IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 3, May 2024

Optimizing Service Placement and Enhancing Service Allocation for Microservice Architectures in Cloud Environments

Roshan Mahant[1], Sumit Bhatnagar[2], Vikas Mendhe[3]
Launch IT Corp, Urbandale, IA USA[1,3]
JP Morgan Chase & Co., New Jersey, USA[2]

Abstract: *With the increasing popularity of microservice architecture, there is a growing need to deploy service-based applications efficiently in cloud environments. Traditional cluster schedulers often fail to optimize service placement adequately, as they only consider resource constraints and overlook traffic demands between services. This oversight can lead to performance issues such as high response times and jitter. To address this challenge, we propose a novel approach to optimize the placement of service-based applications in clouds. Our approach involves partitioning the application into segments while minimizing overall traffic between them, and then strategically allocating these segments to machines based on their resource and traffic demands. We have developed a prototype scheduler and conducted extensive experiments on test bed clusters to evaluate its performance. The results demonstrate that our approach surpasses existing container cluster schedulers and heuristic methods, significantly reducing overall inter-machine traffic and improving application performance.*

Keywords: Microservice architecture, Cloud computing, Service-based applications

I. INTRODUCTION

Microservice architecture is a relatively recent trend that is rapidly gaining popularity in the field of application development. The architecture in question is gaining popularity dueto the fact that it improves the overall scalability and robustness of the system, it reduces complexity by making use of lightweight and modular services and provides flexibility to embrace a wide range of technologies. A strategy for designing a single application as a collection of small services, each of which operates in its own process and communicates with lightweight mechanisms (for example, HTTP resource API), is referred to as the microservice architectural style. This is according to the definition of microservices, which can be found at https://martinfowler.com/tags/microservices.html. As a result, the application is made up of a number of services, which are referred to as service-based applications. These services collaborate with one another to deliver intricate functionality. A significant number of developers have the intention of transforming old monolithic programs into service-based applications. This is because of the benefits that microservices architecture will provide. For example, an application for online shopping may be essentially segmented into product service, cart service, and order service. This configuration has the potential to significantly enhance the application's productivity, agility, and resilience. On the other hand, it also presents difficulties. When setting up a cloud-based service-based application, the scheduler has to carefully plan when each service, which may have different resource needs, will run on each of the distributed compute groups. Also, the network connection between the different services needs to be well managed, since the way they talk to each other has a big effect on the quality of service (for example, how fast a service responds). Because of this, it is becoming more and more important to make sure that service-based applications work the way they should, especially when it comes to how well the networks between the services work.

A lot of complicated, distributed services are used in service-based applications, which usually need more computing power than a single machine can provide. A group of connected computers or cloud computing platforms are often used to run service-based apps. An example of one of these platforms is Amazon Elastic Compute Cloud (EC2), which can be found at https://aws.amazon.com, Microsoft Azure, which can be found at https://azure.microsoft.com, or Google Cloud Platform, which can be found at https://cloud.google.com. What's more, containers are quickly becoming the

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 3, May 2024

cuttingedge technology that effectively hides the running environments of software components and services. This makes it much easier to move apps between cloud environments and makes them work better. In order to successfully install a service-based application in the cloud, it is necessary to take into consideration a total of numerous crucial criteria. To begin, the application's services frequently have a variety of resource requirements, including those for the central processing unit (CPU), memory, and disk space. At the same time as it is responsible for providing unified functionalities, the underlying machine must guarantee that there are sufficient resources to execute each service. It is difficult to allocate resources to each service in an efficient manner, and the difficulty of this task increases when the cluster is comprised of computers that are different from one another. Second, the application's services frequently have traffic needs among them as a result of data exchange, which necessitates careful handling of the situation. As a result of the fact that the response time of a service is directly influenced by the traffic condition associated with that service, poor management of the traffic needs can result in severe performance degradation. Taking into account the traffic demands, a solution that is obvious is to place the services that have big traffic demands among them on the same machine. This can accomplish intra-machine communication and limit the amount of traffic that occurs between machines. Nevertheless, due to the limited resource capacity of the computer, it is not possible to co-locate all of these services on each other. Therefore, the placement of applications that are dependent on services is extremely hard in cloud environments. Cluster schedulers are required to properly position each service of a service-based application in relation to the resource demands and traffic demands of the application in order to achieve the desired level of performance for the application.

Over the last decade, the number of public cloud providers has increased significantly. It is now possible to rent resources located in one or more data-center. Such an infrastructure can satisfy the needs of a large number of users, but it does have the downside that data must be moved (and often stored) in these shared data-centers, raising doubts about the privacy of the data and, potentially, its ownership (for instance, in the case of providers going out of business).

Data-centers are not alone in taking advantage of technological progress: user devices, including personal computer and set-top boxes, have ever increasing computing and storage capabilities. Computer networks offer higher data rates to the wider public, with the spread of Fiber to the Home (FTTH) solutions and high speed wireless networks. Thanks to all this, it is conceivable for the devices hosted on the user premises to offer services to their owners, even remotely. Such a solution has the added advantage of addressing the data privacy and ownership issues mentioned above. An obvious shortcoming of such an approach is that it introduces two single points of failures, namely the user device and the corresponding network connection.

