GeoDeploy: Geo-distributed Application Deployment using Benchmarking

Devki Nandan Jha, *IEEE Member*, Yinhao Li, Zhenyu Wen*, *IEEE Senior Member*, Graham Morgan, Prem Prakash Jayaraman, Maciej Koutny, Omer F. Rana, Rajiv Ranjan, *IEEE Fellow*

Abstract—Geo-distributed web-applications (GWA) can be deployed across multiple geographically separated datacenters to reduce the latency of access for users. Finding a suitable deployment for a GWA is challenging due to the requirement to consider a number of different parameters, such as host configurations across a federated infrastructure. The ability to evaluate multiple deployment configurations enables an efficient outcome to be determined, balancing resource usage while satisfying user requirements. We propose GEODEPLOY, a framework designed for finding a deployment solution for GWA. We evaluate GEODEPLOY using both a formal algorithmic model and a practical cloud-based deployment. We also compare our approach with other existing techniques.

Index Terms—Geo-distributed Web-application, Benchmark, Orchestrator, Multi-objective optimization, Cloud Computing

1 INTRODUCTION

Modern web-applications (WA) like $Google$, $Amazon$ and $Alibaba$ pursue a business model where there is a desire to offer their services to users distributed across the globe and reachable by the Internet. This global service delivery has *two* primary challenges: 1) guaranteeing a uniformity in quality of service (QoS) to all parts of the world; 2) preserving privacy while adhering to sovereign laws regarding data management and distribution [1], [2]. This second point is a more recent challenge as governments start to organise their legal frameworks around international data boundaries. For example, GDPR (General Data Protection Regulation of the EU) restricts the transfer of personal data in consideration of nonEU and EU countries alike [3], [4].

To break the barriers, these services are usually geographically distributed in a multi-cloud environment in a federated manner [5], [6], [7]. An application supported in this way is commonly referred to as a Geo-Distributed Web-Application (GWA). A GWA follows a sophisticated architecture with a set of web servers deployed close to the users in order to provide lower latency. Each web server may have a local database to cache the frequently visited contents. Moreover, these web servers connect to one or more central databases allowing uninterrupted data access while maintaining privacy and ethical issues.

Cloud computing provides an opportunity for WA owners to realise global deployment. Currently, there are more than 267 CPs available in the marketplace, offering numerous host configurations at each datacenter geo-location e.g., AWS (Amazon Web Services) offers 275 instances at the Europe (London) datacenter. Moreover, these hosts are interconnected using WAN (Wide Area Network). However, the available bandwidth between a pair of DCs can be variable. [8] reported that the ratio of the highest pair to the lowest pair bandwidth is greater than 20 in both Amazon EC2 (Elastic Compute Cloud) and Microsoft Azure.

1.1 Deployment challenges

Deployment of GWA in a multi-cloud environment is challenging. A few main challenges considered in this paper are given below:

A. Importance of deployment location. The response time is dramatically affected by the location between the user and the WA host. Fig. 1 shows a significant fluctuation of the response time when a WA is deployed in London and numerous users are accessing it from different geo-locations. It shows that the highest latency is almost six times the lowest one. Complications arise in deriving suitable GWA deployment configurations as: a) the user's location may not be known before deployment and b) the number of users in a particular regulatory domain (provisioned by governmental legislation) may not be predicted before deployment. However, no literature has considered how to tackle this issue.

B. Importance of choosing host types. Users' experience (e.g., response time) is significantly dependent on the capacity of the hosts executing the GWA. Since the CPs offer numerous host configurations and the exact GWA requirements are unknown apriori, the GWA owner may find it difficult to determine which ones to choose without executing them. If the hosts can not provision sufficient

- *D. N. Jha is with Newcastle University, UK and University of Oxford, UK. E-mail: dev.jha@ncl.ac.uk*
- *Y. Li, G. Morgan, M. Koutny and R. Ranjan are with Newcastle University, UK. E-mail: {yinhao.li, graham.morgan, maciej.koutny, raj.ranjan}@ncl.ac.uk*
- *Z. Wen is with the Institute of Cyberspace Security and the College of Information Engineering, Zhejiang University of Technology, China. E-mail: zhenyuwen@zjut.edu.cn (*Corresponding author)*
- *P. P. Jayaraman is with Swinburne University of Technology, Australia. Email: pjayaraman@swin.edu.au*
- *O. F. Rana is with Cardiff University, UK, E-mail: ranaof@cardiff.ac.uk*

This work was supported by the National Nature Science Foundation of China under Grant 62472387, the China Postdoctoral Science Foundation under Grant 2023M743403, 2022M713206; the Zhejiang Provincial Natural Science Foundation of Major Program (Youth Original Project) under Grant LDQ24F020001

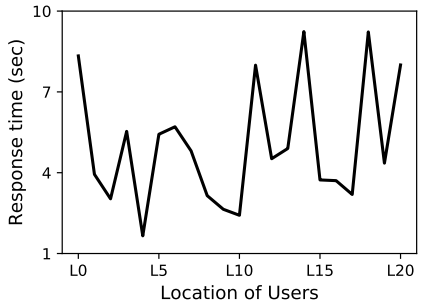


Fig. 1: The response time for users sending requests from different geolocations. L0 represents a specific geolocation of a User. L_n is chosen randomly from the set of 50 users distributed in 10 geolocations

computing resources, it will also cause high response time or even outage problems. In contrast, hiring powerful hosts may cause underutilisation of computing resources, thus leading to resource wastage. As a result, the hosts must be evaluated before deploying the real GWA.

C. Benchmarking geo-distributed web-application. Benchmarking GWA is challenging from both *system* and *algorithm* perspectives.

System challenge requires an ideal benchmark orchestrator that is able to interact with various CPs, which offer different ways to access their computing resources, and automatically run the benchmark application on these resources while providing the required evaluation results. However, most research is focused on single cloud scenarios [9], [10]. Jha et al. tackled the problem of deploying containerised web-applications on multiple clouds belonging to different CPs [11], [12]. However, these tools are unable to orchestrate a GWA benchmarking.

Algorithm challenge on the other hand requires the desired orchestrator to select the host from a mass of available candidates within a limited budget. This challenge is amplified when we aim to maximise the user's experience at the global level. In other words, the orchestrator has not only to consider *which* types of hosts should be evaluated, but also *where* to run the selected hosts. This can be proved to be an NP-hard problem [13].

To address the above-explained challenges, the proposed orchestrator needs to have the following *desirable properties*.

Close to users. The generated benchmarking solutions need to deploy the GWA component close to its users, which is one of the important tricks for reducing the response time significantly.

Increase the diversity of hosts. Depending upon the application type and the workload involved, a solution with a given host type and location may not be suitable. If we can evaluate a more diverse set of deployment solutions, with varying underlying host configurations and locations, we will have a better opportunity to find a solution to satisfy the QoS requirements.

Consider the benchmarking time and budget. To benchmark a GWA, the time and budget must be under consideration. Both directly influence how many *selected* deployment solutions can be obtained and evaluated. Given a budget, the proposed solution should evaluate a maximum and diverse set of configurations.

1.2 Overview of Methods and Contributions

In this paper, we propose, develop and validate GEODEPLOY that automatically 1) generates a set of candidate deployment solutions to maximise users' satisfaction; 2) runs these candidate deployment solutions over the real cloud environments and 3) recommends the most suitable deployment solution based on the requirements. Inspired by the idea of DevOps, we seek to derive a user-centric GWA deployment framework which can source a set of candidate deployment solutions and evaluate these solutions over existing cloud environments [14]. To the best of our knowledge, this is the first work that considers optimising the deployment of GWA in a multi-cloud environment through automated benchmarking.

We summarise the main contribution of our work as follows:

- We formulate the deployment of GWA using a two-phase optimisation problem.
- We propose and implement GEODEPLOY as a novel, user-centric, and cost-efficient deployment orchestrator for GWA. GEODEPLOY tackles the complexities of GWA deployment by considering the given GWA, cloud host information and budget constraints. Initially, the proposed method identifies a set of solutions that optimise both the quality and quantity of solutions while adhering to the specified budget. Subsequently, these solutions are automatically deployed in a real-cloud environment, effectively abstracting any low-level cloud-specific dependencies.
- We performed a comprehensive experiment evaluation for GEODEPLOY in a real-world multi-cloud environment. The evaluation is performed from both the algorithm and the deployment perspective. A scalability analysis is also performed to validate the scalability of GEODEPLOY.

Outline. The remainder of the paper is as follows. To allow our work to be considered in the context of similar works, a set of recent and relevant related works are presented in Section 2. In Section 3, we present the formal model of the GWA deployment problem. A high-level system design of GEODEPLOY is presented in Section 4. Before discussing the details of deployment optimisation in Section 6, an overview of our proposed Adaptive Particle Swarm Optimisation algorithm is given in Section 5. The details of experimental evaluation (numerical analysis and real-world analysis) is presented in Section 7. Before drawing conclusions and indicating our plans for future work in Section 9, we provide a discussion highlighting the limitations of our work in Section 8.

2 RELATED WORK

Benchmark orchestrator. GWA benchmarks need to be orchestrated on different host configurations in the multi-cloud environment. Orchestrating the systematic deployment of WA requires various steps including benchmark and host configuration definition, load generation, continuous performance monitoring and controlling the overall

process which is complex and error-prone to perform manually [15]. A few frameworks are available for orchestrating benchmark applications in the literature. μbench[16], COFFEE [17], Cloud WorkBench [18], Smart CloudBench [19] and SmartDBO [11], allow the orchestrating of WA benchmarks for different cloud providers. μBench [16] allows users to define a flexible microservice benchmark but does not allow it to run on multiple clouds concurrently. Flexibench [20] can also offer a benchmark orchestrator but is specific only for database applications. Cloud WorkBench [18] allows reusable benchmark definition leveraging *Chef* and provisioning the benchmark on heterogeneous clouds. However, defining the benchmark is complex. Smart CloudBench [19] not only automates the benchmark provisioning but also provides a comparison between various cloud providers. COFFEE [17] is an advanced orchestrator which allows orchestrating different container engines. However, all these frameworks ignore the cost of benchmarking which is necessary to consider in cloud environments. SmartDBO [11] orchestrates the benchmarking process with a simple user interface allowing users to interact. It also considers the benchmarking cost and time while selecting the provisioning process. However, none of the given works handle the uncertainty of users' locations and user-directed interaction patterns and thus are not able to support the benchmarking of GWA.

