Online VM Service Selection with Spot Cores for Dynamic Workloads

Nader Alfares, G. Kesidis
Penn State
{nna5040,gik2}@psu.edu

B. Urgaonkar
Penn State and Amazon
buu1@psu.edu

Ata Fatahi Baarzi
LinkedIn
immrata@gmail.com

Aman Jain
Microsoft
aman.jain@microsoft.com

***Abstract*—Over the past ten years, many different approaches have been proposed for different aspects of the problem of cost-effective 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., VMs, serverless functions, data-storage), vertical and horizontal scaling of different microservices, assigning microservices to VMs, task scheduling, or some combination thereof. Herein focusing on selection decisions of on-demand VM services, we consider the problem of creating and actively maintaining a training dataset for supervised machine-learned frameworks like deep neural networks and more light-weight, adaptable online optimization frameworks. For both decision frameworks, we make a case for the usefulness of spot cores and incremental search techniques like simulated annealing to reduce workload preemption while searching the decision space to explore the trade-offs between service-level objectives (SLOs) and cloud-spend. Based on user input, a macroscopic objective that captures both performance and cost will be used. We are particularly interested in scenarios with complex workloads and cloud-service offerings that vary over time.**

***Index Terms*—Annealing, Cloud, Kubernetes, Spark, IaaS**

I. 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" [21], [25]) subject to workload Service-Level Objectives (SLOs, i.e., performance requirements). The problem is particularly challenging when serving 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.

The focus of this paper is the problem of deciding the type and number of on-demand VMs for a cost-effective cluster providing adequate performance. This problem is difficult for complex, diverse and dynamic workloads incident to the cluster, (even if, e.g., a default task scheduling and container sizing policy is used), where a typical user (cloud tenant/customer) may provide little more guidance than "macroscopic" service-level objectives (SLOs) and a cost budget[1].

A. Deep Learning and/or Online Optimization

Classical frameworks for systems identification and control include "black-box" approaches like neural networks (NNs) and simulated annealing. More recently, Deep NNs (DNNs, AIs) have been suggested by researchers as a way to dispense cluster-management advice even when the workloads are complex and dynamic, e.g., [10], [19], [20].[2] For such problems, (supervised) deep learning requires a vast training dataset, typically produced by an extensive "depth of search" study. Such training datasets are often manually curated. Moreover, deep learning requires choosing among a variety of DNN architectures and training hyperparameters. For cluster-management advice, DNN's input would need to include a description of the user's current workload and their performance and cost constraints, i.e., their SLOs. 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.

In contrast, online optimization methods (e.g., [2], [5]) are less responsive to changes than DNN inference (when those changes are well-represented in the DNN's current training dataset). But online optimization methods: do not have associated training costs[3], avoid the need for online workload modeling, and respond more quickly to online model drift.

Considering how training datasets are produced by searching the space of possible decisions they could make, their

[1]Note that existing autoscaling and bin-packing services are typically based on runtime resource-utilization statistics not application-level SLOs, e.g., query response times or whole-job completion times (JCTs), and associated deadlines [11], or the costs of cloud services.

[2]though some public-cloud companies internally forbid the use of AI for control of their cloud-computing infrastructure

[3]The cost of procuring and maintaining the training dataset of a DDN is rarely accounted for in research articles advocating their use, and their reproducibility problems [14] are rarely discussed.

formation is not unrelated to an online optimization process which includes breadth of search. **That is, an online optimization method could be used to produce training samples for deep learning.** Hence, in this paper, we focus on online optimization methods with the understanding that they could be used "directly" to make cloud service procurement decisions or "indirectly" informing a DNN to make them, where the DNN encodes optimal decisions for the current cloud service suite as an explicit function of the model parameters of previously experienced workloads.

B. Online Optimization and Spot Cores

Recently, researchers and practitioners have explored the use of other traditional, more lightweight decision-making methods, such as model-based adaptive control, PID control, and Markov Decision Processes (MDPs), in which DNNs play only a partial role at most (e.g., [22], [30]). For example, particle-swarm optimization has been used for load balancing [23], and genetic algorithms (GAs) have been employed to explore service suites and react to workload and service changes [28], [29].

Simulated annealing (or just annealing) has been widely used for complex, non-convex optimization problems, including practical applications, since its development in the 1980s [1], [13], [16]. Annealing is suitable for resource management problems in dynamic environments with non-convex objectives and indefinite time horizons. It can quickly react to changes by adjusting its temperature parameter and expanding its "local neighborhood" sets. Annealing can also be hybridized with, e.g., GAs or Tabu search, to improve search efficiency.

