairing Y, we also propose a *delay-oriented heuristic* for request dispatch and ART evaluation.

Algorithm 2 shows the procedure of our LNS-based algorithm. The input parameter $iter_{max}$ denotes the maximum number of iterations without improving Y before terminating the algorithm. X_{best} and Y_{best} together denote the best WARDMC solution obtained so far during LNS. The function $destroy(Y_{best})$ in step 4 destroys a copy of Y_{best} by removing a portion of application replicas (introduced in Subsection 4.3.1). The function $repair(Y)$ in step 5 transforms the destroyed replica deployment plan into a new feasible Y and determines the request dispatch plan X accordingly (detailed in Subsection 4.3.2). In step 6, TDC of the new WARDMC solution is evaluated. Based on that, the solution is either rejected or accepted as the current solution for the next search iteration. Specifically, we only accept the cheaper solution and update X_{best} and Y_{best} correspondingly (step 7). Finally, Algorithm 2 returns the best replica deployment plan and request dispatch plan.

Algorithm 3 Destroy heuristic.

```
Input: Replica deployment plan $Y_{best}$, $\rho$, $r$.
Output: New replica deployment plan $Y$ after replicas removal.
1: $k \leftarrow random(1, \lceil \rho |Y_{best}| \rceil)$
2: Randomly select an application replica $a_c$ from $Y_{best}$
3: Initialize a set of application replicas $D \leftarrow \{a_c\}$
4: while $|D| < k$ do
5:     Create a list $L$ including all application replicas from $Y$ not in $D$
6:     Randomly select an application replica $a_c$ from $D$
7:     Sort $L$ such that $i < j \Rightarrow N(a_c, L[i]) < N(a_c, L[j])$
8:     Choose a random number $\xi \in [0, 1)$
9:     $D \leftarrow D \cup L[\xi^r |L|]$
10: end while
11: Update $Y_{best}$ as $Y$ by removing replicas in $D$
12: return $Y$
```

4.3.1 Destroy Heuristic

Our *destroy heuristic* is inspired by [33], which removes the related request-VM assignments based on their similarities, e.g., the deployment location, to improve the performance of multi-cloud service brokering. We propose to measure the relatedness among application replicas for destroying Y_{best}. Thus, $N(a_c, a'_{c'})$ in eq. (12) quantifies the *relatedness* between replicas a_c and $a'_{c'}$:

$$N(a_c, a'_{c'}) = \begin{cases} dt(c, c') & \text{if } a = a' \\ \infty & \text{otherwise,} \end{cases} \tag{11}$$

where $dt(c, c')$ is the RTD between two data centres c and c'. That is, application replicas are related if they belong to the same application and the deployed data centres c and c' are close to each other, while unrelated if they belong to different applications.

Algorithm 3 shows the procedure of the proposed destroy heuristic. The parameter $\rho \in (0, 1)$ is the percentage of application replicas to be removed and r ($r \geqslant 1$) controls random selection of replicas for removal. To discover possibly better WARDMC solutions, our destroy heuristic first *randomly* generates an integer $k \in [1, \lceil \rho |Y_{best}| \rceil]$ as the number of replicas to be removed in step 1. The mechanism has been verified to improve the performance of LNS in our experiments (Subsection 5.7). Then the heuristic randomly selects one replica a_c from Y_{best} (step 2) and adds it to set D (step 3), which is initially empty. Next, we create a list L to include all application replicas not in D (step 5) and randomly pick up a replica a_c from D (step 6). In step 7, the application replicas in L is sorted in the ascending order according to the relatedness with a_c in eq. (12). We select one replica from L with a probability proposed in [33] and add the replica to D (step 8-9). The procedure is repeated until k application replicas have been selected for removal. Finally, Y_{best} is updated by removing all replicas in D.

4.3.2 Repair Heuristic

It is highly likely that the replica deployment plan Y after replica removal cannot satisfy constraint (c). Our *repair heuristic* aims to generate new WARDMC solutions such that $ART \leqslant \mathcal{R}$. Particularly, we must resolve four issues during the repair process: (1) How many new replicas should be deployed? (2) Where should these new replicas be placed? (3) Which types of VMs should be deployed? (4) How to dispatch user requests based on the current replica deployment plan?

To ensure that the number of replicas after repair is not too different from $|Y_{best}|$, we first *randomly* generate the required number of application replicas $l \in [|Y_{best}| - \Delta l, |Y_{best}| + \Delta l]$. Here, Δl is a hyper-parameter which controls the exploration of the repair heuristic. That is, larger Δl creates more optional values for l. It is helpful to consider more different and potentially good WARDMC solutions. The mechanism has also been verified to improve the performance of LNS in our experiments (Subsection 5.7).

To identify appropriate data centres to place new replicas, we design a *greedy*-based mechanism to add data centres until the number of application replicas reaches l. First, we calculate each application's average utilization rate:

$$u_a = \frac{\sum_{u \in U} \gamma_{a,u}}{\sum_{c \in C} \mu_{a,c}}. \tag{12}$$

The application with the highest average utilization rate has the priority to receive new replicas. The rationale is as follows. On the one hand, if $u_a \geqslant 1$, at least one application replica a_c have $u_{a,c} \geqslant 1$, which violates eq. (5). On the other hand, if $u_a < 1$, adding replicas for an application with the higher average utilization rate helps to reduce the average request processing time of application effectively.

Next, we determine the data centre to place a new replica for the selected application. Here, we calculate the *benefit* of each data centre c as follows:

$$benefit_c = \frac{\overline{dt}_{a+c} - \overline{dt}_a}{k_{a,c,v}}, \tag{13}$$

where $\overline{dt}_{a+c}$ and $\overline{dt}_a$ are the average RTD of application a after and before adding a replica for application a using the cheapest VM type v in data centre c, respectively.

