2018 4th Iranian Conference on Signal Processing and Intelligent Systems (ICSPIS)

PESS-MinA: A Proactive Stochastic Task Allocation Algorithm for FaaS Edge-Cloud environments

Abolfazl Danayi
Electrical Engineering department
Amirkabir university of Technology
Tehran, Iran
adanayidet@aut.ac.ir

Saeed Sharifian
Electrical Engineering department
Amirkabir university of Technology
Tehran, Iran
Sharifian_S@aut.ac.ir

***Abstract*—By the advent of FaaS Cloud services and the micro-services programming architecture, designing task allocation algorithms with higher performance has become a crucial task. Motivated by this high-interested challenge, we propose a new allocation algorithm called PESS-MinA based on our novel modular model for FaaS Edge-Cloud environments. In contradiction to widely-used Max-Min and Min-Min algorithms which are both reactive and deterministic, this algorithm is based on stochastic score, and thus provides proactivity considerations. Experiments with Google Cloud Trace dataset show that our algorithm exhibits better performance in both resource load balancing and QoS assurance of FaaS. According to simulations, PESS-MinA decreased the dropped tasks percentage from 2.9% to 0.01%, alongside with a triple balancing score.**

Keywords— Proactive; Task Allocation; FaaS; Edge-Cloud

I. INTRODUCTION

Nowadays, Technology has shaped an ecosystem which is both extensive and complex. To manage complexity and extensiveness of a system, we need more modularized and automatic designs. In Computer Science, Cloud computing is one of the answers to this issue. Despite traditional data center models, a Cloud can be seen as a module that accepts and guarantees performing user requests over net, while the user is not concerned with the infrastructure and the hardware behind it [1]. Furthermore, Cloud computing suggests a new business model by providing several service models (e.g. IaaS, PaaS, SaaS and now FaaS).

In Function as a Service (FaaS) model, user asks a Cloud to execute a function (and return the result) through a standard internet based method (such as REST application programming interface function calls). Although by the advent of micro-services architecture FaaS model has received more attention, improving energy consumption and orchestration efficiency are still under research [2], especially in IoT applications where distributed computing platforms such as Nanodatacenters are proposed and power limits are critical [3].

A FaaS Cloud fulfills this promise by implementing two main functionalities: Resource Auto-scaling and Task to Resource Allocation. The first one provisions hardware resources while the latter assigns a task to the provided VM and allows it to use a specified amount of resource [4]. Considering all the mentioned factors, the problem of designing a task allocation algorithm becomes extremely challenging [1].

This work first proposes a model for FaaS Clouds and then suggests a novel proactive and stochastic task allocation algorithm suitable for both long tasks (like machine learning training processes) and IoT tasks (which need smaller functions to be executed so many times). In order to examine the performance, we compare our algorithm with other existing algorithms.

This paper is structured as follows. *Section 2*, presents the related work. In *Section 3*, we first introduce our model for FaaS Edge-Cloud environment, then discuss the algorithm, and finally, we present our pseudo-code for PESS-MinA. *Section 4*, presents implementation, simulation framework, the Google Cloud Trace dataset, and the results of running all discussed algorithms. The last section includes our conclusions and proposed future work and improvements.

II. RELATED WORKS

Previously, resource sharing and encapsulation was implemented by traditional Virtualization technologies based on an application running on top of the host operating system, called hypervisor, allowing guest OS images to be virtualized over it. Since these methods suffer from considerable setup (and boot) times, setup time is one of the main parameters in providing task scheduling solutions. On the other hand, Linux Container technology, such as Docker, offers an alternative way with less overhead and almost (based on the application of the VM) negligible setup time [4], [5], [6], [7].

A Min-min strategy first estimates the execution time of all unscheduled tasks in a batch, then assigns the task with minimum expected completion time to its corresponding VM and repeats this cycle until all the tasks are scheduled. Max-Min algorithm on the other hand, assigns the task with maximum expected completion time to the VM which can execute it in the minimum time [8], [9].

