BIAS Autoscaler: Leveraging Burstable Instances for Cost-Effective Autoscaling on Cloud Systems

Jaime Dantas, Hamzeh Khazaei and Marin Litoiu
{jaimecjd,hkh,mlitoiu}@yorku.ca
Department of Electrical Engineering & Computer Science
York University
Toronto, Ontario, Canada

ABSTRACT

Burstable instances have recently been introduced by cloud providers as a cost-efficient alternative to customers that do not require powerful machines for running their workloads. Unlike conventional instances, the CPU capacity of burstable instances is rate limited, but they can be boosted to their full capacity for small periods when needed. Currently, the majority of cloud providers offer this option as a cheaper solution for their clients. However, little research has been done on the practical usage of these CPU-limited instances. In this paper, we present a novel autoscaling solution that uses burstable instances along with regular instances to handle the queueing arising in traffic and flash crowds. We design BIAS Autoscaler, a state-of-the-art framework that leverages burstable and regular instances for cost-efficient autoscaling and evaluate it on the Google Cloud Platform. We apply our framework to a real-world microservice workload, and conduct extensive experimental evaluations using Google Compute Engines. Experimental results show that BIAS Autoscaler can reduce the overall cost up to 25% and increase resource efficiency by 42% while maintaining the same service quality observed when using conventional instances only.

CCS CONCEPTS

• **Computer systems organization** → *Cloud computing*.

KEYWORDS

autoscaling, resource provisioning, burstable instances, cloud computing

WoSC '21, December 6, 2021, Virtual Event, Canada

ACM ISBN 978-1-4503-9172-6/21/12…$15.00
https://doi.org/10.1145/3493651.3493667

ACM Reference Format:
Jaime Dantas, Hamzeh Khazaei and Marin Litoiu. 2021. BIAS Autoscaler: Leveraging Burstable Instances for Cost-Effective Autoscaling on Cloud Systems. In *Seventh International Workshop on Serverless Computing (WoSC7) 2021 (WoSC '21), December 6, 2021, Virtual Event, Canada.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3493651.3493667

1 INTRODUCTION

Elasticity is one of the most important concepts of cloud computing. The ability to dynamically adapt the cluster capacity based on the current demand is pivotal for maintaining Quality of Service (QoS) and optimizing the cost. Autoscaling the resources can not only reduce the overall cost for the customer, but also preserve the Service-level Agreements (SLAs) and Service-level Objectives (SLOs) of the services. This is because most of the workloads face unpredicted variations in traffic during their usage. Sometimes, these spikes in usage can cause interference in the QoS metrics, leading to a negative impact on both the cloud providers and the customers.

Many autoscaling solutions try to mitigate these problems by provisioning additional computational resources to handle unpredicted spikes on their workloads. This is commonly known as overprovisioning the number of resources above the minimum required to handle sudden variation in traffic. The downside of this approach, however, is because this extra capacity is often not entirely used during normal demand, which in turn, leads to waste of resources and consequently, increasing the cost. To avoid wasting computational resources, cloud providers such as Amazon Web Service (AWS), Google Cloud Platform (GCP) and Microsoft Azure introduced burstable instances. We believe that this type of instance is the key to designing cost-efficient solutions on the public cloud, especially for small clusters.

Burstable instances are virtual machines whose CPU capacity is limited to a predefined threshold. Even though they are meant to operate under their operational CPU threshold, the CPU capacity can be boosted to the full standard capacity for small periods. Each cloud provider implements its own proprietary system to manage these instances, and they are

usually based on tokens for CPU credits. AWS, for example, controls the frequency by which their burstable instances can operate above the CPU threshold by using a token-based system. Each minute the instance operates below its CPU threshold, the customer receives a token that can then be spent to boost these instances to CPU values above its threshold for one minute. Similarly, Microsoft Azure implements a token-like control system for their burstable virtual machines.

When we compare the cost of burstable instances with regular ones, we can see huge differences in the overall savings one could have. For instance, the Google Compute Engine *N1 shared-core g1-small* instance (with 1 vCPU at 50% sustained rate and 1.7 GB of memory) cost 52% less than the *N1 standard 1* instance (with 1 vCPU with 3.75 GB of memory). This difference can be as high as 10 times depending on the type of instance and the cloud provider.

AWS offers several types of EC2 burstable instances under the families *T2, T3, T3a, and T4g*. On this cloud provider, for on-demand EC2 instances, the cost of burstable instances varies from 90% to up to 10 times less than regular instances. This figure is similar for GCP as well, reaching up to 8 times in savings for some burstable instances. Unlike AWS, though, GCP offers only two families of burstable instances: *E2 shared-core* and *N1 shared-core*.