Content-distribution networks (CDN). CDNs share a geographic user base problem and highlight this issue in the context of a particular type of GWA [21], [22]. Some common examples are Akamai CDN [23] and Amazon CloudFront [24]. CDNs are used to anchor the access of web content, such as video in a streaming service, at the cost of hosting and maintaining the contents at the local edge servers while considering the merging of more widely distributed derived content. CDN typically only hosts and serves static data contents while GWA is used for both static and dynamic data which requires a considered understanding of resource availability in both a network and computational sense to ensure expected application orchestration. Additionally, a CDN belongs to a single cloud provider and focuses on optimising a global network of edge servers together with cloud-enabled services to provide cost-efficient network delivery of data to users, a GWA requires deployment optimisation in a multi-cloud environment. CDN can complement the GWA. Using a CDN can help to reduce the potential load on a GWA's server resources thus allowing it to serve exceeded load capacity. Using the CDN for GWA deployment is out of the scope of this work.

Optimised deployment. Optimised deployment of software artefacts (from application to build support) in cloud environments is an extensively studied problem. Examples of such approaches describe the optimised deployment algorithms in the cloud environment [25], [26], [27]. The main aim of these works is to optimise resource utilisation while deploying tasks/processes. Although such works are intended to ascertain a suitable deployment option, they are not concerned with the analysis of the underlying cloud environment and consider users' QoS requirements before deployment. Any deployment decision there is carried out by evaluating the system metrics provided beforehand. This is in distinct contrast to our approach which not only considers user-related issues but also issues that may span international boundaries and geographically inspired constraints.

Additionally, traditional cloud-edge offloading systems are developed for stream processing applications, which seek to derive minimal computing resources for maintaining stream operations while satisfying QoS requirements [28], [29]. These works do not apply to our purposes as such applications are concerned with large volume data distribution in, primarily, one direction whereas GWAs are interactive and responsive user-driven services.

Geo-distributed orchestration system. Orchestrating the application in a geo-distributed manner has been well studied [30], [31], [32]. Wen et al. focus on optimising the deployment of scientific workflow across federated clouds [31]. These applications have a well-defined computation logic, which is easy to optimise and deploy. However, orchestrating or scheduling big data systems in geo-distributed environments is still a challenge. Huang et al. presented a framework, Yugong, for data and job placement for geo-distributed datacenter for Alibaba cloud considering bandwidth as a main factor [30]. A similar method is proposed by Yang et al. to effectively run the big data systems across multiple cloud datacenters while considering the limitation of bandwidths [33]. Marchese et al. proposed a geo-distributed cloud-to-edge orchestration framework using Kubernetes [32]. Pallewatta et al. designed a framework for scalable placement of microservices-based IoT applications in federated Fog environments [34]. However, these systems are not sensitive to the response time for users, compared to GWA, and therefore do not consider the latency caused when users' requests are submitted from different locations. The proposed GeoDeploy is a novel system that tackles the challenge of benchmarking GWA, which aims to maximise users' satisfaction (e.g., low response time). The users may send their requests from all over the world.

3 System Overview

This section describes the GWA deployment model and formalises the candidate selection problem to an optimisation problem.

3.1 Web-application deployment model

A GWA is given as a directed graph $\mathbb{G}_{web} = (\mathbb{V}_{web}, \mathbb{E}_{web})$, where the nodes $\mathbb{V}_{web} = \{v^1_{web}, \ldots, v^n_{web}\}$ represent the web application components (microservices), and the arcs $\mathbb{E}_{web}$ represent the dependency between components, including data flow and transactions.

The computing resources are given as a complete directed graph $\mathbb{G}_{res} = (\mathbb{V}_{res}, \mathbb{E}_{res})$, where the nodes $\mathbb{V}_{res} = \{v^1_{res}, \ldots, v^m_{res}\}$ represent available hosts and the edges $\mathbb{E}_{res}$ represent the network connections among the hosts. For each host $v^i_{res} \in \mathbb{V}_{res}$:

- $h_{v^i_{res}} \in \mathbb{H}$ is the host type of v^i_{res},
- $loc_{v^i_{res}} \subseteq \mathbb{L}$ is a nonempty set of datacenter geolocations of v^i_{res},
- $\mathcal{P}(v^i_{res})$ is the unit execution cost of v^i_{res},

TABLE 1: A summary of symbols used in the paper

Symbols	Description
$\mathbb{G}_{web} = (\mathbb{V}_{web}, \mathbb{E}_{web})$	Graph of GWA
$\mathbb{G}_{res} = (\mathbb{V}_{res}, \mathbb{E}_{res})$	Fully connected graph of cloud provider hosts
$v^i_{res} \in \mathbb{V}_{res}$	Host i of $\mathbb{G}_{res}$
$h_{v^i_{res}} \in \mathbb{H}$	Host type for host v^i_{res}
$loc_{v^i_{res}} \in \mathbb{L}$	Datacenter geo-location for host v^i_{res}
$\theta \in \Theta$	A deployment mapping/solution in the set of solutions Θ
$\mathcal{P}(\theta^{\mathbf{h}}(v^i_{web}))$	Unit execution cost for host mapping $\theta^{\mathbf{h}}(v^i_{web})$
$\mathcal{O}(v^i_{web})$	Total execution time of v^i_{web}
$\mathcal{N}(\theta^{\mathbf{h}}(v^i_{web}), \theta^{\mathbf{h}}(v^j_{web}))$	Unit data transfer cost from $\theta^{\mathbf{h}}(v^i_{web})$ to $\theta^{\mathbf{h}}(v^j_{web})$
$\mathcal{D}(v^i_{web}, v^j_{web})$	Size of data transferred from v^i_{web} to v^j_{web}
$\mathcal{C}^{\theta}_{exe}, \mathcal{C}^{\theta}_{com}$	Execution cost and transfer cost for a solution θ
$\mathcal{C}^{\theta}$	Total cost of a solution θ
Θ'	Optimal subset of Θ
$\mathcal{B}$	User's budget
s	Population of particle
$\mathbb{P}, \mathbb{P}^{local}, \mathbb{V}el$	Set of position, local best position and velocity of particles
$\mathbb{P}^{global}_{best}$	Global best position vector
ζ	Optimisation objective (maximise or minimise)
$\mathbb{P}_{final}$	Output solution set
$\eta = \lvert\mathbb{P}\rvert_{final}$	Number of output solution
ω	Inertia weight
$C1, C2$	Self recognition and social constant factor
K, K_{max}	Optimal and maximum cluster size
$\mathbb{C}lus = \{Clus_1, \ldots, Clus_K\}$	K disjoint clusters
u_j	Data point representing cluster center for $Clus_j$
a_i	Data point representing host v^i_{res}

- $\mathcal{N}(v^i_{res}, v^j_{res})$ is the unit data transfer cost from v^i_{res} to another host $v^j_{res} \in \mathbb{V}_{res}$.
  Note that we may have $\mathcal{N}(v^i_{res}, v^j_{res}) \neq \mathcal{N}(v^j_{res}, v^i_{res})$.

To model deployments of web applications to cloud providers, we use *deployment* mappings $\theta : \mathbb{V}_{web} \rightarrow \mathbb{V}_{res} \times \mathbb{L}$ and denote $\theta(v^i_{web}) = (\theta^{\mathbf{h}}(v^i_{web}), \theta^{\mathbf{l}}(v^i_{web}))$, for every GWA component v^i_{web}. We also require that the following are satisfied, for all distinct GWA components v^i_{web} and v^j_{web}:

- $\theta^{\mathbf{l}}(v^i_{web}) \in loc_{\theta^{\mathbf{h}}(v^i_{web})}$,
- $h_{\theta^{\mathbf{h}}(v^i_{web})} = h_{\theta^{\mathbf{h}}(v^j_{web})}$,
- $\theta^{\mathbf{h}}(v^i_{web}) \neq \theta^{\mathbf{h}}(v^j_{web})$ and $\theta^{\mathbf{l}}(v^i_{web}) \neq \theta^{\mathbf{l}}(v^j_{web})$.

The set of all deployment mappings is denoted by Θ. Figure 2 shows an example of one of the solutions θ.
Note that, for a deployment mapping, the host types of hosts allocated to all the GWA components do not differ. Moreover, the hosts and locations allocated to distinct components must be different.

Table 2 shows the possible number of deployment mappings by varying the numbers of GWA components and hosts, assuming that there is only one host type, and that each host has only one geo-location (in such a case, there are exactly $n! \cdot \binom{m}{n} = n! \cdot \frac{m!}{n!(m-n)!} = \frac{m!}{(m-n)!}$ different deployment mappings). Note that even for relatively small numbers of GWA components $n = 30$ and hosts $m = 50$, the number of possible deployment mappings is overwhelming, and so the selection of optimal deployment mappings cannot be done by any brute-force approach.

3.2 Problem formulation

Our approach is first to identify a possibly widest range of deployment options with varying host types and locations adhering to the available budget, and then to benchmark these to obtain an optimal deployment. We formulate such a *two-phase* approach in the following way.

Phase I. We aim to find the largest set of deployment mappings with the total cost within a given budget $\mathcal{B}$.