We recently evaluated annealing for container sizing of microservice workloads [3]. Annealing offers the advantage of confining incremental changes to the cluster within the local neighborhood (see the next section), thus minimizing the impact of change on the current workload.

The future availability of "local" spot cores [4], [27] on the same servers as currently employed VMs could also facilitate exploration without workload interruption. Spot cores allow VMs to temporarily acquire additional resources for a fee, which may vary over time. With spot cores, online cluster configuration searches can be conducted to assess cost/performance trade-offs without interrupting the current workload.

C. Contributions

In this paper, we study an online optimization based on an annealing approach to provisioning an IaaS (VM) cluster 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., [26]) and online (runtime), our focus is on the latter.

This paper is organized as follows. Some background on annealing and a description of our Virtual Machine (VM) procurement approach is given in Section II-A. A service procurement problem is formulated and an annealing based approach to it is described in Section II-B. In Section II-C, the results are given of a study using the Apache Spark [24] distributed application and CloudLab [8]. In addition to HiBench workload case-studies, we apply an online annealing method to select the number and sizes of VMs for a deep-learning workload. In Section III, the results of a study are given using hypothetical spot cores on an AWS cluster managed by Kubernetes (K8s). The paper concludes with a brief discussion of future work in Section IV.

II. VM Procurement by Simulated Annealing

A. Background on 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 and ν, the annealing Markov chain on D at temperature $\tau > 0$ has transition probabilities from x to $x' \in \nu(x)$,

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

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 [13], 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) Markov chain Y has (Gibbs) stationary

[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].

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 [18].

B. VM Procurement Approach

Consider a long job stream whose composition and workload profile may change periodically over time. This change could be due to changes in the types of jobs and their proportions and/or the datasets on which the jobs operate. A goal here is to decide on the most performance-and-cost effective IaaS cluster, including consideration of autoscaling costs, dynamic changes to the effective capacities and prices of services considered, or new service offerings.

First, consider a stream of the same type of job, e.g., the jobs are described by the same DAG of component tasks, for a fixed cluster. Let x_k be the cluster configuration for the n^{th} job (e.g., in particular, indicating the total number of cores in the cluster). Starting with a random configuration for x_0, we apply annealing over the course of the job stream with minimizing objective

$$Y_n = t_n + \lambda c_n$$

where t_n and c_n respectively are the total execution time and a "cost" for job n under the currently chosen service configuration. The user specified term $\lambda > 0$ weighs the cost against the execution time.

Second, we formulate the objective to be able to consider a blend of multiple types of jobs. The composition of the blended workload can be described using weighted averages of each type. For example, consider a workload that consists of N types of jobs. The minimizing objective for this workload can be described as

$$Y = \sum_{i=1}^{N} \alpha^{(i)} t^{(i)} + \lambda C, \tag{1}$$

where $\alpha^{(i)} > 0$ is the weight for workload type i ($\sum_{i=1}^{N} \alpha^{(i)} = 1$), $t^{(i)}$ is a measure of its performance to be minimized, and $\lambda > 0$ is a parameter relatively weighing a measure of the cost of the cluster, C. The parameters $\alpha^{(i)}$, which can be interpreted as the relative priority of workload type i, may change dynamically as the workloads experience variations over time; see Section II-G. Alternatively, e.g., use autoregressively estimated averages t_{avg} or 95th percentile execution times (the latter approximated by t_{avg} plus two estimated standard deviations). Upon arrival of a new job (or new set of jobs) n, we run it (them) with the configuration $z_n = x_{n-1} + e_v$, where e_v represents a possible "step size" (incremental change) when exploring configurations and x_{n-1} represents the current "accepted" configuration. We set $x_n = z_n$ (i.e., accept the configuration z_n) with probability

