VASIM: Vertical Autoscaling Simulator Toolkit

Anna Pavlenko, Karla Saur, Yiwen Zhu, Brian Kroth, Joyce Cahoon, Jesús Camacho-Rodríguez
Microsoft, USA
{firstname}.{lastname}@microsoft.com

***Abstract*—In recent years, autoscaling has garnered significant attention in cloud computing, emphasizing cost efficiency, performance optimization, and availability for dynamic workloads. New algorithms for horizontal, vertical, and hybrid scaling, targeting instances, VM specifications, and resources like CPU, memory, and IO, have emerged. Various approaches, including forecasting and custom autoscaling functions, are used. However, conducting comprehensive end-to-end testing remains a complex and costly endeavor due to the variety of technology constraints involved.**

This paper introduces VASIM, an autoscaling simulator toolkit designed for testing recommendation algorithms, with a particular focus on CPU usage in VMs and Kubernetes pods. The toolkit replicates common components found in autoscaler architectures, including the controller, metrics collector, recommender, and resource updater. It enables a comprehensive simulation of the entire autoscaling system's behavior, with the flexibility to customize various parameters. In our demonstration, we showcase VASIM's versatility across multiple use cases, highlighting its effectiveness in evaluating autoscaling strategies, fine-tuning parameters, comparing algorithm performance, and addressing autoscaling-related challenges. This underscores VASIM's critical role in expediting algorithm development and refinement by providing a controlled environment for testing and experimentation.

***Index Terms*—autoscaling, simulation, resource management, cloud computing.**

I. INTRODUCTION

Cloud computing has revolutionized data systems development and management, offering on-demand resources and scalability. In this environment, data systems are commonly deployed using VMs or, more recently, using containers through modern platforms like Kubernetes (referred to as K8s), which has gained widespread adoption for deploying data systems in the cloud. During the provisioning of resources in such deployments, users often need to specify their initial requirements from a myriad of options, including CPU cores, memory sizes, and more, due to the difficulty of accurately estimating these requirements in advance, particularly given the dynamic and ever-changing nature of many workloads. As a result, users tend to fall into two categories: *over-provisioning* increases costs while *under-provisioning* causes performance problems due to "throttling".

In response to these challenges, autoscaling has become fundamental in cloud computing. It dynamically optimizes resource allocation, improving efficiency and cutting costs. In this work, we consider *vertical autoscaling*, which involves the addition or removal of resources from existing instances, i.e., VMs or containers. This is particularly relevant for monolithic data systems with fixed instance counts or limitations in horizontal scaling due to the size of data copy operations required for creating new instances [1].

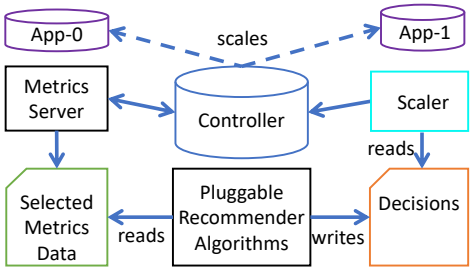

Fig. 1. Common resource autoscaler architecture.

Figure 1 illustrates the key components of the architecture of a centralized *autoscaler* component designed for the dynamic scaling of cluster resources. The *data system* is running within a compute instance. The *Controller* serves load balancing and ensures high availability. It publishes telemetry data related to the data system, including real-time resource usage and allocation (CPU/memory/IOPS), that is managed and stored by the *Metrics Server*. The *Recommender Algorithm*, which is pluggable, analyzes these metrics to make resource allocation decisions. Lastly, the *Scaler* monitors the *Decisions* generated by the algorithm, conducts health and resource safety checks, and instructs the controller to adjust resource allocation as needed. It is important to note that this same autoscaler can be applied in various scenarios, as supported by previous studies [2]–[5]. This includes scaling K8s *pods*[1], optimizing VM capacities, or efficiently managing storage resources.

