”Energy-Aware Server Consolidation for Serverless Computing”

Tech Con 2025 Abstract 616

HPE Go to Market; Hybrid Cloud;

Energy-Aware Server Consolidation for Serverless Computing

Abstract

Serverless Computing (SC) has increasingly been adopted by offering autonomous and flexible scaling of fine-grained functions that alleviate customers from manually managing resources. Its ability to scale according to incoming workload makes it suitable for a wide range of applications, especially for functions that benefit from highly parallel executions. While current SC platforms already offer autoscaling components, we identify an issue prevalent across the cloud computing domain that leads to increased energy consumption and reduced performance. The issue arises from underutilized servers and misconfigured function resource requirements. Approaches are needed to solve these inefficiencies as companies and customers try to minimize energy consumption and carbon emissions. Therefore, we present a proactive server consolidation approach and dynamic function configuration module that works in tandem with existing SC platforms. While server consolidation reduces the number of active physical machines, function configuration tuning helps users find the optimal resource configuration. The server consolidation approach uses request forecasts to proactively turn on or shutdown servers. Preliminary results show that it can lower energy consumption by up to 38% without reducing throughput.

Problem statement

Customers adopt Serverless Computing (SC) for flexible execution of fine-grained functions. However, two caveats arise in SC and other cloud paradigms that negatively impact energy consumption and performance. First, SC promises scalability when customers need massive parallel function execution and a scale-to-zero approach when no functions are required. This presents a waste of idle resources and frequently happens as workload patterns often follow a diurnal pattern [1], which makes times of low workload an ideal candidate to save energy by shutting down servers through server consolidation. The second caveat occurs when customers deploy their functions because they must specify how many resources (e.g., CPU cores) are allocated for each instance. This is not an easy task, and developers tend to overprovision them, thus resulting in low resource efficiency, which can reduce overall throughput because fewer instances can be started. The trend of overprovisioning can be seen in the SC trace dataset from Huawei [1], which contains information about request patterns and resource usage. It shows that the CPU usage's 95*th* percentile, of 80% of deployed functions, is at 20%, which presents an opportunity to increase efficiency by lowering the allocated resources. Delimitrou et al. [2] have conducted a similar analysis based on Virtual Machines, showing that the aggregate CPU utilization is below 20% while up to 80% of resources are reserved. Both conclusions indicate that resources are wasted due to misconfigured applications, thus, wasting space for other applications and increasing energy consumption. Therefore, we identify the problem of resource underutilization in Serverless Computing platforms and introduce the following contributions:

- We propose a proactive server consolidation approach that dynamically minimzes the number of active servers, reducing energy cost by up to 38% compared to the default platform.
- We also include a module that attempts to find the optimal function configuration, increasing resource efficiency. Preliminary results show that optimized configurations can lead to higher resource usage.

Our solution aims to work in tandem with existing customer SC deployments, such as KNative and offers improved energy consumption and resource efficiency, which does not affect overall throughput.

Our solution

Our solution combines a proactive approach for server consolidation and function configuration optimization, to determine a suitable function configuration that requires fewer resources. Which aids us in consolidating more servers and frees up resources. In the following, we focus on the consolidation approach and only outline the function configuration and function migration component that moves running functions from a host to be shutdown to an active one.

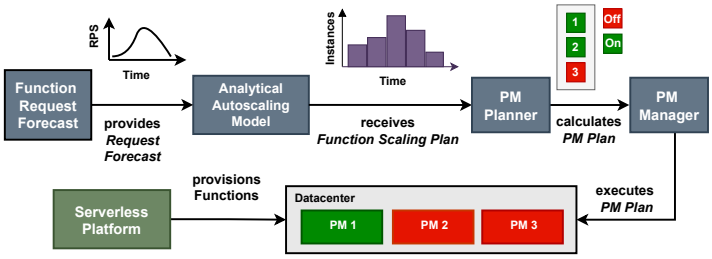

Figure 1: Proactive Server Consolidation System

Proactive Server Consolidation The goal of proactive server consolidation is to manage the life cycle of physical machines (PMs) by shutting them down when not needed and turning them back on in time to serve function requests. However, these actions incur overhead that can take multiple minutes, which means that the consolidation approach must act ahead of time to avoid any performance degradation. In addition, when shutting servers down, currently running functions must be gracefully migrated. Migration entails careful selection of servers as performance degradation can happen due to resource congestion (e.g., overloading a PM with CPU-bound functions). Our consolidation approach is based on historical monitoring data of deployed functions (e.g., throughput), the function requirements (e.g., CPU cores), and a forecast that estimates the number of incoming requests (RPS) for the next hour in 10-minute intervals. This allows us to estimate the required number of function instances. This builds the core idea of our consolidation strategy, and we now explain the components of the current system, which is depicted in Figure 1.

