Cloud-Native Server Consolidation for Energy-Efficient FaaS Deployment

Lu Zhang[1], Yifei Pu[1], Cheng Xu[1], Du Liu[1], Zeyi Lin[1], Xiaofeng Hou[3], Pu Yang[2], Shang Yue[2], Chao Li[1](✉), and Minyi Guo[1]

[1] Shanghai Jiao Tong University
[2] Tencent
[3] The Hong Kong University of Science and Technology
{luzhang,pkq2006,jerryxu}@sjtu.edu.cn, {lichao,guo-my}@cs.sjtu.edu.cn

Abstract. The lack of function-oriented power management scheme has seriously hindered the serverless platform's cost efficiency. In this paper, we analyze the invocation pattern of serverless functions and investigate its implications on server energy efficiency. Rather than using a one-size-fits-all strategy, we propose *DAC*, a software-hardware co-design solution to offer differentiated cloud-native server consolidation. We build a proof-of-concept framework and show that *DAC* can improve the energy efficiency of tail function deployment by up to 23%.

Keywords: Serverless, tail functions, cloud-native consolidation

1 Introduction

In recent years, Function as a Service (FaaS) has attracted considerable attention due to its easy application deployment. To mitigate the overhead of cold start, most FaaS providers adopt a *keep-alive* policy [4,5]. However, Existing scheduling frameworks either deploy functions on servers with idle resources or activate a new server. This often leads to poor data center energy proportionality, due to the significant static power consumption. In addition, functions incur various invocation interval time(IIT) [3]. We refer to functions that can be kept in memory for frequent invocation as *native functions*. However, there are functions that seldom invoke which we call them as *tail functions*. The different IIT of functions bring challenges for optimizing energy efficiency of FaaS infrastructure.

A key insight driving our work is that energy-efficient consolidation scheme should distinguish tail functions from native functions. First, reserving computing resources for tail functions would waste energy. Second, mixing tail functions with native functions causes more servers alive for native functions, thus increasing static power. Thus, it is necessary to treat tail functions differently.

Enlightened by the above observations, in this paper we envision *cloud-native server consolidation* to explore differentiated function management. We maximally consolidate native functions on server cores with the keep-alive policy. Meanwhile, we opportunistically map tail functions on cores that run native functions. Although modern scale-out architecture has strict power limits for

sustained workload, activating more computing resources for a very short time will not cause significant thermal/overloading issues [1, 2]. Short-lived tail functions are well-suited to take advantage of this opportunity. To ensure high performance, we should also treat tail functions differently since some tail functions can still benefit from warm start while some tail functions cannot.

We devise *DAC* (*differentiate and consolidate*), a novel load management strategy for cloud-native server consolidation. First, *DAC* adopts a two-level classifier to classify functions. The first-level classifier aims to distinguish native functions from tail functions. The second-level classifier further identifies the power-performance sensitivity for native functions and the invocation pattern for tail functions. Second, *DAC* uses a consolidation controller to adaptively invoke tail functions. It works to optimize the energy efficiency of servers.

This paper makes the following contributions:

- We classify serverless functions as native or tail functions according to their IIT and discuss the key design considerations of functions' consolidation.
- We propose *DAC*, a novel cloud-native server consolidation strategy that can deploy functions in an energy-efficient way.
- We demonstrate that *DAC* can achieve the efficiency improvement of tail function deployment up to 23% by extensive experiments.

2 Key Design Considerations

Function Invocation Patterns. We investigate invocation patterns from the perspective of invocation rates and invocation intervals using the data set from Azure Functions [3]. Some functions are invoked frequently and we term these functions as *native functions* which can be kept on dedicated servers to ensure warm start. Differently, there are also troublesome functions whose IIT is larger than the given keep-alive value. We refer to them as *tail functions* and the system cost of maintaining warm start for tail functions could be prohibitively high.