Min-Min and Max-Min give the priority to lighter and heavier tasks, respectively. RASA algorithm alternatively switches between them to make use of profits of both algorithms [10]. Another work, suggests an optimized version of RASA by conditioning its execution using a logical tree of conditions [2].

978-1-7281-1194-0/18/$31.00 ©2018 IEEE

It is necessary to notice that all the methods mentioned above, work on the basis of reducing Makespan which is the total time of execution of a batch. But other criteria such as power saving can be used too. As a power saving solution, most efficient server first algorithm (MESF) discussed in [12], tries to give the task to the datacenter (DC) that is nearby available first, and if there is no server available, it sends the task to servers from other regions.

III. Proposed Work

This section first describes our model for FaaS Edge-Cloud environment, and then presents the stochastic proactive algorithm we have proposed for task to resource allocation. We should emphasize on the relation between the Cloud model and designed algorithms. This fact motivates us to propose a suitable model for FaaS in Edge-Cloud environment. The final subsection presents our pseudo-code for PESS-MinA.

A. Proposed Model

Our model is designed based on the fact that the operation of FaaS in cloud environment can be considered as a loop. First, a user sends a request for execution of a task (function), to the Cloud. After that, the task will be analyzed and managed by the Cloud Broker and if it is permitted for execution, it will be sent to a specific VM or Container on the Datacenter. Finally, results will be returned to the User through an inverse path.

In the first level of modularization, User, Broker and Datacenter are three main separable blocks of this loop. Besides, in real-world, they should be isolated from each other for security reasons. According to these factors, in order to modularize our model, we define and use the term "Space". Each space must provide three main conditions: Isolation, High-Performance intercommunication and internet compatible communication with other spaces. We have illustrated our full model in Fig. (1).

1) User Space

As shown in Fig. (1), User space is the medium between the User and the Cloud Broker. User space can offer abilities such as Task catalog panel, Graphical User Interface and Fault correction to the whole FaaS system loop. This Space can be also considered as an Abstraction layer to APIs and other technical operations that the User may not be familiar with.

2) Broker Space

The main component which upgrades a traditional datacenter into a Cloud datacenter is the Cloud Broker. It is responsible for the process of task scheduling. Batch Collector and Allocation are functionally connected and both will be triggered by the same source: User incoming task requests. Auto-scaling can be triggered by a timer whose period time can vary over time. It is notable that since this paper is focused on the Allocation algorithm, in the implementation we have used the "Always On" algorithm for Auto-Scaling module. Each module uses its policies and problem solvers and also has access to the information from Workload Profiling module. Policies and even solvers can vary over time. There are no limits. Our model suggests a Broker Controller module which analyzes Workload Profiling data and changes policies, solvers and etc., when needed. Together these 6 modules create the Broker Space. This modular design can lead to a simpler and more realistic implementation and it can also provide a framework for parallel implementation of Broker server. For example, an Edge-Cloud, NaDa or even a private datacenter with specific applications may use FPGA chips to implement each module and utilize a main computer as the Broker Controller.