Either one can fail, rendering the service unavailable remotely. While modern hardware is reasonably reliable, such a solution is no match for the public cloud providers that can offer highly available services thanks to their redundant infrastructure. A possible solution to overcome these limitations is for several users to cooperate, by pooling their individual resources to form a larger (redundant) system. A few devices (for instance between 10 and 20) can already be enough to significantly increase the availability and, potentially, the performance with respect to a single device. As an example, let us consider a sport club, whose members would like to share pictures and other materials related to the club activities. One member decides to use the storage space available on one of his devices connected at home to let other members store and retrieve their photos. When comparing this solution with the service provided by a public platform, two limitations stand out. First, as these devices are provisioned for individual usage, they might not have enough local resources (CPU, storage, network) for a larger number of users. In our example, the storage space and the residential network will become bottlenecks as members of the club are uploading and browsing more photos. The second limitation comes from the lower availability of these on-premise devices. The photo-sharing storage will be unavailable anytime the device is powered off or experiencing a network outage. If multiple members of the sport club are willing to share their devices and network connectivity, they could significantly improve the situation, provided they have a large enough variety of network providers and usage patterns. Extended functionalities like generating thumbnails and presenting them in a web gallery can be offered to the community as the different devices contribute to the application with their computing and networking resources. Members of the community will fully benefit from the cooperation between the devices as the application can be distributed over these devices. Many frameworks for distributing applications over user-provided devices have been already proposed, such as Cloud@Home[6], Nebulas[4], Community Cloud[13] or CNMC[17]. These solutions aim at creating an Infrastructure-as-a-Service (IaaS) interface on

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 3, May 2024

top of the user provided devices where applications embedded inside virtual machines can be deployed and managed the same way as in a data-center. Such solutions however tends to be complex as they have to adapt to devices and networks which are much more heterogeneous and with a lower availability rate than the infrastructure of a data-center

II. PROBLEM STATEMENT

The problem addressed in this paper is the suboptimal placement of service-based applications in microservice architectures within cloud environments. Traditional cluster schedulers often overlook the intricate balance between resource constraints and traffic demands between services, resulting in performance inefficiencies like high response times and jitter. To enhance system efficiency and scalability, there is a critical need to minimize inter-machine communication overhead while strategically allocating services based on their resource requirements and traffic patterns. This paper aims to tackle these challenges by proposing novel partition and packing algorithms to optimize resource allocation and reduce communication overhead, ultimately improving the overall performance of service-based applications in cloud environments.

Table 1. Notation and Description.

Notation	Description
M	Set of heterogeneous machines in the cluster: $M=\{m1,m2,...,mM\}$
M	Number of machines: (M =
R	Set of resource types: $R=\{r1,r2,...,rR\}$
R	Number of resource types: (R =
Vi	Vector of available resources on machine $Vi=(v1i,v2i,...,vRi)$
vji	Amount of resource *rj* available on machine *mi*
S	A service-based application composed of a set of services: $S=\{s1,s2,...,sN\}$
N	Number of services in the application: (N =
Di	Vector of resource demands of service $Di=(d1i,d2i,...,dRi)$
dji	Amount of resource *rj* that service *si* demands
T	Matrix of communication traffic between services: $T=[tij]N\times N$
tij	Traffic rate from service *si* to service *sj*
X	A placement solution: $X=[xij]N\times M$, where $xij=1$ if service *si* is placed on machine *mj*, otherwise $0xij=0$

III. MODEL DESCRIPTION

In our model, we consider a cloud computer cluster composed of a set of heterogeneous machines $M=\{m1,m2,...,m_M\}$, where $|M=|M|$ is the number of machines. We also consider R types of resources $R=\{r_1,r_2,...,r_R\}$ available on each machine, such as CPU, memory, and disk. For each machine m_i, we represent its available resources as a vector)

$Vi=(v1i,v2i,...,vRi)$,

Wher $ev_{jidenotes}$ the amount of resource rjavailable on machine m_i.

In the Infrastructure as a Service (IaaS) or Container as a Service (e.g., Amazon ECS) models, users specify the resource demands of virtual machines (VMs) or containers when submitting deployment requests. Thus, the resource demands are known upon the arrival of service requests. We consider a service-based application composed of a set of services $S=\{s_1,s_2,...,s_N\}$ to be deployed on the cluster, where $|N=|S|$ is the number of services. For service si, we represent its resource demands as a vector $Di=(d_{1i},d_{2i},...,d_{Ri})$, where dj_i denotes the amount of resource r_j that service s_i demands. We denote the traffic between services as a matrix $T=[t_{ij}]N\times N$, where t_{ij} represents the traffic rate from service s_i to service s_j.A placement solution is modeled as a 0-1 matrix $X=[x_{ij}]N\times M$. If service $s_i$is deployed on machine $x_{ij}=1$; otherwise, $x_{ij}=0$.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 3, May 2024

To achieve the desired performance of service-based applications, our objective is to minimize the overall traffic between services placed on different machines while considering their multi-resource demands. This objective is crucial because network performance directly influences the overall performance of data-intensive services, especially when frequent data transfers occur between services. Given the traffic situation, our goal is to minimize inter-machine traffic by strategically placing services to mitigate network latency or congestion. Although placing services with high traffic rates on the same machine can improve network performance, resource constraints limit the feasibility of this approach. Therefore, we aim to find a placement solution that minimizes the overall traffic between services placed on different machines while satisfying their resource demands. We formulate our objective as follows: Minimize the sum of the traffic rates between pairs of services placed on different machines:

$$\sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{P=1}^{M}\sum_{q=1,q/p}^{M} tij.xip.xjq$$

Subject to the following constraints:

Each service must be placed on exactly one machine:

$$\sum_{J=1}^{M} xij \;=\; 1 \quad 1(\forall i \in \{1,2,\ldots,N\})$$

The resource demands of services placed on a machine must not exceed its resource capacities:

$$\sum_{i=1}^{N} xij \cdot dki \;\leq\; vkj\ (\forall j \in \{1,2,\ldots,M\}, \forall k \in \{1,2,\ldots,R\})$$

Binary decision variables for service placement:

$$x_{ij} \in \{0,1\} (\forall i \in \{1,2,\ldots,N\}, \forall j \in \{1,2,\ldots,M\})$$

This objective seeks to minimize the overall traffic between services on different machines while ensuring that each service is placed on a machine that can satisfy its resource demands.

