UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Ingegneria dell'informazione

Master's Degree in COMPUTER ENGINEERING HIGH PERFORMANCE AND BIG DATA

Final Dissertation

Efficient strategy for microservices scheduling in IAAS architecture

Supervisor

prof. Carlo Ferrari

Candidate

Enrico Sabbatini

Academic Year 2022/2023

12/12/2022

Abstract

Microservices is an architecture developed in 2004 and used commonly for web services: it is easy to develop and it has good fault tolerance. Using this type of architecture you need to split the entire application in the small pieces that need to be independent. The architecture has a lot of advantages but we can have some problems with the application performances, because we need to orchestrate properly all the microservices. If we need more computation power, we can create new microservices instances in a distributed way. In this work we are looking for a way to optmize a workflow of an application with the microservices in a IAAS cloud. This work is based in the following paper [1] that is the starting poing of my work. I will try to optimize the response time of the application and the minimal occupation in the cloud in terms of : cpu usage, machine cycles, ram usage and hard disk usage.

Index

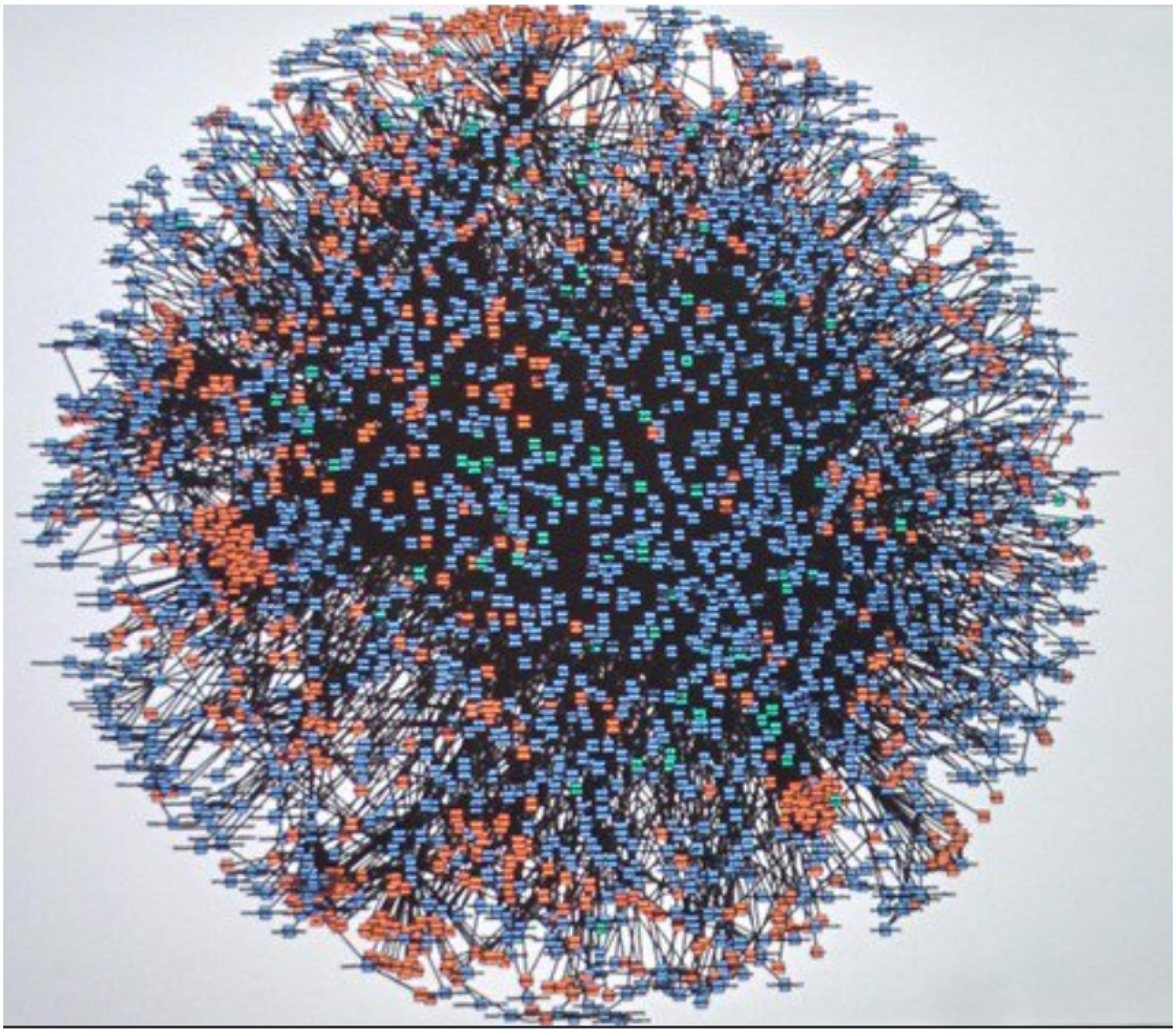


Figure 0.1: Amazon microservices

Chapter 1

Introduction

The microaservice architecture was invented in 2004 and the first time this name was used during a conference about search engine. The main feature of this type of architecture is the fact that we are decomposing the entire system in small atomic pieces so we are loose coupling of the system. Before the 2010 this type of architecture is not used so much, after 2015 it has started to be more used in a lot of companies , like Amazon or Netflix (look at the first page there are a picture of the microservices in the amazon system). If the system is "big" is convenient to split the application in small pieces otherwise it is better to leave it as monolithic. Every day the microservices technology is used by a lot of companies. In the microservices the entire application is divided into a lot of pieces that work indipendently. The main issue of this architecture is to orchestrate these pieces , in this case we need to orchestrate all the microservices in a particular way. In the new era the companies do not care too much ABOUT the physical machine, they rent virtual machines and they work in the IAAS level. In each virtual machine can be a lot of microservices, but if there are a lot of powerful virtual machineS the companies will pay a lot. So the system could be more cheaper in terms of money (and in terms of energy) you need the power of each virtual machine to be the minimum as possible for each microservices.