For each deployment mapping $\theta \in \Theta$, we consider two types of costs (in terms of time), namely the *execution* cost

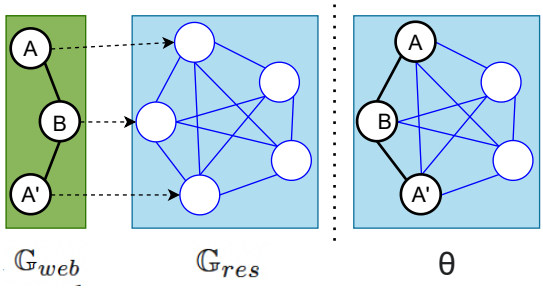


Fig. 2: An example of the deployment mapping θ between $\mathbb{G}_{web}$ and $\mathbb{G}_{res}$

$\mathcal{C}^{\theta}_{exe}$, and the *communication* cost $\mathcal{C}^{\theta}_{com}$, as shown in Eq.1, where $\mathcal{O}(v^i_{web})$ is the total execution time for a GWA component v^i_{web} and $\mathcal{D}(v^i_{web}, v^j_{web})$ is the size of data transferred from a GWA component v^i_{web} to another GWA component v^j_{web}. The cost model is based on the Pay-As-You-Go pricing model. The choice of the Pay-As-You-Go model is based on the wide adoption by all cloud providers, which makes the cost model generalised.

$$\begin{aligned} \mathcal{C}^{\theta}_{exe} &= \sum_{i=1}^{n} \mathcal{O}(v^i_{web}) \cdot \mathcal{P}(\theta^{\mathbf{h}}(v^i_{web})) \\ \mathcal{C}^{\theta}_{com} &= \sum_{i,j=1 \wedge i \neq j}^{n} \mathcal{D}(v^i_{web}, v^j_{web}) \cdot \mathcal{N}(\theta^{\mathbf{h}}(v^i_{web}), \theta^{\mathbf{h}}(v^j_{web})) \\ \mathcal{C}^{\theta} &= \mathcal{C}^{\theta}_{exe} + \mathcal{C}^{\theta}_{com} \end{aligned} \tag{1}$$

We then formulate our goal as an optimisation problem given by Eq. 2 together with the formulas for deployment costs given by Eq. 1.

$$\begin{aligned} &\textbf{maximize:} \quad |\Theta'| \quad \textit{where } \Theta' \subseteq \Theta \\ &\textbf{subject to:} \quad \sum_{\theta \in \Theta'} \mathcal{C}^{\theta} \leq \mathcal{B} \end{aligned} \tag{2}$$

Phase II. In this phase, the set of deployment mappings Θ' obtained in Phase I is deployed and executed on a real-cloud environment with the execution results being collected. The deployment mapping with the "minimal" response time is then selected and used for the actual deployment of GWA components.

4 GeoDeploy architecture

To address the problem outlined in Section 3.2, we designed GeoDeploy. This section summarises the basic architecture.

TABLE 2: Number of deployment mappings generated by varying the number of GWA components and the number of hosts

GWA comp.	Number of hosts				
	30	35	40	45	50
5	2.E+07	4.E+07	8.E+07	1.E+08	3.E+08
10	1.E+14	7.E+14	3.E+15	1.E+16	4.E+16
15	2.E+20	4.E+21	5.E+22	5.E+23	3.E+24
20	7.E+25	8.E+27	3.E+29	8.E+30	1.E+32
25	2.E+30	3.E+33	6.E+35	5.E+37	2.E+39
30	3.E+32	9.E+37	2.E+41	9.E+43	1.E+46

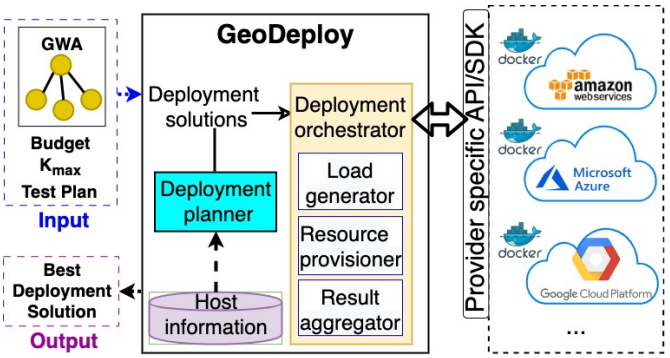


Fig. 3: System Architecture of GEODEPLOY

GEODEPLOY is a user-centric and cost-effective middleware that aims to offer a full-stack deployment solution for GWAs. It is a comprehensive integration tool that receives the test GWA, Budget, maximum cluster size and the test plan as input, generates an optimal set of solutions and automates the deployment of GWA solutions on multiple clouds. The final outcome is the best deployment solution with a minimal response time. There are two main components of GEODEPLOY: *deployment planner* and *deployment orchestrator*, as depicted in Fig. 3.

A. Deployment planner. When GWA, budget and the maximum cluster size are submitted to GEODEPLOY, the *deployment planner* utilises this information to generate a set of deployment solutions. It uses the pre-collected cloud host information and applies either the proposed algorithm (described in Section 6) or other baseline methods such as *Random Selection* and *Greedy* approach.

B. Deployment orchestrator. After receiving the set of deployment solutions generated by the *deployment planner*, it performs a sanity check to validate the solutions. It then automatically deploys the solutions in the real cloud environment. First, the *Resource provisioner* extracts the host details and geo-location and initiates a host. The framework has predefined APIs for each host type and cloud service provider which helps to initiate the host and establish a network connection. All this is performed automatically and is hidden from the user. It then starts the *Load generator* from the pool of host information stored which act as clients for benchmarking. Each client is specified with a *test plan* provided by the user. The test plan contains the request sending rate, request types, the number of total requests and execution time. After starting the clients, communication is established between the clients and the GWA hosts and the tests are performed. Finally, the *Result aggregator* collects the results including users' response time and system statistics, and then returns a suitable deployment solution for the given GWA. More details of the deployment process are provided in Section 6.5.

5 ADAPTIVE PSO ALGORITHM

Particle Swarm optimisation (*PSO*) is a well-known optimisation algorithm used to find a solution in a multidimensional space [35]. However, in the high-dimensional space, the convergence rate is low with the chance of falling into local optima. Numerous updated versions of *PSO* have been proposed for task scheduling in the cloud environment, e.g., Nested PSO [36], Multitask PSO [37].

In this section, we propose *Adaptive Particle Swarm Optimisation (APSO)* algorithm which is able to find a solution without getting stuck in the local optima. The primary novelty of *APSO* lies in its adaptive mechanism, which dynamically adjusts its parameters during optimisation based on problem characteristics and search progress. This contrasts with traditional algorithms like Genetic Algorithms (*GA*) and Ant Colony Optimisation (*ACO*) that employ fixed parameters, potentially hindering their performance in complex multi-cloud scenarios. Compared to these traditional algorithms, *APSO* offers several advantages in terms of scalability, solution quality, and convergence speed as given below:

- Scalability: *APSO*'s adaptive nature empowers it to efficiently handle large-scale optimisation problems with numerous decision variables and constraints, a common challenge in multi-cloud environments. Traditional algorithms like *GA* and *ACO* may struggle with scalability, leading to increased computational costs or suboptimal performance.
- Quality: By balancing exploration and exploitation through its adaptive mechanism, *APSO* frequently discovers higher-quality solutions, often reaching global or near-optimal configurations. This surpasses algorithms like *GA* and *ACO*, which can suffer from premature convergence or slower convergence rates, resulting in less optimal solutions.
- Convergence: *APSO*'s adaptive parameter tuning and efficient search space navigation often lead to faster convergence compared to *GA* and *ACO*. This advantage translates to quicker solutions, particularly valuable in time-sensitive optimisation tasks within multi-cloud environments.

Alg.1 illustrates the key steps of the proposed *APSO*. In concrete terms, we first initialise three sets of vectors, each having s elements:

$$\begin{aligned} \mathbb{P} &= \{\vec{P}_1, \ldots, \vec{P}_s\} \\ \mathbb{P}^{local} &= \{\vec{P}_1^{local}, \ldots, \vec{P}_s^{local}\} \\ \mathbb{V}el &= \{\vec{Vel}_1, \ldots, \vec{Vel}_s\} \end{aligned}$$

$\mathbb{P}$ records the set of current deployment solutions; each $\vec{P}_i$ represents a deployment mapping θ_i. $\mathbb{P}^{local}$ stores the best deployment solution obtained for each $\vec{P}_i$ during any runtime iteration. In $\mathbb{V}el$, each $\vec{Vel}_i$ represents the vector variable to update the value of $\vec{P}_i$.

We initialise $\vec{P}_i$ by randomly mapping a host from $\mathbb{V}_{res}$ for each component of the GWA. The generated $\vec{P}_i$ is checked for the validity of the deployment mapping using a *Valid Particle Generator (VPG)* as specified by the constraints in Section 3.2 and retained only if it is found valid (Alg.1 Line 4-6). Otherwise, it is discarded and, eventually, a valid deployment mapping is derived. Then, $\mathbb{P}^{local}$ is assigned $\mathbb{P}$ for all s components.

To allow a uniform update of the current deployment solution $\vec{P}_i$ to a new $\vec{P}_i^{new}$, $\vec{Vel}_i$ is initialised to a set of randomly generated integer values between $-|\mathbb{V}_{res}|$ and $|\mathbb{V}_{res}|$ (Alg.1 Line 8).

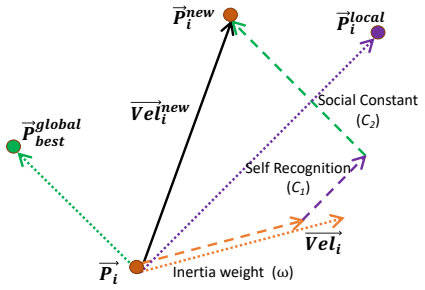


Fig. 4: Movement of a solution P_i in $APSO$

Next, the cost $\mathcal{C}_i$ of each $\vec{P}_i$ is computed by using Eq.1 (Alg.1 Line 10). The cost $\mathcal{C}_i$ represents the *fitness function* for $APSO$. Based on the obtained cost values, we can select η set of solutions $\mathbb{P}_{final}$ as shown in Alg.1 Line 13. An additional parameter ζ is added which represents the objective of the algorithm (i.e., find a list of deployment solutions that $maximises$ or $minimises$ the fitness function and η). We choose the $\vec{P}_i$ with the optimal cost value as the global best solution $\vec{P}_{best}^{global}$ (Alg.1 Line 15).