Placement Algorithm

In this section, they will discuss the algorithms that we have designed and proposed for this study. The purpose of our algorithms is to locate a placement solution that will minimize the amount of traffic that occurs between machines while simultaneously satisfying the requirements of many resources. (1) Application partitioning based on contraction methods, (2) heuristic packing with traffic awareness, and (3) placement finding with threshold adjustment are the three principal components that comprise the design of our methodology.

Application Partition

To begin by normalizing the quantity of accessible resources on machines and the resources that services demand to be a fraction of the maximum ones. This is done in order to accomplish the goal of making the values of various resources similar to one another and easy to manage. It is our understanding that the term vmax−j refers to the greatest quantity of resources rj that are accessible on a system.

$$v_{max-j} = \max_{i \in \{1,2,\ldots,M\}} \left(v_i^j\right)$$

i

Then the vector V_i of available resources on machine m_i and the vector D_i of resource demands of service s_i are normalized as:

$$V_i \;=\; \frac{v_i^1}{v_{max-1}}, \frac{v_i^2}{v_{max-2}}, \frac{v_i^R}{v_{max-R}}$$

$$D_i \;=\; \frac{d_i^1}{v_{max-1}}, \frac{d_i^2}{v_{max-2}}, \frac{d_i^R}{v_{max-R}}$$

Then begin partitioning the service-based application once the normalization process is complete. First and foremost, they inquire about the number of various components that make up the application. In order to identify the number of parts when executing partition algorithms, they provide a threshold α. The demands placed on many resources by a variety of services have led to the establishment of this threshold. The threshold represented by the symbol α represents the upper limit of the resource demands of partitioned portions. In order to ensure that the aggregate resource demands from each part do not exceed α or that no distinct component serves more than one service, it is necessary for us to

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 3, May 2024

carry out partition algorithms in a continuous manner. Using a threshold α that falls within the range of 0 to 1, considering that the resource demands have been normalized, it ensures that every part that has been partitioned can be packed into a machine simultaneously. Presented in Figure 3 is an illustration of an application partition that has a threshold of α equal to zero. Each component's overall demands for both the central processing unit (CPU) and memory do not exceed 0.5, as shown in the figure. The binary partition and the k partition are two partition algorithms that are present, both of which are based on the contraction method. These algorithms are given a threshold α.

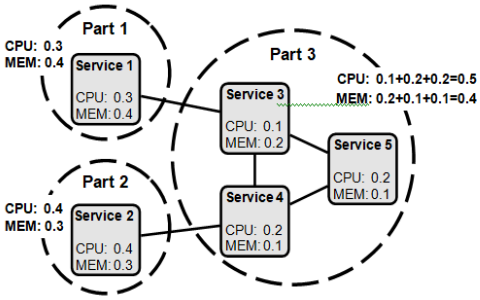

Figure 1. Application partition with threshold α = 0.5.

Binary Partition

The "Binary Partition" and "K Partition" algorithms both aim to partition a service-based application into smaller sections based on resource requirements and a given threshold.

Binary Partition Algorithm:

Initialization:

Start with the entire application represented as a single partition, denoted as P = {S}.

Iterative Partitioning:

Continuously check the resource requirements of each component connected to the current partition (P).

If a part (Si) in the partition exceeds the resource threshold (α) and contains more than one service, it undergoes binary partitioning.

Graph Construction and Contraction:

Build a graph (G = (V, E)) based on the part Si, where nodes represent services and edge weights indicate traffic rates.

Repeatedly perform the contraction method to find a minimum cut (Gmin) in the graph, ensuring efficient partitioning.

Partition Adjustment:

Divide the part Si into two pieces ({Sx, Sy}) according to the minimum cut (Gmin).

Repeat this process until the resource demands from each component do not exceed the threshold α or until no component contains more than one service.

K Partition Algorithm:

Initialization:

Start with the entire application represented as a single partition, denoted as P = {S}.

Iterative Partitioning:

Continuously check the resource requirements of each component within the current partition (P).

If a part (Si) in the partition exceeds the resource threshold (α) and contains more than one service, increase the number of partitions (k).

Graph Construction and Contraction:

Build a graph (G = (V, E)) based on the application S.

Perform the contraction method multiple times (n^2k−2 log n) to achieve a low k-cut with high probability.

Partition Adjustment:

Divide the application into k sections ({S1, S2,..., Sk}) based on the minimum k-cut (Gmin) obtained from the contraction algorithm.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 3, May 2024

Repeat this process until the resource demands from each component do not exceed the threshold α or until no component contains more than one service.

The Heuristic Packing approach aims to efficiently allocate each component of the application onto heterogeneous computers, given a partition of the application. Since the problem resembles a multi-dimensional bin packing problem, which is known to be NP-hard, finding the optimal solution within a polynomial time frame is not feasible, especially when dealing with a significant number of services.

Greedy Heuristics Used:

Traffic Awareness Heuristic: Prioritizes machines based on minimizing traffic rates between services.