1.1 Microservices

Microservices is an architecture where you divide the entire system in small pieces to get a system scalable. Microservices are atomic pieces of the application , it is the best way to create a system like the paradigm "divide and conquer". If you divide all your application in small pieces IT is easIer to build.It more easier to design and to have a good fault tolerance. With microservices is more easier to get a good fault tolerance because if a piece of the application does not work , the whole system does not go down. If you have a monolithic application, this is like one block application , if something goes wrong, theentire system goes down. The microservices is more scalable and more powerful because if you need more power in terms of computation of an application (because there are new user, becAUse you have too much data to manage) you just have to duplicate the microservices AS the system needS and the entire application gets more computing power.

Another thing that distinguishes the microservices system from the monolithic system is the fact that if A microservices application has a new feature to implement, there is no need to reinstall all the application , it required just the installation of the pieces that have changed.

1.2 The advantages

There main advantages of the microservices:

Scalable The application becomes more scalable , so if we need more power in terms of computation we need only to add microservices inside our application

Fault tolerance When some part of the application have a halt, the entire application is still running

Distribuited application It's simple to distribuite this application because that it is already divided in indipendent pieces. So it's simple to move a piece of application in a different machine.

Indipendent develop Each microservices can be developed in a indipendent way, so there can be a lot of teams for each microservices.

and there are a lot of advantages

1.3 The disavantages

Like in every technology we have a lot of problems , in this case we can have some problem in the level of the architecture:

Atomicity When you develop based on a microservices application , you need to take care about the fact that each microservices is small as possible and indipendent one from the others, so is difficult to know the "good" size of each microservices

Decomposing When you develop an application you need to split it in a lot of pieces , this work take a lot of time

Orchestration It is difficult to orchestate the entire application , because each MS is indipendent one from the other, but each microservices can need some data from the others microservices

Api gateway bottleneck The only way to communicate with the clients outside the application is the api gateway

There are other disavantages for this type of architecture: we focus on the orchestration and the api gateway bottleneck. Two problem are connected and if the orchestration is not "good" we can have some delay on the entire system.

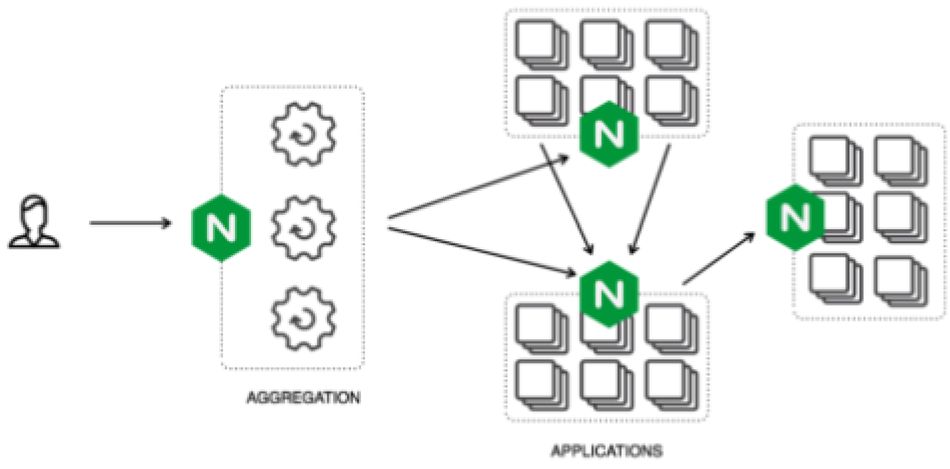


Figure 1.1: Microservices semplification

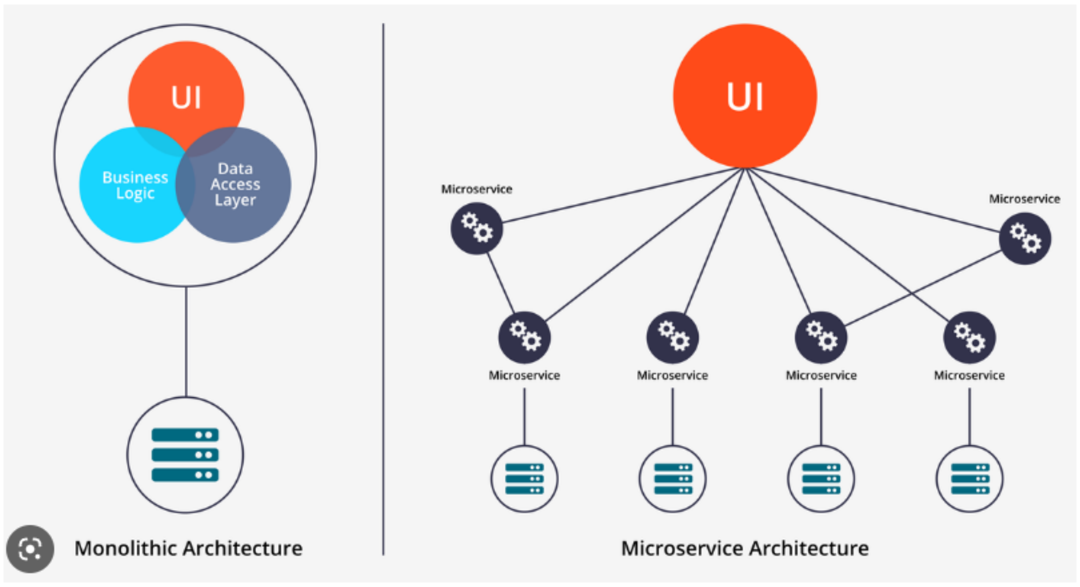


Figure 1.2: Microservices vs monolithic

1.4 Microservices vs monolithic

The monolithic architecture is the architecture of the application build in only one program. This type of architecture is quite used to develop small applications and it is difficult to distribute among different hosts. If one piece of the monolithic system is going down , the entire application will go down as well. It is not scalable and it is difficult to maintain or to perfmorm a new upgrade. If it is developed a new pieces of this application it needs to reinstall all the application in the host. At the same time is easy to develop monolithic if the application is not too big. The microservices are good in terms of scalability and very easy to update. If one piece is update it does not need reinstall all the application and in the same time is not so difficult to be distribuited in varius hosts, becuase the entire system is diveded in small pieces.

