frontiers | Frontiers in Sustainability

TYPE Methods
PUBLISHED 06 January 2023
DOI 10.3389/frsus.2022.1048606



OPEN ACCESS

EDITED BY
M. Affan Badar,
Indiana State University, United States

REVIEWED BY
Giorgos Demetriou,
École des ponts ParisTech (ENPC), France
Bikash Koli Dey,
Hongik University, South Korea

*CORRESPONDENCE
Jason M. Pittman
✉ jason.pittman@umgc.edu

SPECIALTY SECTION
This article was submitted to Sustainable Supply Chain Management, a section of the journal Frontiers in Sustainability

RECEIVED 19 September 2022
ACCEPTED 12 December 2022
PUBLISHED 06 January 2023

CITATION
Pittman JM and Alaee S (2023) A green scheduling algorithm for cloud-based honeynets. *Front. Sustain.* 3:1048606. doi: 10.3389/frsus.2022.1048606



# A green scheduling algorithm for cloud-based honeynets

Jason M. Pittman[1]* and Shaho Alaee[2]

[1]Cybersecurity and Information Technology, University of Maryland Global Campus, Adelphi, MD, United States, [2]AI Futures, Booz Allen Hamilton, McLean, VA, United States

Modern businesses leverage cloud architecture to achieve agile and cost-effective technology services. Doing so comes at the expense of the environment though cloud technologies consume large quantities of energy. Cloud energy consumption is concerning in light of global climate trends and dwindling fossil fuel reserves. Consequently, increasing attention is given to sustainable and green cloud computing, which seeks to optimize compute-resource allocation and usage of virtualized systems and services. At the same time, progress toward sustainable and green cloud technology is impeded because as more enterprises deploy services into cloud architecture, cybersecurity threats follow. Unfortunately, cybersecurity technologies are optimized for maximum service overwatch without regard for compute resources and energy. This negates the energy reduction achieved in recent sustainable technology advancements. In this work, a generalized cybersecurity honeynet scheduling algorithm is proposed, in which power, CPU, and network overhead are operationalized to increase sustainability while balancing defensive mechanisms. The work describes both the mathematical foundation for the algorithm and a pseudocode proof of concept.

KEYWORDS

**green, sustainability, energy, scheduling, algorithm, cloud, honeynet**

## 1. Introduction

Public cloud architectures such as AWS, Microsoft Azure, and Google Cloud represent agile and cost-effective platforms for businesses to deploy infrastructure and services. As of 2020, 94% of enterprise computing occurs in the cloud (Galov, 2022). Clearly, the growth of cloud services, and enterprise adoption thereof, has been meteoric. Yet, while the cloud offers significant savings compared to most on-premise solutions, cloud technologies consume large quantities of energy (Mehta et al., 2021; Ohwo and Thomas, 2021). Consequently, increasing attention is given to sustainable and green cloud computing, which seeks to optimize compute-resource allocation and usage of virtualized systems and services (Yadav et al., 2018; Ohwo and Thomas, 2021). Much of the related literature attempts to optimize cloud orchestration or *schedulers* for green cloud computing.

Moreover, as the enterprise has shifted to cloud architecture, cybersecurity threats have followed (Ohwo and Thomas, 2021). Along with other cybersecurity infrastructure, honeypots and honeynets have made the migration to the cloud (Brown et al., 2012; Hamad and Omara, 2016; Kelly et al., 2021). Honeypots are important tools in research and practice. As intentionally vulnerable systems, honeypots provide researchers with a methodology to collect attacker techniques, tactics, and processes (Chin et al., 2009). Unfortunately, cybersecurity technologies are optimized for maximum service overwatch without regard for compute resources and energy. This negates the energy reduction achieved in recent sustainable technology advancements.

It is not known how much energy honeynets consume. Accordingly, there is a problem insofar as there are no generalized green computing algorithms applicable to honeynets (More and Ingle, 2017). Furthermore, existing honeynet scheduling algorithms optimize toward maximum service exposure and uptime (Kong et al., 2020). As a consequence, the existing scheduling mechanisms do not consider sustainability a core operational principle. Accordingly, the purpose of this work is to demonstrate a potential generalized green scheduling algorithm for honeynets.

## 2. Related work

There are six components in the conceptual framework supporting this research. The necessary background to contextualize the problem is broad and deep because of the number of fields involved. As such, we summarize seminal and timely literature in each of the components while attempting to avoid assumptions in content knowledge. At the same time, we have organized the related work around the common theme of cloud-based honeynets and sustainable, green cloud architecture schedulers.