$$\exp(-\max\{Y_n - Y_{n-1}, 0\}/\tau)$$

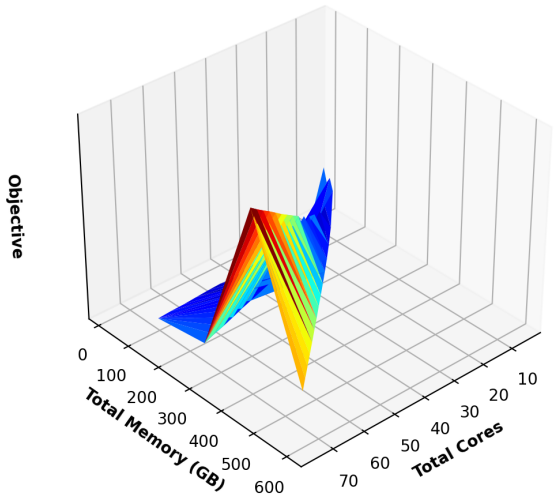


Fig. 1. Blended workload characteristic based on four types of EC2 instances: General Purpose, Compute Optimized, Storage Optimized, and Memory Optimized. Note that the non-convexity of the objective is a consequence of how the categorical VM types are ordered to form "local neighborhoods" of cluster configurations for the annealing mechanism. Note that annealing is suitable for non-convex optimization.

where Y_n is the objective evaluated for job n, and τ is the temperature parameter controlling the degree of exploration and exploitation; otherwise $x_n = x_{n-1}$.

Incremental changes e_v to the existing cluster x define the neighborhood function $\nu(x)$ over specific instances of VMs of different families and the number of each instance. For the example where the user may want to have a homogeneous cluster in which case e_v could is defined so that $x + e_v$ represents different numbers of VMs compared to x but of the same type, or the same number of VMs as x but of different types. In the latter case, the VMs could be in the same family as those of x but smaller or larger (based on number of cores or GPUs), or could be in a different family with say the same number of cores but different hardware accelerators (including lacking them). Alternatively, the user may want to consider heterogeneous clusters in which case $x + e_v$ could differ from x based on only a single VM. We consider heterogeneous clusters in the following experiments.

C. Experimental Results

We evaluate our simulated annealing approach using CloudLab [8] machines. Our set-up consists of six rs630 nodes (a master node and 5 worker nodes). Using Apache Spark (3.1.2v), we exploit the Spark configuration file in the master node to set the new desired configuration when performing annealing.

D. Modeling Cost of EC2

We reference AWS's EC2 per core pricing for different on-demand instance types. For each instance type, a pre-defined memory allocated per core is assumed (e.g., m6g.medium

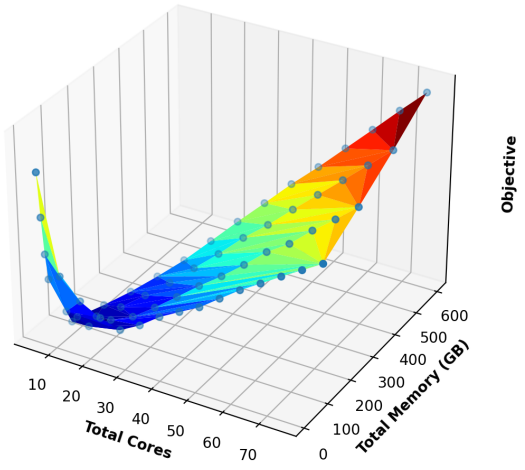


Fig. 2. Blend of HiBench workload types (Wordcount, K-means and PageRank) with adjustment for storage-optimized instances for better comparisons.

instances allocate 4 GB of memory per core procured). We also consider hypothetical instances "between" those offered by AWS with corresponding price adjustments (further details are given in section II-D1).

1) Each job executed independently: Using a blend of three different type of HiBench jobs [15] (we set the weight factor $\lambda = 5$, which we empirically found suitable for our experiments), we evaluate the objective values, under different configurations, for the blended workload as described in section II-A. We generate a stream of independent jobs and record their average execution times under configurations provided by the annealing process, i.e., number of VM instances of a certain type where each VM type is characterized by a number of cores and memory allocated per core. Figure 1 depicts the objective values for different configurations of such blended workload. The peaks in figure 1 evaluate the objective values when using storage-optimized instances[5]. Note that different ordering of the *categorical* instance types in defining the search space may introduce local minimums that are not global. Annealing can overcome local minima more quickly at higher temperatures τ.

We replace the pricing of storage-optimized instances with a hypothetical family of instances for better comparisons (i.e., all families of instances have similar local storage performance) shown in figure 2. Figure 3 shows different simulations of jobs streams while performing annealing under different temperatures. The red dots represent different service-cluster configurations. From figures 3 and 4, we can observe that more exploration is performed by annealing as the temperature τ increases. Furthermore, figure 5 shows that as the temperature increases, the annealing process more quickly finds the configuration that minimizes the objective[6], but with greater service variation due to higher service exploration.

