International Journal of Computer Applications Technology and Research
Volume 12–Issue 09, 01 – 06, 2023, ISSN:-2319–8656
DOI:10.7753/IJCATR1209.1001

Microservices Scheduling Algorithms: A Survey Study

Ghizlane EL HORRI
Information Technology and Engineering Sciences
National School of Computer Science and Systems Analysis Mohammed V University
Rabat, Morocco

Mostapha ZBAKH
Information Technology and Engineering Sciences
National School of Computer Science and Systems Analysis Mohammed V University
Rabat, Morocco

Abstract: The development of many large and complex applications has led to the need to come up with a better solution to manage those applications from one large system into a combination of small services that work together in a cohesive way for a widely used application where it can be easy to deploy, configure, and scale. The popularity of microservices architecture in different fields has made it susceptible to development, especially in task scheduling. We report on the scheduling algorithms that have been used by many researchers and discuss their approaches. This report will help us ameliorate the flexibility of the system in future studies.

Keywords: Microservices architecture, Container, Performance Metrics, Microservices Scheduling Techniques.

I. INTRODUCTION

The microservices architecture is the solution to go from large and complex applications to a combination of small services that can be deployed, configured, or even scaled [1]. Many organizations tried to solve the problem of complex applications that consume time, energy, and cost to deploy, configure, or scale. For this purpose, the concept of microservices can help us divide our application into smaller, interconnected services.

Microservices enable the user to build and maintain the application in the easiest way possible. Before microservice architecture, there was what we call monolithic architecture [2]. The code side of the monolithic architecture is implemented as one large system that shares the same database, which will lead to many problems in terms of management and the redeployment of the whole application.

To solve those problems, we moved from monolithic architecture into SOA architecture, which is an acronym for service-oriented architecture [3]. This architecture separates services into different modules that communicate with each other via a service bus to form the whole application. The problem in SOA architecture is the database storage that is shared with the whole application, as well as the increase in response time and machine load because of the interaction between services. Which leads us to microservices architecture, where the application is created using multiple microservices and each has its own database.

However, there are many tools that support microservices in building applications [4]. Docker, for instance, packages up code and all its dependencies and libraries so that applications can run from one computing environment to another, for instance, from a developer's laptop to another test environment. There are several benefits to containers, such as the fact that they could be lightweight because they share the same OS without the need for a full OS instance per application. Containers could also be portable and platform-independent; they can support modern development and architecture, as well as improve the utilization of application components in the microservices architecture, for instance.

The utilization of containers should be orchestrated by another tool. Kubernetes [4] helps containerized application to be deployed, managed and scaled. There are multiple benefits of orchestration, for instance preventing any unwanted access through using firewalls for example, as well as, having flexible operations and flexible data transfer. Furthermore, it can be economical especially for companies.

The use of microservice architecture has been increasing in the last few years. There are many large companies that have used this particular architecture, such as Netflix, LinkedIn, and Amazon [5]. The application of microservices architecture lies in the fact that it has various benefits in terms of deployment and scalability, as well as the fact that each microservice can be implemented in different languages and be more flexible.

However, the increase in cloud workload, such as Internet of Things (IoT) devices, machine learning applications, cloud storage, and streaming audio and video services, has led to extra demand for several cloud services. For that purpose, the deployment of the applications should meet performance requirements—the response time, for instance—and should also decrease the cost of cloud resources. Many researchers have tried to ameliorate microservices-based applications by working on two main problems: task scheduling and auto-scaling [1]. In the task scheduling context, tools like Docker Swarm and Kubernetes use scheduling strategies for containers and deploy those containers to the proper nodes. However, scheduling can get crucial in terms of cost-efficient operation in the cloud, which led the researchers to develop several scheduling algorithms to fulfill many targets, such as response time, load balancing, resource utilization, reliability, and energy consumption [6]. Therefore, we tried in this paper to highlight the existing scheduling algorithms as well as study their durability and limitations.