We show examples of native functions in Figure 1(a). The IIT is smaller than 10 minutes, which implies that they can be invoked with warm start under the keep-alive policy. Differently, tail functions can be classified into two categories: *explicit tail functions* and *implicit tail functions*. As shown in Figure 1(b), the IIT of explicit tail functions is clearly longer than 10 minutes where all functions would suffer from cold start. However, the implicit tail functions have an uncertain invocation pattern as shown in Figure 1(c). It is not easy to identify whether functions will suffer from cold start or warm start. The existence of implicit tail functions makes function power management a challenging problem.

Limitations of Current Solutions. The current FaaS load management solutions largely ignore the invocation rate of functions. They deploy functions according to the given policy and the current status. There are two key issues: 1) They mix tail functions with native functions. In this case, native functions have to be put on other servers which introduces cold start overhead. 2) It tends to activate more servers to process functions with load balancing. Since servers are far from energy-proportionality, doing so will increase energy consumption.

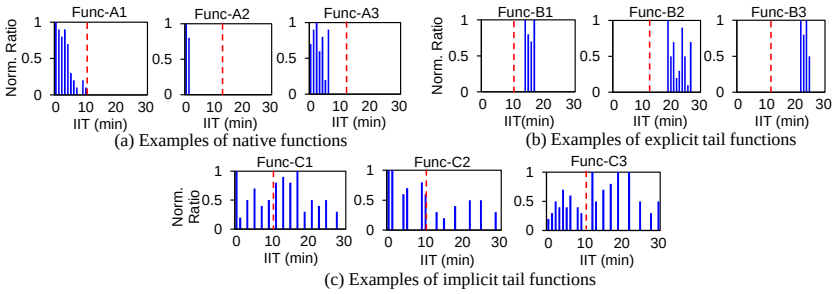


Fig. 1. The invocation interval time (IIT) of different Azure functions [3]

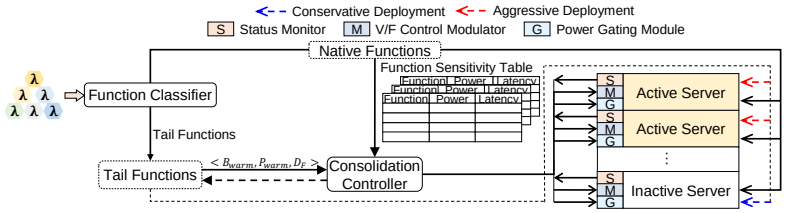


Fig. 2. Overview of the DAC design

We explore *cloud-native server consolidation* for tail functions, there are two deployment methods: *Aggressive Deployment* and *Conservative Deployment*.

Aggressive Deployment places tail functions with native functions. It faces two challenges. First, aggressive deployment would exceed the thermal/power constraint of servers. Servers must scale down the power level of cores, causing server-wide performance degradation. Second, due to the thermal limit, the chip can not support aggressive deployment for long time. Thus, incoming tail functions can not maintain the warm-start state.

Conservative Deployment has its merits. Native functions enjoy sufficient power budget for high performance. If tail functions are invoked successively, they can be deployed in the same server with warm start. The disadvantage of conservative deployment is that it increases the total energy consumption.

3 DAC Design

3.1 System Overview

Figure 2 gives an overview of the *DAC*. It consists of two main components to ensure efficient deployment of tail functions. The function classifier groups invoked functions into native or tail functions according to their IIT distribution. Meanwhile, a consolidation controller is used to coordinate the deployment of functions. The consolidation controller cooperates with a server-level power manager to robustly perform function consolidation. The controller monitors

the status of servers and the collected information can be used to guide function deployment. The power manager can adjust server power consumption by activating/deactivating cores or manipulating the V/F levels of cores.