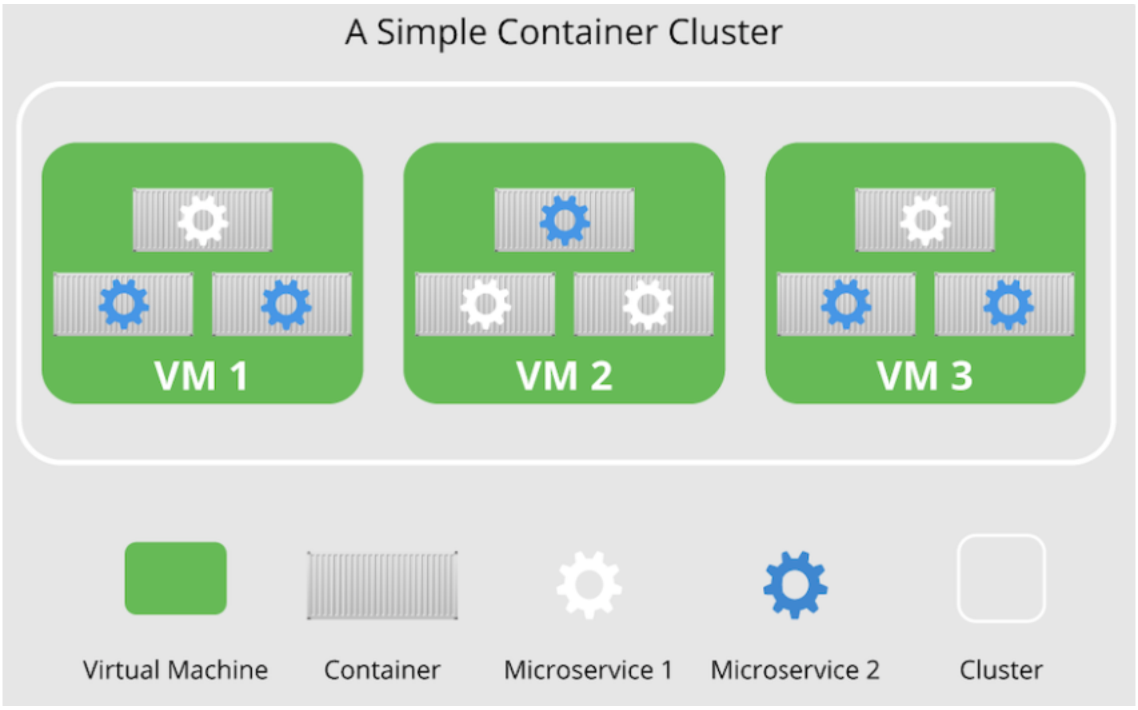


Figure 1.3: Container and microservices

1.5 Virtual Machine

The virtual machineS are like the physical but they are inside a physical one or more. We can have a lot of virtual machine inside the physical and we can split the physical machine in more virtual machine. Usually the server is a virtual machine and we can rent this server like VPs or dedicated host TO keep our server online. A lot of companies rent this virtual machine like Amazon aws or Google cloud TO keep their application server running and online. The main problem is the cost of this virtual machine, becAUse they use energy (for the fAct that they are inside the physical machine). The cost is proportional to the energy and the cost can be a problem FOR the companies. More powerful is the virtual machine more the cost is high. Every virtual machine has a virtual cpu, A virtual ram and A virtual disk, depending on WHERE the physical machine IS and depending ON how the physical machine is divided. The power computing is about the fact that more powerful is the cpu, ram and disk we have, more powerful is the virtual machine (like the physical computer). The virtual machine can be different in terms of bandiwth depending on the place where is the physical machine so the bandwidth available.

1.6 Container

The container is like a virtual machine but in other terms more lightweight. The containers use the same operating system and in the virtual machine THERE can be a lot of containers. The containers divide the virtual machine or the physical machine in different ambient settings. For example we can have a container with the node.js and another container with other dependencies. The containers are a new way to deploy application with all the dependencies needed, we can create a container with inside an application with all the dependencies and the actual state (configuration) of the application. All the containerS are managed by a program nameD Docker, with Docker we can create, destroy or duplicate A container. In this way we will simplIFy the enviroment of the applications and we can create a lot of applicationS eith the same enviroment If an application is needed to BE installED in a machine with container and docker the process is easy. The container is duplicateD and it is copied into the DESIRED machine. The containers,UNLIKE the virtual machineS, can be created in a fast way and they can be more suitable in the case of the microservices regrading the state duplication and its dataset.

1.7 Quality of services

When the system useS microservices architecture, the system needs to be scalable but the users want to be fast. If a user in Amazon want to look to an item in his cart, he or she does not want to wait more than 4 or 5 seconds, otherwise the users will change the application used. So the application needs to be fast enough but at the same time, it needs to be cheaper for who are hosting the system. All there things depends ON the geographic place where the host is situated so this implies the type of bandwidth and the cost of the energy. So who hostS the system or the variOus hosts of the system (AS we are in the distribuited world) need to think about the fact that they have a big impact in the final cost and in the quos of the system. The quality of the system has a lot of variable, not only the time of the response of the system but what response they give to the user. If a user want to take the number of items in a cart in

Amazon and he or she get a different answer the system can be quickly as you want , but users will give a negative evaluation to the system.