The rest of the paper is formulated as follows: Microservices scheduling techniques and performance metrics are introduced in Section 2. Microservices scheduling algorithms are being discussed in Section 3. Section 4 has a comparison of the algorithms referred to in the previous section. Section 5 will conclude the paper.

International Journal of Computer Applications Technology and Research
Volume 12–Issue 09, 01 – 06, 2023, ISSN:-2319–8656
DOI:10.7753/IJCATR1209.1001

II. MICROSERVICES SCHEDULING TECHNIQUES AND PREFORMANCE METRICS

The microservices architecture is widely used nowadays to facilitate the work of many applications. However, it doesn't prevent the researchers from developing the microservices architecture to be more efficient and flexible. Many researchers tried to come up with new solutions for scheduling microservices and reaching optimal performance. As shown in Fig. 1, it is a generalization of how the scheduling of microservices works [7]. The incoming requests from users are treated by the scheduler to find the most suitable placement for those requests using various performance metrics. Each microservice has a specific number of container instances, which vary depending on the user's requests. The requests may be scheduled immediately into containers through physical machines (PM) or into virtual machines (VM) through PM.

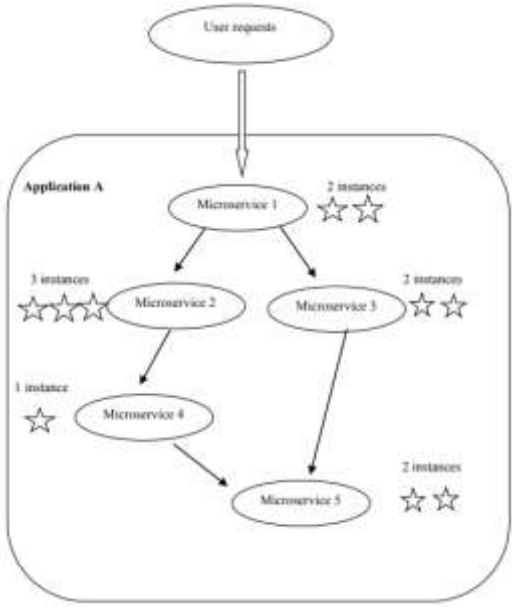

Figure 1 : Application example in the cluster

In this section, we first discuss the most common scheduling algorithm categories as well as the performance metrics used by the researchers to evaluate their algorithms.

A. Scheduling algorithms

The scheduler uses various algorithms that are categorized into four different genres. Each one of those techniques works and operates differently than the others to solve multiple problems, such as scheduling problems [8].

a. Mathematical modeling

The first category finds an optimal solution using different techniques, such as integer linear programming (ILP), where it uses objective functions and equations that are linear as well as constrained variables that are integer. We can also have mixed-integer linear programming (MILP), where some variables are not discrete. Another technique is quadratic programming (QP), which is a technique that tries to find an objective quadratic function with the use of constraints that can be either linear inequality or equations. Quadratic-Constrained Programming (QCP) utilizes an objective quadratic function and quadratic constraints likewise [8].

b. Heuristic techniques

The next category came to solve container scheduling problems, where most of the time it uses the bin packing technique, which is an optimization problem that helps minimize the number of bins, particularly by assigning items of different weights to bins that have specific capacity and trying to minimize the total number of used bins. This category can also use a combination of different techniques in Docker Swarm and Kubernetes to schedule containers to the right nodes. The use of heuristic techniques may be fast and scalable, but optimal solutions are not ensured [8].

c. Meta-heuristic techniques

Meta-heuristic techniques are becoming more useful to solve optimization problems in several fields. Meta-heuristic techniques are categorized into evolutionary algorithms such as genetic algorithms, and swarm intelligence algorithms, such as Ant Colony Optimization, Particle Swarm Optimization, and many others.