Most-Loaded Heuristic: Prioritizes machines based on their load situation, aiming to improve resource efficiency by packing components onto the most loaded machines.

Algorithm Overview:

Matching Factors Calculation:

For each part (Si), calculate two matching factors:

tf (traffic factor): Sum of traffic rates between services in part Si and services already packed into machine mj.

ml (load factor): Scalar value indicating the load situation between the resource demand vector of part Si and the available resources on machine mj.

Heuristic Prioritization:

Initially prioritize machines based on tf to minimize traffic between machines.

If tf factors are identical, prioritize machines based on ml to improve resource efficiency.

Placement Solution:

If it's determined that all components of the partition can be packed into machines, return the placement solution.

Otherwise, return null, indicating that not all components could be packed.

Placement Finding

Algorithm 5: Threshold Determination

Initialization:

Set the initial threshold value α to 1.0.

Define a step value Δ (default is 0.1).

Threshold Adjustment:

Iterate from a large threshold value towards smaller values.

In each iteration:

Partition the given application S using either the binary partition or the k partition algorithm with the current threshold α.

Save the most recent partition results to prevent duplicates.

Placement Solution Attempt:

For each partition obtained:

Attempt to pack every component of the partition into machines using the heuristic packing technique.

Threshold Adjustment Logic:

Starting with a higher threshold α result in fewer partition sections and potentially fewer traffic rates between them.

The algorithm iterates from large to small thresholds, likely aiming to find a balance between partition granularity and traffic minimization.

IV. EXPERIMENTAL METHODOLOGY

Cluster: For the purpose of conducting tests, we establish two distinct testbed clusters in ExoGENI. They employ thirty virtual machines (VMs) that are identical to one another, each of which has two CPU cores and six gigabytes of random access memory (RAM). The second cluster is comprised of ten virtual machines (VMs) with two CPU cores and six gigabytes of random access memory (RAM), and ten VMs with four CPU cores and twelve gigabytes of RAM. There

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 3, May 2024

are 30 virtual machines (VMs) in the homogeneous cluster and 20 VMs in the heterogeneous cluster, but the total resource capacity is the same for both clusters.

Workloads: When conducting the tests, they make use of synthetic applications so that we may evaluate the offered methods in a variety of diverse settings. Taking into account the size of the testbed cluster, we are able to produce service-based applications that are made up of 64, 96, and 128 services respectively. The CPU demand of each service is evenly selected at random from the range [30,100], where 100 represents one CPU core and the memory demand is selected at random from the range [100,300], where 100 represents one gigabyte of random access memory (RAM). This is done for the size of 64. For a size of 96, the need for the central processing unit (CPU) is selected at random from the range [20,67], and the need for memory is selected at random from the range [67,200]. In the case of a size of 128, the demand for the central processing unit (CPU) is selected at random from the range [15,50], and the demand for memory is selected between [50,150]. To the extent that these ranges are accurate, the total resource requirements of various application sizes are roughly equivalent to one another. Ten thousand instances are generated for testing purposes for each and every application size. For the purpose of ensuring that the application graph is connected, everyone has come to the conclusion that the traffic needs between services will be generated with a chance of 0.05. Additionally, the traffic rate will be distributed according to a log-normal distribution, with the mean being 5 Mbps and the standard deviation being 1 Mbps. The work [14] reveals that the log-normal distribution offers the best match to the traffic in the data center, which is the reason why this option was chosen.

V. IMPLEMENTATION

In the initial scheduler's implementation, two distinct configurations emerge from the proposed algorithms: BP-HP (Binary Partition - Heuristic Packing) and KP-HP (K Partition - Heuristic Packing).The BP-HP configuration relies on the Binary Partition algorithm for partitioning the application and subsequently utilizes the Heuristic Packing algorithm to efficiently allocate resources to each partition. This approach emphasizes simplicity and efficiency, making it suitable for applications where straightforward partitioning and practical resource allocation are prioritized.

The two approaches that we provided for application partitioning resulted in the existence of two different configurations. Binary partition (BP) and heuristic packing (HP) are the foundations upon which BP-HP is built. Historical packing and k partition (KP) are the foundations of the KP-HP algorithm.

Baselines: The composite Software as a service (SaaS) placement problem has been the subject of a significant amount of research efforts [6,15], as was indicated earlier. Nevertheless, they aim to achieve placement for a particular group of predetermined service components on the market. The most significant thing to note is that the generation of a placement solution using these metaheuristic-based approaches typically takes minutes or even hours, especially for large-scale clusters. This is something that would be impossible to accomplish using an online response. The problem of traffic-aware virtual machine deployment is the subject of another research work [16,17]. On the other hand, existing solutions are dependent on a particular network topology, whereas our technique is not dependent on the topology of the network. As a result, they have decided to evaluate our scheduler in comparison to the following schemes:

Kubernetes Scheduler (KS): In order to maintain a balanced utilization of the cluster's resources, the default scheduler in Kubernetes [18] container cluster has a tendency to distribute containers evenly across the cluster. To be more specific, we add a soft affinity, which is also known as pod affinity in Kubernetes, to the services that have traffic between them. This is done because the scheduler would attempt to deploy the services that have affinity between them on the own machine.

First-Fit Decreasing (FFD): it is a simple and commonly adopted algorithm for themulti-dimensional been packing problem [19]. FFD operates by first sorting the services in decreasing order according to a certain resource demand and then packs each service into the first machine with sufficient resources.