### 2.1. Honeypots and honeynets

A honeypot is a computing system intended to attract adversaries and designed to be attacked (Zakaria and Kiah, 2013). These purposefully vulnerable systems are qualitatively categorized based on traits such as interaction level (Campbell et al., 2015). In this sense, there are three overarching categories of honeypots: low, medium, and high interactions (Moore and Al-Nemrat, 2015). What services the honeypot emulates and the depth of the emulation constitute the interaction type. Specifically, low-interaction honeypots emulate individual services (e.g., SSH) and limited emulated functionality. A medium-interaction honeypot has more interactivity than a low-interaction honeypot but similarly offers a few services only. High interaction mimics full computing systems.

Each type has associated advantages and disadvantages (Pittman et al., 2020). Perhaps, the most noteworthy disadvantage common to all honeypots is that the deceptive technology is only as good as the traffic it is able to capture. Of course, the ability to capture traffic is, in part, coupled with the service or services offered by the honeypot. To address the capture dependency, researchers developed honeynets.

A honeynet is a collection of honeypots running on the same or adjacent network segments (Fan et al., 2015). Honeynets offer a more diverse application service set than a single honeypot. The aim is to capture more traffic. Moreover, honeynets offer inter-networking capture between discrete honeypots implemented across the honeynet and thus have additional depth than a honeypot.

Honeynets are similar to honeypots insofar as honeynets are also categorized according to *type*. The types—low, medium, and high interactions—denote interaction levels identical to honeypots. Furthermore, because honeynets offer multiple honeypots on the deception network, there is an additional *hybrid* type. Hybrid honeynets are simply two or more honeypot types (Fan et al., 2015; Bao et al., 2018) represented in the honeynet. More specifically, a honeynet is minimally two honeypots and a controller-logging system. Networking equipment is implicit to honeynet architectures but nominally includes a screening router or firewall (Fan et al., 2015; Bao et al., 2018). Finally, the honeynet controller-logging system itself operates similarly to a high-interaction honeypot because of the need to screen and direct incoming network traffic.

Honeynets introduce a different set of disadvantages. Foremost, honeynets are difficult to deploy, configure, and maintain (Fan et al., 2015; Meng et al., 2017; Franco et al., 2021). As an analogy, imagine scaling system and network administration overhead from a single personal computer to a small business office. Furthermore, honeynets consume more compute resources and power compared with a honeypot. This is evidenced by resource management being one of six deployment architecture requirements (Chin et al., 2009).

The latest honeynet literature describes innovations with deployment architectures. These innovations either seek to make honeynets behave more intelligently (Fan et al., 2015; Fraunholz et al., 2017; Meng et al., 2017) or to improve compute-resource management in some fashion (Kong et al., 2020; Washofsky, 2021). Thus, honeypots and honeynets are migrating to cloud architecture (Brown et al., 2012; Hamad and Omara, 2016; Kelly et al., 2021).

### 2.2. Honeynet deployment architectures

Honeynet deployment architectures can be described by their *generation*. Generations i, ii, and iii encompass the transition from physical machines to virtualization (Abbasi and Harris, 2009). The post-generation iii architectures evolved from

host-based virtualization to become hybrid or next-generation implementations. Research (Brown et al., 2012; Kumar et al., 2012; Fan et al., 2019) shows the next-generation honeynets consist of virtual systems or containers in a cloud architecture and software-defined networking (SDN). Based on sample honeynet architectures (Fan et al., 2015; Meng et al., 2017; Bao et al., 2018; Kong et al., 2020; Franco et al., 2021), the minimum number of systems in a honeynet is five—a low-interaction honeypot, a high-interaction honeypot, the honeynet controller and logging facility, a screening router, and SDN controller.

The combination of cloud architecture and SDN enables honeynet deployment architectures to address the deployment, configuration, and maintenance issues older honeynet generations experienced (Kyung et al., 2017). While such an approach has demonstrated success, the trade-off has been in cloud-based compute resources. Processing power and virtual hardware requirements are high for honeynets. The high compute-resource utilization is exacerbated by the need for honeynets to optimize toward maximum service exposure and uptime (Kong et al., 2020). Fortunately, elastic and dynamic resource allocation is a cloud computing specialty.