Note that RTD depends on the request dispatch plan X. Therefore, we develop a *delay-oriented* heuristic to quickly revise X for the purpose of minimizing the total RTD between users and application replicas. Here, we assume that all replicas of the same application have the identical utilization u_a, which can be calculated by eq. (12). Although this assumption based on the capacity-based round-robin scheduling [50] cannot guarantee the optimality of X for given Y, it can effectively prevent any application replica from being heavily utilized, thereby reducing the risk of

Algorithm 4 Delay-oriented heuristic for request dispatch.

```
Input: Replica deployment plan $Y$, $\gamma_{a,u}$, $dt_{u,c}$.
Output: Request dispatch plan $X$.
 1: for all Application $a \in A$ do
 2:     $\lambda_{a,c} \leftarrow \frac{\sum_{u \in U} \gamma_{a,u}}{\sum_{c \in C} \mu_{a,c}} \mu_{a,c}$
 3:     Create two lists $RTDList$ and $DCList$
 4:     while Exist $\gamma_{a,u}$ not dispatch do
 5:         Find the user region $u$ with $min(RTDList)$ and the
            corresponding replica $a_c$ by $DCList$
 6:         if $\lambda_{a,c} \geqslant \gamma_{a,u}$ then
 7:             $r_{a,u,c} \leftarrow \gamma_{a,u}$
 8:             $\lambda_{a,c} \leftarrow \lambda_{a,c} - \gamma_{a,u}$
 9:         else
10:             $r_{a,u,c} \leftarrow \lambda_{a,c}$
11:             $\lambda_{a,c} \leftarrow 0$ and update $DCList$ and $RTDList$
                without considering $a_c$
12:         end if
13:     end while
14: end for
15: return Request dispatch plan $X$
```

long request processing time. After defining the workload, i.e., $\lambda_{a,c} = u_a \mu_{a,c}$, for each application replica, X is generated as detailed below.

Algorithm 4 demonstrates the process of the delay-oriented heuristic. For each application a, the heuristic first calculates the workload of all replicas $\lambda_{a,c}$ (step 2) and creates two lists, i.e., $RTDList$ and $DCList$, to record the minimal RTD among all replicas and the corresponding data centre for each user region (step 3). Then the heuristic iteratively finds the user region with the minimum value in $RTDList$, dispatches the requests from this region to the corresponding replica according to $DCList$. Concretely, if $\lambda_{a,c} \geqslant \gamma_{a,u}$, that is, the workload of a_c is in surplus, the requests rate from u to a_c, i.e., $r_{a,u,c}$ is determined by $\gamma_{a,u}$ in step 7 and $\lambda_{a,c}$ is updated in step 8. Otherwise, $r_{a,u,c}$ is determined by $\lambda_{a,c}$ in step 10 and the two lists $DCList$ and $RTDList$ are updated regardless of a_c in step 11. Finally, the heuristic returns X by setting each of its component $x_{a,u,c}$ to $\frac{r_{a,u,c}}{\gamma_{a,u}}$.

The new application replica is placed in the data centre with the largest benefit defined in eq. (13) until the number of application replicas reaches l. Then we verify whether the new replica deployment plan Y is feasible. If $ART > \mathcal{R}$, we can select new VM types with higher processing speed, e.g., change the VM type from Amazon Web Services (AWS) m5.large to m5.xlarge, for existing application replicas.

Note that the upgrade of VMs will change request dispatch plan X and upgrading a single VM may increase the average RTD. For example, if a Web application has more Asian users than European users, upgrading the application replica in Europe alone will reduce the request processing time for European users. However, the upgrade can increase RTD for some Asian users due to the workload variation based on eq. (12). To decrease ART, multiple replicas should be upgraded simultaneously. We design a *random*-based mechanism to select varied sets of application replicas and upgrade them correspondingly. This mechanism helps to identify an appropriate replica set for upgrade. The overall

Algorithm 5 Repair heuristic.

```
Input: Replica deployment plan $Y$ after replicas removal.
Output: Repaired replica deployment plan $Y$ and request dispatch plan $X$.
1:  $l \leftarrow random(|Y_{best}| - \Delta l, |Y_{best}| + \Delta l)$
2:  while $|Y| < l$ do
3:      Determine the application $a$ with the highest average utilization rate according to eq. (12)
4:      Determine the data centre $c$ with the largest benefit according to eq. (13)
5:      Add a new application replica for $a$ at $c$ with a random VM type
6:  end while
7:  Obtain request dispatch plan $X$ by Algorithm 4 and calculate $ART$
8:  while $ART > \mathcal{R}$ do
9:      Randomly select application replicas from $Y$
10:     Update the selected replicas by upgrading the current VM types
11:     Obtain request dispatch plan $X$ by Algorithm 4 and calculate $ART$
12: end while
13: return $X$ and $Y$
```

repair heuristic is shown in Algorithm 5.

5 Performance Evaluation

In the absence of a publicly available global multi-cloud testbed, many recently published research works mainly rely on simulations to evaluate algorithm performance [3], [51]. In this section, we present the experimental evaluation of our proposed MCApp approach for solving the WARDMC problem.

5.1 Datasets

Based on the latest report regarding the worldwide IaaS public cloud services market share [52], we collect the real VM type descriptions and pricing schemes in February 2021 from three leading cloud providers, i.e. AWS[4], Microsoft Azure[5] and Alibaba Elastic Compute Service (ECS)[6]. 21 different VM types (8 from AWS m5-series, 8 from Azure Dav4-series, and 5 from Alibaba g6-series) have been included in the experiments. We also consider a total of 15 locations for major AWS, Azure and Alibaba data centres, i.e., Northern Virginia, Northern California, Dublin, London, Paris, Frankfurt, Stockholm, Hong Kong, Singapore, Seoul, Tokyo, Sydney, Sao Paulo, Mumbai, and Johannesburg. Furthermore, we adopt 82 user centres from 35 countries on 6 continents in the Sprint IP Network[7] to simulate the global user community.