Best-Fit Decreasing (BFD): it places a service in the fullest machine that still has enough capacity.BFD operates by first sorting the machines in decreasing order according to a certain resource capacity and then packs each service into the first machine with sufficient resources. As an example, consider the Multi-resource Packer (PACK) heuristic [4]. Its basic premise is to schedule services in a way that maximizes the dot product of service resource demands and machine resource availability.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 3, May 2024

Random (RAND): it randomly picks a service in the application and then packs it into the first machine with sufficient resources.

Comparison with Baselines

Figure 4 illustrates the successful placement ratio of various strategies across two distinct clusters. This ratio reflects the proportion of successfully placed applications out of the total number of applications requested, indicating the effectiveness of each algorithm in identifying placement solutions for all involved services. Among the strategies, RAND exhibits the poorest performance due to its lack of a heuristic for service packing. Conversely, KS prioritizes balancing resource consumption across the cluster, while PACK aims to align resource demands with availability. FFD and BFD outperform KS and PACK, attributed to their success in multi-dimensional bin packing problems. Comparatively, BP-HP and KP-HP perform similarly well, marginally outperforming other strategies. This is largely attributed to their iterative partitioning and packing approach with varying thresholds, which increases the likelihood of discovering placement solutions. The most-loaded heuristic employed in the packing algorithm facilitates compact service delivery. In the homogeneous cluster, the successful placement ratio increases with the quantity of services provided. This phenomenon arises because while the total resource demands of applications remain consistent across different sizes, smaller applications exhibit higher resource demands per service, exacerbating resource fragmentation issues.

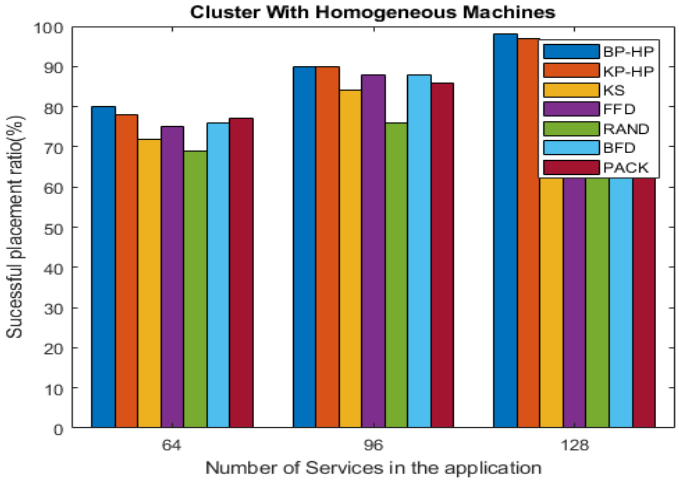

Figure 2. Comparison of successful placement ratio of different scheme in homogeneous machine

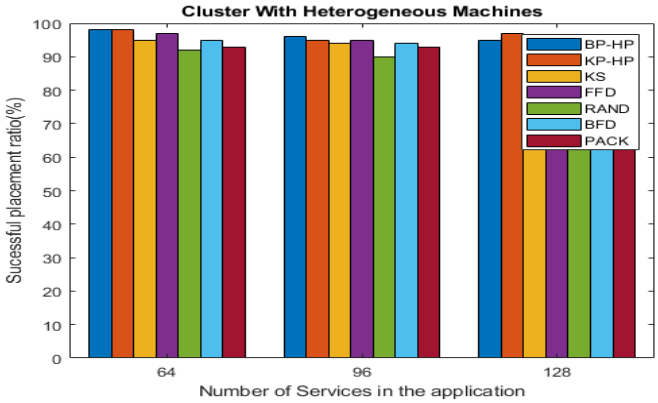

Figure 3. Comparison of successful placement ratio of different schemes in heterogeneous machine

Figure 3 and Figure 4 present the average ratio of co-located traffic for each of the alternative methods, with error bars indicating the highest and lowest possible comparisons. This ratio denotes the amount of traffic occurring between

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 3, May 2024

services deployed on the same machine relative to the total traffic. Higher co-located traffic ratios signify more effective placement solutions in reducing inter-machine traffic.

Table 2 presents the co-located traffic ratio (%) of various schemes in both homogeneous and heterogeneous clusters. For the homogeneous cluster, BP-HP demonstrates the highest average co-located traffic ratio at 47.2%, followed closely by KP-HP at 44.2%, indicating effective placement solutions. In contrast, traditional schemes like KS, FFD, BFD, PACK, and RAND exhibit significantly lower average ratios ranging from 22.1% to 9.4%. The variability in ratios is evident across schemes, with BP-HP and KP-HP showing a wider range of values compared to the other methods. In the heterogeneous cluster, BP-HP maintains a higher average ratio of 49.1% compared to KP-HP's 24.2%, indicating their effectiveness in handling diverse resource constraints. Overall, BP-HP consistently outperforms other schemes in cluster types, highlighting its efficacy in reducing inter-machine communication and optimizing resource allocation in microservices architectures.

