Harmonizing Diverse Compute Resources for Efficiency

Dilina Dehigama
University of Edinburgh
Edinburgh, UK
dilina.dehigama@ed.ac.uk

Shyam Jesalpura
University of Edinburgh
Edinburgh, UK
s.jesalpura@gmail.com

Marios Kogias
Imperial College London
London, UK
m.kogias@imperial.ac.uk

Boris Grot
University of Edinburgh
Edinburgh, UK
boris.grot@ed.ac.uk

Abstract

Online services are characterized by significant load fluctuations at fine-grained intervals even when coarse-grained load measurements indicate a relatively stable load. Running such services on virtual machines (VMs) rented from a cloud provider like AWS, which is a typical way to deploy online applications today, is inefficient due to the need to overprovision VM capacity to meet the SLO under variable load. In contrast, serverless computing is highly elastic but is prohibitively expensive for serving a large volume of requests. We thus argue for combining the different types of compute (i.e., VM and serverless instances) to achieve both cost-efficiency and elasticity. Our results show that hybrid compute is more cost effective than even an optimal VM-only allocation that provisions just enough resource to meet the SLO using perfect knowledge of future load.

1 Introduction

Modern online services are increasingly complex and face challenges in maintaining performance, particularly under fluctuating load conditions. These services, typically deployed in cloud VMs using container orchestration platforms such as Kubernetes, require precise resource allocation to meet stringent Service Level Objectives (SLOs) related to latency and responsiveness. However, traditional resource provisioning models – often based on coarse-grained load measurements and slow-to-react resource allocations – fail to account for the fine-grained variability in workload demands, resulting in resource wastage (due to overprovisioning) or SLO violations (due to underprovisioning).

While VMs do not offer the fine-grained elasticity desired for online services, serverless functions do. Alas, today's cloud cost structures make serverless prohibitively expensive for continuously serving a large volume of requests.

To optimize for both efficiency and cost, we propose a hybrid approach. A stable "base" load can be handled by VMs for predictable performance and cost. Bursts of "excess" load can be addressed by serverless functions for flexibility and pay-per-use pricing. This combination offers the best of both worlds: efficiency and cost-effectiveness.

Our evaluation demonstrates that combining VMs with serverless functions to handle finer-grained fluctuations can reduce total cost by 7.5% compared to a baseline that uses an ideally provisioned VM capacity compute to meet SLO targets. In a more realistic deployment, where VMs are provisioned with 30% additional CPU capacity over an optimal allocation in order to handle likely load fluctuations, the hybrid approach offers cost savings of up to 28.5%. These results highlight the potential of hybrid compute models to enhance resource efficiency and reduce costs while maintaining SLO compliance.

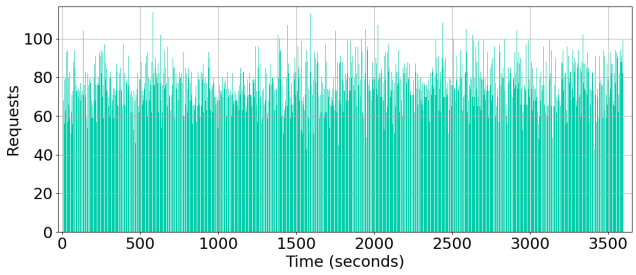
The 2nd Workshop on Hot Topics in System Infrastructure (HotInfra'24), co-located with SOSP'24, November 3, 2024, Austin, TX, USA

2 Motivation

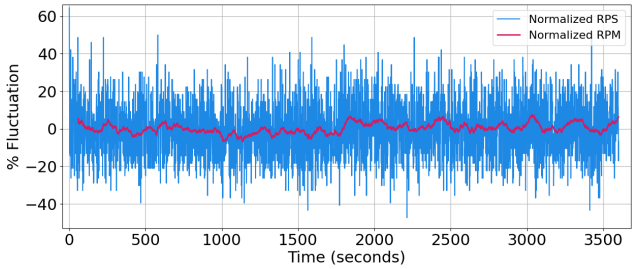
2.1 Modern Online Services & Deployment

Modern online services have grown increasingly complex due to their diverse functionality [1, 2, 9, 10, 12]. These services are subject to highly variable demand patterns due to interactions with users [10]. Despite this variability, meeting Service Level Objectives (SLOs)—particularly in terms of latency and throughput is critical, as failure to do so degrades the end-user experience [11]. Achieving SLO guarantees requires judicious resource allocation that accommodates both fine-grained and coarse-grained load fluctuations while also being cost-effective.