As for the first category, genetic algorithms have become a focus of interest for many researchers because they are influenced by the theory of natural evolution. The individuals of the evolution process are being selected using their fitness to generate the next offspring for the following generation. Moreover, the Ant Colony Optimization algorithms are swarm-based search algorithms that are inspired by the behavior of ants in searching for food. The main goal of this type is to increase resource utilization using suitable load balancing. Concerning Particle Swarm optimization, it is one of the robust techniques that is influenced by the behavior of birds and helps improve the resource utilization and load balancing of the system [8].

d. Machine learning techniques

Machine learning algorithms allow you to build a model from data using algorithms that obtain a predictive analysis using this data. The use of machine learning allows computers to learn without being programmed, i.e., existing data can be used for future behaviors and trends. Many researchers have utilized machine learning techniques to enhance resource utilization by minimizing the number of computing nodes and reducing energy consumption [8].

International Journal of Computer Applications Technology and Research
Volume 12–Issue 09, 01 – 06, 2023, ISSN:-2319–8656
DOI:10.7753/IJCATR1209.1001

B. Performance metrics

In this section, we are going to introduce the most common metrics used by researchers to evaluate the performance of the proposed algorithms for the purpose of formulating algorithms for more efficient container scheduling [8].

Energy efficiency: this type of metric tries to find an adequate scheduler that minimizes the energy consumption of the whole cluster in order to increase revenues as well as upgrade sustainability.

Cost: the communication cost indicates the telecommunication services rented with a variable to run the application. The more the communication are increased, the more the cost becomes greater.

Availability: This metric assures whether the scheduler is able to guarantee the application's availability to the user whenever he or she wants it.

Resource utilization: for this metric, it indicates how the resource utilization of a work node can affect resource efficiency in terms of memory, core, and network bandwidth.

Load balancing: this metric assures that the scheduler is able to evenly distribute the workload across nodes in a way that it won't be overloaded.

Scalability: the metric guarantees that the scheduler is in a position to provide the user with the intended service even though there is an increase in demand on the system.

Makespan/Latency: The scheduler should minimize the makespan or latency in such a way that the required time to run the application from the beginning to the end is reduced.

Throughput: is calculated by dividing the total number of tasks by the amount of time needed to execute the tasks.

Security: This metric tries to assure that the scheduler has the ability to protect data and services from attacks or software bugs.

III. MICROSERVICE SCHEDULING ALGORITHMS

In this section, we introduce the latest scheduling algorithms proposed by researchers and discuss the strengths and limitations of those algorithms.

The authors in [9] proposed an approach called Least Waste, Fast First (LWFF). The concept of this model is to schedule microservices instances in the workload queue S to nodes represented by N. The authors represented the scheduling using the mapping function $sched : S \rightarrow N$. The scheduling algorithm developed by the authors is formulated in the form of a complex variant of the knapsack problem.

The knapsack problem is a combinatorial optimization problem; the idea is to pack a set of items, that have a value and a weight, into a knapsack that has particular capacity under the condition of having a maximized value of the items inside the knapsack. In the scheduling algorithm proposed by the authors, the nodes are going to be packed by microservices instances. The nodes are associated with two computational resources, memory limitation MEM_j and CPU limitation CPU_j, as well as 2-dimensional capacity vector that has memory and CPU capacity [9].

Afterwards, the authors tried to formulate the memory and CPU utilization of each node in a time interval by using equations (1) and (2), as well as define the average utilization of the cluster since each node has different resource capacities by using equations (3) and (4).

$$mem\_util(n_j, \Delta t) = \frac{\sum_{\{\forall s_i \rightarrow n_j\}} (mem_i.part(runtime_j^i, \Delta t))}{MEM_j.\Delta t} \quad (1)$$

$$cpu\_util(n_j, \Delta t) = \frac{\sum_{\{\forall s_i \rightarrow n_j\}} (cpu_i.part(runtime_j^i, \Delta t))}{CPU_j.\Delta t} \quad (2)$$

$$\overline{mem\_util(\Delta t)} = \frac{\sum_{j=1}^{p} MEM_j.mem\_util(n_j, \Delta t)}{\sum_{j=1}^{p} MEM_j} \quad (3)$$

$$\overline{cpu\_util(\Delta t)} = \frac{\sum_{j=1}^{p} CPU_j.cpu\_util(n_j, \Delta t)}{\sum_{j=1}^{p} CPU_j} \quad (4)$$