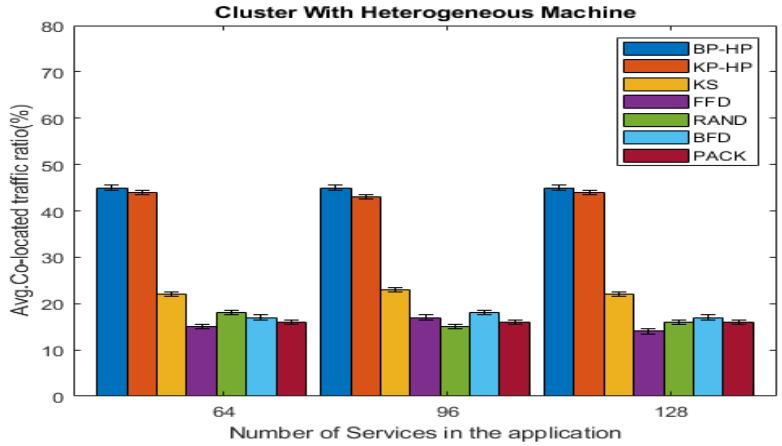

Figure 4. Comparison of average co-located traffic ratio of different schemes in homogeneous machine

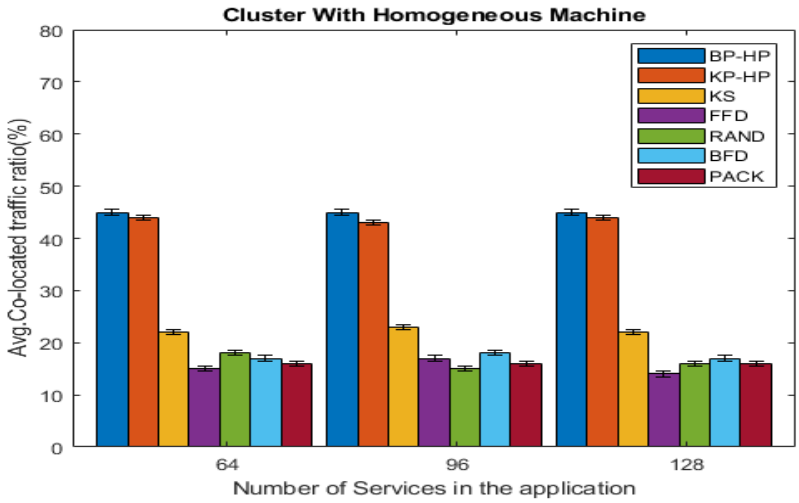

Figure 5. Co-located traffic ratio (%) of different schemes in heterogeneous machine

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 3, May 2024

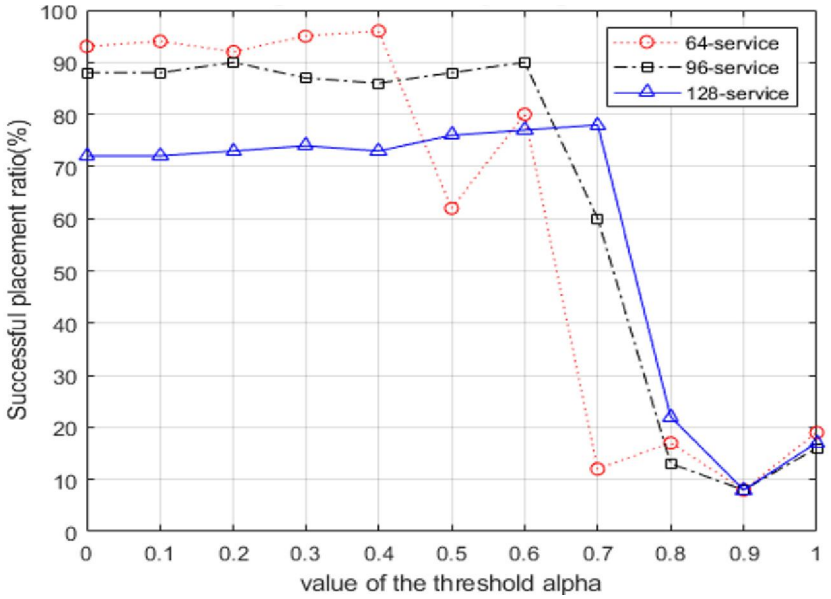

Figure 6. Illustrates the successful placement ratio on the homogeneous cluster when employing the BP-HP algorithm with varying threshold values for α.

Table 2 displays the percentage of co-located traffic for various schemes

Techniques	Homogeneous		Cluster	Heterogenous		Cluster
	Avg.	Min	Max	Avg.	Min.	Max
BP-HP	47.2	34.1	55.8	49.1	36.8	61.4
KP-HP	44.2	29.7	58.2	24.2	34.2	60.2
KS	22.1	16.2	30.1	9.8	5.1	31.7
FFD	9.9	5.5	14.7	10.2	6.2	16.1
BFD	9.8	5.2	16.1	9.7	6.9	15.4
PACK	9.4	5.2	15.2	9.4	5.9	15.4
RAND	9.5	5.1	15.6	5.4	5.7	15.7

Impact of Threshold α

This section discusses the impact of the threshold α on the placement of service-based applications. To illustrate, the threshold α is held constant while applying the BP-HP algorithm to a cluster containing homogeneous machines. Figure 6 illustrates the successful placement ratio for various α values. For instance, when α is set to 0.5, BP-HP achieves a placement solution for 77% of applications comprising 64 services. However, as α increase, the successful placement ratio tends to decrease. Beyond α values exceeding 0.7, only a limited number of applications can be effectively placed. The observed trend can be attributed to the relationship between α and the partitioning process. Higher α values lead to fewer parts in the partition, resulting in larger average resource demands per partition. Consequently, packing these components into machines with multiple resource constraints becomes more challenging. Figure 7 further illustrates the impact of α on the average co-located traffic ratio, with error bars representing the range of ratios. This visualization demonstrates that the co-located traffic ratio tends to increase as α becomes larger. However, higher α values also pose challenges in terms of application pack ability.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 3, May 2024

