Container Sizing for Microservices with Dynamic Workload by Online Optimization

Nader Alfares
Pennsylvania State University
University Park, PA, USA
nna5040@psu.edu

George Kesidis
Pennsylvania State University
University Park, PA, USA
gik2@psu.edu

ABSTRACT

Over the past ten years, many different approaches have been proposed for different aspects of the problem of resources management for long running, dynamic and diverse workloads such as processing query streams or distributed deep learning. Particularly for applications consisting of containerized microservices, researchers have attempted to address problems of dynamic selection of, for example: types and quantities of virtualized services (e.g., IaaS/VMs), horizontal and vertical scaling of different microservices, assigning microservices to VMs, task scheduling, or some combination thereof. In this context, we argue that online optimization frameworks like simulated annealing are highly suitable for exploration of the trade-offs between performance (SLO) and cost, particularly when the complex workloads and cloud-service offerings vary over time. Based on a macroscopic objective that combines both performance and cost terms, annealing facilitates light-weight and coherent policies of exploration and exploitation. In this paper, we first give some background on simulated annealing and then experimentally demonstrate its usefulness for container sizing using microservice benchmarks. We conclude with a discussion of how the basic annealing platform can be applied to other resource-management problems, hybridized with other methods, and accommodate user-specified rules of thumb.

ACM Reference Format:
Nader Alfares and George Kesidis. 2023. Container Sizing for Microservices with Dynamic Workload by Online Optimization. In *The International Workshop on Container Technologies and Container Clouds (WoC '23), December 11–15, 2023, Bologna, Italy.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3631311.3632399

1 INTRODUCTION AND MOTIVATION

Resource management in the public cloud typically involves selecting available services spanning compute, storage and networking to minimize cost (or "cloud spend" [7, 29]) subject to workload Service-Level Objectives (SLOs, i.e., performance requirements). The problem is particularly challenging when serving a plurality of complex time-varying workloads. For a given set of job streams, the optimal service suite may involve a cluster composed of a variety of services. Even within services of the same type, resource instances can vary greatly. For example, VMs can have different amounts and types of memory, CPU cores, hardware accelerators, cross connects (e.g., PCIe, NVLink), and networking. In some cases, users can optionally pay to collocate VMs on the same physical server or to provision networking resources connecting their VMs to storage.[1]

Deploying application software using a microservice architecture, e.g., [5, 20, 28, 35], has the advantage of modular code design and maintenance. Some microservices can be used by different applications. Also, each microservice can be executed in a separate, light-weight container.

Particularly in an edge cloud setting where operating costs are high, the user may wish to economize by explicitly provisioning containerized microservices (vertical scaling)[2] and replicating microservices (horizontal scaling) as necessary. Data locality issues may be involved in how containers are assigned to VMs. Different task scheduling policies have been proposed which can be implemented on cluster managers such as Kubernetes (K8s) [16, 19]. In addition, a default task scheduler and a task assignment policy to containerize microservices can be augmented with simple rules of thumb, e.g., to exploit data locality [23] and to address persistent straggler tasks [25][3].

For complex, diverse and dynamic workloads incident to a cluster, deciding a cloud service suite and a container-sizing and replication policy are difficult tasks (even if, e.g., a default task scheduling is used), and a typical user (cloud tenant/customer) may provide little more guidance than "macroscopic" service-level objectives (SLOs) and a cost budget.

1.1 Deep Learning Versus Online Optimization

For some time, Deep neural networks (DNNs) have been suggested by researchers as a way to dispense cluster-management advice even when the workloads are complex and dynamic, e.g., [6, 24, 28]. For such complex problems, (supervised) deep learning requires a vast training dataset which is typically produced by an extensive "depth of search" study. Such training datasets are often manually