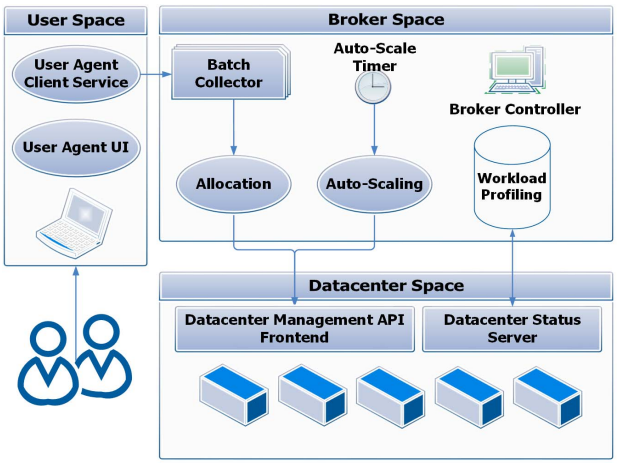

Fig. 1 Proposed Model

3) Datacenter Space

Since we have focused on Edge-Clouds, it is harmless to limit our model to contain just one DC. As discussed above, nowadays Container technology has addressed the issues of the virtualization. Thus, our model suggests using container images on physical hosts, and giving each image one function. This scheme not only supports micro-services architecture, but also reduces "Task to VM mapping and VM to processing element (PE) mapping" into "Container to PE mapping". Thus, in this paper, Container and Task will be treated as equal. As we accept this method, the important topic would be the way by which a Host computer shares its hardware resource between tasks running on it. We have used a time-shared scheduler in our model. Time-shared scheduler migrates container images over CPU cores of a Host, and in this way, each container receives an equal amount of processing power. However, there is a subtle performance degradation due to the container migration.

In this way, the task to resource allocation algorithm will be reduced to "Task to PE mapping" problem.

B. Proposed Algorithm

In this subsection, PESS-MinA is explained mathematically. The name comes from *Proactive Expected Stochastic Score – Min* with *Admission control*. This algorithm is originally a stochastic modification of Max-Min and Min-Min algorithms. This subsection first establishes a mathematical notation over the problem of task to resource allocation (due to the proposed model), then starts from Max-

TABLE I. ALGORITHM SYMBOLS

Symbol	Explanation
ω, v, i, j	*Task, PE, Task index, PE index*
T_s, T_a, T_d	*Submit time, Scheduling time, Deadline interval*
$\alpha, \delta, \gamma, \gamma_n$	*Number of instructions, Minimum needed processing power, Processing power, Shared Processing power for next incoming task*
$\Omega(t)$	*Set of currect running tasks*

Min and Min-Min algorithms and explains 4 modifications leading to our algorithm.

1) Mathematical notation establishment: We will use a set of rules in our notation: All underscripts refer to a property, and all superscripts refer to index. Each *condition* will be considered as a boolean function which returns 1 if the condition is satisfied and 0, otherwise. All parameters are explained in table (I), and we avoid explaining them in text, unless where it can not be avoided.
Equation (1) is used to model Task.

$$\omega = [T_s, T_a, T_d, i, j, \alpha] \tag{1}$$

Since we know the number of instructions or atomic steps of a task, and its deadline time, we can easily calculate the minimum processing power that can execute the task in time, using (2).

$$\delta = \frac{\alpha}{T_d - (Ta - Ts)} \tag{2}$$

We also use (3) as a formulation for PE.

$$v = [j, \gamma, \Omega(t)] \tag{3}$$

Previous section explained the Time-Shared scheduler that we have used. If a PE is running n tasks, each task will receive γ / n processing power. The next task, will change this value to $\gamma / (n+1)$. We call this latter value γ_n and calculate it from (4).

$$\gamma_n(t) = \frac{\gamma}{1 + |\Omega(t)|} \tag{4}$$

2) Designing the algorithm
Admission control (A): When a Min-Min or Max-Min algorithm is scheduling a batch of tasks, it will iterate on all of the tasks and each task will be assigned to a PE. As we use the proposed model of FaaS Cloud, we know the minimum processing power that each task needs (from (2)) and the processing power that each PE grants to its next incoming task (from (4)). Therefore, we will *refuse* a task which none of PEs can execute in its time. Equations (5) shows admission control conditions. If a task satisfies *R-Condition*, it will be *refused*. A *refused* task can be returned back to the user (to allow them retry in future), or be queued for automatic retries.

$$\begin{aligned} A(\omega^i, v^j): &\quad \delta^i < \gamma_n^j \\ R(\omega^i): &\quad \exists! v^j, A(\omega^i, v^j) = 1 \end{aligned} \tag{5}$$