To evaluate the network latency between users and deployed services, we use the network latency information in the Sprint[8] IP backbone network databases.

4. https://aws.amazon.com/ec2/instance-types/
5. https://azure.microsoft.com/en-us/services/virtual-machines/
6. https://www.alibabacloud.com/product/ecs
7. https://www.sprint.net/tools/ip-network-performance (locations are shown as pink dots) and https://github.com/qingdaost/LBARDM
8. https://www.sprint.net/tools/ip-network-performance

5.2 Configuration Settings

In our experiments, the range of Web application processing time for a single request is based on [3], that is, 10-20 ms running on AWS m5.large VM. The number of Web applications in an experimented suite ranges from 1 to 10 to simulate a wide variety of user requirements. It is sufficient for enterprise applications because the meteorological service of New Zealand, i.e., MetService[9], provides 8 Web applications. Referring to [53], we apply Facebook subscribers statistics[10] to simulate the distribution of application requests from every user region. By January 2020, there are approximately 1.3 billion Facebook subscribers from all of 35 countries considered in our simulation. We consider the total number of users for our simulated Web applications as 1/1000th of Facebook subscribers to simulate the Web application providers with millions of users[11]. That is, the number of users in different regions ranges from 2100 to 71800. For each Web application, we select 30 out of all 82 user regions randomly, and each user from the selected user regions makes 25 requests on average daily following [53]. Accordingly, the application request rate spans from 52 to 304 requests per second (that is, approximately 4.5-26.3 millions of requests daily). We assume that new application requests from user regions are generated according to a Poisson distribution, following the common practice in many previous works [3], [54]. $\mathcal{R}$ is set to 150 ms since the response time beyond 200 ms will deteriorate the user experience significantly [9].

5.3 Competing Approaches and Parameter Settings

We adapt the following three approaches to the WARDMC problem and compare MCApp to them.

The LNS-based approach proposed in [33] supports periodical VM selection in multi-cloud. For convenience, we denote the competing algorithm as LNS-MC. This approach applies a greedy-based constructive heuristic, a Shaw-based destroy heuristic [32], and a greedy-based repair heuristic. We apply the same parameter settings as [33]: $iter_{max} = 2000$, $\rho = 0.3$, $r = 4$.

HMOHM [8] is proposed to deploy Web applications to public clouds. The GA-based algorithm in HMOHM applies two evolutionary operators, i.e., mutation and nascency, to balance global and local search. For parameters of GA, we adopt the same settings as recommended in [8]: the population size is 100, the elite size is 10, both the mutation rate and the nascency rate are 0.5, and termination time is 5 minutes.

GA-ARP [7] is recently proposed for budget-constrained application replication and deployment in multi-cloud. The GA-ARP approach minimizes ART of applications subject to a budget constraint. To ensure the quality of deployment solutions, GA-ARP adds a heuristic-based solution to the initial GA population. To apply GA-ARP to the WARDMC problem, we change its time-based fitness function to TDC and budget-related constraint to ART. On the other hand, we change the heuristic in GA-ARP, i.e., GreedyAdd, to a

9. https://www.metservice.com/
10. https://worldpopulationreview.com/country-rankings/facebook-users-by-country
11. https://github.com/qingdaost/LBARDM

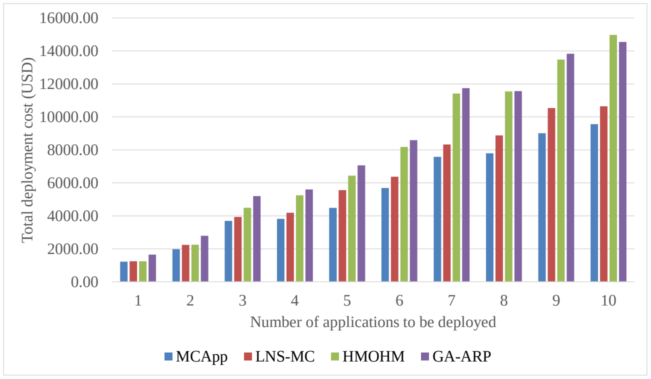


Fig. 3. Comparison of average TDC by competing approaches.

new heuristic that generates a cost-saving solution within the given $\mathcal{R}$ as the seed. For the adapted GA-ARP, our parameter settings include: the population size is 100, the maximum generation is 100, which are sufficient for the search process to converge. The crossover rate and mutation rate are 0.9 and 0.1 respectively following common practice in the literature [55].

We use CBC solvers from Google OR-Tools package version 7.3.7 [48] to implement Algorithm 1. Based on our preliminary simulation studies, we decide to set the parameters: $\Delta u = 1\%$ and $\Delta\mathcal{R} = 5ms$. We set 10 minutes as the termination rule for Algorithm 1 to control the overall computational time of MCApp. For Algorithm 2, we follow [33] to determine its parameter settings: $iter_{max} = 2000$, $\rho = 0.3$, $r = 4$. Besides, we set $\Delta l = 2$, which is sufficient to explore search space, because we cannot gain significant improvements on the quality of the final solutions by using larger Δl. To compare the results, we run each experiment 30 times on the same computer with Intel Core i7-8700 CPU (3.2 GHz and 16 GB of RAM).

5.4 Performance Comparison

Because all competing approaches can satisfy the constraints on average response time, we show the average TDC achieved for the WARDMC problem with different numbers of Web applications in Fig. 3. The mean and standard deviation of TDC are presented in TABLE 2.