WoC '23, December 11–15, 2023, Bologna, Italy

ACM ISBN 979-8-4007-0459-8/23/12…$15.00
https://doi.org/10.1145/3631311.3632399

[1] There are similar efficient resource allocation challenges in "bare metal" and private data center scenarios where virtualized public-cloud services are not in play.
[2] In [28], it's argued that provisioning containers can prevent resource starvation of bottleneck microservices toward improving total job execution times, compared to simply relying on the VM's OS to schedule component tasks as they arrive in a "best effort" fashion without considering end-to-end job execution times.
[3] Also, rather than relying on a default mechanism of a cluster manager (e.g., K8s), a user may wish to control how microservices are "packed" into the VMs of their cluster, to both economize on the number of VMs and, again, to address data locality issues as a job's tasks are executed as a sequence of microservices.

curated. Moreover, deep learning requires choosing among a variety DNN architectures and sizes and training hyperparameters. For cluster-management advice, DNN's input would need to include a description of the user's *current* workload and of the their performance and cost constraints, i.e., their SLOs. The workload description could be the parameters of a general-purpose workload model (which may need to be periodically re-estimated to account for distributional changes in the job arrival process, or the jobs themselves, of a non-stationary workload). If the advice concerned specific cloud services and how to use them, then the DNN may need to be refined or retrained whenever the services change (including just the terms of their service-level agreements (SLAs)). Moreover, DNN refinement may be required if the user's workload changes so that it is anomalous with respect to those represented by the (current) training dataset. (Such problems are sometimes called online "model drift.") Such DNN refinement (a.k.a. online reinforcement or active learning) typically requires producing a new batch of training samples.

For these reasons, it's difficult to make fair comparisons between online optimization methods and those based on DNNs. (Indeed, it's often difficult to fairly compare different DNN based approaches.) We instead point out some qualitative differences.

In contrast to online optimization methods, DNNs are very costly to train and refine and are difficult to precisely reproduce by other parties [11], but once trained, DNNs can perform inference much more quickly. The cost of creating and curating training datasets and the reproducibility issues of DNNs are typically not explained in articles advocating deep learning approaches. Considering how these training datasets are produced by *searching* the space of possible decisions/advice they could make, their formation is not unrelated to an online optimization process which includes breadth of search. (An online optimization method could conceivably be used to identify training samples for deep learning or refinement.) Also, online optimization methods respond more quickly to online "model drift" and do not require modeling the workload.

1.2 Online Optimization Methods

Recently, researchers and practitioners have explored the use of other traditional but more light-weight and adaptable means of decision-making (including using model-based adaptive control, PID control, and Markov decision processes) where DNNs play only a partial role at most, e.g., [30, 38]. For example, particle-swarm type optimization has been used for load balancing, e.g., [31]. Also, genetic algorithms (GAs), e.g., [36, 37], have been used to explore the service suite and dynamically react to changes in the workload and service offerings. But exploration under GAs is rather ad-hoc, like random search.

Simulated annealing (or just annealing) [1, 10, 15] has been widely used for complex, non-convex optimization problems, including for practical applications, since its development in the 1980s. Annealing is suitable for the foregoing resource management problems which operate in a highly dynamic environment over an indefinite time horizon. Annealing can more quickly react to detected changes in operating conditions by simply increasing its temperature parameter to more aggressively (broadly) search the parameters it controls (and/or by expanding its "local neighborhood" sets). Annealing's local neighborhood set can be adjusted to accommodate rules of thumb. Also, annealing can be hybridized with a GA or random search to improve breadth of search, or hybridized with, e.g., a Tabu search mechanism to improve local-search efficiency. Thus, we herein employ annealing as a representative method of online optimization.

In this paper, we take an annealing approach to the problem of sizing the containers within its VMs for a plurality of different streaming workloads. Annealing works to minimize a macroscopic objective that can account for factors like current job execution times and the cost per unit time of the existing cluster. While it can operate both offline (e.g., [33]) and online (runtime), our focus is on the latter. We apply online annealing to the container-sizing problem of a microservice workload relying on the default container assignment mechanism of the K8s cluster manager. Experimental results are reported for these prototypes.