2) Jobs executed in parallel: The annealing approach can also be employed for workloads that consist of jobs that are executed in parallel (i.e., when jobs compete for resources) and a job queue may be present. The minimizing objective can be adjusted for this case by measuring the total sojourn time of jobs instead of just the execution times. The annealing process performs similarly to what is described in section II-D1.

E. Deep Learning Workload

The annealing method can also be utilized in training Deep Neural Networks (DNNs), i.e., deep learning. Figure 6 depicts the characterization of distributed deep learning, on our Spark cluster, using the Keras library [7] to train a Convolutional Neural Network (CNN) to recognize handwritten digits of the MNIST dataset [17]. Using the pricing model described in section II-D, the goal is to find the service configuration that minimizes the objective function as described in section II-A (we set the weight factor $\lambda = 1$). The execution time here refers to training a model for one epoch. In our experiments, we set the initial configuration to 1 core with 4 GB of memory. Figure 7 shows the selected configurations that were explored by the annealing process (shown as red dots). From figure 8, we find that our annealing approach is capable of quickly finding the configuration that minimizes the objective.

F. Adaptation of Annealing

We evaluate the adaptation of annealing for dynamic changes in the blended workload. Figure 9 shows the computed objective values for a stream of blended HiBench jobs. Here, we portray the change in workload as a change in the distribution of the blend (red vertical line depicts the point of time when the change occurs). The blue lines depict the objective values computed before the change occurs, while the orange lines depict after such. The oscillation in the computed objective values is due to the exploration nature of annealing at a positive temperature. After the change, we observe that annealing adapts to the change by finding the new objective-minimizing service configuration through exploration as well.

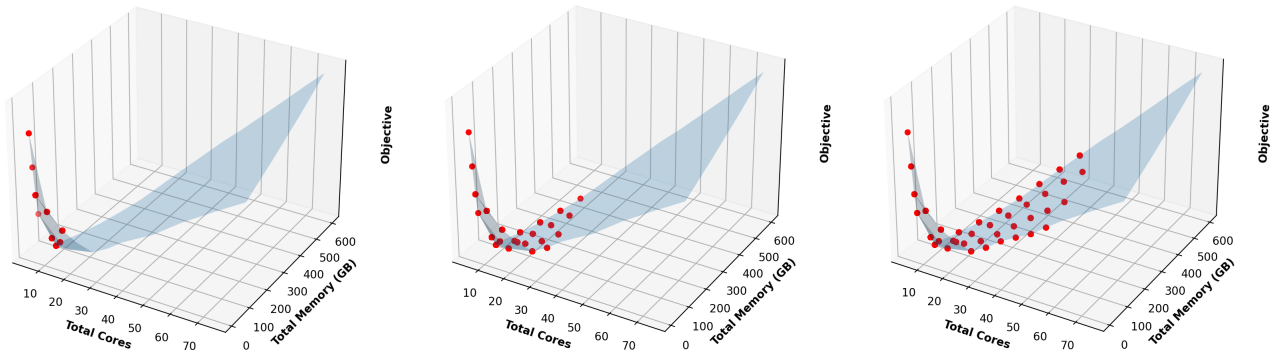
G. Discussion: Adapting τ and λ

For the example of cluster management based on online optimization considered herein, two critical parameters warrant careful consideration: the temperature τ of the annealing algorithm (trading off breadth and depth of search), and the objective Y's parameter λ (trading off its user-specified performance (SLO) and cost component objectives). Some approaches to adjust the temperature are:

- Continuous Logarithmic: The classical logarithmic (slow) cooling schedule for a stationary system, e.g., [1].
- Threshold Based: Predefined thresholds for the objective value (or its components) are established with temperature changes at each threshold. If, e.g., Y itself crosses below a threshold, the temperature is reduced to encourage exploitation. Alternatively, if, e.g., the cost component exceeds a certain threshold, the temperature may be increased to

[5]Latency to local storage was not a significant performance factor in our experiments. Hence, AWS instances that use Elastic Block Store (EBS) are also emulated using SSDs, but with their actual AWS pricing.

[6]In this paper, in order to assess the annealing mechanism, we separately identify by exhaustive search the configurations that globally minimize the objective under consideration.