We introduce the Burstable Instance Autoscaler (BIAS), an application autoscaler that combines different instance types for scaling virtual machines in the public cloud. BIAS Autoscaler was evaluated on GCP, and it uses the already existing infrastructure and services of GCP to add and remove resources as needed as well as distribute the traffic among the different types of instances. We use the Square-Root Staffing Rule to calculate the number of required servers on the fly, and evaluate our framework on a microservice workload. Our work presents two main contributions:

- We use a well-known technique to create a new solution for reducing the cost and increasing the efficiency of autoscaling systems by leveraging burstable instances in combination with conventional ones. We implement and evaluate our solution in our prototype BIAS Autoscaler, which is open-sourced on GitHub[1], and validate our technique under two distinct scenarios: transient queueing arising in traffic, and flash crowds.
- We demonstrate how our framework can be extended to other cloud providers, and how it can be used to manage other serverless services based on containers such as Kubernetes. We also explain how to implement customized scaling policies on BIAS Autoscaler.

[1] https://github.com/BIAS-Cloud/BIAS-Autoscaler

2 BIAS AUTOSCALER DESIGN

In this section, we present the architecture and design of BIAS Autoscaler, and show how it uses a combination of regular and burstable instances to reduce cost on the public cloud. We evaluate BIAS Autoscaler on GCP, and the source code is openly accessible on GitHub[1].

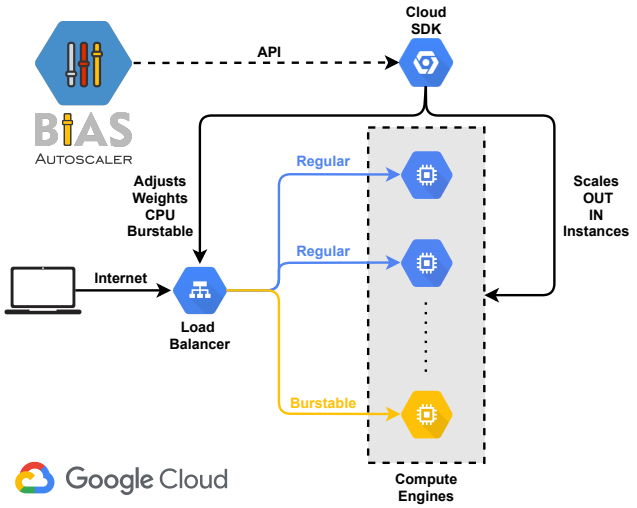

Figure 1: Cluster architecture managed by BIAS Autoscaler on GCP.

2.1 The architecture of BIAS autoscaler

BIAS Autoscaler is a ready-to-use autoscaler with little to no configuration required for cloud systems. Even though it was primarily evaluated and developed on GCP, it can be extended to other public cloud providers such as AWS and Azure. To the best of our knowledge, BIAS Autoscaler is the first open-source autoscaler fully tested and validated on GCP that leverages burstable instances for scaling Google Compute Engine instances. BIAS Autoscaler is also the first autoscaler to use the Google Load Balancer to dynamically change the traffic distribution among the instances. It uses the existing GCP services to manage and monitor the cluster, and it was developed using the Java programming language in combination with the Micronaut Framework. A full step-by-step guide and documentation is provided on GitPages[2]. We used the Google Cloud Java API and SDK for scaling and controlling the cluster, and the Google Cloud Stackdriver Monitoring Client for monitoring the necessary metrics. BIAS Autoscaler can be deployed either on a Google Compute Engine instance or run as a container on Google Kubernetes Engine. It is a reactive autoscaler that uses the Google Load Balancer to adjust the CPU utilization of the

[2] https://bias-cloud.github.io/BIAS-Autoscaler

burstable instances based on traffic distribution. Figure 1 shows how BIAS Autoscaler is used to scale out/in the resources on GCP. Its architecture is divided into three distinct modules: monitor, scaling and controller. The internal architecture of BIAS Autoscaler is shown in figure 2.

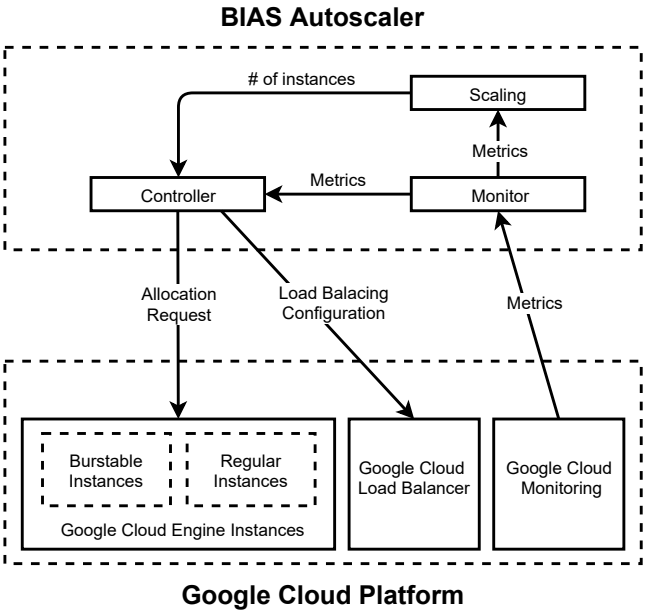

Figure 2: BIAS Autoscaler architecture.