1.8 Docker vs virtual machine

The main difference is that the virtual machine can be see like a container of containers and the container can be see like a subset of the virtual machine. A container uses the operating system whereIT is installed, and it has a small boot time instead of the virtual machine. To install a new container is more easIer than install a new virtual machine because to create a docker with insed an application you can easly copy it and run with docker. So if we have a system with microservices architecture we can easIly implement it with Docker and we have in a fast way a microservices running with all his dependeNcies and it's state. If we create a virtual machine for each microservices as first thing we have a high boot time and second THING we use a lot of physical resources for nothing, with containers we will slipt the virtual machine in "sub-virtual machine" to save the boot time for the fact that we reuse the virtual machine just created.

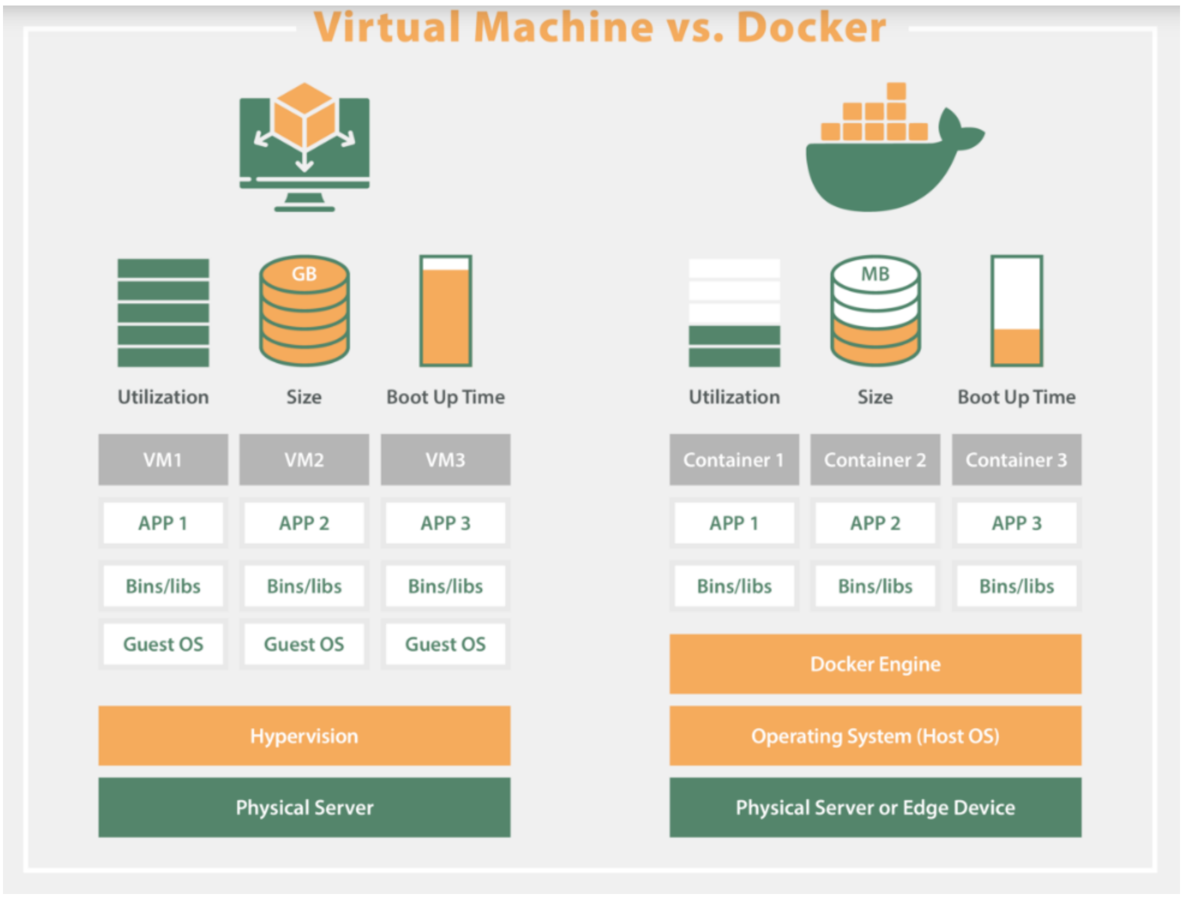


Figure 1.4: Docker vs Virtual machines

1.9 The main idea of the work

In this work we will see how to optimize the utilization of the virtual machine for a system based in microservices architecture , so who host the system will save money. At the same time , we save power computation (in terms of disk space ,utilization cpu and ram space) and we get a good quos in terms of time and quality of response. In this work we will use docker and a lot of tools to simplify the managin of the virtual machine and some known algorithms in literature. In this work is exaplained with more details the objective of the next chapter, after that there are the mathematical model of the problem and the model of the system. We will use some NP-HARD problem to adapt some known algorithm to help the manage of all the microservices and the virtual machine. In the end of the work there are some statistics experiment to show that the idea works. All this work is done with some simulation tools and with some random requests.

Chapter 2

Scheduling microservices in IAAS architecture

In the microservices architecture we have a lot of advantage as I have said, but we need to manage some issue about this type of system architecture. We have to orchestrate a lot of things and we need to take care about the fact that we have no longer one application and we have a lot of "pieces" of this application.If we lose the control of the orchestration, we would have some problem in terms of time of the resposne and in the terms of time due to the occupancy of the power avaible, in this case the virtual machine. When we have a lot of clients that try to get some information, like Amazon we can think, we need to orchestrate or in other words we need to schedule all these actions that the users wants from our application. So we need to schedule all this "pieces" and we need to place these pieces in the right position, otherwise we use too much virtual machine and the cost of the applciation increse in a in terms of power. This algorithm is already created by the autohr of 1 but their algorithms have some problem, it too much heavy. If the system would be big as Amazon to be bigger like Amazon, this algorithm will be slow (we are talking about a O n3, where N is the number of task we have. The proposed algorithm needs to take care about the statistics of the system, so it takes time to be inizialized. We need an algorithm that tries to be quickly and orchestrate all the system in the best way. So in this work I will try to create an algorithm similar to the 1 with better performance in terms of time and results.