To find better deployment solutions (update $\vec{P}_i$ to $\vec{P}_i^{new}$) $APSO$ uses *four* vector variables, $\vec{P}_i$, $\vec{Vel}_i$, $\vec{P}_i^{local}$ and $\vec{P}_{best}^{global}$. First, it computes $\vec{Vel}_i^{new}$, which is the key variable to update $\vec{P}_i$, as *three* variables influence it, as follows. ω defines how much $\vec{Vel}_i$ can contribute to the updating process. $C1$ and $C2$ are the factors which affect similarity between $\vec{P}_i^{new} - \vec{P}_i^{local}$ and $\vec{P}_i^{new} - \vec{P}_{best}^{global}$ respectively. Two randomly generated variables $F1$ and $F2$ are used to reduce the bias introduced by $\vec{P}_i^{local}$ and $\vec{P}_{best}^{global}$. Finally, the search space is restricted by using a modulo operation with $|\mathbb{V}_{res}|$. Eq.3 summarises the changes:

$$\begin{aligned} \vec{Vel}_i^{new} \quad = \quad & (\omega \cdot \vec{Vel}_i \\ & + C1 \cdot F1 \cdot (\vec{P}_i^{local} - \vec{P}_i) \\ & + C2 \cdot F2 \cdot (\vec{P}_{best}^{global} - \vec{P}_i)) \mod |\mathbb{V}_{res}| \end{aligned} \tag{3}$$

Despite the novelty of our $APSO$ approach, the fundamental tasks performed by variables $C1$ and $C2$ in guiding particle movement and convergence remain unchanged. Therefore, to maintain alignment with established methodologies and ensure comparability with prior research, we opted to retain the variable values to *two* as those used in past works [38]. The values of $F1$ and $F2$ are generated randomly between 0 and 1 following a uniform distribution. The new deployment solution $\vec{P}_i^{new}$ is computed using the existing solution $\vec{P}_i$ and the new $\vec{Vel}_i^{new}$, as shown in Eq.4:

$$\vec{P}_i^{new} = (\vec{P}_i + \vec{Vel}_i^{new}) \mod |\mathbb{V}_{res}| \tag{4}$$

The new deployment solutions are checked and the cost for the valid solutions is calculated. If the new cost $\mathcal{C}_i^{new}$ is better than the fitness value of $\vec{P}_i^{local}$ and $\vec{P}_{best}^{global}$, it should be updated (Alg.1 Line 26-29). Finally, we update the solutions $\mathbb{P}_{final}$ by replacing the existing solutions with those in $\vec{P}^{new}$ which have better cost values (Alg.1 Line 31). The described steps (Alg.1 Line 16-29) are repeated until the termination condition is met.

Termination. In this paper, *terminate* is set to *true* after 100 iterations or if there is no updating of the $\vec{P}_{best}^{global}$ in 30 iterations. After the termination, $\mathbb{P}_{final}$ is returned as output.

Algorithm 1: Adaptive PSO

Input: $\mathbb{G}_{web} = (\mathbb{V}_{web}, \mathbb{E}_{web})$ - dependency graph of the web-application, $\mathbb{G}_{res} = (\mathbb{V}_{res}, \mathbb{E}_{res})$ - connection graph of cloud providers host, ζ - optimization parameter
Output: $\mathbb{P}_{final}$ - list of η resulting solutions

1 s - population size
2 **for** $i = 1, \ldots, s$ **do**
3 // Initialize $\mathbb{P}$, $\mathbb{P}^{local}$ and $\mathbb{V}el$
4 $\vec{P}_i \leftarrow$ `rand` $(0, |\mathbb{V}_{res}|)$
5 // Validate $\vec{P}_i$
6 $\vec{P}_i \leftarrow$ `VPG` $(\vec{P}_i)$
7 $\vec{P}_i^{local} \leftarrow \vec{P}_i$
8 $\vec{Vel}_i \leftarrow$ `rand` $(-|\mathbb{V}_{res}|, |\mathbb{V}_{res}|)$
9 // Compute the fitness function $\mathcal{C}_i$ using Eq. 1
10 $\mathcal{C}_i \leftarrow \mathcal{C}_{exe}^{\vec{P}_i} + \mathcal{C}_{com}^{\vec{P}_i}$
11 **end**
12 //Initialize $\mathbb{P}_{final}$
13 $\mathbb{P}_{final} \leftarrow$ `sort` $(\mathbb{P}, \mathcal{C}, \zeta)[0, \eta - 1]$
14 // Initialize the global best solution
15 $\vec{P}_{best}^{global} \leftarrow \mathbb{P}_{final}[0]$
16 //Repeat until the termination
17 **while** ! *terminate* **do**
18 //Update the solutions
19 **for** $i = 1, \ldots, s$ **do**
20 $\vec{P}_i^{new}, \vec{Vel}_i^{new} \leftarrow$ `update` $(\vec{P}_i, \vec{Vel}_i, \vec{P}_i^{local}, \vec{P}_{best}^{global})$
21 // Validate the updated solution $\vec{P}_i^{new}$
22 $\vec{P}_i^{new} \leftarrow$ `VPG` $(\vec{P}_i^{new})$
23 // Compute the fitness function $\mathcal{C}_i^{new}$ using Eq. 1
24 // update the local best solution
25 **if** $\mathcal{C}_i^{new}$ *is better than* $\mathcal{C}_i$ **then**
26 $\vec{P}_i^{local} = \vec{P}_i^{new}$
27 **if** $\mathcal{C}_i^{new}$ *is better than* $\mathcal{C}_{best}^{global}$ **then**
28 //update the global best solution
29 $\vec{P}_{best}^{global} = \vec{P}_i^{new}$
30 **end**
31 //Update the output
32 $\mathbb{P}_{final} \leftarrow$ `Replace`$(\vec{\mathbb{P}}^{new}, \mathbb{P}_{final})$
33 **end**

6 Optimising the Deployment

This section presents a detailed description of the proposed approach which consists of the following four main components: *(A) clustering*, *(B) budget allocation*, *(C) deployment solution generation*, and *(D) benchmarking in real-world environments*.

6.1 Overview of executing GeoDeploy

To obtain the "best" deployment solution, each component is executed sequentially as shown in Alg. 2 and Fig. 5. An input set should be provided, in which the GWA and benchmarking budget $\mathcal{B}$ are provided by the user. The host information $\mathbb{V}_{res}$ including the CPU, memory, location and price, about each provider is pre-collected. In the first step, we partition the hosts into K clusters (see Alg. 2 Line

Algorithm 2: GEODEPLOY's algorithm overview

Input: $\mathbb{G}_{web} = (\mathbb{V}_{web}, \mathbb{E}_{web})$ - dependency graph of the web-application, $\mathbb{G}_{res} = (\mathbb{V}_{res}, \mathbb{E}_{res})$ - connection graph of cloud providers host, K_{max} - maximum cluster size, $\mathcal{B}$ - User's Budget, TP - Test plan,

Output: θ - obtained solution

1 // Partition $\mathbb{V}_{res}$ hosts into K clusters
2 $K, \mathbb{C}lus \leftarrow Clustering(K_{max}, V_{res})$
3 // Allocate $\mathcal{B}$ to K clusters
4 `allocateBudget`($\mathbb{C}lus$, $\mathcal{B}$)
5 // Generate solutions for each cluster
6 $\Theta \leftarrow$ `genSolution` ($Clus_i$, $\mathcal{B}_i$) for each $i \leq K$
7 // Benchmark the selected solutions and choose the best solution
8 $\theta \leftarrow$ `Benchmark` (Θ, TP)

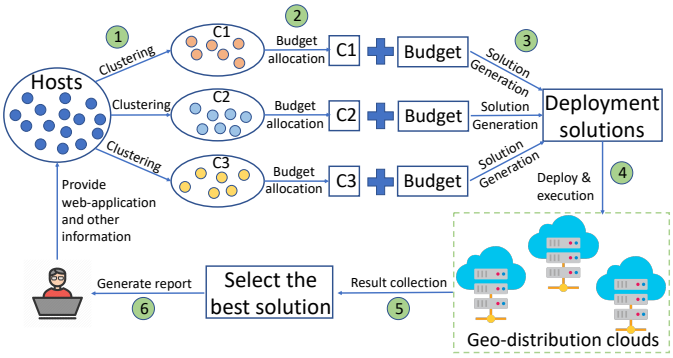


Fig. 5: The execution workflow of the proposed method

2). Next, the given budget $\mathcal{B}$ is allocated to K clusters in step 2 (Alg. 2 Line 4). In step 3, based on the allocated budget, clustered hosts and other provided information, a set of deployment solutions is generated (see Alg. 2 Line 6). The obtained solutions are automatically deployed and executed on the corresponding hosts (step 4). Finally, the benchmarking results are collected and then a report is generated and sent back to the user (illustrated in Alg. 2 Line 8). The following subsections indicate the technical details of each key component.

6.2 Clustering

Our first step is to create a set of clusters to partition the nodes in $\mathbb{V}_{res}$, i.e., each node $v^i_{res} \in \mathbb{V}_{res}$ is mapped to exactly one cluster with all the nodes belonging to the same cluster bearing similar host type characteristics ($h \in \mathbb{H}$). Each host type h is associated with many aspects of resources such as CPU, memory, storage, bandwidth, etc. In this paper, the clustering algorithm considers CPU and memory properties. We employed K-MEANS algorithm to create K disjoint clusters $\mathbb{C}lus = \{Clus_1, \ldots, Clus_K\}$ [39].

Note that the parameter K needs to be found and input to the K-MEANS algorithm, and determining K is essential to achieve optimal partitioning. We employed the *Elbow Method* to find this value [40]. It receives K_{max} as input and starts with different values of $k \in \{0, 1, ..., K_{max}\}$ and computes a total *Intra Cluster Variation* ICV_k for each cluster size k (Alg. 3 Line 6). The difference between ICV_k and

Algorithm 3: Clustering algorithm

Input: K_{max} - maximum cluster size, $\mathbb{V}_{res}$ - cloud hosts

Output: $\mathbb{C}lus = \{Clus_1, \ldots, Clus_K\}$ - obtained clusters

1 // Apply Elbow method to get the optimal K
2 **for** $k = 2, \ldots, K_{max}$ **do**
3 | // Apply K-MEANS
4 | $Clus_k \leftarrow$ K-MEANS($\mathbb{V}_{res}, k$)
5 | // Compute Intra-Cluster Variation
6 | $ICV_k \leftarrow \sum_{j=1}^{k} \sum_{a_i \in Clus_j} ||a_i - u_j||^2$
7 **end**
8 //Find the knee point
$\Delta ICV_{k,k+1} \leftarrow |ICV_{k+1} - ICV_k|$ for each $k < K_{max}$
9 $K = k(kneed_{k,k+1}(\Delta ICV_{k,k+1}))$
10 // Find the final cluster
11 $\mathbb{C}lus \leftarrow$ K-MEANS($\mathbb{V}_{res}, K$)

Function: K-MEANS ($\mathbb{V}_{res}, K$)
1 // Initialise cluster center
2 $u_k \leftarrow$ `rand` (v_i) for $v_i \in \mathbb{V}_{web}$ and $k \in K$
3 **while** *! terminate* **do**
4 | // v_i is allocated to a cluster
5 | **for** *each* $v_i \in \mathbb{V}_{web}$ **do**
6 | | $c_{ik} \leftarrow \begin{cases} 1 \ if \ \ l = arg \ \min_j ||v_i - u_k||^2 \\ 0 \ otherwise \end{cases}$
7 | **end**
8 | //Update the cluster center
9 | **for** $k = 1, \ldots, K$ **do**
10 | | $u_k \leftarrow \frac{\sum_i (c_{ik} \times v_i)}{\sum_i c_{ik}}$
11 | **end**
12 **end**

ICV_{k+1} is determined for all $k, k+1$ and the knee point is detected using *kneed* library (Alg. 3 Line 9). The resulting value of K represents the optimal cluster number.