2.1.1 CPU utilization of burstable instances. BIAS Autoscaler works by maxing out the CPU utilization of the burstable instance only when the cluster is scaling out new resources. It sets the CPU threshold of the burstable instances to its default value (T) as soon as the new resources are added and are ready to be used in the cluster. This CPU threshold is refereed as weights (w_b) on BIAS Autoscaler. By doing this, the full capacity of the burstable instances is used when the cluster requires additional computational power to process the current demand. On this strategy, BIAS Autoscaler will boost the burstable instances only when necessary.

Monitoring: The monitoring component is in charge of acquiring metrics from the load balancer and the instances. Since we validated BIAS Autoscaler on GCP, the Google Load Balancer and the Google Compute Engines were used. It fetches metrics from these cloud services through the Google Cloud Monitoring service.

Scaling: This component is where the scaling algorithm is implemented. It reads the metrics provided by the monitor component, and calculates the number of burstable and regular instances of the current demand. This information is then fed to the controller module so it can perform the scaling of the cluster. Currently, BIAS Autoscaler supports only the Square-Root Staffing Rule (SR Rule) scaling policy, but any policy can be applied.

Controller: This module is the core of BIAS Autoscaler. The number of calculated burstable (k_{b_c}) and regular (k_{r_c}) instances is provided to this component, and it outputs the necessary changes to the cluster. Once again, since we validated BIAS Autoscaler on GCP, it uses the Google Cloud Java API to control the load balancer traffic distribution among the instance groups, and scales out/in the Google Compute Engine instances. The controller module is also responsible for updating the weights of the CPU threshold (w_b) of the burstable instances. Whenever it scales out the regular or burstable instances, it sets w_b to 100% to burst the burstable instances to their maximum capacity while the new resources are being provisioned in the cluster. This helps to reduce the CPU load of the regular instances while the new resources are added to the cluster. As soon as the calculated number of instances (k_{b_c}, k_{r_c}) are identical to the current number of instances (k_b, k_r), BIAS Autoscaler sets w_b to its original threshold value, T.

Although BIAS Autoscaler was primarily designed and validated on GCP, it can be extended to AWS and Azure as well. In order to control EC2 instances on AWS, though, a customized load balancer is required since the AWS Elastic Load Balancer does not support dynamic adjustments in the traffic distribution among different instance groups. The same approach should be applied when using BIAS Autoscaler to control Azure Virtual Machines on Microsoft Azure. A generic interface is provided so users can implement a class to communicate with their customized load balanced using RESTful/gRCP APIs. Additionally, BIAS Autoscaler can be extended to manage services based on containers on GCP and other cloud providers as well. For GCP, a generic interface is provided to implement procedures to control the Google Kubernetes Engine using the Google Cloud SDK.

2.2 Scaling policy

Both predictive and reactive scaling algorithms can be applied on BIAS Autoscaler. However, we chose a reactive approach to scale our resources. Our reactive strategy assumes that the future demand resembles the current state. We use the well-accepted Square-Root Staffing Rule (SR Rule) as our scaling strategy for BIAS Autoscaler. Many works [4, 5, 13] have been developed around autoscaling cloud resources based on the SR Rule in recent years. Since the Google Cloud Load Balancer allows the distribution of the traffic based on the CPU utilization of instance groups, we leverage this feature to control the utilization level of the burstable instances. Based on the previous development done on [10], we consider our system as an M/M/k queueing system.

Theorem 2.1. *(Square-Root Staffing Rule [6]) Given an M/M/k queueing system with arrival rate λ and service rate μ, and $R = \frac{\lambda}{\mu}$, where R is large, let k^*_α denote the least number of servers needed to ensure that the probability of queueing $P_Q^{M/M/k} < \alpha$. Then $k^*_\alpha \approx R + c\sqrt{R}$ where c is the solution for the equation $\frac{c\Phi(c)}{\phi(c)} = \frac{1-\alpha}{\alpha}$ where $\Phi(.)$ denotes the c.d.f. of the standard Normal distribution and $\phi(.)$ denotes its p.d.f.*

The parameter c is related to the probability of queueing, α, which determines the mean response time of our service, $E[T]$. Equation 1 shows how we can calculate $E[T]$ using P_Q where $\rho = \frac{\lambda}{k\mu}$ is the system utilization [6]. We assumed the probability of queueing (P_Q) of our service as 10% for all benchmark tests we performed, but this property can be easily changed on BIAS Autoscaler configuration file.

$$E[T] = \frac{1}{\lambda} \cdot P_Q \cdot \frac{\rho}{1-\rho} + \frac{1}{\mu} \tag{1}$$

Therefore, the SR Rule used to determine the number of servers k required to handle an arrival rate λ is $k_c = R + c\sqrt{R}$. Note that the value of R is known, and it varies depending on the cluster configuration and workload used. We use the approach proposed on [4] to determine the number of and regular ($k_{r\_c} = R$) and burstable ($k_{b\_c} = c\sqrt{R}$) instances.