## 2.3. Cloud computing

The idea of *cloud computing* is to distribute compute resources and persistent data across geographic locations (Dillon et al., 2010). Doing so leverages cost-effective compute models and offers a guarantee of ubiquitous, always-on information technology. According to NIST (Mell and Grance, 2011), such distribution can be categorized in three services: software, platform, or infrastructure. Honeynets are implemented as infrastructure as a service (IaaS) (Kelly et al., 2021) because of the need to configure processors, memory, storage, and networking components for high interaction honeypots.

While cloud computing accounts for 94% of enterprise IT workloads (Kelly et al., 2021), the technology is not free of challenges. Early research (Dillon et al., 2010; Daryapurkar and Bagde, 2014) suggested platform security and cost modeling are the biggest challenges. However, recent work demonstrates a consensus on performance optimization and sustainability (Abd El-Mawla and Ibrahim, 2022; Khan et al., 2022) as significant challenges.

## 2.4. Workload scheduling mechanisms

Workload scheduling is necessary for performance optimization because compute-resource utilization rises sharply as task parallelism increases. There are two scheduling mechanisms to consider in this conceptual framework. The first is related to general cloud workload scheduling. The second is honeynet-specific workload scheduling in IaaS platforms.

### 2.4.1. Cloud scheduling

Regardless of what services a cloud architecture provides—infrastructure, application, or platform—there is a need to do so with a high degree of performance. Thus, when a user or adjacent system initiates a process in the cloud, the cloud scheduler selects available compute resources and allocates the task to those resources (Kumar et al., 2019). The task scheduler mechanism demonstrates a similar concept to CPU scheduling of user or system process requests against the kernel and hardware resources. Such an analogy is appropriate because single metric scheduling (e.g., CPU utilization) has been the core of cloud task scheduling since the beginning.

It should not be controversial to suggest single metric scheduling algorithms are not as effective at optimizing performance as multi-metric algorithms. Moreover, the breadth of research investigating various task-scheduling algorithms indicates the criticality of the mechanism. The plethora of approaches to task scheduling also shows how balancing optimization and implementation is challenging. Overall, the differences between multi-metric scheduling mechanisms are a combination of what metrics are amalgamated and how tasks are allocated to compute resources based on those metrics.

For example, genetic algorithm-based task scheduling leverages evolutionary recombination phases achieve high optimization for performance to the detriment of task latency (Zhao et al., 2009). Comparatively, particle swarm scheduling does not induce task latency but does exhibit load balancing problems despite having high performance (Awad et al., 2015). Other multi-metric algorithms, such as *ant colony* or *cost-based*, share similar trade-offs. Yet, none of these task-scheduling algorithms incorporate sustainability metrics.

To that end, workload optimization and scheduling in the context of sustainable cloud resource management research are nascent topics. Cutting-edge work has focused on combining advanced learning algorithms with the scheduling of application services tasks, virtual machine elasticity, and compute-resource allocation. Furthermore, research (Arunarani et al., 2019) suggests the nature-inspired algorithms hold the most promise and need to be further explored.

### 2.4.2. Honeynet scheduling

Insofar as a honeynet is cloud based, it is constrained by the underlying cloud scheduling algorithm and must implement a higher-layer task scheduler for the routing of network traffic into its honeypots. The reliance on network traffic fosters a sensitivity to both connection and operational latency. Thus, the literature surrounding cloud-based honeynet scheduling focuses on minimizing TCPIP overhead while optimizing compute-resource allocation to the honeypot virtual machines. Hence, software-defined networking (SDN) has emerged as a honeynet scheduling mechanism (Han et al., 2016; Kyung et al., 2017; Meng et al., 2017) attempting to optimize the honeynet network

layer. SDN, in this context, allows for dynamic spin-up of specific low- and high-interaction honeypots based on the incoming network traffic. The advantage is that a single SDN controller can orchestrate connections for the entire honeynet, and control is centralized (Han et al., 2016; Meng et al., 2017). In this way, only the most desirable ingress traffic is directed to the relevant honeypot.

In addition to task scheduling for network traffic, honeynets are also subject to compute-resource task scheduling constraints. Existing research defines *resource scheduling* as including compute resources (i.e., CPU and RAM) (Bao et al., 2018). Resource task scheduling is critical to honeynets because the virtual machine honeynets must always be accessible to maximize capture availability. Obviously, when maximally capturing attack traffic, the honeynet may require significant compute resources under millisecond demand (Bao et al., 2018; Kong et al., 2020). Alternatively, the honeynet idles with minimal compute-resource expenditure.