Today's online services are typically deployed as containers on VMs using container orchestration systems like Kubernetes [6]. These systems enable resource provisioning per-container and facilitate fine-grained resource management, such as allocating CPU in milicores (one-thousandth of a CPU core) to containers. However, it's important to note that in current VM provisions, charges are based on the full allocated capacity, regardless of whether it is actively utilized or idle.



(a) Load measured at 1 second granularity.



(b) Request arrival rate fluctuations from the median request rate. Fluctuation at 1-minute resolution (red) and 1-second resolution (blue). Values are normalized to their respective medians.

Figure 1. Twitter load trace over a one-hour period.

2.2 Fluctuating Load Patterns

One of the primary challenges in resource provisioning for online services arises from fluctuating load patterns. Even at a period of relatively stable load (e.g., several minutes on a typical day), there exists considerable fine-grained load fluctuation.

To illustrate this, we analyze a representative one-hour long portion of Twitter's load trace. Figure 1(a) (top-plot) shows the load while Figure 1(b) (bottom plot) illustrates the variability in load, using both one-minute (requests per minute - RPM) and one-second (requests per second - RPS) intervals. The values in the bottom plot are normalized to their respective medians for the shown period.

At the one-minute granularity, the load appears relatively steady, varying within 6% of the median. However, when the same data is plotted at one-second resolution, sharp deviations of up to 50% from the median load become apparent. This insight reveals that allocating resources based on coarse-grained measurements can overlook critical load fluctuations and potentially lead to SLO violations due to inadequate resource provision. A similar phenomenon has recently been highlighted in the production cluster of YouTube [13], indicating that the problem is not specific to one trace or one type of service. Thus, resource allocation for online services must take into account fine-grained load variation to accommodate the inherent variability in demand and ensure adherence to SLOs.

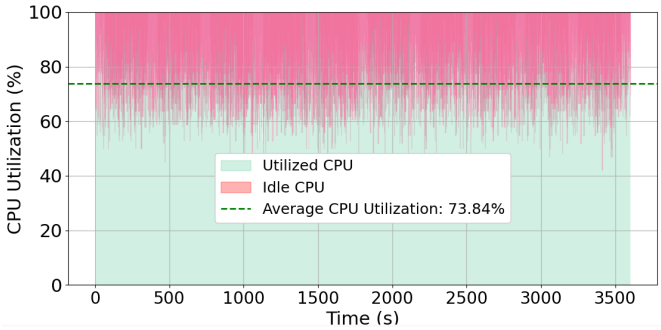


Figure 2. CPU utilization over one hour. Green (bottom-portion) denotes the utilized CPU fraction while the pink (top-portion) denotes idle CPU time.

2.3 Resource Allocation and Utilization

Given the fluctuating nature of online services, allocating resources to meet SLO targets without squandering money on idle compute is a challenge. To demonstrate this, we deployed a containerized toy application on a rented VM from AWS. The application receives requests, does some busy work and returns the response to the user. We profiled the application by replaying the one-hour long load trace specified in Section 2.2 to determine the exact CPU allocation required to meet the SLO target at the 99th percentile latency. We set the SLO target at five times the latency of a single request on an otherwise-idle VM. The CPU allocation required to meet this SLO target is referred to as *Optimal CPU provisioning*.

A CPU allocation lower than this optimum would result in SLO violations (i.e., more than 1% of requests would violate the latency target). A higher CPU utilization would result in more CPU idle time and higher bills without helping SLO.

Figure 2 shows the CPU utilization over the period of trace. Despite an optimal CPU allocation that leverages exact knowledge of the load in the trace, on average only 74% of the CPU time is effectively utilized, leaving 26% of CPU resources wasted (highlighted in pink). This result highlights that even under optimal resource provisioning, where the future load pattern is known precisely and resources are sized accordingly, significant resource under-utilization occurs due to the fine-grained variability in the request stream.

2.4 Deficiencies of Existing Compute Types

In an ideal computing model, resources would be elastic enough to perfectly accommodate fine-grained load fluctuations, providing just enough compute to meet the SLO without keeping CPUs idle in the (brief) periods of lower load. Today's VMs are far from this ideal; regular VMs can take a minute or more to boot [3], while even fast-booting VMs such as microVMs [8] can take a second or more to fully come online and start serving requests. Given such bring-up latencies, VMs cannot effectively cater to fine-grained load variations.