Actual state of the art

In the paper [1], this problem is faced with good results. The algorithm used in the paper [1], determines the container configuration based on the statistics of the workloads and it adapts to the streaming workload. A VSBPP-based scaling algorithm (operation research) is proposed to solve the two-layer scaling problem caused by the introduction of the container layer, and the impact of container images is considered. By simulation experiments using workflow applications, the ability of the proposed algorithm to improve the success ratio and cost for microservices in clouds seems good. There is problem still open: In future works, it is planned to study how to construct a generalized performance model for a microservice-based system to accurately predict the response time of each microservice with different configurations. Taking this into account, it is possible to optimize the process of autoscaling of microservices. The algorithms present by the author are too much complex,so one problem is still open.

2.1 Why is it important?

It is easy to understand that if we have less virtual machine and we keep the system with a good quos in terms of response, we will have less outgoings and we will earn more than before. Another important thing is how we orchestrate the microservices, if the system is dumb, there can be overload of virtual machines with a lot of queue and the quos will be bad. So , having only a fixed number of virtual machine it can introduce a bad user experience and in the same time if we do not remove the virtual machines that are not utilized we cannot save money.

Chapter 3

Model of the system

In this section described how the system is modeled to be apply to the algorithms developed in this work. In the microservices architecture there are some modifications about the system because the system will face not only the occupancy of the virtual machine with the containers but also the scheduling of all the tasks that the user will require to the entire application. The actions of the user are a task , for example if a user want to see how much money there is in his bank account, this action is a task of the system. So we need to model these two problems , the scheduling and where we will place this microservices depending on the request of the users. The "brain" of this system is the api gateway that it has the control of the entire system of the virtual machine and the container engine. The next chapter show how algorithms are implements with this model.

3.1 Hyphotesis

The model on this work does not consider:

Fault tolerance In this work all the microservices are free from bug, it is an hypothesis of the work and it does not consider if the microservices goes down.

Security The system does not model the part of the security or does not consider the security of the microservices.

Physical problem As all this work is done in IAAS architecture, we do not care about physical problem about the machine

All these hypothesis refer to the fact that all this model consider only to manage the resources and getting the best time as possible in an ideal system, where every virtual machine work properly , so there are faults in the hosting system and there are no problems with attacks from the outside.

3.2 Definitions

Computing Power The power is the cpu, ram and disk space that each virtual machine (and subvirtual) has to compute the task.

Execution time The execution time is the time used to execute the task (in this case the microservices job)

Task It is the set of subtask that complete the operation. In the model is a directed graph where each vertex is a subtask

SubTask It is a sub-vertex of the task, it is a single operation

Constraint It is an edge of the directed graph that if a task is not complete the task after that it will not be executable

Queining Netowrk Is the model where each vertex is a Virtual Machine and each virtual machine has a queue .

Task Network Directed graph where each edge is a contraint and the task is a node. It is a directed graph.

Virtual machine It is a set of container, where this set has a limit depends from the power of the machine.

Container The container is a microservices , for the fact that we assign each microservices in a container.

Api Gateway It is the orchestrator of the entire system, it is the algorithm I will study in this work.

Virtual machine It is a machine with some disk space, ram space and cpu usage. Each of this value are in per cent

3.3 Definitions of microservices and virtual machine

For the fact that we are working in the IAAS architecture, we need to take care only of the virtual machine, so the system sees the microservices like a container and like a task. So when in this work we talk about the microservices, it is the same thing to talk about a task or to talk about a container. In each container there is one microservices each sub-task is rapresented by one container. Each microservices (or task or container) is represented by a vector of three numbers. The first number represents the ram in terms os Megabit , the second rapresent the cycle of the cpu and the third represent the disk pace in terms of megabite. The third element depends on the dependencies of the microservices, because usually the microservices needs to be smaller as possible (from the definiton of microservices). So a microservices can be formalized :

$$m_i = \begin{bmatrix} cpu \\ ram \\ disk \end{bmatrix} \tag{3.1}$$

We have three elements, the cpu is the occupation that is calculated depending on the type of the cpu of each virtual machine, it's a measure calculated based on the maxium clock cycle of the virtual machine we have. The ram and the disk are the size in terms of the megabyte and they are getting by the specification of the virtual machine. The same are the virtual machine:

$$vm_i = \begin{bmatrix} cpu \\ ram \\ disk \end{bmatrix} \tag{3.2}$$

So this is the way how the system is represent in the poartion of the model that needs new resources. In the other part of the model , we will see the microservices like a task and we will insert the microservices in the right virtual machine. The second part of the algorithm I will model the system like the firstone, but we have a queue of microservices when all the virtual machine are occupied. We have only one queue and we need to register each task that is not executed or it is waiting to be executed by a virtual machine.

The fixed set of the virtual machine

The set of fixed virtual machine is rapresented by M and it is the types of virtual machine the system can create. Each virtual machine has a cost of power, and this cost is defined by the vector in the previous section. More the vector has high more the cost of the virtual machine.