Current System The *Function Request Forecast* module estimates the RPS in time windows, which the *Analytical Autoscaling Model* module uses to estimate the number of required function instances. The *PM Planner* component takes the *Function Scaling Plan* as input and produces a *PM Plan* that tells the *PM Manager* which servers to turn on or shut down. The *PM Planner* implements the consolidation algorithm, which currently finds the maximum value in the forecast and estimates the servers needed for that. The algorithm is simple and determines by looking ahead whether we require more or less PMs in the near future and only assumes 75% cores of the server to be available. The reason for that is that the servers use hyperthreading, which can introduce performance degradation as soon as all physical cores are allocated to CPU-bound applications. The system integrates into the existing Serverless Computing platform, based on Kubernetes, and builds on common and available monitoring data, such as CPU usage, throughput, etc.

Function Request Forecast This module is implemented using Holt's Exponential Smoothing method [3]. This method uses past observations and forecasts by calculating an exponentially weighted sum over previous observations. We chose this method as an initial solution based on our data analysis of the Huawei dataset [1]. Our analysis concluded that many functions have a weekly and daily reoccurring pattern, which simplifies the forecast process and led us to choose Holt's Exponential Smoothing model and the past week of request observations as input. The model's simplicity stems from its approach of fitting data for each request, which causes minimal inference overhead and, more importantly, does not require us to train models for each function. This makes it scalable and very easy to use for new functions.

Function Migration & Configuration Tuning The *Function Migration* and *Function Configuration Tuning* are not yet ready and therefore not included in Figure 1, but are important for a complete solution. While Kubernetes and KNative already support function migration on their own, it can lead to performance degradation due to resource-unaware function re-scheduling. We want to aid this process by giving hints about which nodes it should prefer when re-scheduling function instances. The *Function Configuration Tuning* module is planned to be based on an analytical model that uses resource usage to estimate the throughput of new function configurations. This module will work together with the consolidation approach to find the optimal plan and function resource configuration.

Evidence the solution works

In the following, we first present the accuracy of our *Function Request Forecast* module and then results from our end-to-end consolidation experiments.

Function Request Forecast Figure 2 shows the median Root Mean Squared Percentage Error (RMSPE) for three forecast windows of 10, 30 and 60 minutes from a test dataset that we created from the Huawei dataset [1], which contains 20% of all functions that had a weekly pattern. Results indicate that the median error is at or below 10%, but outliers can go beyond 30%. The mean RMSPE has been heavily skewed for some functions and tends to be higher than the median. We think these errors come from the fluctuating request patterns. These fluctuations often occur in two forms: abrupt increase in requests or changing average requests per week. Due to the fact that we only have data of within one year, it's not possible to tell whether these fluctuations are common or outliers.

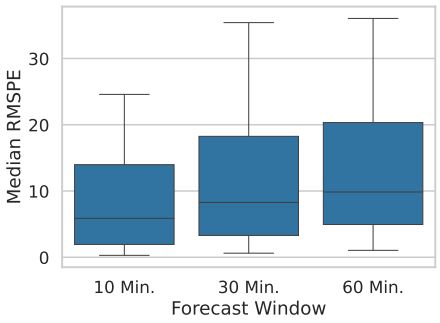

Figure 2: Median Root Mean Squared Percentage Error

End-to-End Consolidation These experiments focus on the ability of our current solution to manage servers and decrease energy consumption while guaranteeing throughput. A limitation of the current evaluation is that we do not shutdown servers and only ex- and include them from the Kubernetes cluster. In addition, our current implementation uses data from the original trace to create the input for the forecast. However, to mimic real-world

behavior, we add a 20-second delay when servers start (in real life, we measured a startup time of around 5 minutes). We compare our *proactive* approach to the *default* Knative setup by using an 8-hour long trace from the Huawei dataset [1] to reproduce a real-world setting, but shorten it to 8 minutes. An image inference function is used as an evaluation application and configured with 24 cores (*wasteful*) and 18 cores (*optimal*). We estimate both to process 20 requests per second, based on previous profiling experiments. However, we had to lower the throughput for the *optimal* one during the experiments to 15. We realized in the end-to-end experiments that we overestimated its throughput. and the consolidation failed to spin up a new server, while leading to lower energy consumption, could cause reduced overall throughput.

Throughput The *wasteful* function was in both approaches, *proactive* and *default*, able to process on average 117 requests per second. The *optimal* configuration achieved in the *default* experiment an average throughput of 115 RPS, while the *proactive* one was able to process 113 RPS. In all cases, the *proactive* consolidation approach did not negatively impact the throughput.

CPU Usage The *proactive* approach leads to a higher average function instance CPU usage in both cases. Specifically, the *wasteful* configuration in the *default* approach achieved an average of 15% and a max value of 41%. While the *proactive* approach achieved an average of 28% and a maximum value of 53%. In the *optimal* function instance case, the *default* approach achieved 17% average and 91% maximum CPU usage, and the *proactive* one achieved on average 29% usage and 100% max CPU usage. Thus, the *optimal* configuration was able to use up all its allocated CPU cores, and the *proactive* approach can increase average and maximum CPU usage.

