RFaaS: Function Scheduling Across Heterogeneous Clusters

Zhihang Tang[1,2], Yiming Li[1], Zezheng Mao[1], Laiping Zhao[1(✉)], and Keqiu Li[1]

[1] College of Intelligence and Computing, Tianjin University, Tianjin 300072, China
{tangzhihang,laiping}@tju.edu.cn
[2] Intelligent Computing Infrastructure Innovation Center, Zhejiang Lab, Hangzhou 311121, China

Abstract. The rapid development of cloud computing has attracted a diverse migration of applications. Serverless computing, with its abstract resource management, on-demand billing, and dynamic scaling, has become a popular cloud computing paradigm. X86 signifies traditional computility, while RISC-V symbolizes new potential. Making the most of existing traditional computing resources while exploring the potential of new computility will be a key challenge. The x86 and RISC-V hybrid computility supply will continue for a long time. Task management under diverse instruction set architectures is a critical issue that needs addressing in this evolving landscape. Current research predominantly concentrates on homogeneous instruction set clusters. In this paper, we propose RFaaS, a function job scheduling methodology tailored for RISC-V + X86 heterogeneous instruction set clusters, leveraging the OpenFaaS serverless computing platform. We delve into the affinity traits of function jobs in RISC-V + X86 amalgamated instruction set clusters and devise an affinity classifier alongside an architecture-aware scheduling algorithm. Our methodology dissects scheduling decisions into resource fulfillment and affinity alignment, underpinned by a meticulously crafted update algorithm to uphold and refine job affinities. Experimental results show that RFaaS can provide at least 3x performance improvement and 2.4x throughput increase compared to existing solutions.

Keywords: Heterogeneous Clusters · Serverless Computing · Function Scheduling · RISC-V

1 Introduction

The rapid development of cloud computing has attracted a diverse migration of applications. Serverless computing, known for its high level of abstraction in resource and programming management, on-demand billing, and dynamic scaling, has become a popular cloud computing paradigm. But with the processor architectures continue to evolve, the trend toward heterogeneous clusters is becoming increasingly evident. Heterogeneous clusters [1], comprising a variety of hardware components, allow for greater

Published by Springer Nature Switzerland AG 2025
X. Chen et al. (Eds.): NPC 2024, LNCS 15527, pp. 448–460, 2025.
https://doi.org/10.1007/978-981-96-2830-8_35

flexibility and efficiency in handling a variety of workloads, not only bolster the development and deployment aptitudes of cloud applications but also lay the groundwork for performance enhancements. Nonetheless, the distinct operational attributes of disparate hardware elements within heterogeneous clusters, coupled with the divergent tasks executed on different hardware nodes, escalate the intricacy surrounding resource allocation and scheduling determinations.

The X86 architecture represents traditional computility, has captured a significant share of the server market (e.g., achieved 88% in 2023 [2]). RISC-V [3] is a fifth-generation RISC (Reduced Instruction Set Computing) architecture, represents emerging and exploratory computility, has garnered significant attention due to its open-source nature, simplicity, modularity, and scalability, According to relevant agency forecasts, it is expected that RISC-V processors will occupy nearly a quarter of the global market share by 2030. In the foreseeable future, fully utilizing traditional computing resources while exploring the potential of new computility will be a key challenge. Therefore, X86 and RISC-V hybrid computing offerings will continue for a long time. In this trend, exploring and improving task management mechanisms under heterogeneous instruction set architectures has become a difficult problem that we must solve.

Existing works mainly focus on the homogeneous instruction set cluster, which can effectively allocate resources and schedule the homogeneous instruction set, but lacks the analysis on the impact of the heterogeneous instruction set [4],cannot exploited the characteristics of different instruction set clusters. This may result in poor execution performance and thus violating the SLO (service level objective) of the job. For example, OpenFaaS [5] is a popular Serverless framework, but has hardware inawareness shortcomings which may lead to latency sensitive functions being deployed on weaker hardware, and scheduling policy limitations.