(a) Low Temperature Annealing ($\tau = 10$) (b) Medium Temperature Annealing ($\tau = 25$) (c) High Temperature Annealing ($\tau = 50$)

Fig. 3. Performing annealing on a blended workload of three types of jobs: Wordcount, K-means, and Pagerank

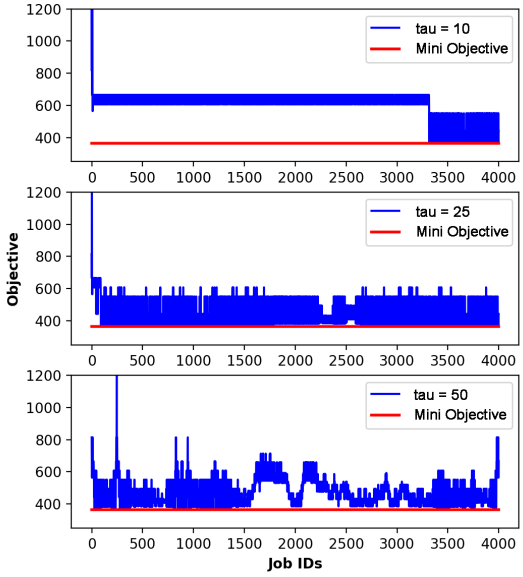


Fig. 4. The occurrences of exploitation and exploration depends on the temperature. Each chart in the above represent performing an annealing process under a fixed temperature for a blended workload.

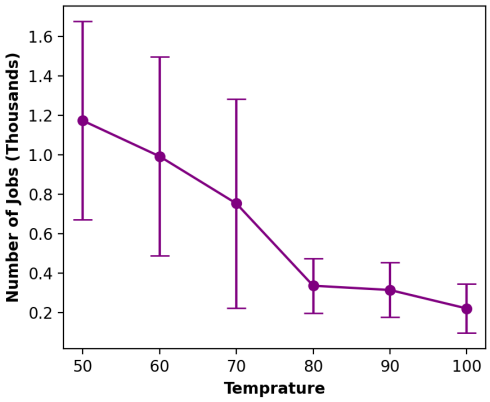


Fig. 5. The number of jobs submitted before a configuration with a minimum objective is selected depends on the temperature. The vertical bars represent 95% confidence intervals (± 2 sample standard deviations).

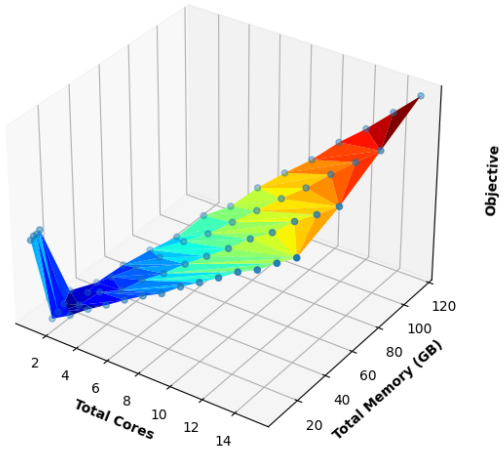


Fig. 6. Training a Deep Neural Network using MNIST dataset for handwritten digits recognition.

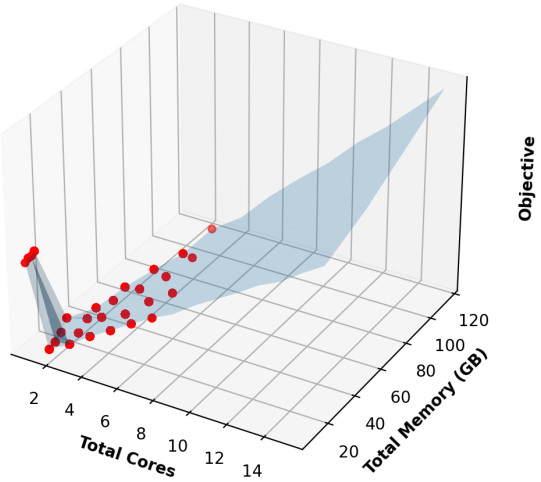


Fig. 7. Performing annealing on a Deep Neural Network using the MNIST dataset.