3.4 knapsack problem

$$\max \sum_{i=1}^{I} v_i x_i$$

$$\text{such that} \sum_{i=1}^{I} w_i x_i \leq W$$

$$x_i \in \{0,1\}\ \forall i \in [I]$$

Figure 3.1: knapsack problem 0-1

In this work, i will model all the systems with the knapsack problem. This problem is well known in the literature as a NP-HARD problem. There is a knapsack problem when we have some items and each item has a cost, the objective function of this problem is to maximize the cost, but we cannot exceed the limit of the knapsack. In this system we will use the Multi knapsack problem which is the same but we can use more than once the available items. I will model this problem in a mathematical form:

$$\begin{aligned}
&\text{maximize} && z = \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij} \\
&\text{subject to} && \sum_{j=1}^{n} w_j x_{ij} \le c_i, \qquad i \in M = \{1, \dots, m\}. \\
& && \sum_{i=1}^{m} x_{ij} \quad \le 1, \qquad j \in N = \{1, \dots, n\}, \\
& && x_{ij} = 0 \text{ or } 1, \qquad i \in M, j \in N,
\end{aligned}$$

$$x_{ij} = \begin{cases} 1 & \text{if item } j \text{ is assigned to knapsack } i; \\ 0 & \text{otherwise.} \end{cases}$$

Figure 3.2: multiknapsack knapsack problem

So , i will model all the systems in two different ways with using the modeled problem: in one instance, when we need to create new resources we interprete the microservices like the knapsack model. Since this problem is NP-HARD, is not possible to develop and algorithm to find and exact solution to the problem, so i will use some heuristics methods , like genetic and particle swarm optimization and an approximation algorithm.

3.5 The api gateway

The Api gateway is the core of all this work, it has the view of the entire system and in this model has some properties:

View of the system It has a complete view of all the microservices system, it look inside to every virtual machine and can see the amounting of computing power

Power to change the system In the apigateway there is the permission to modify the system, so it can manage the virtual machine and the containers

Where the algorithm works The algorithm we will develop it will run inside the api gateway. It has the capability and the various resources needed to the algorithm

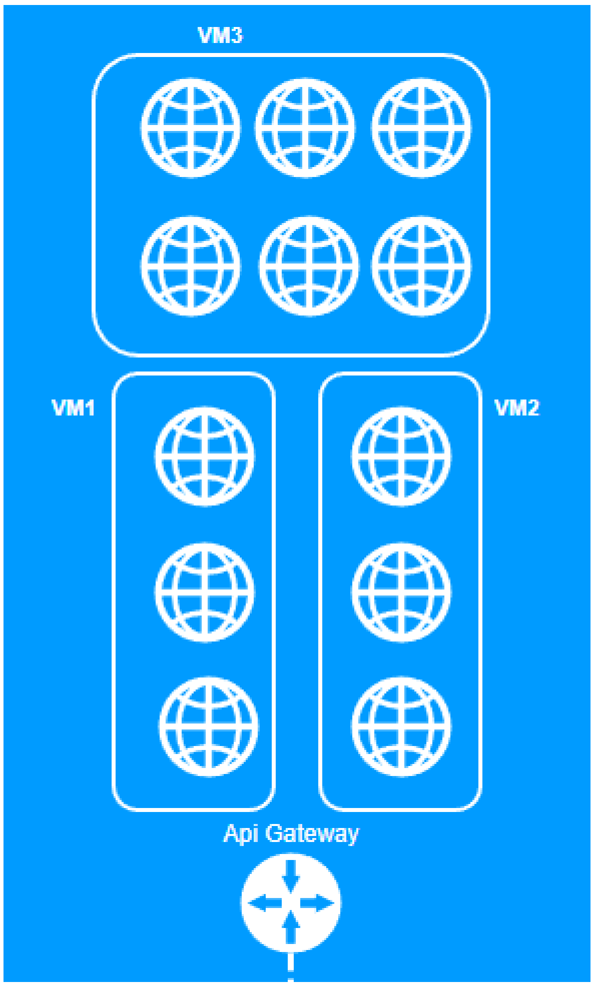


Figure 3.3: The view of the apigateway

The view of api gateway

The api gateway has a subset view of all the system, this means that every virtual machine is a subset of container. Every virtual machine has a limit depending on the type of virtual machine, so it can contains a finite number of microservices (of container).

3.6 The divison of the model

The model is divided in two parts:

Resources model In this part of the model, it is modeled the creation of the virtual machine

Scheduling model It is the model on which the task are executed

The resources model

In the resources model is applied the knapsack problem where the sum of the power compunting of the microservices are the knapsack and the virtual machine are the items. We don't want to exceed the limit of the microservices, because otherwise there are a waste of power computation. So we have some types of virtual machine M and for each virtual machine we want to have the virtual machine with the smallest cost. So we have some set on this part of the model:

The fixed virtual machine types This set rapresent the types of virtual machine that we can create. Each virtual machine is defined like in the section 3.3

The virtual machine Each virtual machine is a set called VM where these virtual machine are rapresented by the i-th element of the vector named VMs

The vector that rapresent the VMs is:

$$y = \begin{bmatrix} vm_1 \\ vm_2 \\ \vdots \\ vm_M \end{bmatrix} \tag{3.3}$$

where M is the number of types of virtual machine and the sum of this vector rapresents the number of virtual machines created.

The scheduling model

In the scheduling model we have a lot of tasks to be exectued, we do not map a task to a single microservices because a microservices can execute a lot of tasks. Every microservices has a queue of task to be executed. The task can occupate the percentage of the microservice if the percentage reaches the maxium it will create another microservices. The scheduling model sees the task as a single like a set of subtask and each subtask So we have this model divided in two parts:

The microservices division The microservices is divided in some parts to fit the number of subtask, where each subtask occuptied a part of the model in percentage.

The execution of the task The execution of the tasks are view like a graph where each task is a vertex of the graph and contains some subtask where each subtask matches to a pieces of microservices

The virtual machine The virtual machine is the container where the new microservice is created and where each microservices works. This container has a limit of number of miccroservices that depends from the power of the virtual machine

Therefore we have a graph G=(V,E) that rapresents the various action of the user and the set of subtask that is the microservices.

V Where V is the set of vertex where each vertex corresponds to the task

E It is the set of edges where each edge corresponds to the constraint

In the other part of the model we have the microservices division and the virtual machine, they are see like the definition on the section 3.3 The virtual machine are the container of the microservices and each microservices has a index that rapresent how is occupied, calculated with some stimation. The microservices has a percentage between 1 and 100, the same has the virtual machine.