Thereafter, the authors attempted to introduce a profit function that will be used later as an objective function to be maximized. The equation (5) defines the profit function as a vector using equations (3) and (4).

$$profit(s_i, n_j) = (\overline{mem\_util(\Delta t)}, \overline{cpu\_util(\Delta t)}) \quad (5)$$

Eventually, the scheduling problem is being formulated by the authors using a bi-objective optimization problem as follows:

$$\forall s_j \in S \wedge \forall n_j \in N \; maximize \; profit(s_i, n_j)$$

$$Subject\ to\ \forall time\ t \wedge \forall n_j \in N \begin{cases} \sum_{\forall s_i \rightarrow n_j \; in \; t} mem_i \leq MEM_j \\ \sum_{\forall s_i \rightarrow n_j \; in \; t} cpu_i \leq CPU_j \end{cases}$$

At the beginning, to choose the microservices, the authors apply the approach first come, first served (FCFS), afterwards, the algorithm will allocate the service to the suitable node by achieving three phases: filtering, producing the Pareto set, and choosing the final solution. At first, the algorithm will generate a set of feasible nodes that meet the requirements of a specific service, and then it will runs a comparison based on a profit equation for all the nodes in the set of feasible nodes. Furthermore, to allocate each service to the proper node, the authors calculate the memory utilization and CPU utilization of the whole cluster. Next, the algorithm will calculate the profit vector of each decision that has been made to use it next in the Pareto set that has the non-dominated solutions after removing all the dominated solutions. In the next step, the solution that has the least execution time will be taken from the Pareto set to assign the service to the selected host as a final step [9].

To evaluate the efficiency of the proposed algorithm, the authors compared it with another two scheduling algorithms, which are Spread and Binpack. The spread approach tries to select the nodes with the least load, while Binpack maximizes the utilization of the nodes. At first, the authors compared the average utilization of memory and CPU in clusters between the three algorithms using nine different classes on six different nodes from AWS EC2, they concluded that the LWFF algorithm overcomes the two other algorithms by having the best memory and CPU utilization simultaneously.

The authors worked on different metrics to evaluate their algorithm. The first metric is scheduling latency and its effect on the execution time of services. The authors discovered that the latency of the other two algorithms is lower than the LWFF algorithm. On the other hand, the execution time of the LWFF algorithm is faster than the other two. The authors measured the throughput of the active nodes per second for the three

algorithms and concluded that LWFF has the highest throughput of the other algorithms. According to the metrics presented earlier, the authors concluded that their approach is the best choice for scheduling microservice instances into nodes [9].

Other researchers [10] suggest a model for multi-objective resource scheduling for vehicle-to-everything (V2X) microservices based on the edge container cloud architecture. The scheduling model that the authors worked on is the multiple fitness genetic algorithm (MFGA).

At first, the authors quantify three major factors: microservices calling distance, resource utilization, and resource utilization balancing. As for the microservices calling distance, the authors formulated an equation (6) that helps determine the calls between containers and measures how many calls have been made between containers. The calling distance generated between containers should be as short as possible than across physical hosts to meet the users' needs.

$$\begin{cases} d(k_i,k_j) = \begin{cases} 1, calls\ across\ physical\ hosts \\ between\ containers\ i\ and\ j \\ 0, else \end{cases} \\ D_{ij} = \sum_{j=1}^{m-1} d(k_i,k_j) \\ D_i = D_{ij} + D_{ji} \\ D = \sum_{i=1}^{m} D_i \end{cases} \quad (6)$$