K-means clustering is then performed on the obtained cluster size K. Initially, each cluster $Clus_k$ is assigned to an arbitrary cluster center u_k (Alg. 3 Line 2). A data point v_i including CPU and memory information is allocated to a cluster $Clus_k$ based on the closeness to the cluster center u_k as shown in Alg. 3 Line 6. Here, $||\cdot||$ represents the function (Euclidean function) to measure the distance between the data point v_i and the cluster center u_k. The value c_{ik} can be either 1 if v_i is allocated to cluster $Clus_k$ or 0 otherwise. With each iteration, u_k is also updated (Alg. 3 Line 10). These steps are repeated until the termination condition reaches which in our scenario is the cluster center u_j remains unchanged.

The pseudo-code for the clustering algorithm is given in Algo.3. It is worth mentioning that the clustering stage is performed only once on a host data set. The stage is repeated only when the dataset is changed.

6.3 Budget allocation

As we discussed above, deployment solutions are generated for each cluster, so it is necessary to fairly distribute the given budget $\mathcal{B}$ to the clusters $\mathbb{C}lus$.

Algorithm 4: Budget allocation

Input: $Clus_1, \ldots, Clus_K$ - clusters of hosts, $\mathcal{B}$ - user's Budget
Output: $\mathcal{B}_1, \ldots, \mathcal{B}_K$ - clustered budget

1 **for** $k = 1, \ldots, K$ **do**
2 | // Get solution with maximum total cost
3 | $\Theta^{max} \leftarrow APSO(Clus_k, \zeta = maximization)$
4 | $\theta_k^{max} = \Theta^{max}[0]$
5 | // Get solution with minimum total cost
6 | $\Theta^{\min} \leftarrow APSO(Clus_k, \zeta = minimization)$
7 | $\theta_k^{min} = \Theta^{min}[0]$
8 | // Compute the average cost $\mathcal{C}_{av}^k$
9 | $\mathcal{C}_{av}^k = \frac{\mathcal{C}_{\theta_k^{max}} + \mathcal{C}_{\theta_k^{min}}}{2}$
10 **end**
11 // Distribute the budget $\mathcal{B}$
12 $\mathcal{B}_k = \frac{\mathcal{C}_{av}^k}{\sum_{i=1}^{K} \mathcal{C}_{av}^i} \cdot \mathcal{B}$ for each $k \leq K$

Since the unit execution cost $\mathcal{P}(v_{res}^i)$ of host v_{res}^i varies from one cluster to another, it is not suitable to divide the budget $\mathcal{B}$ equally among the clusters. The average cost for benchmarking a cluster is a good reference for budget distribution to ensure fairness. However, evaluating the average cost for a cluster requires computing the cost of each possible solution which is not feasible otherwise. Alternatively, we use the mean of solutions which have the maximum and minimum cost as the reference to distribute the budget $\mathcal{B}$. To this end, we interact our Alg. 4 with the *APSO* (discussed in Section 5). For each cluster $Clus_k \in \mathbb{C}lus$, we first obtain a solution with maximum cost $\mathcal{C}_{max}^k$ and minimum cost $\mathcal{C}_{min}^k$ by setting the optimisation constraint ζ as *maximisation* and *minimisation* respectively (see Alg. 2 Line 3-7). Next, the average cost for each cluster k, $\mathcal{C}_{av}^k$ is computed by normalising the maximum and minimum cost (Alg. 2 Line 9). Finally, the user's budget $\mathcal{B}$ is distributed to each cluster $Clus_k$ using the equation in (Alg. 2 Line 12) and $\mathcal{B}_k$ is returned.

6.4 Deployment solution generation

In this stage, the cluster budget $\mathcal{B}_k$ is distributed to find a set of solutions for each cluster $Clus_k$ as given in Alg. 5. First, we generate η set of solution $\mathbb{P}_k$ for each cluster k using the *APSO* algorithm (Line 3). Next, each solution $\mathbb{P}_k[i_k]$ is added individually to a final list Θ_k^{fin} till no solution can be added in the remaining budget $\mathcal{B}_k^{left}$ (Line 6). To maximise the utilisation of the budget, we combine all the remaining cluster budget $\mathcal{B}_k^{left}$ (Line 12). Later, the budget $\mathcal{B}^{left}$ is allocated to each cluster in descending order based on its reference (refer to Alg. 2 Line 9) to compute more solutions (Line 14-18). Finally, the clustered solutions Θ_k^{fin} from each k cluster are merged to get the final list of solution Θ' for the real deployment.

6.5 Benchmarking in real-world environments

The solutions Θ' obtained in the previous step are deployed using the *deployment orchestrator* discussed in Section 4. To execute an optimised solution θ, first the *resource provisioner*

Algorithm 5: *Solution generation*

Input: $Clus_1, \ldots, Clus_K$ - clusters of hosts, $\mathcal{B}_1, \ldots, \mathcal{B}_K$ - clustered budget
Output: Θ' - optimized list of hosts

1 // Get a set of solution Θ_k for each cluster k
2 **for** $k = 1, \ldots, K$ **do**
3 | $\mathbb{P}_k = \texttt{APSO}(Clus_k, \zeta = minimization)$
4 | $\mathcal{B}_k^{temp} = 0, \Theta_k = [], i_k = 0$
5 | **while** $(\mathcal{B}_k^{temp} \leq \mathcal{B}_k)$ **do**
6 | | $\Theta_k^{fin} \leftarrow \mathbb{P}_k[i_k]$
7 | | $\mathcal{B}_k^{temp} \leftarrow \mathcal{B}_k^{temp} + \mathcal{C}^{\mathbb{P}_k[i_k]}$
8 | | $i_k \leftarrow i_k + 1$
9 | **end**
10 **end**
11 // Compute the remaining budget $\mathcal{B}^{left}$
12 $\mathcal{B}^{left} \leftarrow \sum_{k=1}^{K} (\mathcal{B}_k - \mathcal{B}_k^{temp})$
13 // Utilize $\mathcal{B}^{left}$ to add more solutions
14 **for** $k = K, \ldots, 1$ **do**
15 | **if** $\mathcal{B}^{left} \leq \mathcal{C}^{\mathbb{P}_k[i_k]}$ **then**
16 | | $\Theta_k^{fin} \leftarrow \mathbb{P}_k[i_k]$
17 | | $\mathcal{B}^{left} \leftarrow \mathcal{B}^{left} - \mathcal{C}^{\mathbb{P}_k[i_k]}$
18 | | $i_k \leftarrow i_k + 1$
19 | **end**
20 **end**
21 // Merge the solutions Θ_k^{fin} to get the final list Θ'
22 $\Theta' \leftarrow \Theta_k^{fin}$ for each $k \leq K$

extracts the host information including instance configurations (h, loc and p). Next, communication is established with the instance. The communication is performed using cloud-specific APIs defined in the *Go* language. After the successful initialisation of GWA hosts, each host is installed with the specific Docker images and communication is established between various GWA components.

A similar process is followed to initiate the client hosts required for the *load generator*. The *deployment orchestrator* is notified with the successful establishment of GWA and client hosts. Following this, the state of deployment along with the IP address of the hosts is returned.

With the successful establishment of GWA hosts, the client hosts are launched with the defined *test plan* and the IP address of the web server GWA components. Since there are multiple web server GWA components, a simple *geo-location rule-based load generation* was developed, i.e., the client will load the nearest web server host in terms of geo-location. The client hosts are triggered to load the web server hosts and the execution starts.

The completion of an experiment is reported to the *deployment orchestrator* and the *result aggregator* retrieves the results from the clients. The hosts are terminated and the cloud resources are released. Finally, the *result aggregator* compares all the results and returns the best solution θ^{best} to the user.

6.6 Complexity analysis

This section computes the worst-case time complexity of our proposed framework, GeoDeploy. GeoDeploy consists of

four main components as given in Fig.5, the complexity of *clustering*, *benchmarking in real-world environment* is *O(1)*. The complexity of *budget allocation* and *deployment solution generation* depends on the complexity of *APSO*. In our case, the complexity of *APSO* depends on the population space $s << |\mathbb{V}_{res}|$, number of GWA components $|\mathbb{V}_{web}|$ and the maximum number of iteration before termination κ is given as $O((s + s\log s + \kappa) \times |\mathbb{V}_{web}|)$. Here, κ and s are constant thus reducing the complexity of *APSO* to be $O(|\mathbb{V}_{web}|)$.

Since the number of clusters K is a constant, the complexity of *budget allocation* and *deployment solution generation* are also equal to $O(|\mathbb{V}_{web}|)$. Therefore, the overall worst-case complexity of our proposed approach is $O(|\mathbb{V}_{web}|)$. The result is also validated with real experiments as given in Section 7.4.