3.2 Function Classifier

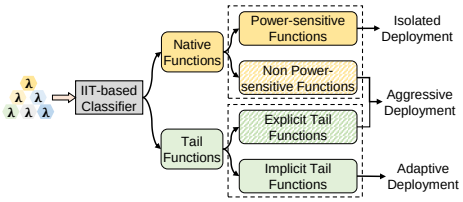


Fig. 3. *DAC* function classification

As shown in Figure 3, the classifier further divides native functions into either *power-sensitive* and *non-power-sensitive* functions. We can opportunistically place tail functions to servers that run non-power-sensitive native functions. Power-sensitive native functions must be processed in an isolated way for high performance. We create a sensitivity table to store how power affects native functions' latency. Explicit tail functions can be deployed with native functions for energy saving purpose. Importantly, we need to carefully choose the deployment decision for implicit tail functions.

3.3 Consolidation Controller

The consolidation controller manage the status of native functions and opportunistically place tail functions with an adaptive algorithm. It requires two parameters to make the proper deployment decision for each function: 1) B_{warm}, the beneifts of functions' warm-start and 2) P_warm, the warm-start probability. B_{warm} is given by: $B_{warm} = \frac{D_{init}}{D_{total}}$, where D_{init} is functions' initialization time while D_{total} is the total duration. P_{warm} of each function is presented as $P_{warm} = \frac{Ivk_{warm}}{Ivk_{total}}$, where Ivk_{warm} is the frequency of invocations whose IIT is less than 10 minutes while Ivk_{total} is the total number of invocation.

The consolidation controller further utilizes an adaptive function deployment algorithm to deploy tail functions. The latency of functions is given by:

$$L_A = D_F, \qquad L_C = D_F * (P_{warm} * (1 - B_{warm}) + 1 - P_{warm})$$

where D_F is the duration of the function, L_A and L_C are the estimated latency of aggressive and conservative deployment. The system energy under the two methods is given by:

$$E_A = Power_{Dynamic} * L_A, \qquad E_C = Power_{total} * L_C$$

To identify the best trade-off between energy and performance, we use the metrics: $Eff = 1/(\alpha\frac{L_A}{L_C} + \beta\frac{E_A}{E_C})$. where α, β presents the importance of functions' latency and energy saving respectively ($\alpha + \beta = 1$). If $Eff > 1$, it implies that the aggressive deployment is more energy-efficient and vice versa.

Table 1. The evaluated serverless functions

Function	markdown	img-resize	sentiment	ocr-img	autocomplete	matmul	linpack	dd
Runtime	python	nodejs	python	nodejs	nodejs	python	python	python
B_{warm}	0.89	0.76	0.5	0.04	0.25	0.2	0.58	0.33

Table 2. Evaluated tail function pool

Tail Function Pool	**Abbr.**	**Functions**
Low Warm Ratio ($0 < B_{warm} < 0.3$)	LWR	ocr-img,matmul autocomplete
Medium Warm Ratio ($0.3 \leq B_{warm} < 0.65$)	MWR	dd, linpack sentiment
High Warm Ratio ($0.65 \leq B_{warm} < 1$)	HWR	img-resize markdown
Hybrid Function Pool ($0 < B_{warm} < 1$)	HFP	All functions combined

Table 3. Mixed function invocations

Case	**Mixed Functions**
Case 1	LWR (P=0), MWR (P=0.3), HWR (P=0.6), HFP (P=0.9)
Case 2	LWR (P=0.9), MWR (P=0), HWR (P=0.3), HFP (P=0.6)
Case 3	LWR (P=0.6), MWR (P=0.9), HWR (P=0), HFP (P=0.3)
Case 4	LWR (P=0.3), MWR (P=0.6), HWR (P=0.9), HFP (P=0)

4 Evaluation

4.1 Methodologies

To validate and evaluate *DAC* under large-scale deployment, we implement a trace-driven evaluation framework. Our test bench takes realistic function execution trace as input. The information of tail functions is shown in Table 1. Table 2 and Table 3 show various function pools (We use P as P_{warm} for brevity). We compare *DAC* with two deployment schemes: *PerFst* and *EnerFst*. Like OpenWhisk, *PerFst* deploys functions on individual servers for the best performance. Differently, *EnerFst* applies aggressive deployment for tail functions to reduce the number of active servers which pursues energy saving [6].