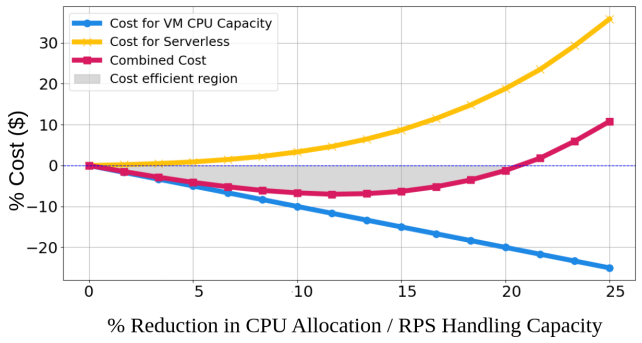


Figure 3. Cost comparison of serverless and VM-based compute classes for varying VM CPU allocation.

In principle, serverless computing can meet the elasticity objective, particularly when instances are kept warm. Serverless encourages fine-grained resource provisioning with short-running tasks, small memory footprints and pay for actual usage (unlike the pay-for-allocation with VMs). Alas, for sustained usage, serverless compute is several times more expensive than VMs, making it unattractive from a cost perspective for sustained serving. Thus, the challenge is to provide a compute substrate that can meet the fine-grained elasticity demands of online services while capitalizing on this elasticity from a cost-perspective.

3 Hybrid Compute for Peak Efficiency

In order to capitalize on the fine-grained load fluctuations from both compute efficiency and cost perspectives, we make the following critical observation. In a given coarser-grained time interval (e.g., several minutes), the load on a typical online service is comprised of two components: a base component and an additional *excess* component. The base component is the minimal load consistently observed at fine grain during the coarser interval. For instance, in Figure 1(a), that would be just over 40 RPS. The rest of the load in the coarser interval is strictly above that and varies at a fine granularity.

Based on this observation, we argue that online services are best accommodated with a hybrid deployment model that combines two fundamentally different types of compute: the base component of the load should be served on VM-based instances that combine predictable performance with low- and predictable pricing. The excess component is best served on truly elastic instance types, such as serverless, that are fast to scale and offer pay-per-use pricing.

4 Preliminary Evaluation

We present an opportunity study to demonstrate the potential of serving online loads with different types of compute for elasticity and cost-efficiency. Toward that goal, we compare a VM-only deployment, in which resources cannot be scaled to accommodate fine-grained load variations (e.g., at 1-second granularity) to a hybrid deployment that uses VMs for a portion of the load and serverless for the excess. We also study a serverless-only deployment using pre-warmed instances offering fully elastic pay-for-use compute.

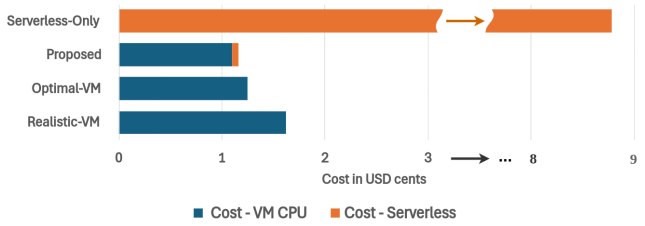


Figure 4. Total cost comparison across deployment models

Configurations. We evaluate our proposed hybrid approach against two VM-based deployments. The first VM-only scenario assumes optimal CPU provisioning (referred to as the *Optimal-VM*), based on perfect knowledge of future load patterns — a condition that is often impractical in real-world environments. The second scenario is a realistic configuration, where VM CPU capacity is overprovisioned by 30% compared to an optimal CPU provisioning, to account for unpredictable load variability. This strategy reflects common industry practices and ensures compliance with SLOs during traffic surges [5, 14].

Methodology. For our evaluation, we utilized Amazon EC2 *t3.medium* instances—one of the most common instance types. In AWS, the cost for these VMs is set at 0.0208 USD per vCPU per hour [4]. For the serverless component, we employed AWS Lambda, which has a different pricing model based on the execution time and memory usage of the functions [7].

Results. We performed sensitivity analysis by gradually lowering the Optimal-VM CPU allocation from its optimal level. As we decrease the CPU allocation for the VMs, both the cost of VMs and the maximum RPS that the VMs can handle are reduced. While the VM load is reduced, the load on serverless functions increases, which proportionately raises the cost of the serverless component. We analyzed the combined cost of the VMs and serverless functions to determine the point at which cost savings are maximized. The results of the sensitivity analysis are shown in Figure 3. In this analysis, all values are normalized to their respective baseline values and are presented as percentages. For the trace under study, we find that the optimal cost point is achieved when CPU allocation is reduced by 11.7% from the Optimal-VM configuration, resulting in a 7.5% overall cost reduction compared to Optimal-VM cost.