Such volatility strains the underlying cloud architecture task scheduler (Kumar et al., 2019; Kelly et al., 2021). This task scheduler strain results from the two layers—cloud and honeynet—not being interconnected. At the same time, neither the cloud scheduler nor the honeynet scheduler are natively oriented toward sustainability.

## 2.5. Green computing

The green computing initiative is a reaction to cloud data centers consuming up to 4% of global electrical power (More and Ingle, 2017; Mehta et al., 2021). The initiative aims to reduce energy consumption by implementing sustainability at various levels of computing infrastructure (Kurp, 2008), particularly the software enabling cloud architectures. The focus on software is reasonable, given that servers—compute nodes and virtual machines alike—make up approximately 70% of cloud power allocation (More and Ingle, 2017). Moreover, the highest consumer of energy within cloud server architecture is the CPU (Abd El-Mawla and Ibrahim, 2022; Khan et al., 2022). Second to the processor, network interface cards (NICs) can consume up to 13% of the total power drawn by the system.

Software layer green computing research can be categorized as consolidation (Srikantaiah et al., 2008), scaling (Dargie and Schill, 2012), and scheduling (Merkel et al., 2010). A consolidation is an overarching approach not unlike the concept of batch processing or network communications insofar as fewer, larger bundles of consolidated operations are more efficient to process than, smaller operations. Scaling aims to optimize energy efficiency in the electronics components, including CPU frequency related to voltage from the power subsystem. Green task schedulers must be at least energy consumption and thermal aware (Abd El-Mawla and Ibrahim, 2022). The literature also explicitly calls for such schedulers to maintain intercommunication with the control plane capable of dynamic virtual machine spin-up, spin-down, and migration (Abid et al., 2020; Mehta et al., 2021).

## 2.6. Green schedulers

Similar to honeypots, green computing schedulers can be described as low level or high level. Low-level green schedulers attempt to optimize operating system processes based on energy consumption in the underlying hardware. The literature (Dargie and Schill, 2012; More and Ingle, 2017; Abid et al., 2020) describes low-level schedulers as dynamic process scaling mechanisms. High-level green schedulers operate as load managers within the cloud compute fabric and within an operating system. In this context, green schedulers can be understood as *controllers* seeking to optimize which cloud virtual machines are powered on or off (Srikantaiah et al., 2008; Merkel et al., 2010; Mehta et al., 2021). Thus, high-level schedulers implement the green computing principles of consolidation and scheduling, whereas low-level schedulers behave according to the scaling principle.

Unfortunately, there are no studies investigating a comprehensive resource allocation techniques workload task scheduling and energy consumption in terms of "...cost, resource utilization, energy, workload, execution time, response time, user satisfaction, and QoS" (Abid et al., 2020, p. 2817). More notably, the literature recognizes increases in sustainability at the green compute layer necessarily limit cloud computing processing capabilities (Abid et al., 2020; Mehta et al., 2021). The trade-off causes a great deal of friction with honeynet implementations based on the honeynet's usage profile and operational mission.

# 3. Mathematical foundation and proposed algorithm

The green scheduling algorithm for cloud-based honeynets takes into consideration five sustainability characteristics. The characteristics were extracted from the cloud, honeynet, and green computing literature. We first operationalized these into discrete mathematical foundational equations. Then, we use critical elements from the equations to aid in the pseudocode description of the green honeynet scheduler algorithm.

## 3.1. Mathematical foundation

### 3.1.1. Resources

There are two sets of resources in the foundation of the proposed algorithm. First, there are the resources in the cloud

architecture. The cloud consists of $N$ compute nodes. Each cloud compute node can be described as follows:

$$C_i = \{CPU, MEM, NET\} \quad (1)$$

where $C_i$ is the set of cloud compute node resources consisting of a CPU, RAM, and network. The proposed algorithm incorporates the cloud hardware because of the implicit relation to virtualized compute resources. In simple terms, if there are no cloud hardware resources available to service a task, the task cannot be routed to the virtualized system regardless of the latter's resource utilization.