Energy Consumption The last part summarizes the energy consumption results, which were calculated by excluding power values when servers were excluded from the cluster by our consolidation system. Results show an overall reduction in energy consumption. The *proactive* approach used in the *wasteful* experiment a an average of 0.133 kWh, while the *default* approach consumed 0.217 kWh. In the *optimal* configuration, the *default* approach used 0.223 kWh, while the *proactive* one used 0.18 kWh. Figure 3 shows one experiment that visualizes our consolidation decisions with respect to the incoming workload, where *17 D* means server *17* has been switched off and *17 U* means it has been turned on. We see that the proactive approach immediately shuts down servers *17* and *18*, and turns them back on when workload is rising. When keeping in mind that the throughput over all cases is the same, we see a clear benefit of using our approach over the *default*.

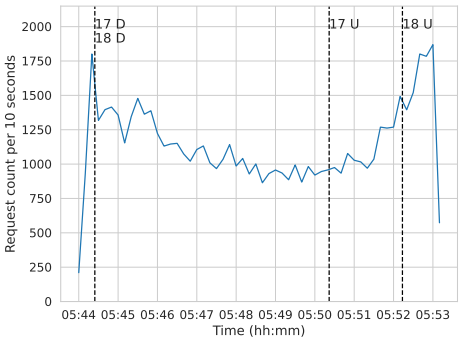

Figure 3: Proactive server consolidation

Competitive approaches

While Server Consolidation has been investigated in research [4, 5, 6, 7, 8], public offerings are more commonly referred to as Cluster Autoscaling (CA). There are several CA offerings available from public cloud vendors, such as Amazon [9, 10], Google Cloud [11], Alibaba [12], and also the built-in CA from Kubernetes [13]. However, these are tied to Big Data platforms [10, 11, 12], such as Hadoop, and focus on dynamically adding and removing Virtual Machines, or use a reactive approach and work with Kubernets [13, 9]. In contrast, our approach builds on a proactive approach with a focus on physical machines to save energy consumption and is applicable to any Serverless Computing platform that runs on Kubernetes.

Current status

The current prototype already works end-to-end with a real-world setup. However, the *Function Configuration Module*, to dynamically determine optimal Function Configurations, and the *Function Migration Module* to support the scheduler, have yet to be integrated in the prototype.

Next steps

Our solution offers a novel approach to allowing customers to build serverless clusters to handle peak workloads. It relieves them from unnecessary energy costs during downtime without compromising on performance. Noticeable improvements must be made towards the consolidation algorithm and the request forecast module. The former to include a reactive fail-safe mechanism in case the forecast does not accurately estimate workload, and the latter one to improve the accuracy. In addition, turning machines on is costly and should be carefully done.

References

[1] A. Joosen, A. Hassan, M. Asenov, R. Singh, L. Darlow, J. Wang, and A. Barker, "How does it function? characterizing long-term trends in production serverless workloads," in *Proceedings of the 2023 ACM Symposium on Cloud Computing*, SoCC '23, (New York, NY, USA), p. 443–458, Association for Computing Machinery, 2023.

[2] C. Delimitrou and C. Kozyrakis, "Quasar: resource-efficient and qos-aware cluster management," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, (New York, NY, USA), p. 127–144, Association for Computing Machinery, 2014.

[3] E. S. Gardner Jr., "Exponential smoothing: The state of the art," *Journal of Forecasting*, vol. 4, no. 1, pp. 1–28, 1985.

[4] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Energy-efficient resource allocation and provisioning framework for cloud data centers," *IEEE Transactions on Network and Service Management*, vol. 12, no. 3, pp. 377–391, 2015.

[5] C. Gu, Z. Li, H. Huang, and X. Jia, "Energy efficient scheduling of servers with multi-sleep modes for cloud data center," *IEEE Transactions on Cloud Computing*, vol. 8, no. 3, pp. 833–846, 2020.

[6] Z. Xiong, M. Zhao, Z. Yuan, J. Xu, and L. Cai, "Energy-saving optimization of application server clusters based on mixed integer linear programming," *Journal of Parallel and Distributed Computing*, vol. 171, pp. 111–129, 2023.

[7] L. Fan, C. Gu, L. Qiao, W. Wu, and H. Huang, "Greensleep: A multi-sleep modes based scheduling of servers for cloud data center," in *2017 3rd International Conference on Big Data Computing and Communications (BIGCOM)*, pp. 368–375, 2017.

[8] A. Varasteh and M. Goudarzi, "Server consolidation techniques in virtualized data centers: A survey," *IEEE Systems Journal*, vol. 11, no. 2, pp. 772–783, 2017.

[9] Amazon, "Karpenter." `https://github.com/aws/karpenter-provider-aws`, 2024.

[10] Amazon, "Aws glue auto scaling." `https://docs.aws.amazon.com/glue/latest/dg/auto-scaling.html`, 2024.

[11] Google, "Dataproc." `https://cloud.google.com/dataproc?hl=en`, 2024.

[12] Alibaba, "Hadoop autoscaler." `https://www.alibabacloud.com/help/en/emr/emr-on-ecs/user-guide/enable-or-disable-auto-scaling-for-hadoop-clusters`, 2024.

[13] Kubernetes, "Kubernetes cluster autoscaler." `https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler`, 2024.