The development of *autoscaling recommender algorithms* within the previous architecture presents a significant challenge, requiring costly testing and meticulous fine-tuning procedures. This complexity arises from multiple factors: (1) the algorithms have numerous parameters, making correct configuration difficult; (2) real-world testing across various scenarios is necessary, including sudden spikes and low demand periods; and (3) appropriate metrics must be considered to assess algorithm effectiveness based on user requirements and budget constraints. These metrics include: (1) *slack*, denoting the extraneous resources, such as CPU and memory, allocated to prevent resource strain during utilization spikes; (2) *throttling*, representing instances where CPU or another resource type lacks sufficient capacity, resulting in performance issues or system crashes that can jeopardize system stability; and (3) *number of scalings*, as excessive scaling can negatively

[1]A *pod* serves as a logical encapsulation for one or more containers that share the same resources within a K8s cluster.

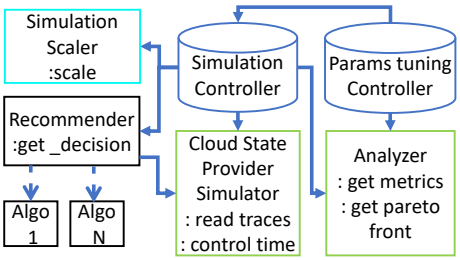

Fig. 2. VASIM infrastructure.

impact system performance due to disruptions in resource allocation. To overcome this challenge, *simulations* are essential for quickly evaluating recommendation algorithms and estimating potential cost savings.

This paper introduces VASIM, a novel tool designed to address the intricacies of autoscaling algorithm evaluation, allowing faster iteration. While the industry already offers a variety of simulation tools focusing on diverse aspects (e.g. energy efficiency, cloud topology optimization, fault tolerance, etc. [6]), VASIM distinguishes itself by placing particular emphasis on the "recommender" component of autoscaling. VASIM makes several key contributions. **(1) Resource Efficiency and Cost Reduction:** VASIM makes a significant impact by reducing costs through the elimination of resource-intensive real-time testing and scaling operations. This approach empowers developers to iteratively fine-tune algorithms within a controlled, cost-effective environment, thus expediting development cycles. **(2) Multi-objective Optimization:** Addressing the challenge of managing multiple conflicting parameters in autoscaling algorithm development, VASIM leverages Pareto optimization techniques that effectively explores parameter configurations, enabling developers to strike a harmonious balance between critical metrics like resource slack and throttling levels. **(3) Recommender Algorithm Testing:** VASIM streamlines the process of testing and evaluating recommendation algorithms across 4000+ workloads. Researchers can gauge scalability and sensitivity to various workload types, facilitating data-driven fine-tuning and algorithmic enhancements. **(4) Versatility and Adaptability:** Unlike restrictive tools tailored to specific autoscaling systems, VASIM stands as a versatile, adaptable solution suitable for a wide range of systems. Users have the flexibility to interchange components, experiment with various recommendation strategies, switch machine learning models, and assess the implications of changes in scaler mechanisms—all within a controlled and customizable environment.

In the rest of the paper, we provide an in-depth exploration of VASIM and outline our plans for the demonstration of its utility across various scenarios.

II. VASIM System overview

VASIM facilitates controlled evaluations of scaling system logic and algorithms, offering a cost-effective alternative to resource-intensive system assessments.

Simulator Components. The components of VASIM are depicted in Figure 2. The *Simulation Controller* orchestrates the entire simulation. It initiates the process by loading simulation parameters and triggering the operation of other components. Upon initialization, the *Cloud State Provider Simulator* analyzes input resource traces and generates metrics that represent resource allocation and utilization at any given moment. The customizable *Recommender* runs the autoscaling algorithm using these metrics and determines scaling actions that are executed in a simulated environment by the *Simulation Scaler*. The *Parameter Tuning Controller* configures the algorithm, specifying argument values employing strategies like grid or random search. Multiple configurations are run in parallel on multicore machines to fine-tune the algorithm, accelerating the process and enhancing scalability. *Analyzer* collects performance metrics on each simulation run, including instances of insufficient resources and scaling events based on the decisions made by the recommender and workload trace.

Pareto Frontier Analysis. The Analyzer component plays a vital role in assessing the system's performance. It allows us to create Pareto frontiers that helps us understand the trade-offs between scaling frequency, resource usage, and cost. A Pareto frontier is a concept in multi-objective optimization that represents the set of optimal solutions where no improvement in any one objective can be achieved without sacrificing performance in at least one other objective. Each point on the Pareto frontier represents a distinct algorithm configuration, offering a range of choices from cost-effective to high-performance settings.