2 BACKGROUND

2.1 Annealing

Simulated annealing was introduced in the 1980s as a generic framework to minimize a complicated function $Y : D \to \mathbb{R}$ over a very large discrete bounded domain D, where "complicated" here means that Y has plural local minima in addition to global ones and D may be a finite discretization of a continuous domain.

A *local* neighborhood function $\nu(x)$ for all $x \in D$ is defined, where $x \notin \nu(x)$. A collection of possible transitions between x and elements of $\nu(x)$ are also defined, often taken as all equally likely as assumed in the following (i.e., each with uniform probability $1/|\nu(x)|$). A key requirement of the neighborhood function ν is that it has to produce a connected graph in D[4]. Typically, the neighborhood function ensures only *incremental* one-step changes to the current configuration state, but this is not a requirement.

Given Y, ν, one can define an annealing Markov chain on D at temperature $\tau > 0$ with transition probabilities from x to $x' \in \nu(x)$ being:

$$\frac{1}{|\nu(x)|} \exp\left(-\frac{\max\{Y(x') - Y(x),\ 0\}}{\tau}\right).$$

We see that a possible transition from the current state x to the next state x' is "accepted" with positive probability even when the objective Y is increased, i.e., when $Y(x') > Y(x)$, and always accepted when $Y(x') \leq Y(x)$ – this is the "heat bath" rule. When the temperature parameter τ increases, this acceptance probability increases, i.e., there is more exploration and less exploitation which is particularly useful when trying to avoid poor local minima. If the temperature τ is initially sufficiently high and slowly (logarithmically) decreases to zero over time, it can be shown that the (time-inhomogeneous) Markov chain Y will converge in probability to its global minimum on D [1]. But this limiting result is not very useful in practice. Even early on, some authors pointed out that it may be better not to thus "cool" the annealing chain [10], particularly when considering a finite time-horizon. If the temperature is fixed $\tau > 0$ and the neighborhoods all have the same size ($|\nu(x)|$ is a constant function of $x \in D$) then (time-homogeneous)

[4]That is, the Markov chain resulting from the "base" transition probabilities associated with the neighborhood function is "irreducible". These transition probabilities should be chosen so that the base Markov chain is also time-reversible [1].

Markov chain Y has (Gibbs) stationary distribution proportional to $\exp(-Y(x)/\tau)$. Note that as the temperature $\tau \to 0$, only transitions that reduce Y are accepted, i.e., pure exploitation.

In the past, annealing was successfully applied to complex optimization problems such as placement and routing of VLSI circuits and large-scale bin-packing problems [22].

2.2 Kubernetes

Kubernetes (K8s) is an open-source container orchestration platform. It has gained popularity due to its ability to automate the deployment, scaling, and management of containerized applications in complex distributed systems. A K8s *deployment* is an abstraction layer that manages the creation, scaling, and updating of *pods*, which are the smallest deployable units in K8s. Deployments provide a declarative way to manage the state of the application, ensuring that the desired number of replicas is always available. Pods, on the other hand, are single instances of a containerized application that run in a shared environment. They provide a lightweight and flexible way to manage containers and their resources. Each pod has a unique IP address and a set of shared resources, including storage volumes and network interfaces. By setting resource requests and limits, K8s can allocate resources more efficiently and ensure that each pod has enough resources to run effectively.

The K8s Application Programming Interface (API) is an interface that enables developers to manage K8s resources by providing a mechanism to create, read, update, and delete resources, such as deployments, pods, and services. To update resources requests and limits, developers can use the K8s API directly or the `kubectl` command-line tool. The API enables developers to retrieve the current deployment configuration, modify the resources requests and limits in the configuration file, and subsequently update the deployment. The K8s API ensures that the updated deployment is automatically rolled out to the cluster, maintaining the desired state of the system. In this paper, we exploit the API to size the different deployments of microservices with the resource requests of the annealing process.