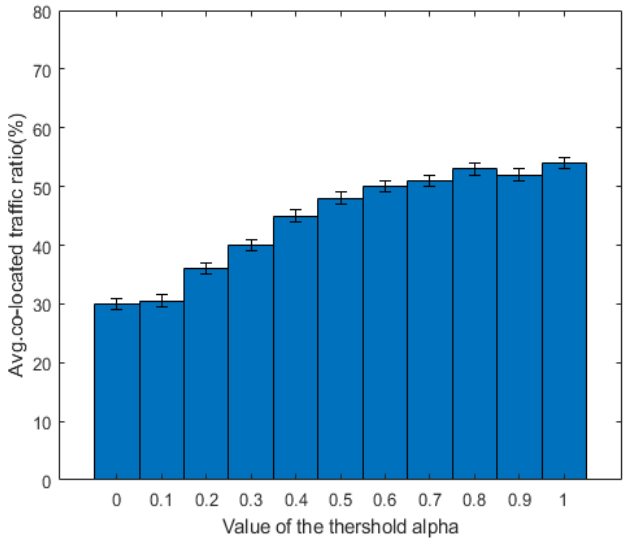

Figure 7. Average co-located traffic ratio on the homogeneous cluster by using BP-HP with different values of threshold α.

VI. OVERHEAD EVALUATION

The estimation of overhead is conducted by measuring the algorithm's runtime and comparing it with results from KS and RAND algorithms. A Python implementation of the KS scheduling algorithm is provided for comparative purposes. The experiments are executed on a dedicated server equipped with an Intel Xeon E5-2630 2.4 GHz CPU and 64 GB of RAM.For the heterogeneous cluster, Figure 8 illustrates the average algorithm runtime for various schemes, with error bars representing the maximum and minimum runtimes. RAND exhibits low overhead due to its straightforward nature, while KS is slightly more complex as it handles service affinity and employs predicated rules and prioritization policies. Baseline algorithms are simpler compared to BP-HP and KP-HP, which are more sophisticated and consequently have larger overheads. Moreover, the algorithm's runtime is significantly influenced by the threshold α value, with a substantial difference observed between maximum and minimum runtimes. Higher α values result in fewer iterations, while lower α values lead to more iterations. However, both BP-HP and KP-HP are able to provide solutions within seconds for a variety of requirements, making them suitable for online scheduling, especially for applications with fewer than one hundred services. Additionally, the application partitioning step is identified as the most time-consuming component, suggesting that the algorithm's runtime for large-scale clusters with the same number of services would not significantly differ.

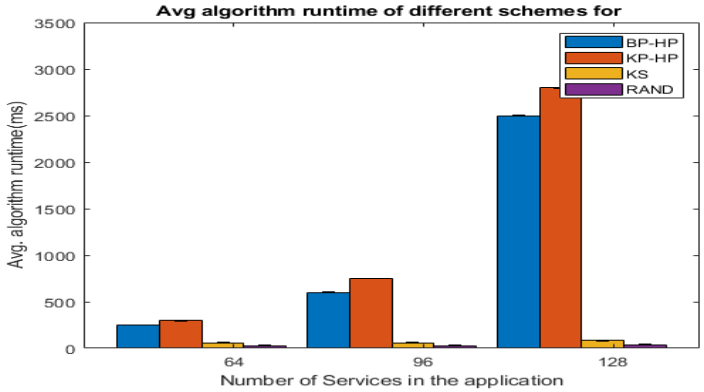

Figure 8. Average algorithm runtime of different schemes for the heterogeneous cluster.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 3, May 2024

VII. CONCLUSION

In conclusion, our research has addressed the challenges associated with service placement in microservice architectures within cloud environments. Through the development and evaluation of novel partition and packing algorithms, we have made significant strides in optimizing resource allocation and minimizing communication overhead. Firstly, we introduced two partition algorithms, Binary Partition and K Partition, both leveraging a well-designed randomized contraction algorithm. These algorithms effectively identify high-quality partitions of service-based systems, laying the groundwork for efficient resource allocation. Furthermore, we incorporated the most loaded heuristic and traffic awareness into the packing process, ensuring that applications are packed as efficiently as possible. By considering both resource demands and traffic patterns, our packing algorithms minimize inter-machine communication and improve overall system performance.

Through extensive evaluation on testbed clusters, we demonstrated the effectiveness of our proposed algorithms. Not only did they improve the ratio of successfully placed applications on the cluster, but they also significantly reduced co-located traffic, thereby enhancing system efficiency and scalability. While our algorithms incur some overhead, our evaluation indicates that this overhead is acceptable within real-world scenarios. With careful consideration, we have determined that our algorithms are suitable for practical deployment, offering tangible benefits in terms of resource utilization and communication efficiency.Looking ahead, we aim to further enhance our implementation by exploring problem-specific optimizations and incorporating dynamic resource management techniques. By adapting to evolving resource dynamics and addressing more complex scenarios, we can continue to improve the effectiveness and applicability of our algorithms in real-world cloud environments.

REFERENCES

[1]. H. Dinh-Tuan, F. Beierle, and S.R. Garzon, “Maia: A microservices based architecture for industrial data analytics,” in 2019 IEEE International Conference on Industrial Cyber-Physical Systems (ICPS). IEEE, 2019, pp. 23–30.

[2]. H. Dinh-Tuan, M. Mora-Martinez, F. Beierle, and S. R. Garzon, “Development frameworks for microservice-based applications: Evaluation and comparison,” in Proceedings of the 2020 European Symposium on Software Engineering, 2020, pp. 12–20.