When the number of applications $|A|$ is 1, the superiority of MCApp over LNS-MC and HMOHM is about 2%. As $|A|$ increases, MCApp can generate solutions with much lower TDC than LNS-MC and HMOHM. Particularly, when deploying more than 4 applications simultaneously, the performance improvement of MCApp over HMOHM is substantial, i.e., the cost reduction is more than 30%. We also find that except $|A| = 10$, GA-ARP always has the worst performance, which demonstrates that dispatching the requests from the same user region to different application replicas rather than the closest one results in better solutions. Comparing to some competing approaches, the cost reduction of MCApp can be more than 35% in some problem instances. For example, when $|A| = 5$ and 7 comparing with GA-ARP and $|A| = 10$ comparing with HMOHM.

From TABLE 2, we can calculate that the average improvement of MCApp in terms of TDC is 10.47% lower than LNS-MC, 25.55% lower than HMOHM, and 32.33% lower than GA-ARP among all problem instances. The observed performance differences between MCApp and three competing approaches are verified by a statistical test (Wilcoxon Rank-Sum test) with significance level of 0.05 for all problem instances. We also find that MCApp has small standard deviation, confirming its stability and reliability for the WARDMC problem.

5.5 Effectiveness of Application Replication

In this subsection, we evaluate the effectiveness of application replication for satisfying constraints on ART. As reviewed in Subsection 2.1, H-GA is proposed for multi-cloud application deployment without replication in [3]. We show the ART achieved by MCApp and H-GA in Fig. 4. For all problem instances, H-GA cannot generate solutions having ART within 150 ms, i.e., the red dotted line in Fig. 4. For example, the minimal ART achieved by H-GA is 238.18 ms for the problem instance with $|A| = 5$ and the maximal ART achieved by H-GA is 244.93 ms for the problem instance with $|A| = 9$. MCApp is capable of replicating and deploying Web applications with ART strictly below 150 ms.

If most users of an application are geographically localized, H-GA may generate adequate solutions without replication. When serving the requests from widely distributed users, H-GA cannot bring ART under 150 ms even selecting

TABLE 2
Performance comparison of the approaches for WARDMC with different application diversities (TDC per month in USD, the best is highlighted).

No. of Apps	MCApp		LNS-MC [33]		HMOHM [8]		GA-ARP [7]	
	TDC	n^*	TDC	n^*	TDC	n^*	TDC	n^*
1	**1219.68±0**	3	1244.16±0	2	1246.08 ±10.52	2	1649.76±5.46	6
2	**1978.56±0**	4	2240.64±0	3	2251.70 ±84.40	4	2787.14±71.34	10
3	**3694.25±41.49**	7	3935.81±7.77	6	4492.32 ±179.33	6	5198.9±127.63	18
4	**3816.96±30.11**	8	4190.50±0.53	6	5244.10 ±333.92	8	5595.98±216.57	22
5	**4481.28±0**	9	5558.40±0	6	6434.93 ±355.30	10	7059.34±154.57	30
6	**5687.16±40.48**	13	6373.44±0	7	8182.58 ±587.35	12	8586.84±143.12	41
7	**7580.52±43.15**	13	8323.20±0	8	11413.37 ±819.92	14	11745.65±210.32	47
8	**7794.19±197.69**	15	8876.16±0	9	11549.78 ±709.09	16	11564.04±175.6	54
9	**9009.41±308.45**	15	10535.04±0	10	13481.40 ±1091.05	18	13829.18±198.22	61
10	**9558.6±237.74**	19	10644.48±0	11	14968.92 ±1421.10	20	14547.79±248.01	68

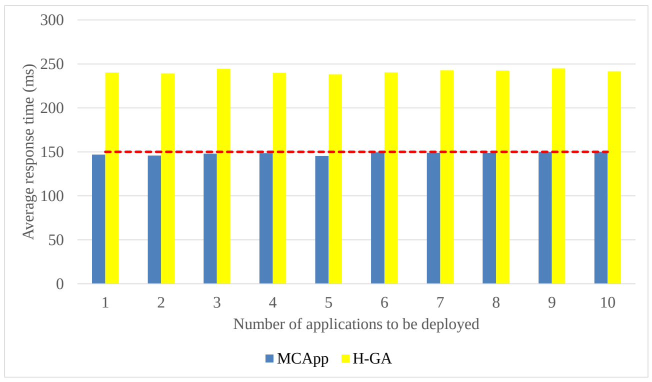


Fig. 4. Comparison of ART for evaluating the effectiveness of application replication.

the most expensive VMs for all Web applications. In that case, deploying application replicas at different data centres is imperative to reduce response time [56]. Therefore, we study the important problem of application replication for Web application deployment in this paper.

5.6 Replica Amount

Next, we investigate the effectiveness of MCApp by comparing the numbers of application replicas in the final replica deployment plans. The replica number n^* of the *best* solution among 30 runs is presented in TABLE 2.

n^* obtained by GA-ARP is always significantly greater than the other competing approaches. Each application has approximately 6 replicas on average. Too many replicas cause the leased VMs heavily under-utilized. In contrast, LNS-MC tends to use smaller n^* than those decided by MCApp. The smaller replica number decided by LNS-MC means more expensive VMs with higher processing speed have to be rented to serve the user requests, which also restrict the utilization of VMs.

Comparing MCApp and HMOHM, n^* are different for 8 out of 10 problem instances. From above results, we can conclude that the number of replicas seriously impacts TDC, and MCApp can effectively search for appropriate n for replica deployment plans. For example, when $|A| = 1$, MCApp determines n as 3 and deploys the three application replicas in Northern Virginia, Seoul, and Mumbai data centres respectively.

5.7 Effectiveness of Algorithm 1

As shown in TABLE 3, we also examine the performance of our proposed MILP-based algorithm, i.e., Algorithm 1, using two baseline methods, i.e., the greedy-based constructive heuristic in [33] (GreedyCon for convenience) and GreedyAdd in [7], in terms of TDC. The values of n of the the base solutions are also reported in TABLE 3.