7 Evaluation

In this section, we evaluate GeoDeploy, aiming to assess the quality and quantity of solutions obtained at each step of the proposed approach from both algorithmic and deployment perspectives. Initially, we conducted numerical analysis on real data collected from various cloud providers, resulting in a refined set of deployment solutions subsequently implemented in a real multi-cloud environment. Our comparison involved evaluating the diversity and scalability of our proposed algorithm in contrast to baseline methods. Our experiment section is divided into the following subsections. In Section 7.1, we discuss the experiment setup detailing the benchmark application, load generator, and the comparative evaluation methodology. Section 7.2 presents the results of numerical analysis showing the variation of solutions generated and budget utilised by GeoDeploy. It also highlights the diversity of results obtained by GeoDeploy compared to the state-of-the-art methods. Section 7.3 provides insights from the real experiment, offering a comprehensive view of the project's performance in practical scenarios. Finally, to demonstrate the effectiveness of the proposed in a scaled environment, Section 7.4 presents the scalability test results.

7.1 Experiment setup

Dataset and benchmark application. We considered three main cloud providers i.e., AWS[1], Microsoft Azure[2] and Google Cloud[3] and for each of them we selected four datacenters located in *UK South (London)*, *US West (Oregon)*, *South America (Sao Paulo)* and *Asia Pacific (Singapore)*. As a result, the input of our algorithm includes 776 host configurations together with their execution and communication costs, which are available on GitHub[4].

We consider a simple GWA which consists of three components; one central database and two web servers. Each web server is associated with its local database. The web servers are configured to handle both $text$ and $image$ data, with 50% of the data distributed across the two servers, while the entire dataset is stored in the central database. All these components are dockerised and the images are stored on the Docker hub.

Load Generator. In our evaluation, we used a load generator to emulate realistic workloads. The load generator was designed to mimic the statistical properties of real-world application workloads. The load generator is implemented as a web application deployed on AWS $t2 - medium$ *UK South (London)* datacenter. A pool of 50 users is emulated depending on the given datacenter geo-location. Each user is specified to execute and generate load from any given geographical area. To access the GWA application, a random set of users is selected based on a *normal distribution*. Again, the users are mapped to the closest web-server using *geo-location rule-based load generation* as specified in Section 6.5.

Evaluation methodology. Since *Random* and *Greedy* approaches are the most straightforward solutions directly applicable to such optimisation problem [41], [42], this paper considers them as a baseline approach and compares the proposed approach with them. More advanced algorithms can be adopted to work with our system in the future. We compare our algorithm with the following three adaptive **baseline** algorithms:

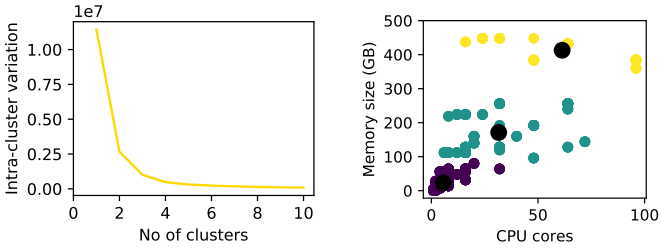
- *Random Selection (Random)* – we randomly select the solution until the budget is exhausted.
- *Greedy with Computation Cost (Greedy)* – we select the solutions starting from the cheapest computation cost until the budget is exhausted. The fitness function is $\Theta' \leftarrow \sum_k \theta_k^{min}$ where, θ_k^{min} is a valid solution with cheapest computation cost. In each round, the selected solution is popped from the list and the step is repeated till the budget is available.
- *Clustered Greedy with Computation Cost. (Clus.Greedy)* – we first create K clusters and for each cluster. we apply the Greedy strategy to obtain the solutions.

7.2 Numerical analysis

Environment settings. We evaluated GeoDeploy and the baseline comparison algorithms on a PC with Intel(R) Core(TM) i5-6200U CPU @2.3GHz - 2.4GHz with 16 GB memory and 512 GB SSD.

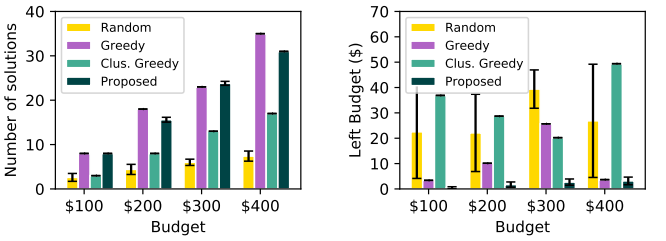
Parameter settings. We set the same inputs for all the evaluated algorithms. The *budget* is varied from $100 to $400. The selected budget of $100 to $400 strikes a balance between affordability and effectiveness, allowing for a range of testing activities while remaining mindful of budgetary considerations. To find the data transfer size, we performed a simple analysis in a controlled environment. The result shows that each user request consumes 25–500 KB per second. For evaluation, we assume that approximately 50–200 users attempt to access a component of the GWA in any given hour. This is randomly chosen based on the average number of users accessing a web server for a small organisation. Combining these two values, the *size of the data* transferred among the components of a GWA can vary between 4.3 and 344 GB/hour. Following this result, in our evaluation, we set the *size of the data* transferred among the components of a GWA via randomly selecting values between 4.3 and 344 GB/hour.

1. https://aws.amazon.com/ec2/pricing/on-demand/
2. https://azure.microsoft.com/en-gb/pricing/details/virtual-machines/linux/
3. https://cloud.google.com/compute/vm-instance-pricing
4. https://github.com/DNJha/GeoDeploy



(a) Elbow Method (b) K-means Clustering Result

Fig. 6: Clustering the given data to find the best cluster size (a) and clustering result (b).



(a) Number of solutions generated (b) Average total unutilized budget

Fig. 7: Numerical evaluation result comparing the number of solutions generated and the unutilized budget. Black bar on top represents the standard deviation.

7.2.1 Clustering

We apply a clustering technique to partition the host configurations into distinct clusters, a crucial preliminary step in our approach aimed at enhancing diversity. To maintain a manageable size for each cluster and facilitate effective analysis, we set the *max cluster size* at 10. Utilising the *Elbow method*, an intermediate step in our clustering process, we analyse the intra-cluster variation, as illustrated in Fig. 6a. This analysis shows that 3 represents the optimal cluster size, subsequently guiding our clustering procedure. With this optimal value determined, we proceed to cluster our data using $K = 3$. The resulting clusters, along with their respective centroids denoted by black circles, are visualised in Fig. 6b. This comprehensive clustering approach enables us to effectively manage and analyse the diverse host configurations within our system.

7.2.2 Number of solutions vs. budget

In this subsection, we show the number of solutions generated by each approach. Fig. 7a shows that our algorithm obtained more solutions compared to *Random* and *Clus.Greedy* for all the cases. On average our algorithm generates 72.5% more solutions than *Random* and 40.5% more solutions than *Clus.Greedy*. Moreover, the proposed algorithm is comparable with the *Greedy approach* with an average of 6% fewer obtained solutions. Since *Greedy* concentrates on host configurations with low computation cost, the expected number of solutions is high, however, GWAs always have a high amount of data transfer which is not covered by this approach leading to a comparable number of solutions.

Fig. 7b shows our algorithm outperforms in utilising budget compared to others, being more than 17 times better than *Clus.Greedy*. The wasted budget for *Random* is too variable, indicated by its high error rate (standard deviation) in Fig. 7b, and on average the wastage is almost $14\times$ as compared to our proposed approach. Again, the unutilised budget with *Clus.Greedy* is very high with an average of $16\times$ higher as compared to the proposed method. The result shows that *Greedy* and *Clus.Greedy* are not able to generate diverse solutions.

7.2.3 Effectiveness of diversity

The key to finding the most suitable hosts for a GWA is to increase the diversity of the hosts in the generated solutions. Fig. 8 shows the diversity of results computed by different algorithms with the budget $400, where the yellow dots are the available hosts and the triangles represent the hosts selected by different algorithms.

The results (Fig. 8d) clearly show that our proposed approach is scattered better than others. The *Random Selection approach* also provides good scatter but the total number of solutions is 68% lower that the proposed algorithm. The *Greedy* approach has the worst diversity and the total number of solutions is 85% lower than the proposed approach. *Clus.Greedy* has better diversity, compared to the *Greedy* approach. However, there are no solutions selected for smaller size host configurations as shown in Fig. 8c. The reason behind this is that the total budget is distributed according to the average computation cost only. The communication is established only if the solution is valid and selected which can not be predicted beforehand. Since GWA have high data communication costs, none of the solutions can be selected in the allocated budget. This results in the selection of an average of 75% fewer host configurations as compared to the proposed approach.

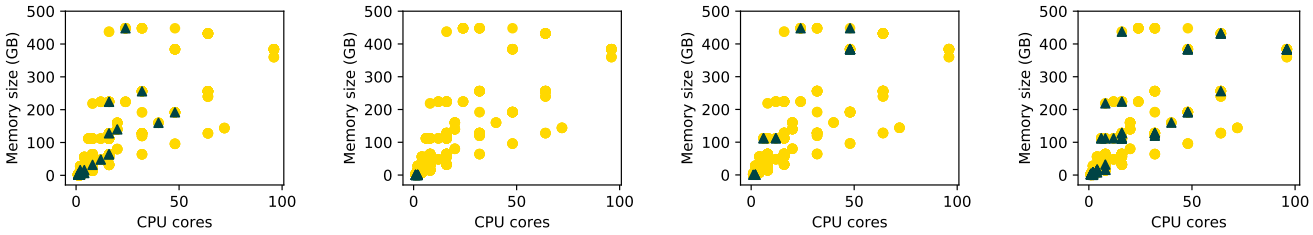
7.3 Real cloud environment analysis

Execution environment. Once the *selected* deployment solutions are obtained by our algorithm and baseline algorithms, GEODEPLOY can automatically deploy the obtained solutions, based on the provided host configurations and locations.

To evaluate the advances of our proposed algorithm, we deploy all the solutions generated from different algorithms in Section 7.2.2 with budget $100 on the real cloud environment using the GEODEPLOY *orchestrator* as discussed in Section 4.

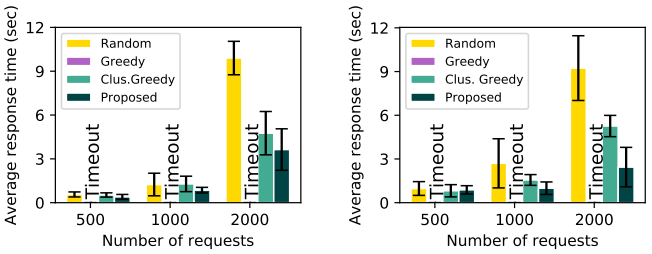
In each experiment, we evaluated the solutions by using both *text data* and *image data* and computed its response time. Each experiment is performed with three different request rates of 500, 1000 and 2000 per minute specified in the *Test plan*. The maximum permissible response time for GWA is set to 60 seconds after which a *Timeout* is notified.