Second, there are virtual machine resources comprising the honeynet. At a minimum, the virtual compute resources are, in a three-to-one relation to the cloud hardware resources. In other words, there are at least three virtual machines running on a single physical cloud compute node to form a honeynet. These can be described as follows:

$$C_h = \{vCPU, vMEM, vNET\} \quad (2)$$

Finally, the proposed algorithm must be aware of the maximum resources available on a given individual compute node $c_i \in C_i$ such that the cloud compute node is optimally utilized for the honeynet. This can be described as follows:

$$c_i \in C_i | sum(c_h \in C_h) <= max(c_i) \quad (3)$$

### 3.1.2. Tasks

Any compute instruction sent by the cloud architecture to the honeynet is encoded as a task. Likewise, instructions from the honeynet to the cloud architecture are tasks. The algorithm does not need to differentiate between task directions because the green honeynet scheduler multiplexes all tasks. Thus, we can simply state the set of tasks in the following form:

$$T = \{t_1, t_2, t_3 ... t_n\} \quad (4)$$

### 3.1.3. Network interfaces

The honeynet requires at least seven NICs—one physical and six virtual—based on the minimum number of cloud compute nodes and virtual systems involved. The virtual NICs are attached in pairs to virtual machine honeypots as $c_h N_a, N_b$. These NICs impact power consumption, as discussed, and also impact honeynet latency negatively. Honeynet latency is considered as a sum of individual honeypot latencies as follows:

$$L = \{(c_h N_a, N_{b1}) + (c_h N_a, N_{b2}) + (c_h N_a, N_{b3}) ... (c_h N_a, N_{bn})\} \quad (5)$$

### 3.1.4. Power

Virtual machine honeypots in the honeynet can be in one of three states: active, sleep, and off. This can be expressed as $S_a$, $S_s$, and $S_o$, respectively. The $S_a$ state can be further expanded into *running* as $S_{ar}$ or $S_{ai}$. As well, transitions between states must be incorporated into the algorithm because of the spike in compute resources incurred during any such operations. The set of state transitions can be expressed as follows:

$$X = \{(S_a \rightarrow S_s), (S_a \rightarrow S_o), (S_s \rightarrow S_a), (S_s \rightarrow S_o), (S_o \rightarrow S_a)\} \quad (6)$$

Furthermore, the algorithm considers the collective baseline power consumption through the sum of all active and sleeping honeynet virtual machines. This can be expressed as follows:

$$P(C_i) = \{P(S_{ar}) + P(S_{ai}) + P(S_s)\} \quad (7)$$

### 3.1.5. Utilization

Utilization is an expression of the power consumed as observable through task allocation to a compute resource. From the perspective of the honeynet controller, a green algorithm monitors honeynet utilization according to the summation below. Functionally, utilization has a threshold $d$ which serves as a demarcation for green power consumption.

$$U_{c_h} = \sum_{j=1}^{T} c_{j,d} \quad (8)$$

The algorithm also has a reference lookup for power consumption associated with $T$, which is accessible through the utilization instantiation of $U_T$. Such allows the algorithm to determine whether an incoming task can be processed optimally by an active, running honeypot without exceeding the green threshold $d$.

## 3.2. The algorithm

The aforementioned expressions can be operationalized into a green honeynet task-scheduling algorithm (Algorithm 1). In simple terms, incoming tasks are mapped from a cloud compute node to a honeypot in the honeynet when the honeypot has available compute resources. Otherwise, if there is honeypot in an *off* state, the honeynet should make it active and map the incoming task onto the honeypot.

When this is not possible, the honeynet checks whether the cloud compute node has enough compute resources available to support an additional virtual machine. If so, the honeynet will request a new honeypot virtual machine be created and map the task onto the new honeypot. When the current cloud

compute node is near the threshold, the honeynet requests the new honeypot virtual machine to be moved to a different compute node in the cloud infrastructure.

In all cases, the honeynet returns an updated compute resources utilization value.

```
for all t ∈ T do
  while (c_i ∈ C_i ∧ c_h ∈ C_h) ∈ S_sr do
    if (U_T < U_{c_h}) and (L is True) then
      t → c_h
      return U_{c_h}
    else
      if (S_o → S_a) then
        t → c_h
        return U_{c_h}
      else if P(c_i) < d then
        new c_h(S_ar) → C_h ⇒ c_i ∈ C_i
        t → c_h
        return U_{c_h}
      else if P(c_i) ≥ d then
        move c_h(S_ar to c_i ∈ C_i
        t → c_h
        return U_{c_h}
      end if
    end if
  end while
end for
```

Algorithm 1. Green honeynet task scheduler.