3 EVALUATION

We conducted two different experiments to evaluate the performance of BIAS Autoscaler for scaling virtual machines using Compute Engines on GCP. In order to evaluate the performance of our framework, we created the Load Microservice and made it open-sourced on GitHub[3]. This microservice simulates a web-server application with a RESTFul API with adjustable CPU load and processing time. The full documentation of the Load Microservice is available on GitPages[4]. We use the Locust[5] benchmark tool for our performance tests. Locust is an open-sourced, scriptable and scalable performance testing tool that allows customized use test cases written in Python. We analyzed the performance of our autoscaler during a transient queueing in traffic and a flash crowd scenario. For the former case, we evaluated BIAS Autoscaler in three distinct configurations, and then compared our performance to the rule-based GCP autoscaler. Finally, we tested BIAS Autoscaler for handling flash crowds and analyzed the QoS metrics and the SLOs violations for each test. All the experiments are openly accessible on GitHub[6].

[3] https://github.com/BIAS-Cloud/Load-Microservice
[4] https://bias-cloud.github.io/Load-Microservice
[5] https://locust.io
[6] https://github.com/BIAS-Cloud/Experiments

3.1 Transient Queueing

For this experiment, we simulated a fixed increasing rate in traffic for a long period, and we analyzed the QoS metrics and SLOs violations when running BIAS Autoscaler with both burstable and regular instances compared to regular instances only and burstable instances only. In addition to evaluating QoS metrics and SLOs violations, we compare the computational power and efficiency of the burstable instances with their equivalent regular ones.

Experimental Setup: We created a cluster on GCP with burstable and regular on-demand instances. We used *N1 shared-core g1-small* instances as our burstable instances, and *N1 standard 1* as our regular ones. Both these instance types have 1 identical vCPU (Intel(R) Xeon(R) CPU @ 2.30GHz). The main difference between them is that the burstable instances are CPU limited to 50% utilization, but they can boost themselves up to 100% of 1 vCPU for small periods. For memory, however, our instances differ a bit. Whereas our regular instances have 3.75 GB RAM, our burstable ones have only 1.7 GB RAM. The CPU utilization target of the burstable instances was set to 40%. We first perform three benchmark tests: one with BIAS Autoscaler scaling regular instances along with burstable ones, one scaling regular instances only where $k_r = R + c\sqrt{R}$, and another one scaling burstable instances only where $k_b = R + c\sqrt{R}$. We then compare the results of these three tests with another performance test using the GCP autoscaler with regular instances only set to scale out each time the CPU utilization reaches 50%.

SLOs: The SLO for the average response time was set to 150 ms with 95% of the requests below 300 ms, and no error is allowed.

Service rate μ: We run a benchmark test to determine the service rate experimentally. This test consisted of running a regular instance for 60 minutes under a fixed arrival rate. The service rate for our evaluation tests was set to $\mu = 17$ requests/s for each regular instance. The probability of queueing P_Q for all tests we performed is 10%.

Load generation: For simulating the user traces, we created a test scenario on Locust where the arrival rate λ increases linearly from 10 to 75 request/s in a window of 108 minutes.

Results: We reduced the cost by 25% when replacing some conventional instances with burstable ones. To achieve this, we compared the cost of running BIAS Autoscaler with regular and burstable instances (figure 3) against running it with regular instances only (figure 4). The scaling algorithm used in both tests was the SR Rule, and we considered $c\sqrt{R}$ as the number of burstable instances. This scaling strategy differs from a pure reactive scaling algorithm since more than one instance can be added at once under the SR Rule as can be

seen on figure 5(c) and figure 6(c). Another surprising finding was the outstanding performance reached when running BIAS Autoscaler with burstable instance only. This is because this configuration resulted in savings of 56% compared with regular instances only with almost no impact in the SLOs. However, this high saving in cost may be questionable since all burstable instances ran above their CPU threshold of 50% during the entire test as can be seen in figure 6(d). Therefore, since the burstable instances on GCP are highly workload dependable, they may not sustain long periods running at full CPU capacity.

This 25% cost savings can be understood better when we analyze the resource utilization during the two test scenarios. While we maintained an average CPU usage of our resources of approximately 45% when running with regular instances only, this figure was roughly 64% when we combined burstable and regular ones (considering we rate the CPU of the burstable instances at 50%). As a result, we increased our resource efficiency by 42% when using a combination of these two instance types. This demonstrates how BIAS Autoscaler can be used to not only reduce the cost, but also to increase the overall resource efficiency. Although relying solely on burstable instances appears to be the best cost-effective option at first glance, the black-box managing system of GCP states that there is no guarantee it can sustain long periods running on maximum CPU capacity. Thus, we do not advocate that cluster administrators should replace all their conventional instances with burstable ones instead. However, this demonstrates that burstable instances can indeed be used for replacing some regular instances as long as their CPU load is correctly managed by the autoscaler to avoid long runs at their maximum CPU capacity, as demonstrated in figure 3(d).