Response result. Fig. 9a shows the average response time obtained for the best solution. The result shows that the obtained solution outperforms the comparators with an average of $3.3\times$ and $1.3\times$ lower response time compared to *Random* and *Clus.Greedy* respectively. We observe that in



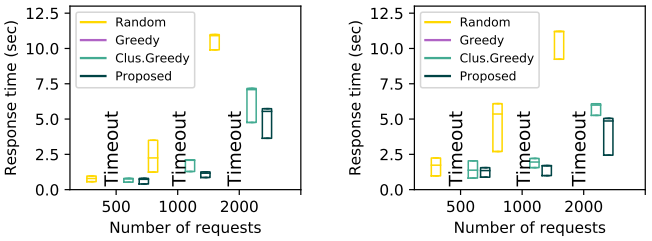
(a) Random selection algorithm (b) Greedy algorithm (c) Clustered greedy algorithm (d) Proposed algorithm

Fig. 8: The host diversity comparison between GEODEPLOY and the state-of-the-art methods. Each host configuration is represented as a circle in 2D space of CPU and memory values. The triangle shows the host configurations selected by the respective methods.



(a) For text data (b) For image data

Fig. 9: Average response time obtained by executing the methods for the given data. *Timeout* represents the requests are not responded. Black bars on the top represents the standard deviation.



(a) For text data (b) For image data

Fig. 10: Figure comparing the average, 95th and 99th percentile response time for the given data. Lower, middle and upper lines of the box represent the average, 95th percentile and 99th percentile of the response time respectively. *Timeout* represents the requests are not responded.

the *Greedy* approach, results tend to *timeout* in all cases as the host sizes are very small and are not able to handle even 500 requests. Fig. 10a also compares the average value with the 95th and 99th percentile values. It shows that for lower numbers of requests (e.g., 500 requests), the high percentile response time (95th and 99th) values are close to the average value. For the case of a high request size (e.g., 2000), the high percentile response time deviates from the average value.

A similar trend is observed for the best solution obtained for the image data as shown in Fig. 9b and Fig. 10b. In general, the proposed solution offers a lower response time, i.e., 2.1× and 1.3× less, compared to *Random* and *Clus.Greedy* respectively.

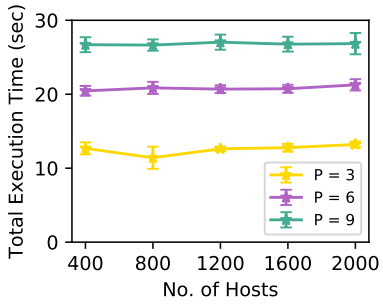


Fig. 11: Total execution time obtained by varying the number of hosts and replicas for GWA. P represents the problem size with number of GWA replicas

7.4 Scalability test

It is necessary for our proposed approach to scale with an increasing number of hosts and replicas of GWA components (represented as problem size). We evaluated the scalability of our proposed approach by varying the number of replicas for the GWA's web server which is represented as Problem Size (P). The three GWA cases are a) Problem Size $P = 3$, b) Problem Size $P = 6$ and c) Problem Size $P = 9$. The number of cloud providers is set to five. We increase the number of hosts varying from 200 to 2000. We also set the number of datacenter geo-locations to be 12 so that a valid solution is always generated. It's important to emphasize that these values are selected independently from the Cloud providers' offered locations and are exclusively utilised for the purpose of scalability assessment. To maintain consistency, we increased the budget for each case such that $\frac{Budget}{ProblemSize} = 50$ in every case.

Fig. 11 depicts the result obtained. The figure clearly shows that the total execution time does not significantly increase when increasing the number of host configurations for any case. The maximum increase is visualised for P = 3 with a value of 6% as compared to the average value. The figure also shows that the execution time increases with the increase in the problem size. With the increased problem size, the algorithm complexity increases as firstly it needs to search more elements, secondly, for each new element, the validity is tested and finally, the fitness function is computed. However, the total execution time does not increase exponentially with the increasing problem size.

8 Discussion

The experiment results demonstrate broad applicability across various Geo-Distributed Web Application (GWA) scenarios, specifically evaluating the performance of leading cloud providers such as AWS, Azure, and Google Cloud. Our methodology is particularly well-suited for scenarios utilising the pay-as-you-go pricing model, enabling efficient optimisation of resource utilisation and cost management. However, it is important to note that our approach may not be suitable for other pricing models, such as spot pricing. Additionally, while our results are based on a comprehensive test GWA, it is essential to recognise that outcomes may vary depending on the specific characteristics and requirements of individual GWA types.

Flexible execution. Since the cloud environment is dynamic, to analyse the exact performance variations, benchmark execution needs to be performed for longer durations, however, this is very costly. [12] used a flexible execution that schedules the benchmark for longer durations while actually executing for very short time periods thus covering the long-term variation in a defined budget. Developing similar techniques for GeoDeploy's execution plan under the defined budget is part of our future work plan.

Large-scale execution. Conducting experiments on a large-scale system provides invaluable insights into the behaviours and performance of our proposed framework for GWAs. Leveraging large-scale GWA benchmarks, such as the Online Boutique[5], one can aim to adapt and enhance these benchmarks to align with our proposed approach. By utilising such benchmarks in our experiments, we seek to simulate realistic and diverse workload conditions representative of real-world GWA deployments, enabling us to comprehensively evaluate the efficacy and scalability of our framework in handling varying demands and workload patterns. Also, we plan to extend our framework to include cloud service providers such as IBM, Alibaba, and DigitalOcean, thereby expanding the scope of our experiments to assess the quality of solutions obtained across multiple cloud environments.

Variable pricing models. One of our objectives is to advance the development of GeoDeploy frameworks to accommodate the evolving spectrum of cloud pricing models, including reserved instances, spot instances, and emerging alternatives, and facilitate efficient resource management in GWA benchmarking. Modelling various types of cloud pricing models for GWA is challenging due to several factors. Firstly, the dynamic nature of spot pricing introduces unpredictability, as prices fluctuate based on real-time supply and demand, making cost estimation and budget management complex. It can also be terminated by the provider with a little notice. Additionally, the upfront commitment required by reserved instances necessitates accurate long-term resource forecasting, which is difficult and can result in either underutilisation or overprovisioning of resources. Moreover, the geographical distribution of applications adds another layer of complexity, as different regions may have varying pricing structures, leading to inconsistencies in budget allocation strategies. The requirement to maintain performance and reliability despite these dynamic pricing fluctuations is crucial. Therefore, GeoDeploy will need new capabilities to track spot and reserved instances in real time. This requires designing algorithms that interact with cloud provider APIs to dynamically update GeoDeploy cost values. Additionally, GeoDeploy will need to dynamically adjust benchmarking decisions to utilize cheaper configurations as prices change frequently.

5. https://github.com/GoogleCloudPlatform/microservices-demo

9 Conclusion

We consider the problem of finding a suitable deployment option for a given GWA in a multi-cloud environment. We suggest that in order to find the most suitable deployment solution, it is necessary to *increase the diversity of hosts* while maintaining the *dependency of GWA components* for evaluation. We designed GeoDeploy, a novel GWA deployment orchestrator that incorporates a variety of novel heuristics for the problem at hand. We implemented GeoDeploy with Amazon AWS, Microsoft Azure and Google cloud platforms, though it can be easily extended to other cloud providers. Experimental results confirm that GeoDeploy outperforms *baseline methods*. Our proposed approach can generate up to 85% more solutions with 16 times better budget utilisation. The real-cloud evaluation also shows that with the proposed approach there is a better chance to find a suitable deployment solution, providing a response time that is half the response time of other *baseline methods*.

References

[1] A. Vulimiri, C. Curino, P. B. Godfrey, T. Jungblut, J. Padhye, and G. Varghese, "Global analytics in the face of bandwidth and regulatory constraints," in *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, 2015, pp. 323–336.

[2] Y. Yuan, D. Ma, Z. Wen, Y. Ma, G. Wang, and L. Chen, "Efficient graph query processing over geo-distributed datacenters," in *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020, pp. 619–628.

[3] J. P. Albrecht, "How the gdpr will change the world," *Eur. Data Prot. L. Rev.*, vol. 2, p. 287, 2016.

[4] P. Voigt and A. Von dem Bussche, "The eu general data protection regulation (gdpr)," *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 2017.

[5] O. Almurshed, O. Rana, Y. Li, R. Ranjan, D. N. Jha, P. Patel, P. Jayaraman, and S. Dustdar, "A fault-tolerant workflow composition and deployment automation iot framework in a multicloud edge environment," *IEEE Internet Computing*, vol. 26, no. 04, pp. 45–52, 2022.

[6] H. Wang, H. Shen, Z. Li, and S. Tian, "Geocol: A geo-distributed cloud storage system with low cost and latency using reinforcement learning," in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2021, pp. 149–159.

[7] L. Luo, G. Zhao, H. Xu, Z. Yu, and L. Xie, "Achieving cost optimization for tenant task placement in geo-distributed clouds," *IEEE/ACM Transactions on Networking*, 2023.

[8] R. Viswanathan, G. Ananthanarayanan, and A. Akella, "{CLARINET}: Wan-aware optimization for analytics queries," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 435–450.

[9] E. Cecchet, V. Udayabhanu, T. Wood, and P. Shenoy, "Benchlab: an open testbed for realistic benchmarking of web applications," in *Proceedings of the 2nd USENIX conference on Web application development*. USENIX Association, 2011, pp. 4–4.

[10] J. Scheuner, J. Cito, P. Leitner, and H. Gall, "Cloud workbench: Benchmarking iaas providers based on infrastructure-as-code," in *Proceedings of the 24th International Conference on World Wide Web*, 2015, pp. 239–242.

[11] D. N. Jha, M. Nee, Z. Wen, A. Zomaya, and R. Ranjan, "Smartdbo: smart docker benchmarking orchestrator for web-application," in *The World Wide Web Conference*, 2019, pp. 3555–3559.