To pinpoint the optimal state with minimized slack, throttling, and the number of scalings, denoted as L, we introduce the following extended objective function G:

$$\mathrm{G}(\alpha, \beta, p) = \alpha \cdot K(p) + \beta \cdot C(p) + L(p), \tag{1}$$

where α and β are scalar coefficients that represent the penalties for having slack and throttling, respectively. By adjusting these coefficients, the preference for higher slack/throttling/number of scalings can be tailored. The parameter p represents the combination of parameters, while $K(p)$, $C(p)$, and $L(p)$ denote the observed (simulated) total slack, insufficient CPU, and the number of scalings, respectively. This function computes an objective value by combining the three observed metrics, assigning a weight of α to slack, β to throttling, and a weight of 1 to the number of scalings.

The optimal parameter combination set can be estimated by iterating over all possible values of α and β:

$$\hat{p} = \arg\min_{p} G(\alpha, \beta, p) | \forall \alpha, \beta \in D, \tag{2}$$

where we sample random numbers from a log-uniform (reciprocal) distribution with $\ln(D) \sim \mathcal{U}(-100, 100)$ to encompass a broad range of values.

The Pareto curve helps us choose the ideal parameter configuration that aligns with our business requirements.

Simulator Accuracy. To validate the VASIM's accuracy, we conducted extensive experiments comparing the VASIM's

scaling decisions with real benchmarks using statistical analysis, such as pairwise t-tests. The consistent alignment between the real and simulated runs affirms the VASIM's reliability. For instance, in the scenario depicted in Figure 3 and Figure 4, the only notable inconsistency occurred after 20 hours when the actual production system failed to downscale as intended due to a transient error.

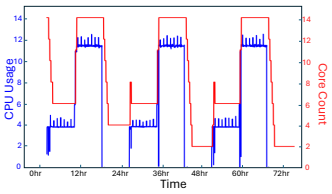

Fig. 3. Real run.

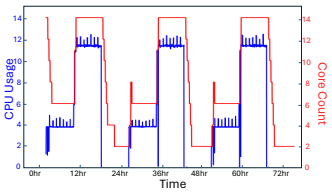

Fig. 4. Simulated run.

Versatility and Adaptability. Our simulator is versatile, allowing for the evaluation of machine learning algorithms for cloud resizing, migration, and recommendations. It supports component replacement, model switching, and testing of different scaling mechanisms, enabling comprehensive testing, capacity planning, and performance optimization. VASIM can use input traces from Alibaba [7] and various database systems' workloads, accommodating for differences in their scaling behaviors by adjusting parameters for scaling duration.

III. Implementation and Evaluation

VASIM can be used on standard machines and is implemented as a Python package. One simulation run can take just minutes depending on the selected algorithm and input trace data. Additional algorithms can be incorporated as Python classes that conform to a predefined interface. Currently, VASIM has been used to validate and tune three different recommendation algorithms using Alibaba traces [7], internal Microsoft database traces, and K8s traces generated by running BenchBase benchmarks [8].

In a representative experiment, spanning 8 days, we conducted 10,000 simulation runs using distinct algorithm configurations, completed on an Azure Standard D96ads v5 instance (96 vCPUs, 384 GiB RAM) in less than an hour. This showcases VASIM's efficiency in exploring diverse autoscaling algorithms and strategies swiftly.

IV. Demonstration

VASIM will be demonstrated using an interactive Python web application built with the Streamlit framework. While we acknowledge the potential utility of VASIM in different contexts and for various user roles, our demonstration specifically explores scenarios encountered by researchers and engineers, referred to as *users*, during the development, fine-tuning, and operation of recommender algorithms.

Scenario 1 (Autoscaling Strategy Evaluation). *Consider a cloud-based data system running fluctuating workloads, resulting in occasional resource over-provisioning and under-provisioning. In this context, a user aims to implement an autoscaling algorithm. Their objective is to assess the effects of the dynamic autoscaling strategy on resource utilization, cost, and performance metrics, with the intention of refining and tuning the algorithm accordingly.*

Initially, the user chooses a representative workload and gather historical CPU trace data to capture real-world variations in resource demand. Subsequently, they integrate a sophisticated recommender algorithm that leverages ML techniques to make scaling decisions based on this trace data.