In this paper, we will explore the function job scheduling problem of heterogeneous instruction set cluster based on X86 + RISC-V. This hardware combination selection aims to show how cross-node heterogeneity and intra-node heterogeneity should work together efficiently in concrete practice, and provide examples for heterogeneous computing research on representative hardware platforms. However, it is not easy to achieve, mainly due to the following challenges: Firstly, the computing efficiency and computing characteristics of heterogeneous cluster based on X86 + RISC-V are significantly different from those of traditional cluster, so it is necessary to profile it to obtain rich computing characteristics; Secondly, how to design an efficient scheduling algorithm according to the computing power characteristics of heterogeneous clusters to improve the utilization rate of computing power resources of heterogeneous clusters?

To address challenges above, our main contributions are as follows:

- We analyze the deployment problems and execution characteristics of jobs on RISC-V + X86 clusters, and profile the functional job characteristics of heterogeneous instruction sets.
- We design RFaaS, a function job scheduling method for RISC-V + X86 heterogeneous instruction set clusters, aimed at improving throughput while ensuring job execution times.

- We implement the prototype system of RFaaS, and evaluate its performance. Experimental results show that RFaaS can provide at least 3x performance improvement and a 2.4x throughput increase compared to existing solutions.

2 Background and Motivation

X86 is a traditional architecture with a Complex Instruction Set Computing (CISC) design, is known for its widespread use in personal computers and servers. In contrast, RISC-V is an emerging open-source architecture following a Reduced Instruction Set Computing (RISC) approach, offers simplicity and flexibility. One significant difference in function execution lies in the complexity of instruction sets: X86's CISC design allows for more complex instructions per operation, while RISC-V's RISC design favors a simpler and more streamlined approach to function execution.

To explore the relationships between different functions in RISC-V + X86 heterogeneous cluster scenarios and enhance the scheduling efficiency of heterogeneous instruction set clusters, we analyzed the characteristics of function tasks for two architectures. We selected some of the most popular current processor-level benchmarks (such as microbench [6], ServerBench [7], DeathStarBench [8], FunctionBench [9]) to test machines in both architectures. Combined with the experiments of Zorun et al. [10]. The test environment configuration is the same as in Sect. 4.2.

Figure 1 shows the results of a computationally structured microbench test, the Performance of RISC-V is low, with almost all functions performing less than half as well as on X86. The reason for the above results is mainly due to the differences in processor levels. It can be concluded that existing RISC-V processors still lack the same performance as mainstream cloud vendors' processors, showing disadvantages in compute-intensive tasks.

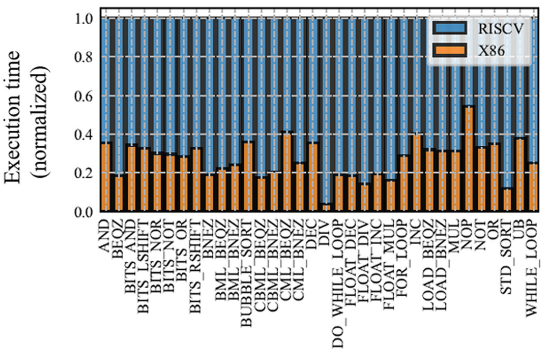


Fig. 1. Computational benchmark compares two architectures

In order to explore suitable tasks and fully leverage the advantages of RISC-V architecture, we make a more detailed exploration of the above benchmark. The first is that because RISC-V has simple instruction set makes context switching and interrupt handling more efficient, it may have an advantage when making operations with heavy

system calls. The experimental results in Fig. 2 show that, almost all system calls show better results on RISC-V than on X86. That is because although X86 processors have more mature hardware optimization technology, but if the function involves system calls, the execution of the function will not only occur on the CPU, but will involve interrupts, context switches, IO waiting and other scenarios, and in these scenarios, RISC-V architecture has some unique designs (for example, simple instruction set of RISC-V makes context switching and interrupt handling more efficient), so RISC-V architecture may have advantages when performing operations with a large number of system calls. Therefore, according to the characteristics of RISC-V + X86 heterogeneous cluster, designing a suitable function job scheduling method has certain research value.