Stochastic Proactivity (P): Min-Min and Max-Min are intrinsically greedy algorithms and this causes a reactive behaviour. To fix this issue, we first fit a PDF using batch data, then take some *virtual-task* samples from it, and finally append them into the batch which is being scheduled. This method forces the algorithm to consider a number of most probable forthcoming tasks, and allows refusing some of current tasks if they avoid future tasks from being accepted.
Stochastic Expectation (E): Another issue of Min-Min algorithm is being memoryless. In order to overcome this issue, instead of using only current batch to fit the PDF model, we suggest considering both current batch, and a number of previous batches.
Stochastic Score (SS): Max-Min gives priority to heavier tasks while Min-Min grants it to lighter tasks. However, we recommend putting the highest priority on the most statistically expected tasks. Since these tasks will be repeated in future. So as soon as a task is finished, another similar task will be replaced with it. We expect this scheme to result in a better load-balancing. As a solution, the Log-Likelihood will be considered the stochastic score of each task (6).

$$ss(\omega^i) = P(\omega^i \mid PDF) \tag{6}$$

C. Pseudo-Code and algorithm implementaion

In this subsection, we propose and explain the Pseudo-code for PESS-MinA. *Kappa* and *M* are proactivity and memory factors, respectively. *B* and *V* refer to the input batch of tasks and a copy of PEs set information. A copy is needed because of virtual-tasks that need to change PEs set information, without really begin assigned to a PE. *Refuse-Allocation* is the output vector whose every element shows if corresponding task in the batch is refused (-1) or, if admitted, it is equal to the index of the allocated PE (j > 0).

The highlighted line in the pseudo-code, refers to the SS (Stochastic Score) feature. If we replace this line with alternative lines from Min-Min and Max-Min algorithms, we can acquire PEMin-MinA and PEMax-MinA algorithms. Alongside these algorithms, Min-MinA and Max-MinA can be

PESS-MinA Algorithm Pseudo-code

Parameters: **Kappa**, **M**
Arguments: **B, V**
Output: **Refuse-Allocation**

N <- Length of **B**
Calculate ***Delta*** *for each task in* **B** *using (2), and store them in* ***Deltas***
Append ***Deltas*** *to* ***Mem***
Fit the ***PDF*** *with* ***Deltas*** *of last* **M** *batches from* ***Mem***
N_P <- **Kappa** * ***N***
Generate ***N_P*** *samples from* ***PDF*** *and append them to* **B**
For each Task in **B,** *Calculate Log-likelihood score*

While **B** *is not empty do*:
    i *= Index of the task in* **B** *with the highest score*
    j *= Index of the PE with the highest* ***Gamma***
    *If (****Gamma*** *of* **V[j]** < ***Deltas[i])***
        If ***(i)*** *does not refer to a virtual-task:*
            Refuse the ***(i)*** *task and eliminate it from* **B.**
        End if
    Else
        If ***(i)*** *does not refer to a virtual-task:*
            Allocate the ***(i)*** *task to the* **V*(j)***.
        End if
        Eliminate ***(i)*** *from* **(B).**
        Update **V[j]***'s parameters due to its new task or virtual-task.*
    End if
End While

Return the ***Refuse –Allocation*** *vector*

easily achieved by applying just **A** (Admission) feature to classical Min-Min and Max-Min algorithms. (The procedure is straight-forward and we avoid explaining in details in this paper).

Although we have introduced 5 algorithms, PESS-MinA is chosen for the paper title, since as it will be discussed in section (IV), this algorithm provides the best performance among all implemented algorithms.

IV. Implementation and Experimental Results

In this section, implementation details and the dataset that is used in the simulation, are explained. Then the results of running the simulation and a comparison between algorithms are given.

In order to simulate the datacenter, we have used the academic CloudSim 3.0 toolkit [12]. We modified some classes of CloudSim to support our model. Besides, we developed a number of new classes to add dynamic simulation ability to it. To cover our modular design, Cloud broker is completely implemented (Developed) as a Python 3.6 application. The datacenter (CloudSim) and Cloud broker (Python application) are connected using standard TCP/IP protocol over localhost address. We have used Google Cloud Trace dataset [13]. One datatype that this dataset provides is the task status table. Table (II) explains how we have used the task status table of Google Cloud Trace to derive our simulation dataset. Each entry in the derived dataset is a task request, with its specified mega-instructions, submission time and deadline time. (We have also derived priority number which is not used in this work). Fig. (2) depicts the number of task requests per seconds vs time. Our computer with an Intel core-i7-6500U CPU and 8 G-Bytes of RAM, is used for running the cloud-broker server program and the simulation toolkit (CloudSim) for our dataset consisting of 42298 tasks. Each simulation takes about 10 minutes.