Even though we observed a slightly better average response time (7% only) when using regular instances only, the SLOs were not impacted when using burstable instances. The 95th percentile performance was also approximately equivalent for the two tests. The reason for that is because the maximum 95th percentile response time reached when running burstable and regular instances combined was less than 25% higher than running BIAS Autoscaler with regular instances only. Despite this small difference, both tests met the required SLOs for the 95th percentile.

Table 1 compiles the results of all four tests performed. Note that when we compare side-to-side BIAS Autoscaler with the rule-based GCP autoscaler, we can see that BIAS Autoscaler reduces the cost by approximately 18% while maintaining roughly the same SLOs.

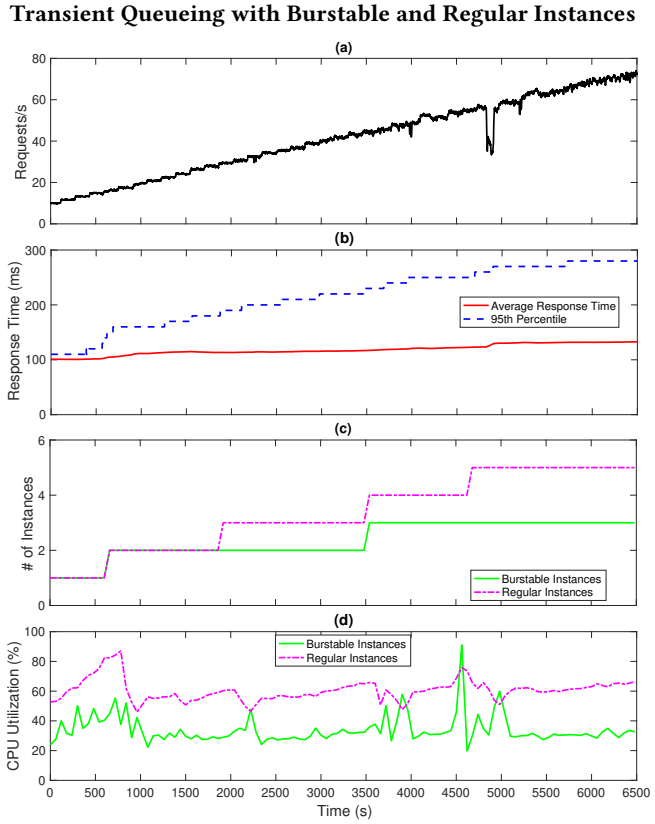

Figure 3: Results for the transient queueing experiment with burstable and regular instances.

3.2 Flash Crowd

For this experiment, we simulated a flash crowd for a short period, and we analyzed the QoS metrics and SLOs violations when running BIAS Autoscaler with both burstable and regular instances compared to regular instances only using the same configuration as for the transient queueing experiment. We performed a benchmark test where BIAS Autoscaler runs with regular and burstable instances combined. The key difference of this experiment is the workload used. The SLO for the average response time was set to 300 ms with 95% of the requests below 1000 ms, and no error is allowed.

Load generation: For simulating a flash crowd, we created a test scenario on Locust where the arrival rate λ increases from 10 request/s to three different picks with a maximum of 85 requests/s in a window of 33 minutes in total. We run this load against two distinct configurations of BIAS Autoscaler. The figure 7(a) shows the load used for the flash crowd experiment.

Results: The outcome of this experiment was similar to the transient queueing benchmark test, with approximately

Table 1: Results of performance tests for transient queueing.

Test Scenario	Average Response Time (ms)	Maximum 95th Percentile (ms)	Cost (10^{-3} USD)
Regular instances only	110	210	493
Rule-based GCP autoscaler	108	220	450
Burstable and regular instances	118	280	371
Burstable instances only	120	220	218

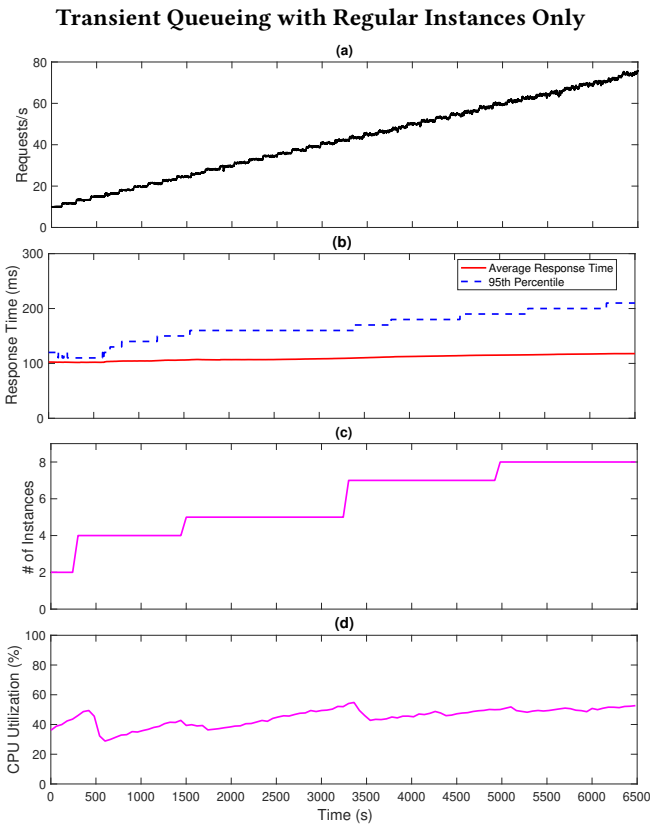

Figure 4: Results for the transient queueing experiment with regular instances only.