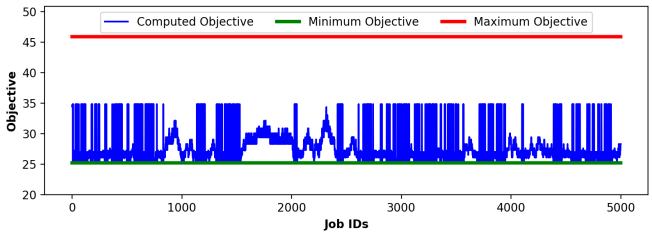


Fig. 8. Computed objectives for selected configurations by the annealing process (blue line). The red and green lines correspond to configurations with the maximum and minimum objective values, respectively.

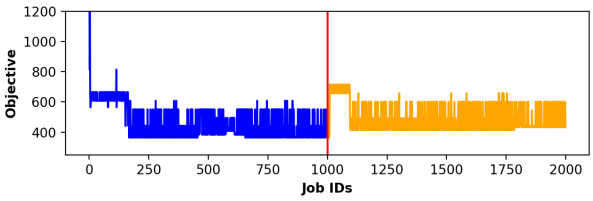


Fig. 9. Annealing adapting to change in the blended Hibench workload.

encourage exploration. More specifically, temperatures can, e.g., increase exponentially (e.g., $\tau \to 2\tau$) and decrease additively (e.g., $\tau \to (\tau - \delta)^+$ for $\delta > 0$).

- Iteration Based: The temperature adjustment is based on the number of iterations or the convergence behavior. E.g., if no significant change in the cost objective is observed after a certain number of iterations, and user-specified performance or cost objectives are not met, then the temperature is increased to encourage further exploration.

The λ parameter is similarly adjusted. E.g., after a certain number of iterations: If the performance (SLO) is met but a user-specified constraint is not, then λ can exponentially increase. Conversely, if cost budget is met but the SLO is not, then λ can decrease additively decrease. If there are plural job types as in (1), the parameters α can be similarly adjusted.

III. Using Spot Cores

In the future, VMs could be dynamically augmented with local cores (or hardware accelerators like GPUs) which are unreserved or reserved but unused, where in the latter case they can be revoked. E.g., Google K8s Engine (GKE) [12] provides a service wherein the user is "not charged" for unused cores in their cluster. Such unused cores could serve as "spot cores" for another local VM. The added cost of using spot cores is weighed against the convenience of not subjecting the workload to preemption or added queueing delays when deciding whether to change one or more VMs in a cluster.

We emulate spot cores within a K8s environment, where a Social Media network microservice from DeathStarBench [9] is deployed onto the cluster. The cluster configuration comprises a master node and three K8s worker nodes (noted as w_i, $i \in \mathbb{Z}$), along with an additional node dedicated to workload generation (using Locust [6], a HTTP benchmarking tool). Throughout our experiments, we adopt the pricing of an `m5.4xlarge` instance (and the pricing for `m5.8xlarge` instance for spot cores) as a reference for cost calculation across the entire cluster. Notably, we assume that the cost associated with Spot Cores exceeds that of the cores procured with the virtual machines. We use K8s default resource management mechanisms for the deployed application. Define the objective

$$Y = t_{\text{avg}} + \lambda(C_{\text{v}} + C_{\text{s}}),$$

where t_{avg} is the (autoregressively estimated) average execution time of a request incident to the application and C_{v} and C_{s} respectively are the cost of VM and spot cores.

Table I
A comparison of the objective values across various Kubernetes cluster configurations, both with and without spot cores. Note that it is assumed that the cost of spot cores is greater than that of procured cores within a VM.

Configuration (cores for w_1,w_2, and w_3)	Avg. Exec. Time (ms)	Avg. Obj. ($\lambda = 10$)	Avg. Obj. ($\lambda = 1$)
(16, 16, 16)	33	56.04	35.30
(16, 16, 32)	20	79.08	24.60
(16, 16, [16, 16 spot])	20	102.12	26.91

Table I shows a comparison of three different cluster configuration of the worker nodes for the deployed application (indicating the number of cores w_i for each worker i). The first configuration considers using `m5.4xlarge` as reserved instances for the K8s workers. The second configuration considers replacing the third worker node with a `m5.8xlarge`. Lastly, the third configuration has the same number of cores of the second configuration. However, we consider the pricing of spot cores of the remaining 16 cores of the the third worker node when evaluating the objective value (for our experiments, we set the pricing of the spot cores to be double of the pricing VM cores). When $\lambda = 1$, the second configuration is preferable to the first, which is not the case when user is more price-sensitive with $\lambda = 10$.