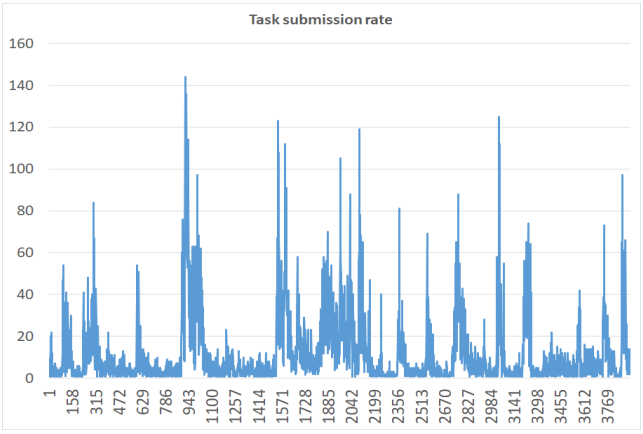

Fig. 2 Task submission rate

In the simulation, datacenter has 256 host computers where each host has a 2400 MIPS CPU with 4 Cores, 4 Gigabytes of RAM and 1 Terabytes of storage. Each batch consists of 300 tasks. As mentioned before, we have used an "Always On" strategy in the Auto-Scaling module. A Gaussian Mixture Model (GMM) with 2 Gaussian components is considered as the stochastic PDF that our algorithm needs. Kappa and M are chosen from {0, 0.05} and {0, 1}, respectively.

Results are given in Table (III) and fig. (3). In table (III), Light gray, dark gray and the black cells of each column are respectively first, second and the last ranks of that attribute. **S-Rate** shows the Success rate and It is the percentage of tasks which are successfully executed in time. **A-Rate** stands for Admission rate and is equal to the admitted tasks percentage.

TABLE II. Deriving dataset from Google Cloud Trace

Symbol	How to read from Google Trace
ω	*Each entry line is a task request*
T_s, T_d	*Submit time of each entry, Average of execution time of each class of tasks*
α	*Average of utilization*2400*execution time for each class of tasks (2400 MIPS is our assumption)*

TABLE III. EXPERIMENTAL RESULTS

Algorithm	S Rate (%)	A Rate(%)	P Rate (%)
Round-Robin	91.4	100	91.4
Min-Min	97.1	100	97.1
Max-Min	96.6	100	96.6
Min-MinA	97.4	97.5	99.1
Max-MinA	97.2	97.2	99.99
PEMin-MinA	96.3	96.3	99.99
PEMax-MinA	97.3	97.3	100
PESS-MinA	98.1	98.1	99.99

P-Rate is the percentage of successful admitted tasks. **B-Value** demonstrates the balancing score and is the square root of variance of idle-times of all PEs. A lower **B-Value** is related to a more balanced and utilized algorithm.

As shown in Table (III) and fig. (3), PESS-MinA has the best **S-Rate**. It means that this algorithm has the most successfully executed tasks. Although it has the second rank in **A-Rate**, the fact that Round-Robin, Min-Min and Max-Min do not use admission control, makes their **A-Rate** number valueless; And between other algorithms, PESS-MinA is the first in the case of **A-Rate**. We accept its **P-Rate** of 99.99 as a very high value because it had only 2 dropped tasks, among 41517 admitted tasks. Finally, Its **B-Value** is the second rank. Tough PEMin-MinA with the best **B Value**, has the worst **A-Rate**.