The next factor, which is resource utilization, is being introduced by the authors using the first equation (7) that defines the total number of physical hosts held by deploying containers with microservices. Then comes formula (8), which indicates the overall resource utilization rate of physical hosts that are triggered to deploy container microservices. It is necessary to occupy the shortest number of physical hosts to successfully use the computing resource and minimize energy consumption.

$$Z = \sum_{i=1}^{n} P_i \quad (7)$$

$$U = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{l=1}^{s} p_i \times k_{ij} \times r_{jl}}{\sum_{i=1}^{n}\sum_{l=1}^{s} p_i \times c_{il}} \quad (8)$$

Where k_{ij} indicates whether the container j is placed in host i or not, as for p_i specify if the host is activated or not. For r_{jl} and c_{il} are respectively the resources requested by microservices container and the type of resources that the host can supply. The authors worked with four types of resources which are CPU, memory, disk, and bandwidth.

The resource utilization balancing for a server is defined using equation (9), as is the resource utilization balancing for the entire edge cloud, as denoted in equation (9). The authors claim that the smaller the values, the more load is balanced.

$$\begin{cases} N_i = \sqrt{\frac{1}{s}\sum_{l=1}^{s}(u_{i,l} - u_i)^2} \\ N = \sqrt{\frac{1}{z}\sum_{i=1}^{s} p_i \sum_{l=1}^{s}(u_{i,l} - u_i)} \end{cases} \quad (9)$$

Where $u_{i,l}$ is the utilization of type l resources on host i, and u_i is the average utilization of all resources on the physical host i.

Using those three factors, the authors introduced the goal of this study under the formula of an objective function, as shown in equation (10).

$$maxAim(aim_1, aim_2, aim_3) = \max\left(\frac{1}{D}, \frac{U}{Z}, \frac{1}{N}\right) \quad (10)$$

After introducing the three factors, the authors combined those factors to come up with a solution to the scheduling problem. The first step in the MFGA algorithm is chromosome coding, where the code divides the containers into H groups using resource utilization, and then each group is allocated to a particular host. Next, the authors defined a fitness function, which has a value that indicates if the solution to the problem is weak or not. The function is calculated using weight parameters, which are microservice dependencies, resource utilization, and resource utilization balancing [10].

Afterwards, the gene evaluation function is created to evaluate the load balance of hosts. The authors mentioned that the function accelerates the algorithm's convergence as well as improves the performance of each machine. The function uses the same parameters as the previous function.

Another operation is generated by the authors, which is a crossover operation. The crossover operation uses the gene evaluation function on each host to speed up the convergence of the algorithm and also reach the crossover efficiency of the task set. For that purpose, the authors determined three main steps for this operator. The first step is to select the initial solutions, then exchange the most adaptable genes. Afterwards, it will delete the same microservices container from the new chromosome. The final step is to re-add the containers that are missing because of gene exchange using the fitness function and the gene evaluation function. The authors have grouped all the steps mentioned above to formulate the MFGA algorithm [10].

To evaluate their algorithm, the author used the tool CloudSim as well as three other algorithms: the round-robin algorithm (RR), the most-utilization first algorithm (MF), and the first come, first served algorithm (FCFS). At first, the authors made a comparison concerning the number of hosts occupied by the four algorithms, and they concluded that MFGA is the algorithm that used fewer hosts compared to the others. As for resource utilization, MFGA has the highest values among the four algorithms. The authors also made a comparison of microservices calling distance and concluded that MFGA has the smallest calling distance among the four algorithms, and the same result goes for resource utilization balancing. The authors came to the conclusion that the MFGA algorithm has better performance than the RR algorithm, the MF algorithm, and the FCFS algorithm [10].

The authors in [11] used another approach, which also consists of an optimization problem. The authors in this article worked on two algorithms: the first is a scheduling algorithm, and the second is an auto-scaling algorithm. Our concern is the scheduling algorithm proposed by the authors under the name Urgency-based Workflow Scheduling (UWS).