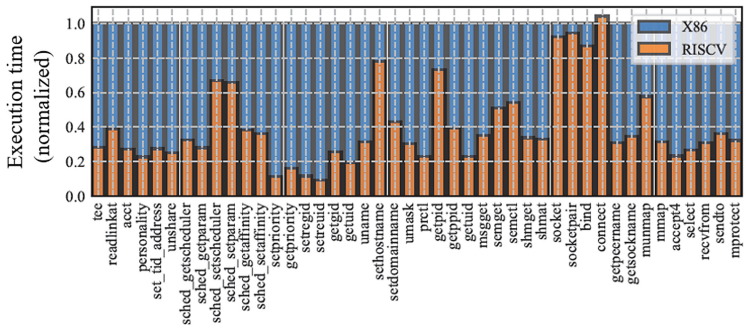


Fig. 2. Comparison of common system execution time between two architectures

3 RFaaS Design

3.1 Overview

The basic design idea of RFaaS is to optimize system performance by using the features and characteristics of different instruction set machines (such as system call time) combined with characteristics such as job affinity.

Figure 3 shows the overall design architecture of RFaaS. When a function is deployed in a cluster, RFaaS will characterize a single run of the function in an offline manner, record and maintain data from various dimensions during job execution ❶, and then train and maintain a classification model based on the results ❷. For online services, job requests that arrive at the gateway will enter classifier ❸. The classifier uses a trained classification model to classify the requested function to obtain its affinity, and places the job into the corresponding classification queue. The classifier will also dynamically update the classification model based on its offline characterization results. When a request enters the classification queue, it cannot determine its final deployment location because the classifier cannot perceive the status of the cluster, especially the queuing situation of the job. After passing through the classifier, the request will be sent to the scheduler module ❹. The scheduler module will periodically synchronize the cluster status ❺ and use heartbeat detection and other methods to check whether the service is

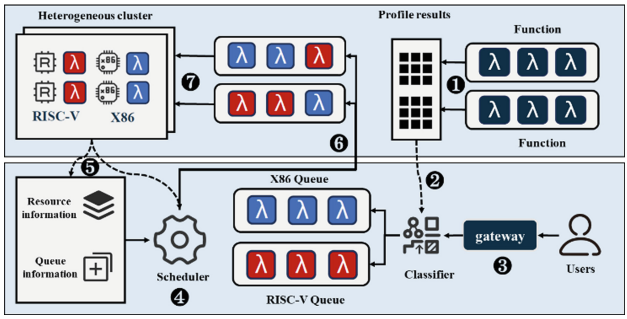


Fig. 3. RFaaS system architecture

alive. Finally, combined with the final completion time and other information carried by the job, the job will be sent to the queue maintained by the corresponding node ❻, and the node will decide which job to run on its own ❼.

3.2 Affinity Classifier

Metric Selection: We selected a series of performance metrics through correlation analysis, using Pearson [13], Kendall [14], and Spearman [15] correlation coefficients to evaluate the correlation between the target placement node and various metrics. Higher coefficients indicate stronger correlations. Table 1 shows the correlation scores between the target placement node and metrics under the three correlation analysis methods, based on metrics with a correlation score greater than 0.1. We selected 10 highly correlated metrics as inputs for the classification model.

Application Profile: We used tools like perf, sar, and strace to profile jobs. However, not all collected metrics can be directly applied to the model. If the correlation between metrics is low, it may lead to model overfitting, affecting generalization and accuracy. On the other hand, high input dimensionality can prolong model training and prediction time, increasing operational costs. Therefore, it is necessary to extract performance metrics highly correlated with inherent characteristics.

Incremental Model: To continuously optimize and update the classification model for optimal performance, we used online incremental learning [16]. Initially, we built a small workload metric dataset and corresponding labels to train the predictor. During execution, the predictor continuously expands the dataset by inserting new metrics and deployment nodes. This allows the model to self-update, improving prediction accuracy and handling dynamic changes effectively.