3 EXPERIMENTAL EVALUATION - CONTAINER SIZING FOR MICROSERVICES

3.1 Experimental Set-Up

For our container sizing experiments (Figure 1), we use CloudLab [4] to set up a cluster of r320 nodes. Specifically, we configure the cluster to have five nodes, with one node serving as the Kubernetes master and three nodes as Kubernetes workers. Additionally, we designated one node as the workload generator, which also includes the annealing process that communicated with the Kubernetes API at the master node to adjust and modify the resources allocated (configurations) to the deployments. We run our experiments on two microservice applications provided by DeathStarBench [5] (using helm-chart): *Social Network* and *Hotel Reservation*. The former is an implementation of a social media network that allows users to read, post, and react to social media posts, whereas the latter is an implementation of a browser-based application that allows users to search and make reservations for hotels. We employ the annealing approach to size the resources allocated to microservices' deployments. We generate the workload for the two microservices using `wrk2` [26], along with workload generator scripts (the benchmark provides both). In addition, we use Locust [3] to generate a mixture of request types provided by the microservice benchmark.

3.2 Defining the Annealing Objective

We define the objective function as $Y_n = t_n + \lambda(U_{\text{cpu}} + U_{\text{mem}})$ where t_n is the average execution time for epoch n, U_{cpu} and U_{mem} are CPU and memory utilization, respectively, and $\lambda > 0$ is a user-specified factor that weighs the average latencies of the requests against the utilization of resources by the microservices. For experiments of this section, we set $\lambda = 1$, which we empirically found to be suitable for our demonstrations.

3.3 Experimental Results

For each experiment, we deploy the microservice application to the Kuberentes cluster while setting the resource request and limit to the minimum configuration for cores and memory for each deployment (we set the minimum for CPU and memory to be at 0.1 CPU and 0.1 MiB). Second, we set up our workload generator to send requests at one-minute intervals (epochs) and measure the mean latency of the requests. After every epoch, the annealing process evaluates the objective value and explores a random microservice to patch with a new resource request and limit at steps of 0.1 CPU/MiB for CPU/memory. When maximum resources are allocated, the annealing exploration phase can either modify resources allocated to a random microservice, or reallocate resources from one microservice to another.

Figure 2 depicts the response times of the read requests to the social network application while performing annealing at different temperatures. The figure shows that a minimizing configuration of the microservices resources can be quickly discovered at a relatively high temperature. In practice, when a configuration with an acceptably small evaluated objective is found, the temperature can be lowered, retaining the configuration desired for a longer period of time, as depicted in figure 3.

Figure 4 depicts the response time of read requests made to the social network application while performing annealing at temperature $\tau = 10$. This experiment describes a scenario where an annealing process selects and remains at a configuration that minimizes the objective value (e.g., a local minimum) for a period of time before finding the optimal configuration through exploration. Note that to find the globally minimizing configuration, configurations that *increase* the objective are first explored.

Figure 5 illustrates the objective values obtained through simulated annealing when handling a mixed set of social network request types. Specifically, the mixture comprises 25% `read_home_timeline` requests, 25% `read_user_timeline` requests, and 50% `compose_post` requests. For this experiment, we employed Locust [3] to generate this blend of requests directed at the social network application. The figure highlights that, for handling a blend of workload streams, a higher annealing temperature facilitates a quicker discovery of an optimal configuration, albeit with increased variations.

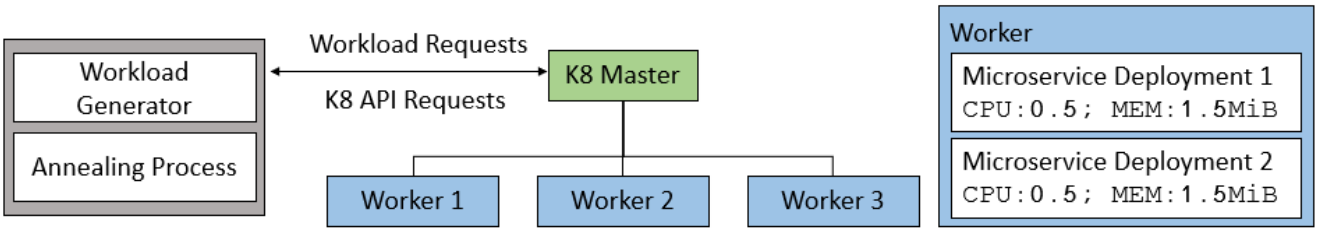

Figure 1: Container sizing experimental setup.