## 4. Conclusion

Cloud services have expanded exponentially over the past decade. However, cloud technologies consume large quantities of energy (Mehta et al., 2021; Ohwo and Thomas, 2021). Consequently, research is increasingly exploring sustainable and green cloud computing options. More specifically, there is growing interest in optimizing cloud orchestration or *schedulers* for green cloud computing.

Furthermore, as enterprises have shifted to cloud architecture, cybersecurity threats have followed (Ohwo and Thomas, 2021). Defensive cybersecurity technologies such as honeypots and honeynets have followed the migration (Brown et al., 2012; Hamad and Omara, 2016; Kelly et al., 2021). This is because honeypots are important tools in research and practice. Unfortunately, cybersecurity technologies are optimized for maximum service overwatch without regard for compute resources and energy. This negates the energy reduction achieved in recent sustainable technology advancements.

Accordingly, we presented a mathematical foundation for a green honeynet scheduling algorithm. The foundation considered five sustainability characteristics as evidenced in the related work. Notably, the characteristics were cloud compute node and honeynet node resources, tasks, network interfaces, power, and utilization. We used those characteristics as integral elements in a conceptual green honeynet scheduler consisting of 21 steps.

Overall, the green honeynet scheduling algorithm may be of theoretical significance to researchers exploring mechanisms for sustainability and green computing. The proposed green algorithm for honeynet scheduling is illustrative of one possible research pathway which investigators can use to build on. In addition, this work may hold practical significance for cloud and cybersecurity engineers interested in a solution to schedule resource allocation sustainably. From a managerial perspective, a successful implementation of the proposed algorithm would have practical significance insofar as operational costs are reduced through sustainable, green operations.

This work is based on four assumptions. Foremost, we assume a net positive sustainability impact can be made in honeynets operating in a cloud architecture. While there is a healthy literature basis for sustainability in cloud and cloud task schedulers, this work is the first to describe how sustainability standards might be implemented for honeynet scheduling. Furthermore, the proposed scheduling algorithm is assumed to incorporate the correct set of variables related to sustainability and green computing. It is possible for extraneous factors to be present in the cloud architecture which are not encapsulated in the proposed algorithm. Along those lines, we assume honeypot resource utilization correlates positively with attack detection and capture. Finally, we assume the introduction of the scheduling algorithm does not itself increase resource utilization significantly. Future work, if conducted, might bear out the findings necessary to resolve these assumptions.

Given the proposed algorithm and the stated assumptions, this work, as well as any future work, is limited in three important ways. For instance, existing cloud architecture and algorithm design serve as limitations. If the cloud architecture was to change, say to limit network overhead for sustainability reasons, not only would the algorithm design potentially not function appropriately but also a honeynet may not be possible at all. Another limitation affecting this work is the lack of existing research in applying sustainability and green computing standards to cybersecurity tools in general and honeynets specifically. A final limitation, assuming future work is conducted to bring the algorithm to practical fruition, is the availability of proper instrumentation to measure sustainable effectiveness.

Taking the assumptions and limitations into account, future work may explore the potential relationship between attack detection and capture with the goal of discovering sustainable

optimizations in the capture mechanisms. In addition, while a honeynet depends on having constant service exposure, individual honeypots should only be active when there is an attack to capture. There may be future work possible here in reducing the power consumption overhead by turning on a honeypot or reviving one from a sleep state. Another suggestion for future work is for optimizations in the honeynet controller and logging system. This system cannot be reasonably kept in a sleeping state or off. Some areas for investigation include more efficient signaling architectures, condensed logging to reduce network overhead, and learning algorithms for optimized honeypot control. Each area may be investigated through experimentation in future studies. Finally, future exploration of how the algorithm proposed in this research operates in various cloud platforms (e.g., AWS and Azure) may be of benefit to cloud and cybersecurity practitioners. Such work could include resource allocation profiling of the scheduling algorithm itself to ensure its execution does not adversely impact sustainability.

## Data availability statement

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author.

## Author contributions

JP and SA contributed to the conception and design of the study and wrote the equations and pseudocode. JP wrote the first draft of the manuscript. SA was responsible for the related work analysis. Both authors contributed to the manuscript revision and read and approved the submitted version.

## Conflict of interest

SA was employed by Booz Allen Hamilton.

The remaining author declares that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

## Publisher's note



## References

Abbasi, F. H., and Harris, R. (2009). "Experiences with a generation III virtual honeynet," in *2009 Australasian Telecommunication Networks and Applications Conference (ATNAC)* (IEEE), 1–6.