Model Selection: We summarized the characteristics of several popular incremental machine learning models, such as IKNN [17], ISVR [18], and IMLP [19], and used these models to build the classification model. We tested the accuracy results of k-nearest neighbor (KNN), logistic regression (LR), random forest regression (RFR), support vector machine (SVR), and multilayer perceptron (MLP) predictors. The evaluation metrics include Mean Absolute Error, Mean Squared Error, R2 (Coefficient of Determination),

Table 1. Correlation Scores Between Target Placement Node and Metrics.

Metric	Pearson Coefficient	Kendall Coefficient	Spearman Coefficient
Branch Mispredictions	0.416	0.603	0.738
CPU Migrations	0.337	0.314	0.361
CPU Clock Speed	0.183	0.649	0.792
Page Faults	0.114	0.580	0.662
Context Switches	0.382	0.669	0.797
Branch Instructions	0.256	0.701	0.857
CPU Utilization	0.283	0.197	0.231
Disk IO	0.427	0.295	0.346
Network Bandwidth	0.431	0.438	0.566
System Call Time Ratio	0.965	0.866	0.944
Branch Mispredictions	0.416	0.603	0.738
CPU Migrations	0.337	0.314	0.361

and Explained Variance Score. Among these, RFR significantly outperformed the other methods, achieving an accuracy as high as 99.8%. Therefore, we ultimately chose RFR as the algorithm for the classification model.

3.3 Architecture Aware Gittins Priority Scheduling Algorithm

Analysis of Request Scheduling Problem: Due to resource heterogeneity, sensitivity variations between requests and resources, and potential request concurrency, finding an optimal scheduling scheme for timely requests is highly complex. This complexity arises from the need to consider factors like resource heterogeneity, job sensitivity, node preferences, and request timing. Overcoming these challenges requires an intelligent scheduling algorithm and resource allocation strategy that maximizes cluster resource usage while meeting job SLO.

The Multi-Architecture Cluster Function Scheduling (MA-CFS) problem is defined as follows: Given the different architecture of server set $\{S_1, S_2, ..., S_m\}$ and a variety of different affinity function $\{f_1, f_2, ..., f_n\}$. Each server has a different resource capacity, and each function has different resource requirements. For the heterogeneous cluster composed of these multi-architecture servers, it is required to accurately identify each function and allocate resources reasonably in order to ensure the maximum degree of job SLO.

MA-CFS Problem Modeling: The objective function of MA-CFS problem can be set

as follows: $min\frac{\sum_{i=1}^{F}(d_i-b_i)}{F}$, where F represents the set of all requests, d_i represents the time when the execution of the request i starts, and d_i represents the time when the execution of the request i finishes.

This is a Linear Programming (LP) problem with many constraints. First of all, in order to ensure the request latency requirements, time d_i must ensure that the request has been completed no more than the requested deadline e_i, so we can draw the following constraints: $1 \leq d_i \leq e_i, \forall i \in F$. For the start of each request execution time b_i must be less than the request has been completed time e_i, so we can draw the following constraints: $1 \leq b_i \leq e_i, \forall i \in F$. For each request, the start execution time b_i must also be less than the request completion time d_i, so the following constraints can be derived: $b_i \leq d_i, \forall i \in F$. We also define the two binary variables x_{ij}^t and y_{ij}, x_{ij}^t indicates whether job i is running on server j at time t, and y_{ij} indicates whether job i is running on server j. The constraints related to these two variables and the relationship between them can be obtained: $x_{ij}^t \in \{0,1\}, y_{ij} \in \{0,1\}. x_{ij}^t \leq y_{ij}, \forall i \in F, \forall j \in S, \forall t = 1,2, ..., e_i$. To ensure that any request i is processed by only one server j at the same time, the following constraints can be obtained: $\sum_{j=1}^{S} y_{ij} = 1, \forall i \in F$. For each server, it is necessary to ensure that the sum of memory resource m_i^j and CPU resource c_i^j used by request i allocated on the server at any time t does not exceed the maximum server resource R_{mem}^j and R_{cpu}^j, so the following constraints can be obtained: $\sum_{i=1}^{F}\sum_{j=1}^{S} x_{ij}^t \cdot m_i^j \leq R_{mem}^j, \forall i \in F, \forall j \in S, \forall t = 1,2, ..., e_i, \sum_{i=1}^{F}\sum_{j=1}^{S} x_{ij}^t \cdot c_i^j \leq R_{cpu}^j, \forall i \in F, \forall j \in S, \forall t = 1,2, ..., e_i$.