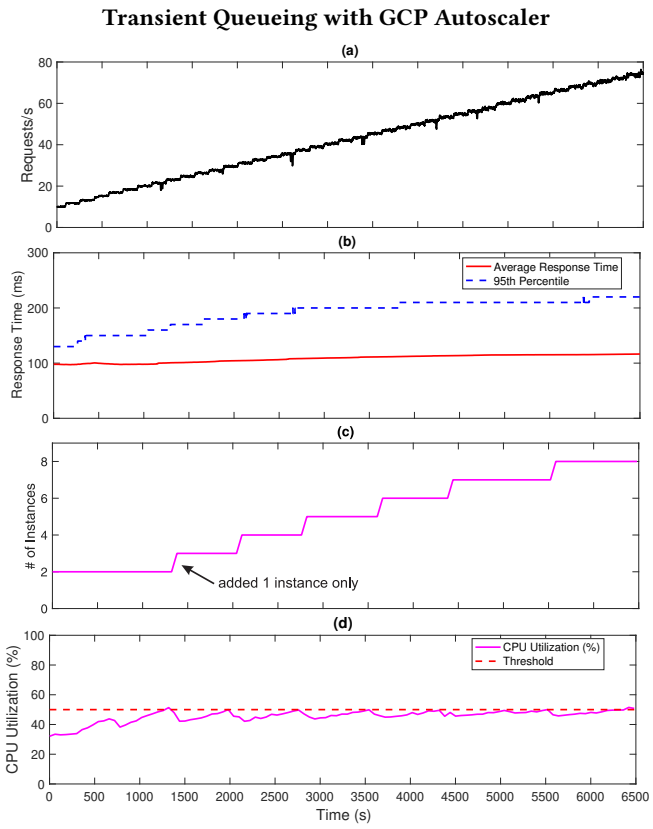

Figure 5: Results for the transient queueing experiment with the rule-based GCP autoscaler set to 50% using regular instances only.

25% reduction in cost when replacing some conventional instances with burstable ones. Even though both performance tests where BIAS Autoscaler ran with burstable and regular instances and the one with regular instances only met all requited SLOs, the average response time achieved by the later configuration outperformed the former one by almost 2 times. Since we did not overprovision the cluster, the average and the 95th percentile of the response time for the flash crowd experiment running with both burstable and regular instances was almost double the figure seen on the transient queueing experiment. The average response time for the flash crowd experiment when BIAS Autoscaler ran with burstable and regular instances was 232 ms whereas for regular instances only was 141 ms.

Although the response time achieved when combining burstable and regular instances was almost two times higher than when using regular instances only, the cost savings should be considered when using this approach for scaling cloud resources. Overall, using the SR Rule algorithm for scaling regular and burstable instances was adequate for scenarios where the traffic increases at a steady and constant rate. For this application, BIAS Autoscaler was able to maintain roughly the same SLOs as for regular instances only at

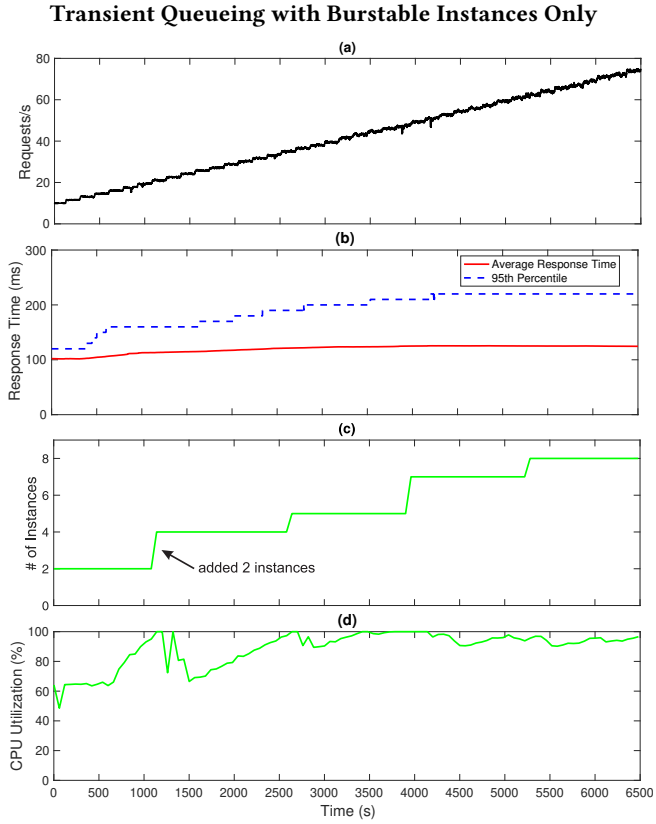

Figure 6: Results for the transient queueing experiment with burstable instances only.