V. CONCLUSIONS AND FUTURE WORK

In this paper we proposed a model for FaaS Edge-Cloud environments which splits the FaaS Cloud loop to 3 main spaces: Datacenter, Cloud Broker and the User. In our simulation and implementation, the Datacenter and User spaces are simulated using an academic simulation tool called CloudSim. As a proof of the usability of our modular design, we implemented and developed our Cloud Broker program in python 3.6 and connected CloudSim to it using a standard TCP/IP connection. We also proposed and implemented a novel algorithm named PESS-MinA. Using Google Cloud Trace dataset, we simulated this algorithm besides Round-Robin and extensions of Min-Min and Max-Mina algorithms. Experimental results show that our algorithm not only is the best in the case of number of successful tasks but also it provides a very good load balancing.

We suggest the following items, as the future work of this paper:

1- Kappa and M parameters should be set adaptively
2- This version of PESS-MinA is based on CPU power, but Bandwidth, Latency, Memory and Storage are other parameters of a function that should be considered

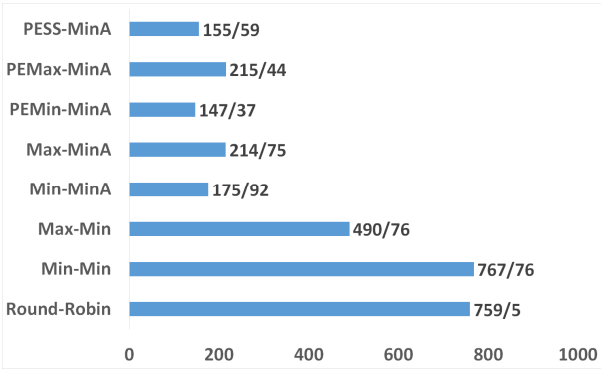

Fig. 3 Balancing Score (B Value)

3- In a micro-services architecture, functions of an application are related to each other, and providing a new model for application, coinciding micro-services architecture is valuable.

VI. REFRENCES

[1] S. Mittal and A. Katal, "An Optimized Task Scheduling Algorithm in Cloud Computing," *2016 IEEE 6th International Conference on Advanced Computing (IACC)*, 2016.

[2] M. Gribaudo, M. Iacono, and D. Manini, "Performance Evaluation of Replication Policies in Microservice Based Architectures," *Electronic Notes in Theoretical Computer Science*, vol. 337, pp. 45–65, 2018.

[3] V. Valancius, N. Laoutaris, L. Massoulié, C. Diot, and P. Rodriguez, "Greening the internet with nano data centers," *Proceedings of the 5th international conference on Emerging networking experiments and technologies - CoNEXT 09*, 2009.

[4] I.-D. Filip, F. Pop, C. Serbanescu, and C. Choi, "Microservices Scheduling Model Over Heterogeneous Edge-Cloud Environments As Support for IoT Applications," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2672–2681, 2018.

[5] A. M. Joy, "Performance comparison between Linux containers and virtual machines," *2015 International Conference on Advances in Computer Engineering and Applications*, 2015.

[6] K.-T. Seo, H.-S. Hwang, I.-Y. Moon, O.-Y. Kwon, and B.-J. Kim, "Performance Comparison Analysis of Linux Container and Virtual Machine for Building Cloud," 2014.

[7] C. Pahl and B. Lee, "Containers and Clusters for Edge Cloud Architectures -- A Technology Review," *2015 3rd International Conference on Future Internet of Things and Cloud*, 2015.

[8] B. Taneja, "An empirical study of most fit, max-min and priority task scheduling algorithms in cloud computing," *International Conference on Computing, Communication & Automation*, 2015.

[9] O. M.elzeki, M. Z. Reshad, and M. A. Elsoud, "Improved Max-Min Algorithm in Cloud Computing," *International Journal of Computer Applications*, vol. 50, no. 12, pp. 22–27, 2012.

[10] Parsa, "RASA: A New Grid Task Scheduling Algorithm," *International Journal of Digital Content Technology and its Applications*, 2009.

[11] Z. Dong, N. Liu, and R. Rojas-Cessa, "Greedy scheduling of tasks with time constraints for energy-efficient cloud-computing data centers," *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 4, no. 1, 2015.

[12] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2010.

[13] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale," *Proceedings of the Third ACM Symposium on Cloud Computing - SoCC 12*, 2012.