At first, the authors tried to formulate the scheduling problem into a task scheduling scheme as defined in (11). Afterwards, the authors defined the optimization probing the execution time and finish time when a task is allocated to a microservice instance, as well as calculating the lease start time, the lease finish time caused by deploying containers into the VM, and the earliest start time for executing a task in the microservice instance at a timestamp. Then the authors introduced the optimization problem, which is defined as minimizing the cost of VMs by meeting the deadline constraints of all requests using the formula (12).

$$M = \left\{ m_{i,j,k,l} \mid m_{i,j,k,l} = (t_i, WF_l, ms_{j,k}, ST(t_i, ms_{j,k})) \right\} \quad (11)$$

Where $m_{i,j,k,l}$ denotes that task t_i which belongs to the workflow WF_l is allocated to the instance $ms_{j,k}$ starting from the start time $ST(t_i, ms_{j,k})$.

$$\min cost$$
$$s.t. \quad rt_l \le D_l, \forall WF_l \quad (12)$$

Where $cost = \sum_{vm_x \in VM} price_x * \left\lceil \frac{duration_x}{interval} \right\rceil$ and $rt_l = max_{t_i \in WF_l}\{AFT(t_i)\} - T$.

Eventually, the authors presented the algorithm by calculating in the first place the cost-effective configuration (CE) of each type of microservice using the statistical information of the computation workload of tasks. Then, the urgency-based workflow scheduling algorithm UWS is performed based on the CE. Moreover, in the algorithm, the deadline allocates a sub-deadline for each task according to the CE. The scheduling urgency is being calculated in the urgency calculation, and tasks are being prioritized according to their scheduling urgency. The task mapping will select the proper service instance for each task in the order of priority. As a final step, the set contains all newly created service instances [11].

The authors introduced another metric beside the objective function, which is the success ratio, which defines the ratio between the number of workflows that meet the deadline and the overall number of scheduled workflows. The success ratio is presented under the following formula (13).

$$ratio = \frac{\sum_{WF_l \in WF} succ_i}{|WF|} \quad (13)$$

Where $succ_i = \begin{cases} 1 & rt_l \leq D_l \\ 0 & rt_l > D_l \end{cases}$.

To observe the performance of the algorithm, the authors tried to work with four different workflow applications: Montage, LIGO, GENOME, and SIPHT. The number of tasks is about 50 per workflow. Regarding the information about the workflows, it is given in DAX format files that have the name, computation workload, data transfer amount, and dependencies between tasks, as well as using 8 different types of VMs with different prices. The authors picked two workflow scheduling algorithms which are ProLiS and IC-PCPD2 to make a comparison between their algorithm and the two algorithms. Afterwards, the authors implemented the three algorithms into the four workflows and observed the variation of each metric. As for the success ratio, the authors concluded that this metric is increasing for the three algorithms; however, the UWS has the highest rank among the other two for all the workflow applications. On the other hand, the authors confirm that their algorithm has the superior value of finding a number of feasible solutions with different workflows. Concerning the cost, the authors observed that it has a lower value compared to the other algorithms, which makes it the most appropriate algorithm for scheduling microservices [11].

Another algorithm has been proposed in [7]. The authors propose a multi-objective optimization model that aims to solve scheduling problems. The objective here is to improve the system by reducing the network transmission overhead through microservices, balancing the load of clusters, and ameliorating the reliability of cluster services. At first, to reduce the network transmission overhead among microservices, the authors introduced three factors: the network distance between nodes, the number of requests between microservices, and the quantity of data transmission. The authors utilized the equation (14) to calculate data transmission overhead.

$$COMM(X) = \sum_{j=1}^{n}\sum_{i=1}^{m}\frac{x_j}{Scale_i}\sum_{l=1\wedge l\neq j}^{n}\sum_{ms_k\in CON\_SET_l}\frac{x_{k,l}}{Scale_k}Link(ms_i, ms_k) * * Trans(ms_i, ms_k) * Dist(pm_j, pm_l) \quad (14)$$