Abd El-Mawla, N., and Ibrahim, H. (2022). *Green Cloud Computing (GCC), Applications, Challenges and Future Research Directions*. Available online at: https://njccs.journals.ekb.eg (accessed December 11, 2022).

Abid, A., Manzoor, M. F., Farooq, M. S., Farooq, U., and Hussain, M. (2020). Challenges and issues of resource allocation techniques in cloud computing. *KSII Trans. Internet Inform. Syst.* 14, 2815–2839. doi: 10.3837/tiis.2020.07.005

Arunarani, A., Manjula, D., and Sugumaran, V. (2019). Task scheduling techniques in cloud computing: a literature survey. *Fut. Generat. Comput. Syst.* 91, 407–415. doi: 10.1016/j.future.2018.09.014

Awad, A., El-Hefnawy, N., and Abdel_kader, H. (2015). Enhanced particle swarm optimization for task scheduling in cloud computing environments. *Proc. Comput. Sci.* 65, 920–929. doi: 10.1016/j.procs.2015.09.064

Bao, N. K., Ahn, S. W., and Park, M. (2018). "An elastic-hybrid honeynet for cloud environment," in *CSEIT, NCS, SPM, NeTCoM*, 117127.

Brown, S., Lam, R., Prasad, S., Ramasubramanian, S., and Slauson, J. (2012). *Honeypots in the Cloud*. Wisconin: University of Wisconsin-Madison, 11.

Campbell, R. M., Padayachee, K., and Masombuka, T. (2015). "A survey of honeypot research: trends and opportunities," in *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)* (IEEE), 208–212.

Chin, W., Markatos, E. P., Antonatos, S., and Ioannidis, S. (2009). "Honeylab: large-scale honeypot deployment and resource sharing," in *2009 Third International Conference on Network and System Security* (IEEE), 381–388.

Dargie, W., and Schill, A. (2012). "Analysis of the power and hardware resource consumption of servers under different load balancing policies," in *2012 IEEE Fifth International Conference on Cloud Computing* (IEEE), 772–778.

Daryapurkar, J. U., and Bagde, K. G. (2014). Cloud computing: issues and challenges. *Int. J. Recent Innov. Trends Comput. Commun.* 2, 770–773.

Dillon, T., Wu, C., and Chang, E. (2010). "Cloud computing: issues and challenges," in *2010 24th IEEE International Conference on Advanced Information Networking and Applications* (IEEE), 27–33.

Fan, W., Du, Z., Smith-Creasey, M., and Fernandez, D. (2019). Honeydoc: an efficient honeypot architecture enabling all-round design. *IEEE J. Select. Areas Commun.* 37, 683–697. doi: 10.1109/JSAC.2019.2894307

Fan, W., Fernández, D., and Du, Z. (2015). "Adaptive and flexible virtual honeynet," in *International Conference on Mobile, Secure and Programmable Networking* (Springer), 1–17.

Franco, J., Aris, A., Canberk, B., and Uluagac, A. S. (2021). A survey of honeypots and honeynets for internet of things, industrial internet of things, and cyber-physical systems. *IEEE Commun. Surv. Tutor.* 23, 2351–2383. doi: 10.1109/COMST.2021.3106669

Fraunholz, D., Zimmermann, M., and Schotten, H. D. (2017). "An adaptive honeypot configuration, deployment and maintenance strategy," in *2017 19th International Conference on Advanced Communication Technology (ICACT)* (IEEE), 53–57. doi: 10.23919/ICACT.2017.7890056

Galov, N. (2022). *Cloud Adoption Statistics for 2022*. Web Tribunal. Available online at: https://webtribunal.net/blog/cloud-adoption-statistics/#gref (accessed April 06, 2022).

Hamad, S. A., and Omara, F. A. (2016). Genetic-based task scheduling algorithm in cloud computing environment. *Int. J. Adv. Comput. Sci. Appl.* 7, 550–556. doi: 10.14569/IJACSA.2016.070471

Han, W., Zhao, Z., Doupé, A., and Ahn, G.-J. (2016). "Honeymix: toward SDN-based intelligent honeynet," in *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks and Network Function Virtualization*, 1–6.