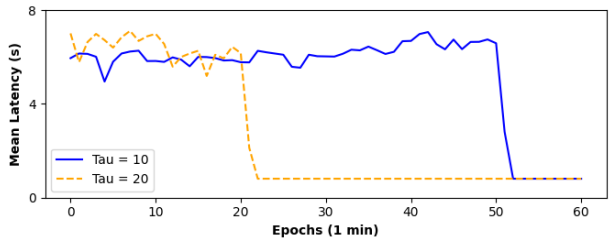

Figure 2: Social Network: a configuration with a minimizing objective is found quickly at a relatively high temperature.

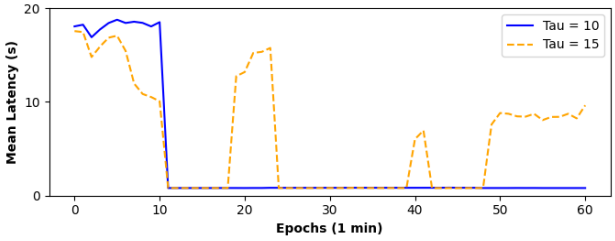

Figure 3: Hotel Reservation: an optimal configuration is retained at a longer period of time for annealing with relatively low temperatures.

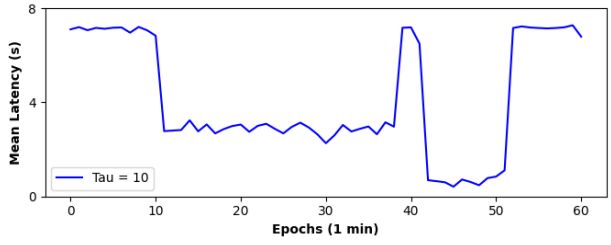

Figure 4: Social Network: performing annealing

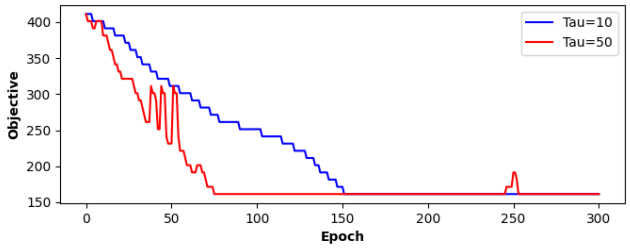

Figure 5: Performing simulated annealing for a mixture of request types incident to the social network microservice application.

3.4 Discussion: Adaptive Temperature Adjustment in Simulated Annealing

When a running average of the cost objective value improves, indicating progress towards the desired optimization goals, the temperature can be reduced. This reduction in temperature corresponds to a decrease in the likelihood of accepting worse solutions, leading to a more focused (depth of) search around the currently nearby optimum. Several different strategies can be employed for adaptive temperature adjustment in SA, depending on the nature of the optimization problem and the specific cost and performance metrics involved, for example:

- Continuous Logarithmic: The classical logarithmic (slow) cooling schedule [1].
- Threshold Based: Predefined thresholds for the objective value are established with temperature changes at each threshold. If the current objective value crosses a threshold, the temperature is reduced to encourage exploitation. Alternatively, if the cost objective exceeds a certain threshold, the temperature may be increased to encourage exploration. The method of the previous subsection is an example. For another example, at any given threshold, temperatures can increase exponentially (e.g., $\tau \longrightarrow 2\tau$) and decrease additively (e.g., $\tau \longrightarrow (\tau - 1)^{+}$).
- Iteration Based: The temperature adjustment is based on the number of iterations or the convergence behavior. For instance, if no significant in the cost objective is observed after a certain number of iterations, the temperature can be incremented to encourage further exploration.