[3]. Y. Shkuro,Mastering Distributed Tracing: Analyzing performance in microservices and complex systems. Packt Publishing Ltd, 2019.

[4]. G. Somashekar and A. Gandhi, “Towards optimal configuration of microservices,” in Proceedings of the 1st Workshop on Machine Learning and Systems, 2021, pp. 7–14.

[5]. R. Ramakrishna, “Bayesian optimization of Gaussian processes with applications to performance tuning,” InfoQ, Oct. 19, 2019

[6]. T. H. Thakkar, J. J. Amalraj, M. S. Sakthivel, and M. Ramkumar, “Spread scheduling strategy in docker container using meta-heuristic optimization,” Design Engineering, pp. 5793–5807, 202

[7]. Balalaie, A. Heydarnoori, A. Jamshidi, P. Migrating to cloud-native architectures using microservices: An experience report. In Proceedings of the European Conference on Service-Oriented and Cloud Computing, Taormina, Italy, 15–17 September 2015; Springer: New York, NY, USA, 2015; pp. 201–215.

[8]. Amaral, M.Polo, J.Carrera, D.Mohomed, I.Unuvar, M.Steinder, M. Performance evaluation of microservices architectures using containers. In Proceedings of the 2015 IEEE 14th International Symposium on Network Computing and Applications, Cambridge, MA, USA, 28–30 September 2015; pp. 27–34.

[9]. Wu, Z.Lu, Z. Hung, P.C.Huang, S.C.Tong, Y.Wang, Z. QaMeC: A QoS-driven IoVs application optimizing deployment scheme in multimedia edge clouds. Future Gener. Comput. Syst. **2019**, 92, 17–28.

[10]. Chen, X.Tang, S.Lu, Z.Wu, J.Duan, Y.Huang, S.C.Tang, Q. iDiSC: A New Approach to IoT-Data-Intensive Service Components Deployment in Edge-Cloud-Hybrid System. IEEE Access **2019**, 7, 59172–59184.

[11]. Selimi, M.Cerdà-Alabern, L.Freitag,F.Veiga, L.Sathiaseelan, A.Crowcroft, J. A lightweight service placement approach for community network micro-clouds. J. Grid Comput. **2019**, 17, 169–189.

[12]. Guerrero, C.; Lera, I.; Juiz, C. A lightweight decentralized service placement policy for performance optimization in fog computing. J. Ambient Intell. Humaniz. Comput. **2019**, 10, 2435–2452.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 3, May 2024

[13]. Biran, O.Corradi, A.Fanelli, M.Foschini, L.Nus, A.Raz, D.; Silvera, E. A stable network-aware vm placement for cloud systems. In Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid), Ottawa, ON, Canada, 13–16 May 2012; pp. 498–506.

[14]. Dong, J.Jin, X.Wang, H.; Li, Y.Zhang, P.Cheng, S. Energy-saving virtual machine placement in cloud data centers. In Proceedings of the 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, Delft, The Netherlands, 13–16 May 2013; pp. 618–624.

[15]. Brandolese, C.Fornaciari, W.; Pomante, L.;Salice, F.Sciuto, D. Affinity-driven system design exploration for heterogeneous multiprocessor SoC. IEEE Trans. Comput. **2006**, 55, 508–519.

[16]. Hu, Y.Zhou, H.de Laat, C.Zhao, Z.Concurrent container scheduling on heterogeneous clusters with multi-resource constraints. Future Gener. Comput. Syst. **2020**, 102, 562–573.

[17]. Meng, X.Pappas, V.Zhang, L. "Improving the scalability of data center networks with traffic-aware virtual machine placement. In Proceedings of the 2010 IEEE INFOCOM, San Diego, CA, USA, 15–19 March 2010; pp. 1–9.

[18]. Wang,M.Meng, X.Zhang, L. Consolidating virtual machines with dynamic bandwidth demand in data centers. Infocom **2011**, 201, 71–75.

[19]. Goldschmidt, O.Hochbaum, D.S. Polynomial algorithm for the k-cut problem. In Proceedings of the 1988 29th Annual Symposium on Foundations of Computer Science, White Plains, NY, USA, 24–26 October 1988; pp. 444–451.

[20]. Woeginger, G.J. There is no asymptotic PTAS for two-dimensional vector packing. Inf. Process. Lett. **1997**,64, 293–297

[21]. Karger, D.R. Global Min-cuts in RNC, and Other Ramifications of a Simple Min-Cut Algorithm. SODA **1993**,93, 21–30.

[22]. Baldin,I.Chase,J.Xin, Y.Mandal, A.Ruth, P.Castillo,C. Orlikowski, V.; Heermann, C.Mills, J. Exogeni: A multi-domain infrastructure-as-a-service testbed. In The GENI Book; Springer: New York, NY, USA, 2016; pp. 279–315.

[23]. Benson,T.Anand,A.Akella,A.Zhang, M. Understanding data center traffic characteristics. In Proceedings of the 1st ACM Workshop on Research on Enterprise Networking, Barcelona, Spain, 21 August 2009; ACM: New York, NY, USA, 2009; pp. 65–72.

[24]. Huang, K.C.Shen, B.J "Service deployment strategies for efficient execution of composite SaaS applications on cloud platform. J. Syst. Softw. **2015**, 107, 127–141

[25]. Alicherry, M.Lakshman, T. "Network aware resource allocation in distributed clouds. In Proceedings of the 2012 IEEE INFOCOM, Orlando, FL, USA, 25–30 March 2012; pp. 963–971.