Kelly, C., Pitropakis, N., Mylonas, A., McKeown, S., and Buchanan, W. J. (2021). A comparative analysis of honeypots on different cloud platforms. *Sensors* 21, 2433. doi: 10.3390/s21072433

Khan, T., Tian, W., Zhou, G., Ilager, S., Gong, M., and Buyya, R. (2022). Machine learning (ml)—centric resource management in cloud computing: a review and future directions. *J. Netw. Comput. Appl.* 2022, 103405. doi: 10.1016/j.jnca.2022.103405

Kong, T., Wang, L., Ma, D., Xu, Z., Yang, Q., Lu, Z., and Lu, Y. (2020). "Automated honeynet deployment strategy for active defense in container-based cloud," in *2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)* (IEEE), 483–490.

Kumar, M., Sharma, S. C., Goel, A., and Singh, S. P. (2019). A comprehensive survey for scheduling techniques in cloud computing. *J. Netw. Comput. Appl.* 143, 1–33. doi: 10.1016/j.jnca.2019.06.006

Kumar, S., Singh, P., Sehgal, R., and Bhatia, J. (2012). Distributed honeynet system using GEN III virtual honeynet. *Int. J. Comput. Theory Eng.* 4, 537. doi: 10.7763/IJCTE.2012.V4.527

Kurp, P. (2008). Green computing. *Commun. ACM* 51, 11–13. doi: 10.1145/1400181.1400186

Kyung, S., Han, W., Tiwari, N., Dixit, V. H., Srinivas, L., Zhao, Z., et al. (2017). "Honeyproxy: design and implementation of next-generation honeynet via sdn," in *2017 IEEE Conference on Communications and Network Security (CNS)* (IEEE), 1–9.

Mehta, J. A., Nanavati, P. K., and Mehta, V. K. (2021). A survey on green cloud computing. *Int. J. Eng. Appl. Sci. Technol* 6, 425–429. doi: 10.33564/IJEAST.2021.v06i01.067

Mell, P., and Grance, T. (2011). The NIST definition of cloud computing. Special Publication 800-145. doi: 10.6028/NIST.SP.800-145

Meng, X., Zhao, Z., Li, R., and Zhang, H. (2017). "An intelligent honeynet architecture based on software defined security," in *2017 9th International Conference on Wireless Communications and Signal Processing (WCSP)* (IEEE), 1–6.

Merkel, A., Stoess, J., and Bellosa, F. (2010). "Resource-conscious scheduling for energy efficiency on multicore processors," in *Proceedings of the 5th European Conference on Computer Systems*, 153–166.

Moore, C., and Al-Nemrat, A. (2015). "An analysis of honeypot programs and the attack data collected," in *International Conference on Global Security, Safety, and Sustainability* (Springer), 228–238.

More, N. S., and Ingle, R. B. (2017). "Challenges in green computing for energy saving techniques," in *2017 International Conference on Emerging Trends and Innovation in ICT (ICEI)* (IEEE), 73–76.

Ohwo, O. B., and Thomas, A. (2021). Towards an efficient and effective cloud computing architecture: a review. *IUP J. Comput. Sci.* 15.

Pittman, J. M., Hoffpauir, K., and Markle, N. (2020). Primer—a tool for testing honeypot measures of effectiveness. arXiv [Preprint] arXiv:2011.00582.

Srikantaiah, S., Kansal, A., and Zhao, F. (2008). "Energy aware consolidation for cloud computing," in *USENIX HotPower'08: Workshop on Power Aware Computing and Systems at OSDI*.

Washofsky, A. D. (2021). Deploying and Analyzing Containerized Honeypots in the Cloud With T-Pot (PhD thesis). Monterey, CA: Naval Postgraduate School.

Yadav, R., Zhang, W., Kaiwartya, O., Singh, P. R., Elgendy, I. A., and Tian, Y.-C. (2018). Adaptive energy-aware algorithms for minimizing energy consumption and SLA violation in cloud computing. *IEEE Access* 6, 55923–55936. doi: 10.1109/ACCESS.2018.2872750

Zakaria, W. Z. A., and Kiah, M. L. M. (2013). A review of dynamic and intelligent honeypots. *Sci. Asia* 39S, 1–5. doi: 10.2306/scienceasia1513-1874.2013.39S.001

Zhao, C., Zhang, S., Liu, Q., Xie, J., and Hu, J. (2009). "Independent tasks scheduling based on genetic algorithm in cloud computing," in *2009 5th International Conference on Wireless Communications, Networking and Mobile Computing* (IEEE), 1–4.