For all of the above, annealing may recall the *most recent* adequate solution x, and may return to x when temperatures increase, bearing in mind that x may not be adequate for the current operating conditions. Also, some or all of the decisions of the above methods could also be based on other statistics such as running average mean (rather than the current value) or variance of the cost objective.

4 DISCUSSION AND FUTURE WORK

This paper describes an online-optimization approach (simulated annealing) to the management of the performance and cost (resources) of containers deployed to a cluster of virtual machines in the public cloud which is agnostic to both services and dynamic workload. Annealing has very low complexity and, at moderate temperatures, is responsive to changes in the workload or available service suite. Considering its ease and low cost of implementation and reasonably good performance, simulated annealing and its variants and hybrids form a compelling class of baseline frameworks for online resources management of clusters.

In this paper, to demonstrate its ease of deployment and use and its good performance, we evaluated an implementation of annealing for pod or container sizing, i.e., vertical scaling, evaluated using Kuberenetes (K8s) for microservice workloads, with costs based on efficiency of use of the existing instances. In [2], we apply annealing for VM service selection with experimental results for Apache Spark implementations of HiBench and distributed deep learning workloads, with costs based on AWS VM instance pricing. (Also, an attempt is made therein to compare online optimization with supervised learning (e.g., deep learning) methods by accounting for the cost of forming the training dataset of the latter.) Note that service selection can be expanded to consider the possible benefits of serverless functions, e.g., [13], particularly for stateless microservices, or to consider different storage options particularly for data intensive tasks.

As mentioned in Section 1, online simulated annealing can also be (jointly or independently) used:

- to adjust the task-scheduling policy;
- for assigning containers or pods to VMs (rather than K8s' default "bin packing" [19]); or
- for horizontal scaling of microservices (i.e., creation of service replicas),

e.g.,[8, 12, 21, 27, 32, 34]. All such annealing mechanisms can work with a common temperature parameter or separately adjustable ones, and some obviously will need to operate at different timescales: fast (task scheduling), moderate (container sizing, assigning containers to VMs), slow (horizontal scaling), slowest (service selection).

Rules of thumb can be easily incorporated into simulated annealing neighborhood function ν, e.g., to constrain assignment of service replicas to the same VM to take advantage of statistical multiplexing [23], or to require that certain microservices be assigned to VMs with GPU support, or to constrain assignment of certain groups of different microservices to the same VM to take advantage of data locality, where the last type of requirement may also constrain horizontal scaling.

The foregoing discussion is **not** meant to suggest that annealing, or some other method of automatic online optimization, needs to be applied to task scheduling or for the assignment of containers/pods to VMs. The default ones provided by, e.g., Spark and K8s (and used in our experiments) may be adequate in some use cases. Note that under K8s, such policies are user configurable [14, 16–19]. That is, the K8s API can be used to create affinities between certain types of VMs and certain microservices and among microservices. Moreover, the K8s API can be used to configure the "bin packer" so that, e.g., the fewest possible VMs are used (allowing some VMs to be released to save cost), or to balance the pods among the existing VMs, subject to any stipulated constraints.

Simulated annealing (or another online optimization framework like Bayesian optimization [9]) can easily be modified, e.g., to remember the best states visited in the recent past and return to them with lower temperature if "exploration" at higher temperatures has not recently yielded good results (smaller objective). Also, simulated annealing can be hybridized with other known methods, e.g., with random restart or genetic algorithms to improve breadth of search (exploration), or by excluding recently visited states to improve efficiency of depth of search (exploitation).

OPEN SOURCING

The scripts we developed for our online optimization method are posted on GitHub [2]. The repository consists of two major sections. The first section focuses on service selection, employing simulated annealing to identify optimal configurations at the VM service level, as detailed in [2]. This segment encompasses performance results from a few HiBench workloads and an illustrative example of a deep learning application. The accompanying scripts employed to produce the dataset and perform simulated annealing are included as well. The second section focuses on sizing containers of the DeathStarBench microservice applications. This segment encompasses two different software implementations of performing simulated annealing for container sizing. The first implementation utilizes `wrk2`, a HTTP benchmarking tool [26], to generate requests of a single type to the microserivce application instance. At every epoch, the script performs simulated annealing using the performance metrics provided by `wrk2`. The second implementation of container sizing uses Locust, also a HTTP benchmarking tool [3]. The workload generated is a mixture of different types of requests incident to the microservice application. In this software implementation, Locust leverages `wrk2`'s endpoints of the microservice application to generate the requests. In addition, our Locust implementation is containerized and can also be deployed on a kubernetes cluster.