IV. Conclusions

In this paper, we advocated for an online optimization approach, such as simulated annealing, for management of a cluster of on-demand-VM resources based on user-specified performance (SLOs) and cost objectives. We gave performance results for the problem of VM service selection where the topography of the annealing objective depends on how the VM services are encoded for purposes of search. In particular, we showed how breadth of search can be used to overcome sub-optimal local minima in the search objective. We also described how the optimization hyperparameters (annealing temperature and objective parameters) can be set toward meeting the user's requirements on performance and cost. Finally, we studied how "spot" cores (or hardware accelerators) could be used in the service-selection process.

Acknowledgements

NSF grants 2212201 & 2122155 supported this research.

References

[1] E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines*. Wiley, 1989.

[2] N. Alfares, A.F. Baarzi, George Kesidis, and A. Jain. Iaas procurement by simulated annealing. http://arxiv.org/abs/2207.04594, 11 July 2022.

[3] N. Alfares and G. Kesidis. Container Sizing for Microservices with Dynamic Workloads by Online Optimization. In *In Proc. ACM Workshop on Container Technologies and Container Clouds (WoC)*, Bologna, Italy, Dec. 2023.

[4] Pradeep Ambati, Inigo Goiri, Felipe Frujeri, Alper Gun, Ke Wang, Brian Dolan, Brian Corell, Sekhar Pasupuleti, Thomas Moscibroda, Sameh Elnikety, Marcus Fontoura, and Ricardo Bianchini. Providing SLOs for Resource-Harvesting VMs in Cloud Platforms. In *Proc. USENIX OSDI*, 2020.

[5] Romil Bhardwaj, Kirthevasan Kandasamy, Asim Biswal, Wenshuo Guo, Benjamin Hindman, Joseph Gonzalez, Michael Jordan, and Ion Stoica. Cilantro: Performance-Aware Resource Allocation for General Objectives via Online Feedback. In *Proc. USENIX OSDI*, 2023.

[6] C. Bystrom, J. Heyman, and T. Gustafsson. Locust. https://github.com/locustio/locust.

[7] Francois Chollet and al. Keras. https://github.com/fchollet/keras, 2015.

[8] Cloudlab. https://www.cloudlab.us/.

[9] C. Delimitrou and al. Deathstarbench. https://github.com/delimitrou/DeathStarBench.

[10] C. Delimitrou and C. Kozyrakis. HCloud: Resource-Efficient Provisioning in Shared Cloud Systems. In *Proc. ASPLOS*, 2016.

[11] Wei Gao, Zhisheng Ye, Peng Sun, Yonggang Wen, and Tianwei Zhang. Chronus: A Novel Deadline-Aware Scheduler for Deep Learning Training Jobs. In *Proc. ACM SoCC*, 2021.

[12] GKE. Google Kubernetes Engine pricing. https://cloud.google.com/kubernetes-engine/pricing.

[13] B. Hajek and G. Sasaki. Simulated annealing - to cool or not. *Systems and Control Letters*, 12(5):443–447, June 1989.

[14] S. Higginbotham. Show Your Machine-Learning Work. *IEEE Spectrum*, Dec. 2019.

[15] Shengsheng Huang, Jie Huang, Yan Liu, and Jinquan Dai. HiBench : A Representative and Comprehensive Hadoop Benchmark Suite. https://www.intel.com/content/dam/develop/external/us/en/documents/hibench-wbdb2012-updated.pdf, 2012.

[16] G. Kesidis and E. Wong. Optimal acceptance probability for simulated annealing. *Stochastics and Stochastics Reports*, 29:221–226, 1990.

[17] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] Spark. http://spark.apache.org, last accessed, Sept. 2017.

[25] TCMFLEXERA. State of the Cloud Report. https://info.flexera.com/CM-REPORT-State-of-the-Cloud, 2023.

[26] 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.

[27] Yawen Wang, Kapil Arya, Marios Kogias, Manohar Vanga, Aditya Bhandari, Neeraja J. Yadwadkar, Siddhartha Sen, Sameh Elnikety, Christos Kozyrakis, , and Ricardo Bianchini. SmartHarvest: Harvesting Idle CPUs Safely and Efficiently in the Cloud. In *Proc. ACM EuroSys*, 2021.

[28] 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.

[29] 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.

[30] H. Zhang, Y. Tang, A. Khandelwal, J. Chen, and I. Stoica. Caerus: Nimble Task Scheduling for Serverless Analytics. In *Proc. USENIX NSDI*, Apr. 2021.