GreedyCon achieves the same performance as Algorithm 1 for single application deployment. However, when $|A|$ is between 2 and 6, Algorithm 1 outperforms GreedyCon significantly. Besides, Algorithm 1 can generate better solutions than GreedyAdd with fewer n in all problem instances, benefiting from the flexible request dispatch plans.

TABLE 3
Base solutions' comparison for deploying applications with different application diversities (TDC per month in USD, the best is highlighted).

No. of Apps	Algorithm 1		GreedyCon [33]		GreedyAdd [7]	
	TDC	n	TDC	n	TDC	n
1	**1244.16**	2	**1244.16**	2	2376.00	3
2	**2052.00**	5	2240.64	3	4354.56	4
3	**3767.04**	8	4193.28	6	10091.52	6
4	**4033.44**	9	4193.28	6	7672.32	8
5	**5353.92**	11	5558.40	6	9440.64	10
6	**6134.40**	12	6373.44	7	18869.76	11
7	9385.92	13	**8323.20**	8	33730.56	13
8	9650.88	17	**8876.16**	9	19560.96	15
9	11269.44	17	**10535.04**	10	21041.28	17
10	11721.60	19	**10644.48**	11	64258.56	18

Note that for the problem instances where $|A| > 6$, there is not enough time for Algorithm 1 to generate good base solutions. However, using the information obtained from the solutions generated by Algorithm 1, e.g., the appropriate replica number for applications, MCApp still can obtain high quality final solutions by our proposed LNS-based algorithm, e.g., Algorithm 2. Overall, Algorithm 2 improves the base solutions from Algorithm 1 for all 10 problem instances, whereas GreedyCon solutions can only be improved for 2 out of 10 problem instances ($|A| = 3, 4$). Therefore, the hybrid approach MCApp is effective in finding good WARDMC solutions.

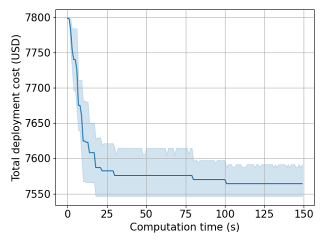
5.8 Effectiveness of Algorithm 2

Since there are several newly developed heuristics in Algorithm 2, we perform ablation studies to analyse its effectiveness. For the destroy heuristic, we compare Algorithm 3 with the destroy heuristic in [33] where $k = \lceil \rho |Y_{best}| \rceil$. We also compare Algorithm 5 with the repair heuristic in [33] that sets $l = |Y_{best}|$. For application replica placing, we compare our greedy mechanism with the random mechanism. For VM type selection, we compare our random mechanism with the greedy mechanism. The mean TDC of these ablation heuristics are presented in TABLE 4.

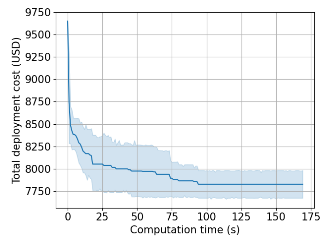
First, we can observe that Algorithm 3 achieves better performance than [33] with determinate k (shown as Determinate k in TABLE 4) except the single application deployment. This means that removing a random number of replicas in step 2 of Algorithm 3 helps to explore search space, especially for high application diversity (the improvements exceed 10% when $|A| = 8, 9, 10$). Second, the better exploration by setting l randomly in step 1 of Algorithm 5 also can lead to better performance than determinate l in [33] (shown as Determinate l in TABLE 4) for the majority of problem instances ($|A| = 1, 2, 5, 7, 8, 10$). Third, the greedy-based mechanism for placing new replicas in steps 3 and 4 of Algorithm 5 performs significantly better than the random mechanism (shown as Random placing in TABLE 4). This supports our analysis in Subsection 4.3.2, e.g., replicating the application with the highest average utilization rate. Lastly, the random-based mechanism for VM selecting in steps 5 and 9 of Algorithm 5 clearly outperformed the greedy mechanism (shown as Greedy selecting in TABLE 4). The random mechanism works effectively in the scenario where

TABLE 4
Ablation of Algorithm 2 across each contribution in mean TDC

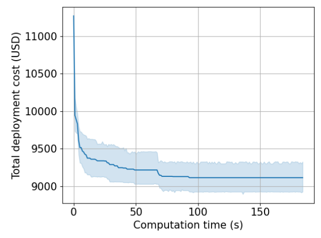
No. of Apps	Determinate k	Determinate l	Random placing	Greedy selecting
1	1219.68	1244.16	1232.54	1244.16
2	1979.14	1995.84	1978.66	1978.56
3	3724.75	3684.96	3757.44	3767.04
4	3877.56	3815.83	3918.50	4021.92
5	4484.78	4488.12	4483.34	4492.51
6	5732.11	5606.21	5720.74	5714.64
7	7798.32	7610.81	7642.32	7664.40
8	8779.99	7860.34	8164.97	8190.94
9	10043.30	8983.90	9498.12	9657.65
10	10733.28	9658.63	10218.10	10419.07



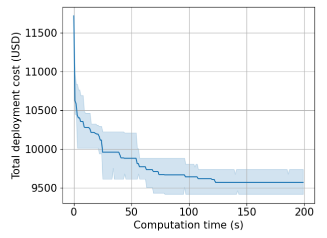
(a) 7 applications to be deployed



(b) 8 applications to be deployed



(c) 9 applications to be deployed



(d) 10 applications to be deployed

Fig. 5. Changes of TDC of Algorithm 2

upgrade of a single VM has the indeterminate impact on ART as analysed in Subsection 4.3.2.

We depict the change of TDC obtained by Algorithm 2 in Fig. 5. While the experiment results are obtained on the problem instance with 7, 8, 9 and 10 applications to be deployed, the same convergence behavior has also been witnessed on other problem instances. In Fig. 5, the convergence of TDC can be found after the computation time of Algorithm 2 exceeds about 100-130s.