ACKNOWLEDGEMENTS

This work was supported by NSF grants 2212201 and 2122155. We also acknowledge helpful discussions with Aman Jain and Ata Fatahi Baarzi.

REFERENCES

[1] E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines.* Wiley, 1989.
[2] N. Alfares, G. Kesidis, A.F. Baarzi, and A. Jain. Cluster resource management for dynamic workloads by online optimization. http://arxiv.org/abs/2207.04594.
[3] C. Bystrom, J. Heyman, and T. Gustafsson. Locust. https://github.com/locustio/locust.
[4] Cloudlab. https://www.cloudlab.us/.
[5] C. Delimitrou and al. Deathstarbench. https://github.com/delimitrou/DeathStarBench.

[6] C. Delimitrou and C. Kozyrakis. HCloud: Resource-Efficient Provisioning in Shared Cloud Systems. In *Proc. ASPLOS*, 2016.
[7] FLEXERA. State of the Cloud Report. https://info.flexera.com/CM-REPORT-State-of-the-Cloud, 2023.
[8] G.-N. Gan, T.-L. Huang, and S. Gao. Genetic simulated annealing algorithm for task scheduling based on cloud computing environment. In *Proc. Int'l Conf. on Intelligent Computing and Integrated Systems*, 2010.
[9] R. Garnett. *Bayesian Optimization.* Cambridge University Press, 2023.
[10] B. Hajek and G. Sasaki. Simulated annealing - to cool or not. *Systems and Control Letters*, 12(5):443–447, June 1989.
[11] S. Higginbotham. Show Your Machine-Learning Work. *IEEE Spectrum*, Dec. 2019.
[12] A. Ibrahim, A. Mohamed, and X. Shengwu. Job Scheduling in Cloud Computing Using a Modified Harris Hawks Optimization and Simulated Annealing Algorithm. In *Proc. CSIT*, 2020.
[13] A. Jain, A.F. Baarzi, N. Alfares, G. Kesidis, B. Urgaonkar, and M. Kandemir. SplitServe: Efficient Splitting Complex Workloads across FaaS and IaaS. In *Proc. ACM/IFIP Middleware*, Dec. 2020; *Proc. ACM SoCC* (extended abstract, poster presentation), Nov. 2019.
[14] A. Kanso. Scheduling Extensions in Kubernetes. https://github.com/akanso/extending-kube-scheduler.
[15] G. Kesidis and E. Wong. Optimal acceptance probability for simulated annealing. *Stochastics and Stochastics Reports*, 29:221–226, 1990.
[16] kube-scheduler. https://kubernetes.io/docs/reference/command-line-tools-reference/kube-scheduler/.
[17] Kubernetes - affinity and anti-affinity. https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#affinity-and-anti-affinity.
[18] Kubernetes - pod priority and preemption. https://kubernetes.io/docs/concepts/scheduling-eviction/pod-priority-preemption/.
[19] Kubernetes - resource bin packing. https://kubernetes.io/docs/concepts/scheduling-eviction/resource-bin-packing/.
[20] U. Kulkarni, A. Sheoran, and S. Fahmy. The Cost of Stateless Network Functions in 5G. In *Proc. Symposium on Architectures for Networking and Communications Systems*, 2021.
[21] X. Liu and J. Liu. A task scheduling based on simulated annealing algorithm in cloud computing. *Int'l J. of Hybrid Information Technology*, 9:403–412, 06 2016.
[22] Z. Liu, Q. Zhang, M.F. Zhani, R. Boutaba, Y. Liu, and Z. Gong. DREAMS: Dynamic resource allocation for MapReduce with data skew. In *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2015.