[12] D. N. Jha, Z. Wen, Y. Li, M. Nee, M. Koutny, and R. Ranjan, "A cost-efficient multi-cloud orchestrator for benchmarking containerized web-applications," in *International Conference on Web Information Systems Engineering*. Springer, 2019, pp. 407–423.

[13] X. Tang, W. Cao, H. Tang, T. Deng, J. Mei, Y. Liu, C. Shi, M. Xia, and Z. Zeng, "Cost-efficient workflow scheduling algorithm for applications with deadline constraint on heterogeneous clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 9, pp. 2079–2092, 2021.

[14] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, "Devops," *Ieee Software*, vol. 33, no. 3, pp. 94–100, 2016.

[15] D. Weerasiri, M. C. Barukh, B. Benatallah, Q. Z. Sheng, and R. Ranjan, "A taxonomy and survey of cloud resource orchestration techniques," *ACM Computing Surveys*, vol. 50, no. 2, p. 26, 2017.

[16] A. Detti, L. Funari, and L. Petrucci, "μbench: an open-source factory of benchmark microservice applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 3, pp. 968–980, 2023.

[17] M. Straesser, J. Mathiasch, A. Bauer, and S. Kounev, "A systematic approach for benchmarking of container orchestration frameworks," in *Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering*, 2023, pp. 187–198.

[18] C. Davatz, C. Inzinger, J. Scheuner, and P. Leitner, "An approach and case study of cloud instance type selection for multi-tier web applications," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2017, pp. 534–543.

[19] M. B. Chhetri, S. Chichin, Q. B. Vo, and R. Kowalczyk, "Smart cloudbench—a framework for evaluating cloud infrastructure performance," *Information Systems Frontiers*, vol. 18, no. 3, pp. 413–428, 2016.

[20] G. Kousiouris and D. Kyriazis, "Enabling containerized, parametric and distributed database deployment and benchmarking as a service," in *Companion of the ACM/SPEC International Conference on Performance Engineering*, 2021, pp. 77–80.

[21] G. Yan and J. Li, "Towards latency awareness for content delivery network caching," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, 2022, pp. 789–804.

[22] Z. Song, K. Chen, N. Sarda, D. Altınbüken, E. Brevdo, J. Coleman, X. Ju, P. Jurczyk, R. Schooler, and R. Gummadi, "{HALP}: Heuristic aided learned preference eviction policy for {YouTube} content delivery network," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 1149–1163.

[23] "Akamai: Content delivery solutions (cdn)," https://www.akamai.com/solutions/content-delivery-network, accessed: 2023-01-20.

[24] "Amazon cloudfront," https://www.amazonaws.cn/en/cloudfront/, accessed: 2023-01-20.

[25] J. Ru, Y. Yang, J. Grundy, J. Keung, and L. Hao, "A systematic review of scheduling approaches on multi-tenancy cloud platforms," *Information and Software Technology*, p. 106478, 2020.

[26] W. Wang, M. Tornatore, Y. Zhao, H. Chen, Y. Li, A. Gupta, J. Zhang, and B. Mukherjee, "Infrastructure-efficient virtual-machine placement and workload assignment in cooperative edge-cloud computing over backhaul networks," *IEEE Transactions on Cloud Computing*, 2021.

[27] I. Attiya, M. Abd Elaziz, L. Abualigah, T. N. Nguyen, and A. A. Abd El-Latif, "An improved hybrid swarm intelligence for scheduling iot application tasks in the cloud," *IEEE Transactions on Industrial Informatics*, 2022.

[28] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli, "Optimal operator placement for distributed stream processing applications," in *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*, 2016, pp. 69–80.

[29] D. N. Jha, P. Michalák, Z. Wen, R. Ranjan, and P. Watson, "Multiobjective deployment of data analysis operations in heterogeneous iot infrastructure," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 11, pp. 7014–7024, 2019.

[30] Y. Huang, Y. Shi, Z. Zhong, Y. Feng, J. Cheng, J. Li, H. Fan, C. Li, T. Guan, and J. Zhou, "Yugong: geo-distributed data and job placement at scale," *Proceedings of the VLDB Endowment*, vol. 12, no. 12, pp. 2155–2169, 2019.

[31] Z. Wen, T. Lin, R. Yang, S. Ji, R. Ranjan, A. Romanovsky, C. Lin, and J. Xu, "Ga-par: Dependable microservice orchestration framework for geo-distributed clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 1, pp. 129–143, 2019.

[32] A. Marchese and O. Tomarchio, "Application and infrastructure-aware orchestration in the cloud-to-edge continuum," in *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*. IEEE, 2023, pp. 262–271.

[33] S. Yang, L. Jiao, R. Yahyapour, and J. Cao, "Online orchestration of collaborative caching for multi-bitrate videos in edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 4207–4220, 2022.

[34] S. Pallewatta, V. Kostakos, and R. Buyya, "Microfog: A framework for scalable placement of microservices-based iot applications in federated fog environments," *Journal of Systems and Software*, vol. 209, p. 111910, 2024.

[35] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-International Conference on Neural Networks*, vol. 4. IEEE, 1995, pp. 1942–1948.

[36] A. Song, W.-N. Chen, X. Luo, Z.-H. Zhan, and J. Zhang, "Scheduling workflows with composite tasks: A nested particle swarm optimization approach," *IEEE Transactions on Services Computing*, vol. 15, no. 2, pp. 1074–1088, 2020.

[37] H. Han, X. Bai, Y. Hou, and J. Qiao, "Multitask particle swarm optimization with heterogeneous domain adaptation," *IEEE Transactions on Evolutionary Computation*, vol. 28, no. 1, pp. 178–192, 2024.

[38] D. Bratton and J. Kennedy, "Defining a standard for particle swarm optimization," in *2007 IEEE swarm intelligence symposium*. IEEE, 2007, pp. 120–127.

[39] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An efficient k-means clustering algorithm: Analysis and implementation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 24, no. 7, pp. 881–892, 2002.

[40] M. A. Masud, J. Z. Huang, C. Wei, J. Wang, and I. Khan, "I-nice: A new approach for identifying the number of clusters and initial cluster centres," *Information Sciences*, vol. 466, pp. 129–151, 2018.

[41] S. Sahoo, B. Sahoo, and A. K. Turuk, "A learning automata-based scheduling for deadline sensitive task in the cloud," *IEEE Transactions on Services Computing*, 2019.

[42] Y. Han, D. Guo, W. Cai, X. Wang, and V. Leung, "Virtual machine placement optimization in mobile cloud gaming through qoe-oriented resource competition," *IEEE transactions on cloud computing*, 2020.

Devki Nandan Jha is currently a Lecturer at Newcastle University, Newcastle Upon Tyne, UK. He is also a visiting researcher at the Oxford e-Research Centre, University of Oxford. Previously, he was a Research Associate with Oxford e-Research Centre, University of Oxford, Oxford and CyberHive Ltd., Newbury, UK. He has a PhD in Computer Science from Newcastle University, Newcastle Upon Tyne, UK. His research interests include cloud computing, internet of things, trust and security, and machine learning.

Yinhao Li is a Lecturer in the School of Computing at Newcastle University, United Kingdom. He has a PhD in computer science from Newcastle University, United Kingdom. His research interests include cloud computing, edge computing and internet of Things. He previously received his MSc in Computer Science from the China University of Geoscience.

Zhenyu Wen (Member, IEEE) received the MSc and PhD degrees in Computer Science from Newcastle University, Newcastle upon Tyne, United Kingdom, in 2011 and 2016, respectively. He is currently a Postdoc Researcher with the School of Computing, Newcastle University, United Kingdom. His current research interests include IoT, crowd sources, AI system, and cloud computing. For his contributions to the area of scalable data management for the Internet of Things, he was awarded the the IEEE TCSC Award for Excellence in Scalable Computing (Early Career Researchers) in 2020.

Graham Morgan works in the field of distributed systems and created Game Lab at Newcastle University. Game Lab is a research and teaching laboratory that works with the video games industry on optimised resource management, streamed/networked gaming and real-time graphical simulations. Members of the lab regularly work on many of the top selling global video games. His work has won best paper awards in leading IEEE and ACM conferences and he has published in leading IEEE and ACM journals.

Prem Prakash Jayaraman works in the area of distributed systems in particular Internet of Things (IoT), Cloud and Mobile Computing. He has published over 70 papers including 28 journal papers (Transactions on Cloud Computing, Elsevier Computational Science, Transactions on Large-Scale Data- and Knowledge-Centred Systems, IEEE Journal on Selected Areas in Communications, The Scientific World Journal) in the related area of his research.

Maciej Koutny is a Professor of Computing Science in the School of Computing at Newcastle University, United Kingdom. His research interests centre on the theory of distributed and concurrent systems, including both theoretical aspects of their semantics and application of formal techniques to the modelling, synthesis and verification of such systems. He is the chair of the Steering Committee of the International Conferences on Application and Theory of Petri Nets and Concurrency, and an editor-in-chief of the LNCS Transactions on Petri Nets and Other Models of Concurrency. He is an Adjunct Professor at McMaster University, Canada, and in 2011 he held the Pascal Chair at Leiden University, The Netherlands.

Omer F. Rana received the B.S. degree in information systems engineering from the Imperial College of Science, Technology and Medicine, London, United Kingdom, the M.S. degree in microelectronics systems design from the University of Southampton, Southampton, United Kingdom, and the Ph.D. degree in neural computing and parallel architectures from the Imperial College of Science, Technology and Medicine. He is a Professor of performance engineering with Cardiff University, Cardiff, United Kingdom. His current research interests include problem solving environments for computational science and commercial computing, data analysis and management for large-scale computing, and scalability in high performance agent systems.

Rajiv Ranjan is a Professor of Computing Science in the School of Computing at Newcastle University, United Kingdom. Before moving to Newcastle University, he was Julius Fellow (2013-2015), Senior Research Scientist and Project Leader in the Digital Productivity and Services Flagship of Commonwealth Scientific and Industrial Research Organization (CSIRO C Australian Government's Premier Research Agency). Prior to that he was a Senior Research Associate (Lecturer level B) in the School of Computer Science and Engineering, University of New South Wales (UNSW). Professor Ranjan has a PhD (2009) from the Department of Computer Science and Software Engineering at the University of Melbourne.