With VASIM, the user can observe the progression of CPU utilization over time as well as the autoscaling decisions overlayed onto the graph. Figure 5 displays two runs with the same recommender and workload but different parameters. VASIM calculates crucial metrics to evaluate the autoscaling results, such as the count of idle CPU cores, the degree of application throttling, and the frequency of scaling events. An analysis of these metrics yields valuable insights into the effectiveness of the selected autoscaling strategy.

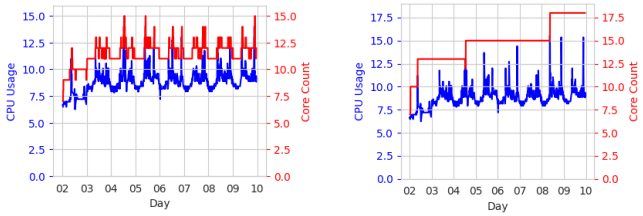

Fig. 5. Simulations: same autoscaling algorithm, different parameters

Operating offline, VASIM enables an iterative approach to refining the recommender algorithm's parameters. These parameters, including scaling thresholds, time-to-scale, and recovery times, can be adjusted to fine-tune critical settings. By systematically exploring the parameter space and employing search techniques like random and grid search, or advanced methods such as Bayesian approaches [9], users can optimize the autoscaling strategy to align with the unique workload characteristics.

This scenario not only illustrates the practical value of VASIM but also highlights its efficacy in the process of fine-tuning and optimizing autoscaling strategies within a controlled environment, allowing users to customize these strategies to align with the specific needs of their workloads and the end users of their data systems. It is important to note that multiple algorithms may be available, and within the same scenario, a user has the option to quantitatively assess and compare algorithms' performance.

Scenario 2 (Parameter Tuning Based on Objective Function). *The objective or optimization function for the autoscaling process is adaptable, depending on the specific user requirements, often involving metrics that directly conflict with each other. For instance, the pursuit of minimizing resource slack may conflict with the objective of minimizing throttling. In this scenario, a user aims to thoroughly explore the vast space of configuration values for the parameters exposed by their algorithm. They also consider metric weights that reflect the importance of specific aspects, such as slack or throttling,*

TABLE I
CORRELATION BETWEEN PARAMETER AND PERFORMANCE.

Params	sum slack	sum insufficient cpu	num scalings	num insufficient cpu	insufficient observations
a	0.40	-0.03	-0.12	-0.13	0.13
b	-0.31	0.39	0.21	0.38	0.38
c	0.29	-0.04	0.04	-0.08	-0.08
d	0.61	0.44	0.72	0.48	0.48
e	0.22	0.08	-0.12	-0.12	0.12

in their optimization goal. The ultimate aim is to identify configurations that align with the user's objectives.

VASIM uses Pareto optimization to concurrently reduce resource slack, throttling levels, and the number of scaling events (as depicted in Figure 6, which shows that decreasing slack results in an increase in throttling or scaling). This approach can be seamlessly expanded to accommodate additional dimensions as necessary. Through experiments conducted using VASIM, the user gains valuable insights into how various parameters interact and directly impact autoscaling decisions, which in turn allows them to fine-tune the algorithm.

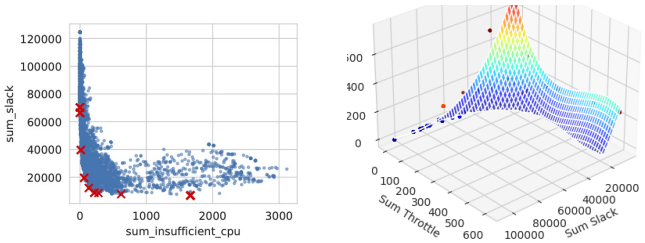

Fig. 6. Illustration of the Pareto frontier analysis, showing algorithm configurations based on trade-offs. This visualization is based on an extensive dataset of 10,000 experiments with different parameter configurations.