The authors moved on to load balancing the system, where they presented equation (15) that helps in maximizing the resource utilization rate within the nodes, in which the resource utilization reflects the load balancing of the system. This means an unbalanced resource load with high resource utilization will lead to the worst load within the system.

$$RESRC_{CONS(X)} = \frac{1}{\sigma_1 + \sigma_2} max_{1\leq j\leq n} \max\left(\sum_{i=1}^{m} x_{i,j}\frac{Link_i \times Cal_{Reqst_i}}{Scale_i \times Cal_{Reqst_j}}\sigma_1, \sum_{i=1}^{m} x_{i,j}\frac{Link_i \times Cal_{Reqst_i}}{Scale_i \times Cal_{Reqst_j}}\sigma_2\right) \quad (15)$$

The other factor is request failure within the cluster, where the authors tried to calculate the average number of request failures to evaluate the cluster services using equation (16).

$$LINKFAIL(X) = \sum_{j=1}^{n}\sum_{i=1}^{m} Fail_j \times x_{i,j}\frac{Link_i}{Scale_i} \quad (16)$$

The authors used those three factors to present their multi-objective function under the constraints of resource capacity and microservice deployment requirements using the formula (17).

$$minimize\ COMM(X)$$

$$minimize\ RESRC\_CONS(X)$$

$$minimize\ LINK\_FAIL(X)$$

$$s.t. \quad \sum_{i=1}^{m} x_{i,j}\frac{Link_i}{Scale_i}Cal\_Rest_i \leq Cal\_Rest_j \quad \forall pm_j \quad (17)$$

$$\sum_{i=1}^{m} x_{i,j}\frac{Link_i}{Scale_i}Str_{Rest_i} \leq Str_{Rest_j} \quad \forall pm_j$$

$$x_{i,j} = \begin{cases} 1 & if\ ms_i \in alloc(mp_j) \\ 0 & if\ \ ms_i \notin alloc(mp_j) \end{cases}$$

$$\sum_{j=1}^{n} x_{i,j} = Scale_i, \quad \forall ms_i$$

$$\sum_{i=1}^{m} x_{i,j} = 1 \quad \forall pm_j$$

Afterwards, the authors explained how the ant colony optimization algorithm works. The latter is performing the feeding process of an ant to help schedule microservices, in which the algorithm applies several steps to reach the goal. The first step is placing a variable ant into the microservice, and then the ant chooses a path with a definite probability to attain the node that satisfy the constraints of the model. The allocation of microservices is linked to the number of containers in the cluster, and if the node that was selected is different each time, then the microservice will be put in the tabu list. As for the next step, the ant will return to the next microservice and perform the second step again. Eventually, the ants will complete allocating all the microservices, and the algorithm finishes when the maximum number of iterations is reached [7].

To compare the algorithms mentioned above, the authors took three related algorithms for scheduling: Multiopt, GA_MOCA, and Spread algorithm where each of those algorithms is a multi-objective container scheduling that considers different factors such as CPU usage, threshold distance, memory usage, balanced use of resource utilization, and so on. The authors made a comparison between the four algorithms using three factors: network transmission, cluster load balancing, and reliability of services. As for the network transmission overhead result with different numbers of user requests, the authors algorithm shows the best outcome took entire consideration of the network data transmission between microservices as well as network distance among nodes and optimized the scheduling using the ant colony algorithm. Concerning the result for resource load in the cluster, the algorithm proposed by the authors has the best performance in terms of resource utilization of nodes and load distribution of each resource. At last, the result of the reliability of the cluster

is being measured by the number of failures for microservices requests, where the algorithm made by the authors has the best performance among the other algorithms in view of the fact that the algorithm utilized the average number of failures to allocate the microservices with more requests to the nodes with lower for improving the reliability of the cluster [7].