Architecture-Aware Gittins Priority Scheduling Algorithm, AGPSA: The Architecture-Aware Gittins Priority Scheduling Algorithm (AGPSA) addresses the NP-hard MA-CFS problem by minimizing execution and queuing times. The algorithm schedules jobs without relying on precise duration estimates. When a request arrives, the job classifier assesses its affinity based on job characteristics, and the most compatible node is selected. If it's the job's first execution, the algorithm initializes the Gittins weight to 1.0. The algorithm then checks if the node's resources suffice; if so, the job runs immediately. If not, the job is queued with priority based on its cut-off and queuing times. Periodic checks determine if the job's Gittins weight exceeds a threshold for execution; otherwise, it waits.

Algorithm 1: Gittins weight update algorithm

```
Input: Job set J
Output: Job set J.
foreach job∈J do
    marking index completed on time?
    number of job executions times
    scheduling threshold α
    improve threshold β
     //wheatherJob is completed on time
    if job.realEndtime<job.endTime then
        index=1;
    end
    times++
   //update Gittins index
    if index==1 then
        if job.gittinsIndex< β then
            job.gittinsIndex←job.gittinsIndex*2;
        end
        else
            job.gittinsIndex←job.gittinsIndex +0.05;
        end
        job.gittinsIndex←job.gittinsIndex / Number of calls;
    end
    else
        if job.gittinsIndex ==1.0 then
            Job.gittinsIndex=α;
        end
        else
            job.gittinsIndex←job.gittinsIndex - 0.05;
        end
    end
end
return Job set J
```

4 Evaluation

4.1 System Implementation

We implemented RFaaS based on the OpenFaaS [5] serverless computing platform using approximately 4400 lines of Java code and 2000 lines of Python code. OpenFaaS is an open source serverless computing platform that simplifies the process of function deployment.

4.2 Experimental Setup

We employed two physical clusters, one based on X86 and the other on RISC-V architecture, as our experimental setups. Each node was equipped with 8 GB RAM and 32 GB storage. The processor frequencies of X86 machines and RISC-V machines are 2.5 GHz and 1.5 GHz respectively. In order to ensure the fairness of the experimental results, we reduced the frequency of X86 processors to the same as that of RISC-V machines. We

selected test functions from FunctionBench [20] and DeathStarBench [8] benchmark suites. Given the lack of a serverless platform tailored for RISC-V + X86 heterogeneous clusters, we adapted OpenFaaS, a server-agnostic platform, as a benchmark. To showcase the effectiveness of our proposed scheme, we implemented variants of RFaaS alongside OpenFaaS for comparison.

At present, there is no work to provide a serverless computing platform for RISC-V + X86 heterogeneous instruction set cluster, so we ported the existing server-unaware computing platform OpenFaaS as a benchmarking scheme. In order to reflect the effectiveness and advancement of the proposed scheme, we also implements variants of RFaaS, RFaaS^{-} and OpenFaaS.

4.3 RFaaS Component Evaluation

Classifier Effect Evaluation: We adjusted the input vector S described in formula (1), set it as zero vector, and fed the modified input into different machine learning models, thus obtaining each machine learning classification model of RFaaS after removing the index of kernel state time, and denoting it as RFaaS-wo-sc. In addition, Since the inherent execution characteristics of system calls inevitably lead to some redundancy between the kernel mode time indicator and other indicators, such as disk I/O and network bandwidth, this paper further evaluates the effect of removing disk I/O and network bandwidth two indicators and three indicators. The resulting classifier versions are denoted as RFaaS-wo-dn and RFaaS-wo-scdn.