Figure 4 presents a comparison of the total costs between the proposed hybrid design, the Optimal-VM configuration, the Realistic-VM configuration and a serverless-only configuration. We find that the hybrid design can achieve cost

savings of up to 28.5% compared to the Realistic-VM configuration, where resources are allocated to maintain a safety margin in the face of a variable load. A serverless-only configuration is even more expensive, costing 8x more than Hybrid, using today's AWS pricing.

5 Conclusion

Online services exhibit significant load fluctuations at fine-grained intervals, even when coarse-grained measurements suggest stability. Traditional VM-based deployments necessitate overprovisioning to handle peak demands resulting in resource waste, while serverless computing offers elasticity at a higher cost. We propose a hybrid approach that combines VMs and serverless functions for solution that is elastic, performant and cost-effective. The results demonstrate that this hybrid model can achieve significant cost savings, reducing expenses by up to 7.5% compared to an optimally provisioned VM-only setup, and by 28.5% when compared to a more realistic overprovisioned configuration.

References

[1] [n. d.]. Q&A with Jim Brikman: Splitting Up a Codebase into Microservices and Artifacts. https://engineering.linkedin.com/blog/2016/02/q-a-with-jim-brikman--splitting-up-a-codebase-into-microservices

[2] 2016. The Opportunities Microservices Provide at Uber Engineering. https://www.uber.com/en-GB/blog/building-tincup-microservice-implementation [Online; accessed 8. Jan. 2024].

[3] 2024. AMI types - Amazon Elastic Compute Cloud. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ComponentsAMIs.html#storage-for-the-root-device [Online; accessed 9. Jan. 2024].

[4] 2024. AWS VM Instances cost. https://aws.amazon.com/ec2/instance-types/t3/ [Online; accessed 27. Jan. 2024].

[5] 2024. Kubernetes Clusters Have Massive Overprovisioning Of Compute And Memory. https://www.nextplatform.com/2024/03/04/kubernetes-clusters-have-massive-overprovisioning-of-compute-and-memory/ [Online; accessed 7. Sep. 2024].

[6] 2024. Production-Grade Container Orchestration. https://kubernetes.io [Online; accessed 9. Jan. 2024].

[7] 2024. Serverless Function, FaaS Serverless - AWS Lambda - AWS. https://aws.amazon.com/lambda [Online; accessed 11. Jan. 2024].

[8] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. 2020. Firecracker: Lightweight Virtualization for Serverless Applications. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 419–434. https://www.usenix.org/conference/nsdi20/presentation/agache

[9] TC Currie. 2017. Airbnb's 10 Takeaways from Moving to Microservices — thenewstack.io. https://thenewstack.io/airbnbs-10-takeaways-moving-microservices/. [Accessed 08-01-2024].

[10] Jenny Qiu Hylbert and Steve Cosenza. 12 August 2020. Rebuilding Twitter's public API. https://blog.twitter.com/engineering/en_us/topics/infrastructure/2020/rebuild_twitter_public_api_2020

[11] Jianshu Liu, Qingyang Wang, Shungeng Zhang, Liting Hu, and Dilma Da Silva. 2023. Sora: A Latency Sensitive Approach for Microservice Soft Resource Adaptation. In *Proceedings of the 24th International Middleware Conference* (Bologna, Italy) *(Middleware '23)*. Association for Computing Machinery, New York, NY, USA, 43–56. https://doi.org/10.1145/3590140.3592851

[12] Web Team. 2023. Microservices at netflix: Lessons for architectural design. https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/

[13] Bartek Wydrowski, Robert Kleinberg, Stephen M. Rumble, and Aaron Archer. 2024. Load is not what you should balance: Introducing Prequal. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, Santa Clara, CA, 1285–1299. https://www.usenix.org/conference/nsdi24/presentation/wydrowski

[14] Xiao Zhang, Eric Tune, Robert Hagmann, Rohit Jnagal, Vrigo Gokhale, and John Wilkes. 2013. CPI2: CPU performance isolation for shared compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems* (Prague, Czech Republic) *(EuroSys '13)*. Association for Computing Machinery, New York, NY, USA, 379–391. https://doi.org/10.1145/2465351.2465388