IV. MICROSERVICES SCHEDULING ALGORITHMS COMPARAISON

The container scheduling problem has become a major obstacle to effectively managing runtime cloud resources. The examination of microservice scheduling techniques has led to the conclusion that not all the algorithms can address the factor performance of the cluster; consequently, many challenges still remain to be solved as research opportunities. After introducing the scheduling algorithms in the previous section, we can observe that for each algorithm, it has its own scheduling factors that can vary from one algorithm to another, as well as each algorithm has its own parameter settings. However, the algorithms tried to solve a specific optimization problem using an objective function to properly improve the cluster. The first article tried to solve an optimization problem by maximizing the profit of using the memory and CPU of the resource, which leads to better throughput. As for the second article, they tried to maximize a multi-objective function in order to increase the chances of resource utilization, as well as adjust the load balancing of the cluster and the calling distance between microservices. The other multi-objective problem is concerned with the minimization of the cost and, on the other hand, the increment of the success ratio in the cluster. The last algorithm worked on minimizing three objective functions, which are network transmission, resource utilization, and the number of failures for microservice requests. As cited, each algorithm works with different multi-objective functions that have a different purpose in finding the ultimate solution.

V. CONCLUSION

The utilization of containers has become the focus of attention lately, and for that reason, many researchers are trying to find an efficient solution to various problems that prevent the improvement of the application. In this article, we introduce a comprehensive survey concerning microservice scheduling techniques. At the beginning, we tried to classify the scheduling techniques into four categories, and then we examined the most common performance metrics used by researchers. Afterwards, we presented four different algorithms that work with a multi-objective optimization problem, yet each one of the algorithms has its own objective function to work with. Eventually, we made a comparison between the four algorithms to observe the similarities between them. We confirm that this survey is intended to provide a future perspective regarding ameliorating the usage of applications within the cloud computing community.

VI. REFERENCES

[1] Mohammed Khatiri, 26 Nov 2020, Task scheduling on heterogeneous multi-core, pp. 87-96.

[2] Haihua Gu, Xiaoping Li, Muyao Liu, Shuang Wang,22 July 2021, Scheduling method with adaptive learning for microservice workflows with hybrid resource provisioning.

[3] Raja Mubashir Munaf, Jawwad Ahmed, Faraz Khakwani and Tauseef Rana, 2019, Microservices Architecture: Challenges and Proposed Conceptual Design.

[4] Markos Viggiato, Ricardo Terra, Henrique Rocha, Marco Tulio Valente, September 2018, Microservices in Practice: A Survey Study

[5] Işıl Karabey Aksakalli, Turgay Çelik, Ahmet Burak Can, Bedir Tekinerdoğan, 2021, Deployment and communication patterns in microservice architectures: A systematic literature review.

[6] Gabriel Araújo, Arthur Sabino, Luiz Lima, Vandirleya Costa, Carlos Brito, Paulo Rego, Iure Fé, Francisco Airton Silva, Energy Consumption in Microservices Architectures: A Systematic Literature Review.

[7] MIAO LIN, JIANQING XI, WEIHUA BAI, AND JIAYIN WU, 24 June 2019, Ant Colony Algorithm for Multi-Objective Optimization of Container-Based Microservice Scheduling in Cloud.

[8] Imtiaz Ahmad, Mohammad Gh. AlFailakawi, Asayel AlMutawa, Latifa Alsalman, 11 March 2021, Container scheduling techniques: A Survey and assessment.

[9] Hamid Mohammadi Fard, Radu Prodan, Felix Wolf, 30 December 2020, Dynamic Multi-objective Scheduling of Microservices in the Cloud.

[10] Yanjun Shi, Yijia Guo, Lingling Lv and Keshuai Zhang, 15 October 2020, An Efficient Resource Scheduling Strategy for V2X Microservice Deployement in Edge Servers.

[11] Sheng Wang, Zhijun ding, Changjun Jiang, 2020, Elastic Scheduling for Microservice Applications in Clouds.