5.9 Further Analysis

Since our proposed approach, i.e., MCApp, combines the iterative MILP and the domain-tailored LNS, the total overhead of MCApp includes the computation time (CT) of Algorithm 1 and Algorithm 2. In TABLE 5, we present the observed CT spent by the two algorithms with respect to different numbers of applications. The corresponding variable numbers (i.e., request dispatch plan X and replica deployment plan Y) and numbers of constraints are also included in TABLE 5. We find that the total computation time (TCT) of MCApp increases with the number of applications to be deployed, because every application will produce some variables and constraints to be handled by MCApp. As shown in TABLE 5, Algorithm 2 spends most of

TABLE 5
Variable numbers, numbers of constraints, and overhead of MCApp (TC and TCT in s.)

No. of Apps	Variable No. of X	Variable No. of Y	No. of constraints	CT of Alg. 1	CT of Alg. 2	TCT of MCApp
1	450	120	61	1.45	14.26	15.71
2	900	240	121	183.9	21.75	205.65
3	1350	360	181	296.47	40.87	337.34
4	1800	480	241	468.87	52.73	521.6
5	2250	600	301	600.00	55.23	655.23
6	2700	720	361	600.00	133.98	733.98
7	3150	840	421	600.00	146.06	746.06
8	3600	960	481	600.00	169.57	769.57
9	4050	1080	541	600.00	183.3	783.3
10	4500	1200	601	600.00	196.51	796.51

TCT when $|A|$ = 1. For $|A| > 1$, CT of Algorithm 1 takes up a majority of TCT. The increasing number of variables and constraints has a greater impact on CT of Algorithm 1 than Algorithm 2. Overall, MCApp can generate a WARDMC solution within 15 minutes. The competing approach LNS-MC can generate solutions within 10 minutes. We attempt to extend the termination time of HMOHM, i.e., 5 minutes, without obtaining noticeably better performance. The computation time required by GA-ARP varies from 20 to 6000 seconds subject to the number of applications.

As confirmed by many existing studies [39], [56], the workload of many Web applications does not change significantly within an hour. MCApp can be periodically applied to re-optimize the replica deployment plan and request dispatch plan, e.g., every hour, to reduce the total deployment cost while satisfying the constraint on average response time. The reactive strategy has been shown to be highly effective in many existing works [57], [58].

6 Conclusions and Future Works

In this paper, we studied the Web application replication and deployment problem in multi-cloud that considers the average response time, including both request processing time and network latency, to reflect the real performance of Web applications. To solve the problem, we propose an approach, i.e., MCApp, which can decide the number of replicas and select VMs from multi-cloud for Web applications subject to the given average response time so that the total deployment cost is minimized. We first transform the nonlinear WARDMC problem into a series of MILP problems through bounding the utilization rate of VMs for application replicas. Further, we propose a MILP-based algorithm to effectively generate a base solution with good resource utilization. Furthermore, to explore the search space, we design a problem-specific LNS-based algorithm to further optimize the base solution. The experiments based on the datasets collected from the real world cloud providers, network environment, and Web applications show that our approach can achieve up to 35% reduction in terms of TDC comparing with the existing approaches.

In this work, we consider the constraint on the average response time, because it seriously affects the user satisfaction with applications [9]. We believe it is a promising future direction to use worst-case response time to evaluate the performance of Web applications. Accordingly, new monitoring and reactive mechanisms will be proposed to satisfy the performance requirement. The other promising direction is to explicitly consider data sovereignty [59] during application deployment. Designing novel approaches for solving the problem of security-aware application deployment and replication is an interesting future research topic.

References

[1] N. Grozev and R. Buyya, "Inter-cloud architectures and application brokering: taxonomy and survey," *Software: Practice and Experience*, vol. 44, no. 3, pp. 369–390, 2014.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica *et al.*, "Above the clouds: A berkeley view of cloud computing," Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Tech. Rep., 2009.

[3] T. Shi, H. Ma, G. Chen, and S. Hartmann, "Location-aware and budget-constrained service deployment for composite applications in multi-cloud environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 8, pp. 1954–1969, 2020.

[4] A. N. Toosi, R. N. Calheiros, and R. Buyya, "Interconnected cloud computing environments: Challenges, taxonomy, and survey," *ACM Computing Surveys (CSUR)*, vol. 47, no. 1, pp. 7–53, 2014.

[5] T. Shi, H. Ma, and G. Chen, "A genetic-based approach to location-aware cloud service brokering in multi-cloud environment," in *2019 IEEE International Conference on Services Computing (SCC)*. IEEE, 2019, pp. 146–153.

[6] G. Shipley. (2018) Best practices for deploying your apps in the cloud. [Online]. Available: https://developer.ibm.com/articles/cl-best-practices-deploying-apps-in-cloud/

[7] T. Shi, H. Ma, G. Chen, and S. Hartmann, "Location-aware and budget-constrained application replication and deployment in multi-cloud environment," in *2020 IEEE International Conference on Web Services (ICWS)*. IEEE, 2020, pp. 110–117.

[8] M. Menzel, R. Ranjan, L. Wang, S. U. Khan, and J. Chen, "Cloudgenius: a hybrid decision support method for automating the migration of web application clusters to public clouds," *IEEE Transactions on Computers*, vol. 64, no. 5, pp. 1336–1348, 2014.

[9] Y. Wu, C. Wu, B. Li, L. Zhang, Z. Li, and F. C. Lau, "Scaling social media applications into geo-distributed clouds," *IEEE/ACM Transactions On Networking*, vol. 23, no. 3, pp. 689–702, 2014.

[10] Y. Mansouri, A. N. Toosi, and R. Buyya, "Cost optimization for dynamic replication and migration of data in cloud data centers," *IEEE Transactions on Cloud Computing*, vol. 7, no. 3, pp. 705–718, 2017.

[11] S. Wang, X. Li, and R. Ruiz, "Performance analysis for heterogeneous cloud servers using queueing theory," *IEEE Transactions on Computers*, vol. 69, no. 4, pp. 563–576, 2019.