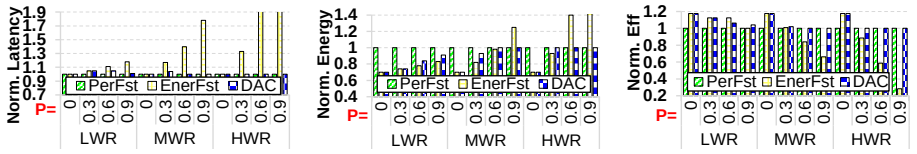
4.2 Evaluation Results

Figure 4 shows the latency, energy and efficiency of different schemes. All results are normalized to *PerFst* which yields the best performance. As seen in Figure 4(a), *DAC* achieves similar performance (up to 5% latency increase) under various warm-start probability values.

As for energy (Figure 4(b)), *DAC* consumes less energy than *PerFst*, but more energy than *EnerFst* in some cases. When P=0.9 in MWR and P=0.6/P=0.9, the latency of *EnerFst* could be very large. Although the dynamic power is smaller than the total server power, significant latency degradation may neutralize the benefits of power saving. Figure 4(c) shows the efficiency of different schemes. *DAC* outperforms all other schemes.

In addition, we evaluate our design using hybrid function pools as shown in Figure 5. The efficiency of *DAC* outperforms *PerFst* by 18% when P=0 while the improvement is 45% compared with *EnerFst* (when P=0.9).

We also investigate *DAC* under mixed functions shown Table 3. As shown in Figure 6, *DAC* achieves the best efficiency among all the evaluated schemes. On average, *DAC* improve the efficiency by 6% and 23% compared to *PerFst* and *EnerFst*, respectively. Although the latency of *DAC* is 2% larger than *PerFst*, *DAC* can save 13% more energy. The high efficiency of *DAC* implies that it can achieve a better design tradeoff between performance and energy.



(a) Normalized Latency (b) Normalized Energy (c) Normalized Efficiency

Fig. 4. Comparison of different schemes under functions with different warm ratios

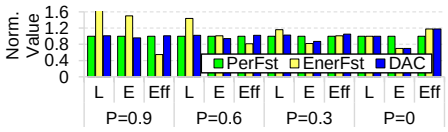


Fig. 5. Comparison of different schemes under *HFP*(L:Latency, E:Energy)

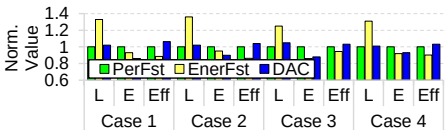


Fig. 6. Comparison of different schemes under mixed function invocation

5 Conclusion

To optimize the efficiency of FaaS infrastructure, we introduce cloud-native server consolidation and propose *DAC*. *DAC* differentiates functions according to invocation patterns and carefully consolidate tail functions. We hope that this paper can give insights to the design of energy-proportional FaaS platforms.

Acknowledgements. This work is supported in part by the National Natural Science Foundation of China (No.61972247), and a Tencent Research Grant.

References

1. Esmaeilzadeh, H., Blem, E., Amant, R.S., Sankaralingam, K., Burger, D.: Dark silicon and the end of multicore scaling. In: ISCA (2011)
2. Hou, X., Hao, L., Li, C., Chen, Q., Zheng, W., Guo, M.: Power grab in aggressively provisioned data centers: What is the risk and what can be done about it. In: ICCD (2018)
3. Shahrad, M., Fonseca, R., Goiri, Í., Chaudhry, G., Batum, P., Cooke, J., Laureano, E., Tresness, C., Russinovich, M., Bianchini, R.: Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In: ATC (2020)
4. Shilkov, M.: Cold starts in aws lambda (2021)
5. Shilkov, M.: Cold starts in azure functions (2021)
6. Zhang, L., Feng, W., Li, C., Hou, X., Wang, P., Wang, J., Guo, M.: Tapping into nfv environment for opportunistic serverless edge function deployment. IEEE Transactions on Computers (2021)