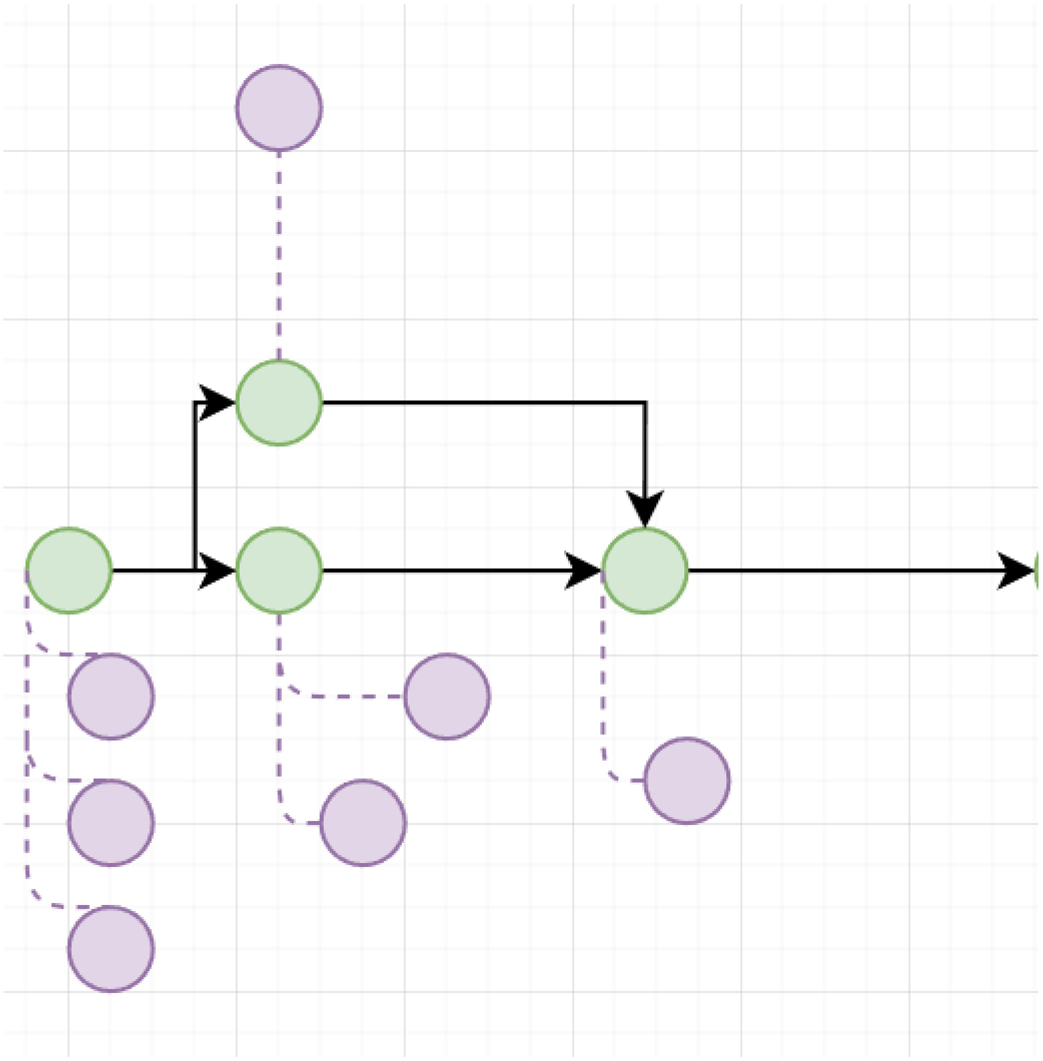


Figure 3.4: The graph of the scheduling model where the green are the vertex of task and the violet is the vertex of subtask

3.7 Constraint

In this model we have two constraints:

[**1**]The time maximum time of the excecution of the task.

[**2**]The types of the virtual machine in terms of power computing.

The first constraint depends from the fact of the Quos , if a user make an operation, the user do not want to waint too much. In the second constraint we can say that we want to minimaze the power in terms of computing, we don't want to use too much power that we do not need.

Chapter 4

The algorithms

In this chapter there are information about the three algorithms that I have used in my work:

The choice algorithm

The newer algorithm

The scheduler algorithm

This three algorithm are the core of the entire work. The choice algorithm , decides when it is the right moment to manage the resources or to schedule the task. The newer algorithm is the algorithm that creates new resources or delete the existing The scheduler algorithm is the algorithm that it schedule the task and fit the microservices in the right virtual machine.

4.1 Genetic Algorithm

We have rapresent the problem in a linear model of the knapsack problem. Where the element M are the types of virtual machine and the other elements are the microservices. So the N is the set of types of microservices and the M is the set of virtual machine.

The coding

The idea of the coding is to create a vector where each position matches to the i-th type virtual machine instead of the set VM. In the end of this algorithm the result is a vector that represent the number of types of virtual machine needed to the system.

Calculate the fitness of individual

In this part we will calculate the fitness. The fitness is:

$$f(x) = \min c(x)$$

$$c(x) = \sum_{i=1}^{T} vm_i$$

Because we want to minimaze the utilization of the virtual machines.