[12] NIST. Cloud computing definitions. [Online]. Available: https://csrc.nist.gov/projects/cloud-computing

[13] S. KardaniMoghaddam, R. Buyya, and K. Ramamohanarao, "Adrl: A hybrid anomaly-aware deep reinforcement learning-based resource scaling in clouds," *IEEE Transactions on Parallel and Distributed Systems*, 2020.

[14] A. Erradi, W. Iqbal, A. Mahmood, and A. Bouguettaya, "Web application resource requirements estimation based on the workload latent features," *IEEE Transactions on Services Computing*, 2019.

[15] S. Slimani, T. Hamrouni, and F. Ben Charrada, "Service-oriented replication strategies for improving quality-of-service in cloud computing: a survey," *Cluster Computing*, pp. 1–32, 2020.

[16] M. S. Aslanpour, M. Ghobaei-Arani, and A. N. Toosi, "Auto-scaling web applications in clouds: A cost-aware approach," *Journal of Network and Computer Applications*, vol. 95, pp. 26–41, 2017.

[17] J. Jiang, J. Lu, G. Zhang, and G. Long, "Optimal cloud resource auto-scaling for web applications," in *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. IEEE, 2013, pp. 58–65.

[18] A. Gandhi, P. Dube, A. Karve, A. Kochut, and L. Zhang, "Providing performance guarantees for cloud-deployed applications," *IEEE Transactions on Cloud Computing*, vol. 8, no. 1, pp. 269–281, 2017.

[19] D. P. Bertsekas, "Nonlinear programming," *Journal of the Operational Research Society*, vol. 48, no. 3, pp. 334–334, 1997.

[20] Y. Hassanzadeh-Nazarabadi, A. Küpçü, and O. Ozkasap, "Decentralized utility-and locality-aware replication for heterogeneous DHT-based P2P cloud storage systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 5, pp. 1183–1193, 2019.

[21] X. Ren, P. London, J. Ziani, and A. Wierman, "Datum: Managing data purchasing and data placement in a geo-distributed data market," *IEEE/ACM Transactions on Networking*, vol. 26, no. 2, pp. 893–905, 2018.

[22] Y. Xia, M. Zhou, X. Luo, Q. Zhu, J. Li, and Y. Huang, "Stochastic modeling and quality evaluation of infrastructure-as-a-service clouds," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 1, pp. 162–170, 2013.

[23] H. Khazaei, J. Misic, and V. B. Misic, "Performance analysis of cloud computing centers using m/g/m/m+ r queuing systems," *IEEE Transactions on parallel and distributed systems*, vol. 23, no. 5, pp. 936–943, 2011.

[24] M. F. Mohamed, "Service replication taxonomy in distributed environments," *Service Oriented Computing and Applications*, vol. 10, no. 3, pp. 317–336, 2016.

[25] N. Bonvin, T. G. Papaioannou, and K. Aberer, "Autonomic sla-driven provisioning for cloud applications," in *2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2011, pp. 434–443.

[26] B.-Y. Ooi, H.-Y. Chan, and Y.-N. Cheah, "Dynamic service placement and replication framework to enhance service availability using team formation algorithm," *Journal of systems and Software*, vol. 85, no. 9, pp. 2048–2062, 2012.

[27] J. Wu, B. Zhang, L. Yang, P. Wang, and C. Zhang, "A replicas placement approach of component services for service-based cloud application," *Cluster Computing*, vol. 19, no. 2, pp. 709–721, 2016.

[28] M. Björkqvist, L. Y. Chen, and W. Binder, "Dynamic replication in service-oriented systems," in *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*. IEEE, 2012, pp. 531–538.

[29] K.-T. Tran and N. Agoulmine, "Adaptive and cost-effective service placement," in *2011 IEEE Global Telecommunications Conference-GLOBECOM 2011*. IEEE, 2011, pp. 1–6.

[30] M. Björkqvist, L. Y. Chen, and W. Binder, "Opportunistic service provisioning in the cloud," in *2012 IEEE Fifth International Conference on Cloud Computing*. IEEE, 2012, pp. 237–244.

[31] H. Ghanbari, M. Litoiu, P. Pawluk, and C. Barna, "Replica placement in cloud through simple stochastic model predictive control," in *2014 IEEE 7th International Conference on Cloud Computing*. IEEE, 2014, pp. 80–87.

[32] P. Shaw, "Using constraint programming and local search methods to solve vehicle routing problems," in *International conference on principles and practice of constraint programming*. Springer, 1998, pp. 417–431.

[33] L. Heilig, R. Buyya, and S. Voß, "Location-aware brokering for consumers in multi-cloud computing environments," *Journal of Network and Computer Applications*, vol. 95, pp. 79–93, 2017.

[34] L. Xiao, M. Johansson, and S. P. Boyd, "Simultaneous routing and resource allocation via dual decomposition," *IEEE Transactions on Communications*, vol. 52, no. 7, pp. 1136–1144, 2004.

[35] T. Loukopoulos and I. Ahmad, "Static and adaptive distributed data replication using genetic algorithms," *Journal of Parallel and Distributed Computing*, vol. 64, no. 11, pp. 1270–1285, 2004.

[36] H. Khalajzadeh, D. Yuan, J. Grundy, and Y. Yang, "Improving cloud-based online social network data placement and replication," in *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. IEEE, 2016, pp. 678–685.

[37] VMware. (2021) Mercedes-benz.io. [Online]. Available: https://www.vmware.com/company/customers/index/fullpage.html?path=/content/web-apps-redesign/customer-stories/2021/building-an-entirely-virtual-consumer-experience

[38] ——. (2021) Gsl general hospital medical college. [Online]. Available: https://www.vmware.com/company/customers/index/fullpage.html?path=/content/web-apps-redesign/customer-stories/2021/leveraging-vmware-solutions-for-an-enhanced-patient-experience

[39] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, "Workload prediction using ARIMA model and its impact on cloud applications' QoS," *IEEE Transactions on Cloud Computing*, vol. 3, no. 4, pp. 449–458, 2014.