Scenario 3 (Scalability and Performance Tailored to Diverse Workloads). *The development and testing of new recommendation algorithms can be a challenging and resource-intensive undertaking. Users often find themselves in need of a practical and efficient means to evaluate their algorithms across a spectrum of workloads and shapes, all without the necessity of real-time execution.*

A user wants to leverage VASIM's capabilities to conduct comprehensive evaluations of their recommendation algorithms using a broad variety of workloads. This process begins by collecting a set of traces that accurately represent their operational context. To illustrate this, we clustered a collection of more than 4000 CPU workload traces from Alibaba [7] and extracted 9 traces (c_29759, c_10235, c_12104, c_23544, c_24173, c_26742, c_29247, c_29345, c_48113) representing the most common scenarios.

VASIM allows the user to perform a thorough examination of algorithm stability across various workload types, including cyclical, bursty, and monotonic variations. Additionally, it enables a detailed assessment of various recommendation metrics, revealing their relative importance in specific scenarios.

Furthermore, this scenario delves into algorithm sensitivity, revealing which parameters the recommender algorithm responds to most and identifying parameters that can be reduced without compromising performance (e.g., as illustrated in the parameter heat map in Table I). This rigorous analysis equips users with the knowledge needed to craft recommendation algorithms that are robust and tailored to diverse scenarios.

In summary, this scenario illustrates VASIM's ability to expedite the development and testing of recommendation algorithms. It allows users to efficiently evaluate algorithm performance across a wide array of workloads, make precise refinements, and create recommendations that perform well under diverse conditions, all within minutes rather than days.

V. CONCLUSION

In this paper, we introduced VASIM, a comprehensive simulation framework designed for the evaluation and fine-tuning of autoscaling algorithms within a controlled environment. Evaluated through numerous real-world use cases involving popular data systems, VASIM shows its adaptability and effectiveness, promising a significant impact on the development and testing of algorithms. While VASIM's effectiveness relies on the accuracy of workload traces and simulation assumptions, it represents a valuable step forward, streamlining development and providing valuable insights and optimization prospects. In fact, VASIM has already played a key role in the development and tuning of algorithms that we have integrated into the K8s vertical pod autoscaling module [1]. This is is facilitated by the close alignment between the VASIM architecture and commonly used resource management architectures. In our future work, we plan to expand support to additional resources beyond CPU and incorporate additional metrics, potentially emitted by the specific systems being tuned.

REFERENCES

[1] A. Pavlenko, J. Cahoon, Y. Zhu, B. Kroth, M. Nelson, A. Carter, D. Liao, T. Wright, J. Camacho-Rodríguez, and K. Saur, "Vertically Autoscaling Monolithic Applications with CaaSPER: Scalable Container-as-a-Service Performance Enhanced Resizing Algorithm for the Cloud," in *ACM SIGMOD*, 2024.

[2] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle, "Elasticity in Cloud Computing: State of the Art and Research Challenges," *IEEE Transactions on Services Computing*, vol. 11, no. 2, pp. 430–447, 2018.

[3] 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 *EuroSys*, 2020.

[4] T. Lorido-Botran, J. Miguel-Alonso, and J. Lozano, "A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments," *J Grid Computing*, vol. 12, pp. 559–592, 2014.

[5] H. Fernandez, G. Pierre, and T. Kielmann, "Autoscaling Web Applications in Heterogeneous Cloud Infrastructures," in *IEEE International Conference on Cloud Engineering*, 2014.

[6] S. Piraghaj, A. Dastjerdi, R. Calheiros, and R. Buyya, "ContainerCloudSim: An environment for modeling and simulation of containers in cloud data centers," *Software: Practice and Experience*, vol. 47, no. 4, pp. 505–521, 2017.

[7] Alibaba Inc., "Alibaba Open Cluster Trace," 2018, https://github.com/alibaba/clusterdata.

[8] D. E. Difallah, A. Pavlo, C. Curino, and P. Cudré-Mauroux, "OLTP-Bench: An Extensible Testbed for Benchmarking Relational Databases," *PVLDB*, vol. 7, no. 4, pp. 277–288, 2013.

[9] C. M. Bishop and N. M. Nasrabadi, "Pattern Recognition and Machine Learning," *J. Electronic Imaging*, vol. 16, no. 4, 2007.