Figure 4 illustrates the impact of excluding the kernel-state time measure on the accuracy of different machine learning classifiers. The KNN and LR models saw accuracy drops of 14% and 12%, respectively, while the RFR model experienced only an 8% decrease. This decline is expected, given the high correlation coefficient (0.965) of kernel-state time. KNN's heavy reliance on local data structure weakens its spatial recognition when a core feature is removed, and LR loses accuracy due to its dependence on linear correlations. In contrast, RFR's multi-decision tree strategy compensates for missing key features, making it the most resilient. Additionally, the comparison between RFaaS-wo-dn and RFaaS-wo-scdn indicates that kernel-state time, disk I/O, and network bandwidth, though somewhat redundant, are not entirely independent, with kernel-state time having a more significant effect on classifier performance.

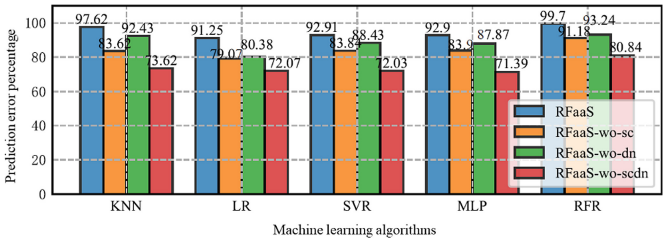


Fig. 4. Classification accuracy under different machine learning models

Scheduler Effectiveness Evaluation: Before evaluating scheduling algorithms, we first evaluated the way machine learning algorithms are used to predict time directly in existing work [16] to verify that "predictors of job execution times cannot make accurate predictions."

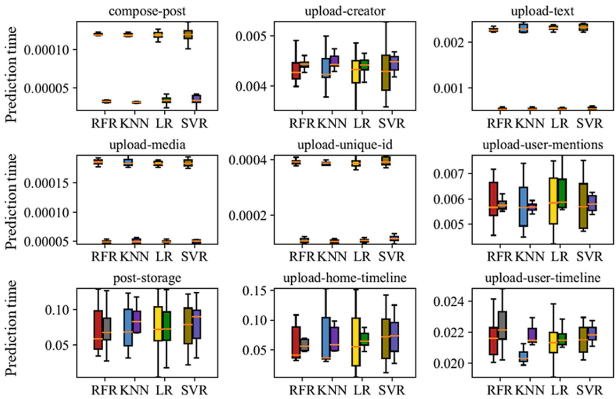


Fig. 5. Prediction of machine learning predictor

Figure 5 compares the prediction results of nine social network functions using four machine learning models: RFR, KNN, LR, and SVR. Each model, trained on the features listed in Table 1 and based on input size, is evaluated on both RISC-V and x86 architectures. The box plots illustrate that, while the prediction time can effectively indicate the optimal execution node for some functions (e.g., compose-post, upload-text), it fails to do so for others (e.g., upload-creator, post-storage), particularly with the RFR model, where classification errors range from 23% to 60%.

To evaluate the AGPSA scheduling algorithm, we integrated a classifier into OpenFaaS to identify function affinity, leading to two OpenFaaS variants: OpenFaaS-RRLB and OpenFaaS-FCFS, based on Round Robin Load Balancing (RRLB) and First Come First Service (FCFS) algorithms, respectively.

4.4 Overall RFaaS Evaluation

Overall Job Completion Time Evaluation: Figure 6 shows a comparison of the job completion time of all functions of RFaaS, RFaaS$^-$ and OpenFaaS solutions under three different loads. These functions are all from the previously mentioned applications, among which CP, UC, UUM and other functions with capital abbreviations are all components of social network applications. CP, UT, UM, UUI, CAP, matmul, float, dd, json belong to X86 affinity functions, UC, UUM, PS, UUT, UHT belong to RISC-V affinity functions.