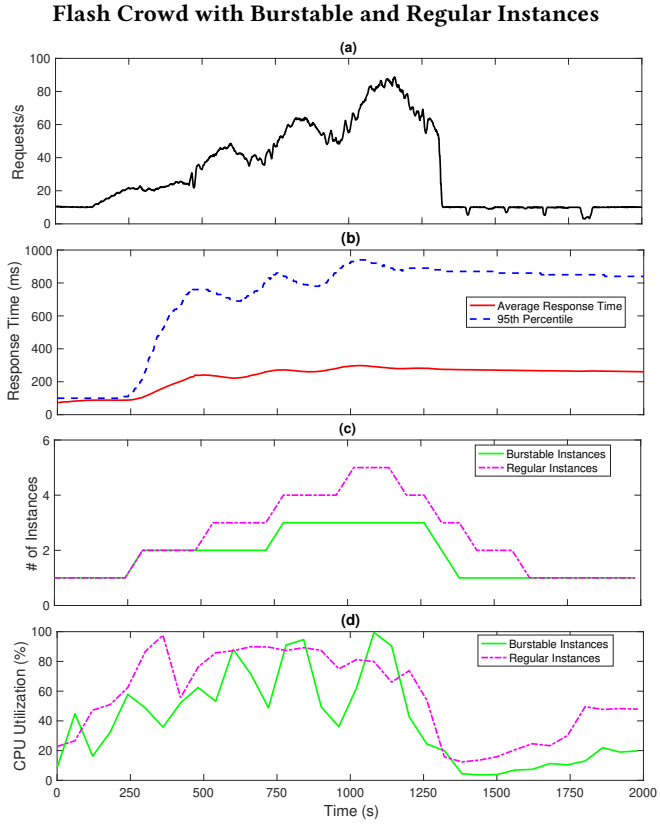

Figure 7: Results for the flash crowd experiment with burstable and regular instances.

the same time the cost was reduced by 25%. However, the same outcome was not reached when using this strategy for handling flash crowds. For these sudden variations in traffic, a fine-granular tuning of the scaling frequency and the number of burstable instances used should be performed to avoid impacting the SLOs.

4 RELATED WORK

Many works [2, 7, 11] have been done on the theoretical analysis of burstable instances, and some frameworks were proposed to AWS. The framework CEDULE developed on [2] and [11] investigates the usage of burstable instances on AWS, and proposes an adaptive scheduling framework to optimize performance and reduce cost on cloud providers. Even though the authors of CEDULE claimed it could be used in any cloud provider, they only tested it on AWS. Unlike our framework, CEDULE focuses only on token-based systems to manage burstable instances (present on AWS and Azure) while BIAS Autoscaler addresses the practical applications of these instances on non-token-like systems such as the one on GCP. Also, CEDULE is not open-source, and no information about its internal architecture is provided by its authors.

Similar to CEDULE, BurScale [4] leverages burstable instances to reduce cost and handle flash crowds on AWS. Unlike our solution, BurScale is validated and applied only on AWS, and it uses a customized load balancer. Even though BurScale is an open-source solution, there is no information on how it could be used on a non-token-like system such as GCP. When using BIAS Autoscaler on GCP, however, the load balancer used is the Google Cloud Load Balancer for controlling the traffic of the burstable instances.

The authors on [7] advocate the usage of burstable instances for workloads that do not require large amounts of computational resources to run, and that occasionally need to run with additional resources for small periods. They also propose the first analytical model for burstable instances that takes into account the QoS metrics and CPU credits of burstable instances to derive a mathematical model that maximizes cost and resource efficiency for customers and cloud providers for IaaS (Infrastructure as a Service) clouds. Although their framework can be used to model burstable instances on AWS and Azure, there is no information on how these instances can be used on GCP.

Some solutions [9, 12] were developed on the efficiency of reactive rule-based autoscaler on Google Compute Engine, whereas others [1, 8] proposed scaling policies for Google Kubernetes Engine. However, no studies were found on the practical usage of burstable instances on GCP. The analysis on [14] presents a complete overview and comparison between burstable instances from AWS and GCP, and explains in detail the token mechanisms used by AWS to manage these instances.

MRburst, which was developed on [3], is also a performance scheduler to control, among other things, the CPU utilization of burstable instances in the network level on AWS to maximize cost efficiency. However, this approach differs from ours since we control our resource utilization on the application level performing load balancing configuration changes.

5 CONCLUSION AND FUTURE WORK

Burstable instances can be the key to improving resource efficiency and reducing costs on the public cloud. We presented BIAS Autoscaler, an autoscaler that leverages burstable instances on the public cloud as the only of its kind fully validated and integrated on the Google Cloud Platform. We applied a known scaling model to validate our concept, and achieved promising results on both savings and resource efficiency. By replacing some of the conventional instances with burstable instances, BIAS Autoscaler was able to reduce the cost by 25% while maintaining the same service SLOs compared with traditional approaches using regular instances only.