[40] IBM Cloud Education. (2020) Three-tier architecture. [Online]. Available: https://www.ibm.com/cloud/learn/three-tier-architecture

[41] S. Wang, Z. Ding, and C. Jiang, "Elastic scheduling for microservice applications in clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 98–115, 2020.

[42] W. Vogels, "Eventually consistent," *Communications of the ACM*, vol. 52, no. 1, pp. 40–44, 2009.

[43] MySQL. (2021) Eventual consistency. [Online]. Available: https://dev.mysql.com/doc/mysql-cluster-manager/1.4/en/mcm-eventual-consistency.html

[44] J. Vilaplana, F. Solsona, I. Teixidó, J. Mateo, F. Abella, and J. Rius, "A queuing theory model for cloud computing," *The Journal of Supercomputing*, vol. 69, no. 1, pp. 492–507, 2014.

[45] J. D. Little and S. C. Graves, "Little's law," in *Building intuition*. Springer, 2008, pp. 81–100.

[46] H. Marchand, A. Martin, R. Weismantel, and L. Wolsey, "Cutting planes in integer and mixed integer programming," *Discrete Applied Mathematics*, vol. 123, no. 1-3, pp. 397–446, 2002.

[47] J. E. Mitchell, "Branch-and-cut algorithms for combinatorial optimization problems," *Handbook of applied optimization*, vol. 1, pp. 65–77, 2002.

[48] Google. Ortools. [Online]. Available: https://developers.google.com/optimization

[49] D. Pisinger and S. Ropke, "Large neighborhood search," in *Handbook of metaheuristics*. Springer, 2010, pp. 399–419.

[50] N. Ghani, A. Shami, C. Assi, and M. Raja, "Intra-onu bandwidth scheduling in ethernet passive optical networks," *IEEE Communications Letters*, vol. 8, no. 11, pp. 683–685, 2004.

[51] W. Li, X. Yuan, K. Li, H. Qi, X. Zhou, and R. Xu, "Endpoint-flexible coflow scheduling across geo-distributed datacenters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 10, pp. 2466–2481, 2020.

[52] C. Stamford. (2020) Gartner says worldwide iaas public cloud services market grew 37.3% in 2019. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2020-08-10-gartner-says-worldwide-iaas-public-cloud-services-market-grew-37-point-3-percent-in-2019

[53] B. Wickremasinghe, R. N. Calheiros, and R. Buyya, "Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications," in *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*. IEEE, 2010, pp. 446–452.

[54] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "Pias: Practical information-agnostic flow scheduling for commodity data centers," *IEEE/ACM Transactions on Networking (TON)*, vol. 25, no. 4, pp. 1954–1967, 2017.

[55] K.-F. Man, K.-S. Tang, and S. Kwong, "Genetic algorithms: concepts and applications [in engineering design]," *IEEE Transactions on Industrial Electronics*, vol. 43, no. 5, pp. 519–534, 1996.

[56] C. Qu, R. N. Calheiros, and R. Buyya, "Auto-scaling web applications in clouds: A taxonomy and survey," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–33, 2018.

[57] A. G. Kumbhare, Y. Simmhan, M. Frincu, and V. K. Prasanna, "Reactive resource provisioning heuristics for dynamic dataflows on cloud infrastructure," *IEEE Transactions on Cloud Computing*, vol. 3, no. 2, pp. 105–118, 2015.

[58] D. J. Dubois and G. Casale, "Optispot: minimizing application deployment cost using spot cloud resources," *Cluster Computing*, vol. 19, no. 2, pp. 893–909, 2016.

[59] C. Esposito, A. Castiglione, F. Frattini, M. Cinque, Y. Yang, and K.-K. R. Choo, "On data sovereignty in cloud-based computation offloading for smart cities applications," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4521–4535, 2018.

Tao Shi is working toward the PhD degree in the School of Engineering and Computer Science, Victoria University of Wellington, New Zealand. His main research interests include cloud computing and distributed system. His research focuses on resource management and combinatorial optimization in clouds.

Hui Ma received her B.E. degree from Tongji University and her B.S. (Hons.), M.S. and Ph.D degrees from Massey University. She is currently an Associate Professor in Software Engineering at Victoria University of Wellington. Her research interests include service computing, conceptual modelling, database systems, and resource allocation in clouds. Hui has more than 100 publications, including leading journals and conferences in databases, cloud computing, service computing, evolutionary computation, and conceptual modelling. She is a steering committee member of ER, and has served as a PC member for more than 70 international conferences, including seven times as a PC chair for conferences such as ER, DEXA, and APCCM. She has also served as local co-chair for Australian AI 2018 and CEC 2019.

Gang Chen received the Ph.D degree from Nanyang Technological University (NTU), Singapore. He is currently a Senior Lecturer with the School of Engineering and Computer Science, Victoria University of Wellington, New Zealand. His research interests include reinforcement learning, evolutionary computation and their application to optimization and scheduling problems, resource management and load balancing in networked systems.

Sven Hartmann received his Ph.D. in 1996 and his D.Sc. in 2001, both from the University of Rostock (Germany). From 2002 to 2007 he worked first as an associate professor, then full professor for information systems at Massey University (New Zealand). Since 2008 he is a full professor of computer science and chair for databases and information systems at Clausthal University of Technology (Germany). There he is also serving as academic dean at the Faculty of Mathematics, Informatics and Mechanical Engineering. Sven has more than 150 publications. He served as a PC member for more than 80 conferences, including 10 times as PC chair. His research interests include database systems, big data management, conceptual modelling, and combinatorial optimization.