Figure 6(a) shows that RFaaS and RFaaS$^-$ outperform OpenFaaS significantly in handling sparse workloads, reducing job completion times by at least half. This is due to OpenFaaS's inefficient classifier, which relies on Docker Swarm's default polling

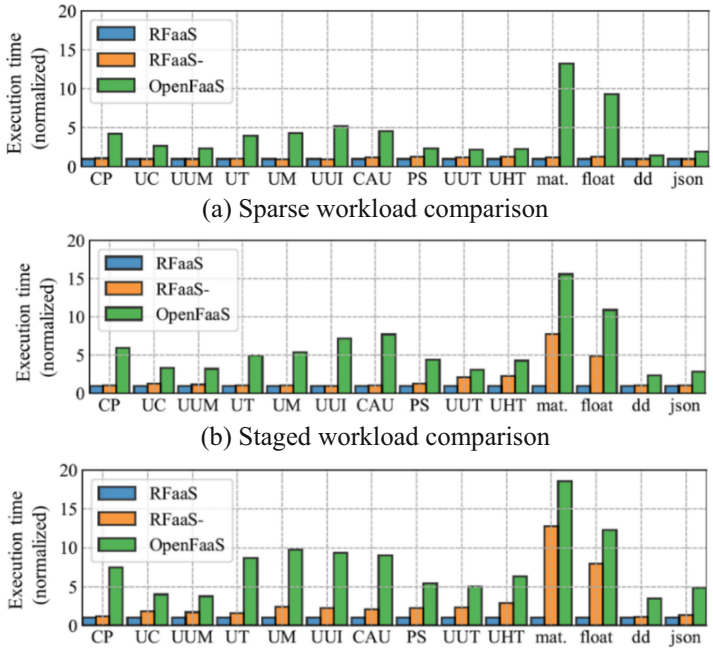
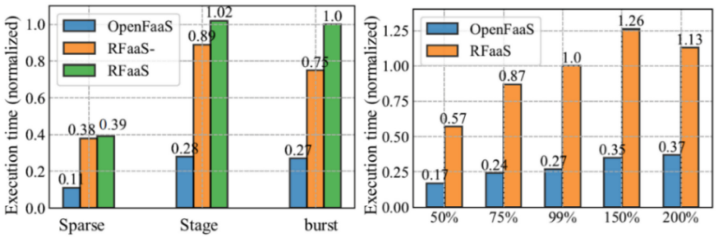


Fig. 6. Different solutions compare job completion times under three workloads

load balancing, leading to poor scheduling and inefficiency, especially when functions with high affinity for x86 architecture are run on RISC-V nodes. Figure 6(b) reveals an even greater performance advantage for RFaaS under staged workloads, as OpenFaaS's polling strategy increases scheduling errors and queue blocking, prolonging execution times. However, RFaaS$^-$ starts to show misclassification under high loads, impacting longer tasks like matrix multiplication. Figure 6(c) demonstrates that these issues are more pronounced under burst workloads.

Overall Throughput Evaluation: Throughput is a key metric for platform capability. This section compares the throughput of RFaaS, RFaaS$^-$, and OpenFaaS under three production loads. Figure 7(a) shows that RFaaS improves throughput by 3.54x, 3.64x, and 3.85x over OpenFaaS, and by 1.04x, 1.15x, and 1.33x over RFaaS$^-$. RFaaS achieves higher throughput due to its job classifier and efficient scheduling algorithm, which accurately identifies request affinity and avoids assigning jobs to slow nodes. In contrast, RFaaS$^-$ lacks effective classification, leading to reduced scheduling accuracy under heavy loads. Figure 7(b) illustrates that RFaaS improves throughput by 3.35x to 3.6x compared to OpenFaaS across various SLO settings.



(a) Throughput comparison of different solutions (b) Throughput comparison of different SLOs

Fig. 7. Throughput comparison

4.5 Results and Analysis

According to the complete evaluation of RFaaS, the performance of RFaaS and comparison methods is mainly measured by classifier accuracy, scheduling algorithm accuracy, component cost, job completion time and throughput. The following experimental conclusions can be obtained:

(1) The technical selection of RFaaS in each component is the optimal solution, in which the classifier can reach 99.7% accuracy, and can accurately identify the affinity of the job; The designed scheduling algorithm can guarantee the scheduling success rate of 63.6% even under a large number of sudden workloads. The importance of the selected system call time ratio index is verified. The accuracy of classifiers that removed this index decreased by up to 14%.
(2) RFaaS also fully guarantees the completion time of the job, regardless of the type of load, RFaaS has shown far better performance than OpenFaaS, and can provide at least 3 times the performance lead for each function; RFaaS also provides the largest throughput increase, at least 2.4x, compared to existing OpenFaaS solutions.