We evaluated BIAS Autoscaler under a transient queueing and a flash crowd experiment, and showed its efficiency on Google Cloud Platform on a microservice workload. The outcome of these performance experiments showed great potential to increase resource efficiency and reduce the cost. These results demonstrated that BIAS Autoscaler can increase resource efficiency by 42% without interfering with the quality of the service when using burstable instances.

In the future, we want to make BIAS Autoscaler fully compatible with AWS using a customized load balancer. We also intend to develop a performance modeling for using burstable instances on GCP. Many models [2, 7, 11, 14] have been developed for the token-like system on AWS, but no study was found on analytical modelling of burstable Google Compute Engines on GCP.

REFERENCES

[1] A. Abdel Khaleq and I. Ra. 2019. Agnostic Approach for Microservices Autoscaling in Cloud Applications. In *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*. 1411–1415. https://doi.org/10.1109/CSCI49370.2019.00264

[2] A. Ali, R. Pinciroli, F. Yan, and E. Smirni. 2018. CEDULE: A Scheduling Framework for Burstable Performance in Cloud Computing. In *2018 IEEE International Conference on Autonomic Computing (ICAC)*. 141–150. https://doi.org/10.1109/ICAC.2018.00024

[3] Ahsan Ali, Riccardo Pinciroli, Feng Yan, and Evgenia Smirni. 2019. It's Not a Sprint, It's a Marathon: Stretching Multi-Resource Burstable Performance in Public Clouds (Industry Track). In *Proceedings of the 20th International Middleware Conference Industrial Track* (Davis, CA, USA) *(Middleware '19)*. Association for Computing Machinery, New York, NY, USA, 36–42. https://doi.org/10.1145/3366626.3368130

[4] Ataollah Fatahi Baarzi, Timothy Zhu, and Bhuvan Urgaonkar. 2019. BurScale: Using Burstable Instances for Cost-Effective Autoscaling in the Public Cloud. In *Proceedings of the ACM Symposium on Cloud Computing* (Santa Cruz, CA, USA) *(SoCC '19)*. Association for Computing Machinery, New York, NY, USA, 126–138. https://doi.org/10.1145/3357223.3362706

[5] A. Gandhi, M. Harchol-Balter, R. Raghunathan, and M. A. Kozuch. 2011. Distributed, Robust Auto-Scaling Policies for Power Management in Compute Intensive Server Farms. In *2011 Sixth Open Cirrus Summit*. 1–5. https://doi.org/10.1109/OCS.2011.6

[6] Mor Harchol-Balter. 2013. *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press.

[7] Y. Jiang, M. Shahrad, D. Wentzlaff, D. H. K. Tsang, and C. Joe-Wong. 2020. Burstable Instances for Clouds: Performance Modeling, Equilibrium Analysis, and Revenue Maximization. *IEEE/ACM Transactions on Networking* 28, 6 (2020), 2489–2502. https://doi.org/10.1109/TNET.2020.3015523

[8] A. A. Khaleq and I. Ra. 2021. Intelligent Autoscaling of Microservices in the Cloud for Real-Time Applications. *IEEE Access* 9 (2021), 35464–35476. https://doi.org/10.1109/ACCESS.2021.3061890

[9] M. N. A. H. Khan, Y. Liu, H. Alipour, and S. Singh. 2015. Modeling the Autoscaling Operations in Cloud with Time Series Data. In *2015 IEEE 34th Symposium on Reliable Distributed Systems Workshop (SRDSW)*. 7–12. https://doi.org/10.1109/SRDSW.2015.20

[10] H. . Lin and C. S. Raghavendra. 1992. An analysis of the join the shortest queue (JSQ) policy. In *[1992] Proceedings of the 12th International Conference on Distributed Computing Systems*. 362–366. https://doi.org/10.1109/ICDCS.1992.235020

[11] R. Pinciroli, A. Ali, F. Yan, and E. Smirni. 2021. CEDULE+: Resource Management for Burstable Cloud Instances Using Predictive Analytics. *IEEE Transactions on Network and Service Management* 18, 1 (2021), 945–957. https://doi.org/10.1109/TNSM.2020.3039942

[12] V. Podolskiy, A. Jindal, and M. Gerndt. 2018. IaaS Reactive Autoscaling Performance Challenges. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. 954–957. https://doi.org/10.1109/CLOUD.2018.00144

[13] A. Suresh, G. Somashekar, A. Varadarajan, V. R. Kakarla, H. Upadhyay, and A. Gandhi. 2020. ENSURE: Efficient Scheduling and Autonomous Resource Management in Serverless Environments. In *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*. 1–10. https://doi.org/10.1109/ACSOS49614.2020.00020

[14] Cheng Wang, Bhuvan Urgaonkar, Neda Nasiriani, and George Kesidis. 2017. Using Burstable Instances in the Public Cloud: Why, When and How? *Proc. ACM Meas. Anal. Comput. Syst.* 1, 1, Article 11 (June 2017), 28 pages. https://doi.org/10.1145/3084448