[23] A. Mahgoub, Edgardo Barsallo Yi, Karthick Shankar, Eshaan Minocha, Sameh Elnikety, Saurabh Bagchi, and Somali Chaterji. WiseFuse: Workload Characterization and Optimized Execution Plans for Serverless DAG Workflows. In *Proc. ACM/IFIP SIGMETRICS/Performance*, 2022.
[24] H. Mao, M. Schwarzkopf, S.B. Venkatakrishnan, Z. Meng, and M. Alizadeh. Learning scheduling algorithms for data processing clusters. https://arxiv.org/pdf/1810.01963.pdf.
[25] J. Mohan, A. Phanishayee, A. Raniwala, and V. Chidambaram. Analyzing and mitigating data stalls in DNN training. In *Proc. VLDB*, Jan. 2021.
[26] W. Morgan. wrk2: a HTTP benchmarking tool based mostly on wrk. https://github.com/giltene/wrk2.
[27] D. Pandit, Samiran S. Chattopadhyay, M. Chattopadhyay, and N. Chaki. Resource allocation in cloud using simulated annealing. In *Proc. Applications and Innovations in Mobile Computing (AIMoC)*, 2014.
[28] H. Qiu, S.S. Banerjee, S. Jha, Z.T. Kalbarczyk, and R.K. Iyer. FIRM: An Intelligent Fine-grained Resource Management Framework for SLO-Oriented Microservices. In *Proc. OSDI*, 2020.
[29] RightScale. State of the Cloud Report: Public Cloud Adoption Grows as Private Cloud Wanes. https://cdn2.hubspot.net/hubfs/2582046/MSPs/RightScale-2017-State-of-the-Cloud-Report.pdf, 2017.
[30] K. Rzadca, P. Findeisen, J. Swiderski, P. Zych, P. Broniek, J. Kusmierek, P. Nowak, B. Strack, P. Witusowski, S. Hand, and J. Wilkes. Autopilot: Workload autoscaling at Google. In *Proc. ACM EuroSys*, 2020.
[31] S. Singhal and A. Sharma. Load balancing algorithm in cloud computing using mutation based PSO algorithm. In *Proc. Int'l Conf. on Advances in Computing and Data Sciences*, 2020.
[32] S. Telenyk, E. Zharikov, and O. Rolik. Consolidation of virtual machines using simulated annealing algorithm. In *Proc. CSIT*, volume 1, pages 117–121, 2017.
[33] P.K. Thiruvasagam, A. Chakraborty, and C.S.R. Murthy. Latency-aware and Survivable Mapping of VNFs in 5G Network Edge Cloud. https://arxiv.org/abs/2106.09849, 2021.
[34] P.J.M. van Laarhoven, E.H.L. Aarts, and J.K. Lenstra. Job shop scheduling by simulated annealing. In *Proc. INFORMS*, Linthicum, MD, USA, Feb. 1992.
[35] D. Vaquero-Melchor, A.M. Bernardos, and L. Bergesio. SARA: A Microservice-Based Architecture for Cross-Platform Collaborative Augmented Reality. *Appl. Sci. Special Issue Augmented Reality: Current Trends, Challenges and Prospects*, 10(6), 19 March 2020.
[36] D. Wilcox, A. McNabb, and K. Seppi. Solving virtual machine packing with a reordering grouping genetic algorithm. In *Proc. IEEE Congress on Evolutionary Computation (CEC)*, 2011.
[37] K. P. Wong and Y. W. Wong. Genetic and genetic/simulated-annealing approaches to economic dispatch. *IEEE Proc. Gener. Transm. Distrib.*, 141(5):507–513, 1994.
[38] H. Zhang, Y. Tang, A. Khandelwal, J. Chen, and I. Stoica. Caerus: Nimble Task Scheduling for Serverless Analytics. In *Proc. USENIX NSDI*, Apr. 2021.