5 Conclusion

Aiming at the problem of job deployment on RISC-V + X86 cluster and the characteristics of job running on RISC-V + X86 cluster, in this paper,we designed a function job scheduling method RFaaS for RISC-V + X86 heterogeneous instruction set cluster. RFaaS includes two main components: a job affinity classifier based on system call time ratios and an architecture-aware Gittins priority scheduling algorithm. Through component testing and overall testing of RFaaS, it is verified that the job classifier and job scheduler can maximize throughput compared with existing solutions while guaranteeing job execution time.

References

1. Li, X., Mitchell, S., Fang, Y., Li, J., Perez-Ramirez, J., Lu, J.: Advances in heterogeneous single-cluster catalysis. Nat. Rev. Chem. **7**(11), 754–767 (2023)
2. Gartner. Market Share Analysis: Servers, Worldwide, 2023 (2024)

3. Waterman, A., Lee, Y., Patterson, D., Asanovi, K.: The RISC-V Instruction Set Manual. Volume 1: User-Level ISA, Version 2.0 (2014)
4. Zhao, L., Li, F., Qu, W., et al.: Aiturbo: unified compute allocation for partial predictable training in commodity clusters. In: Proceedings of the 30th InternationalSymposium on High-Performance Parallel and Distributed Computing, pp. 133–145 (2021)
5. Le, D.-N., Pal, S., Pattnaik, P.K.: OpenFaaS. Cloud Computing Solutions: Architecture, Data Storage, Implementation and Security, pp. 287–303 (2022)
6. RISC-V Linux kernel analysis. https://gitee.com/tinylab/riscv-linux
7. Yu, T., Liu, Q., Du, D., et al.: Characterizing serverless platforms with serverlessbench. In: Proceedings of the 11th ACM Symposium on Cloud Computing, pp. 30–44 (2020)
8. Gan, Y., Zhang, Y., Cheng, D., et al.: An open-source benchmark suite for microservices and heir hardware-software implications for cloud & edge systems. In: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 3–18 (2019)
9. Shahrad, M., Fonseca, R., Goiri, I., et al.: Serverless in the wild: characterizing and optimizing the serverless workload at a large cloud provider. In: 2020 USENIX Annual Technical Conference (USENIX ATC 2020), pp. 205–218 (2020)
10. Benchmarking RISC-V: VisionFive 2 vs the world (2024). https://blog.bitsofnetworks.org/riscv-performance-power-usage/
11. Jindal, A., Gerndt, M., Chadha, M., et al.: Function delivery network: extending serverless computing for heterogeneous platforms. Softw. Pract. Exp. **51**(9), 1936–1963 (2021)
12. Yu, G., Chen, P., Zheng, Z., et al.: FaaSDeliver: cost-efficient and QoS-aware function delivery in computing continuum. IEEE Trans. Serv. Comput. (2023)
13. Sedgwick, P.: Pearson's correlation coefficient. BMJ **345** (2012)
14. Abdi, H.: The Kendall rank correlation coefficient. Encyclopedia of Measurement and Statistics, pp. 508–510. Sage, Thousand Oaks (2007)
15. De Winter, J.C., Gosling, S.D., Potter, J.: Comparing the Pearson and Spearman correlation coefficients across distributions and sample sizes: a tutorial using simulations and empirical data. Psychol. Methods **21**(3), 273 (2016)
16. Zhao, L., Yang, Y., Li, Y., Zhou, X., Li, K.: Understanding, predicting and scheduling serverless workloads under partial interference. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–15 (2021)
17. Cover, T., Hart, P.: Nearest neighbor pattern classification. IEEE Trans. Inf. Theory **13**(1), 21–27 (1967)
18. Smola, A.J., Schölkopf, B.: A tutorial on support vector regression. Stat. Comput. **14**, 199–222 (2004)
19. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Netw. **2**(5), 359–366 (1989)
20. Kim, J., Lee, K.: Practical cloud workloads for serverless FaaS. In: Proceedings of the ACM Symposium on Cloud Computing, pp. 477–477 (2019)