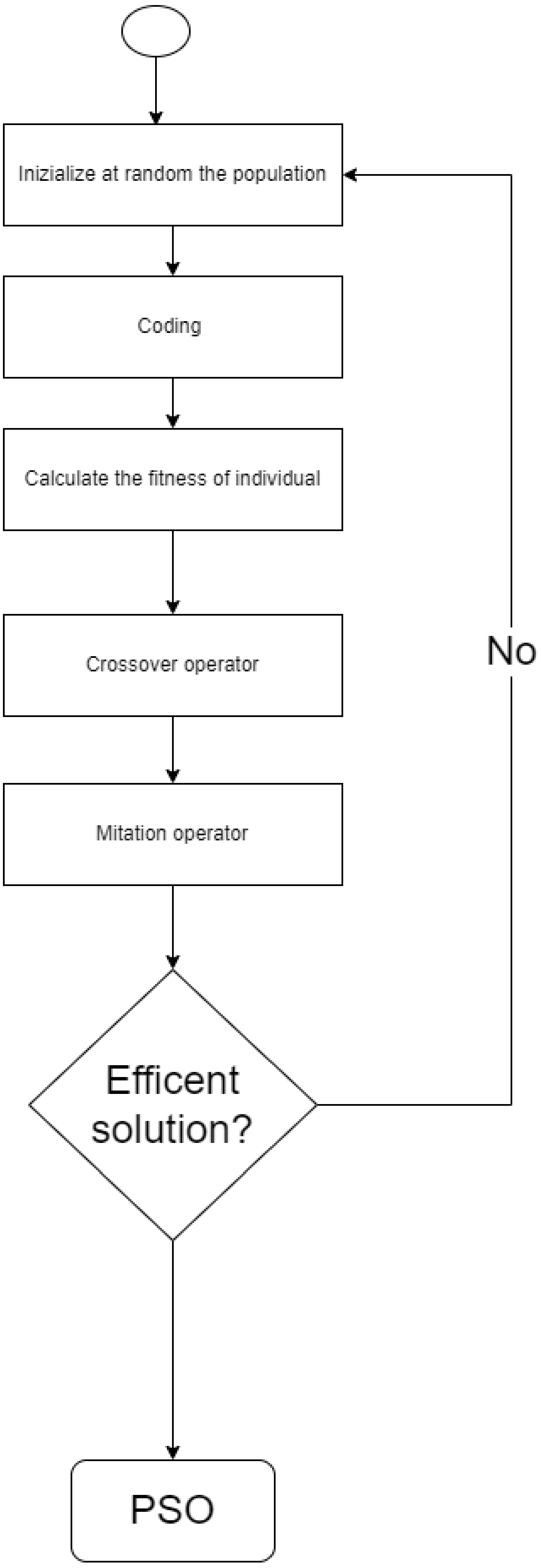

Inizialize at random the population
Coding
Calculate the fitness of individual
Crossover operator
Mitation operator
Efficent solution?
No
PSO

Figure 4.1: Description of the genetic algorithm

Crossover Operator

In the P2 formula is applied if and only if

$$f_a < f_b \tag{4.1}$$

otherwise the algorirthm will aply P2. The crossover operator is neede to improve the global search capabilty and to help to get the best possible solution. In thi formula we have:

f_{max}	= The maximum fitness value
f_a	= The maximum fitness value between two individual
f_b	= The average fitness value of all population
k_1	= Parameter between 0 and 1
k_2	= Parameter between 0 and 1

We will calculate the corssover operation and as:

$$P1 - crossover = k_1 * \frac{f_{max} - f_a}{f_{max} - f_b}$$

$$P2 - crossover = k_2$$

Mutation operator

In the P2 formula is applied if and only if

$$f_b < f_c \tag{4.2}$$

otherwise the algorirthm will aply P2. The mutant operator is neede to improve the local search capabilty and to help to get the best possible solution. In thi formula we have:

f_{max}	= The maximum fitness value
f_b	= The average fitness
f_c	= The fitness value of who do the mutation
k_3	= Parameter between 0 and 1
k_4	= Parameter between 0 and 1

$$P1 - Mutation = k_3 * \frac{f_{max} - f_c}{f_{max} - f_b}$$

$$P2 - Mutation = k_4$$

PSO

The result of this algorithm will be transfered to the pso part to get the final result of this part of the algorithm.

4.2 PSO

The particle swarm optimization was proposed by Kennedy and Eberhart in 1995. As mentioned in the original paper, sociobiologists believe a school of fish or a flock of birds that moves in a group “can profit from the experience of all other members”. In other words, while a bird is flying and searching randomly for food, for instance, all birds in the flock can share their discovery and help the entire flock get the best hunt. This is a heuristic algorithm like the genetic with a lot of difference, as the birds it try to get a optimal solution utsing a lot of agents that they will try to get an optimal one. Some components of this algorithm is similar to the genetic one, the difference is that it is a gradient based algorithm instead of the genetic one, but like the the genetic algorithm it uses the natural things of the world.

The fitness function

The fitness is the same of the genetic algorithm, but in this algorithm change a little bit other factors.

Velocity update and position update Swag

The velocity is like the physics in real world , it is the velocity of the particle, there is a velocity of the single particle and the velocity of all the particle. The velocity of the single particle is named Pbest and the velocity of all the particles is the Gbest. This is an import part of the PSO becase if the velocity is too high, it will be give to the system too much changes in the system, this means that the solution can be missed (not the optimal one) and a too slow velocity it can be a slow down the system too much slow to find a solution. The w is the inherth weight , it adjust the global and optimal search capability. The r is a random variable between 0 and 1 and c is the variable of learnability of the velocity (like the learning rate in the learning algorithm). The updating the position like the physics in real world is how the system change the position of each particle. So the postion in this case is the solution of the algorithm.

$$x_{ij} = \begin{cases} 0 & rand < \dfrac{1}{1+\exp(-v_{ij})} \\ 1 & rand > \dfrac{1}{1+\exp(-v_{ij})} \end{cases}$$

Figure 4.2: Binary variable

$$x_{ij} = \begin{cases} 0 & rand < 0.5 \\ 1 & rand > 0.5 \end{cases}$$

Figure 4.3: Binary random variable

$$v_{ij}^{k+1} = \omega \bullet v_{ij}^{k} + c_1 \bullet r_1 (Pbest_{ij}^{k} - x_{ij}^{k}) + c_2 \bullet r_2 \bullet (Gbest_{j}^{k} - x_{ij}^{k})$$

Figure 4.4: Binary variable

$$x_{ij}^{k+1} = x_{ij}^{k} + v_{ij}^{k+1}$$

Figure 4.5: Position update

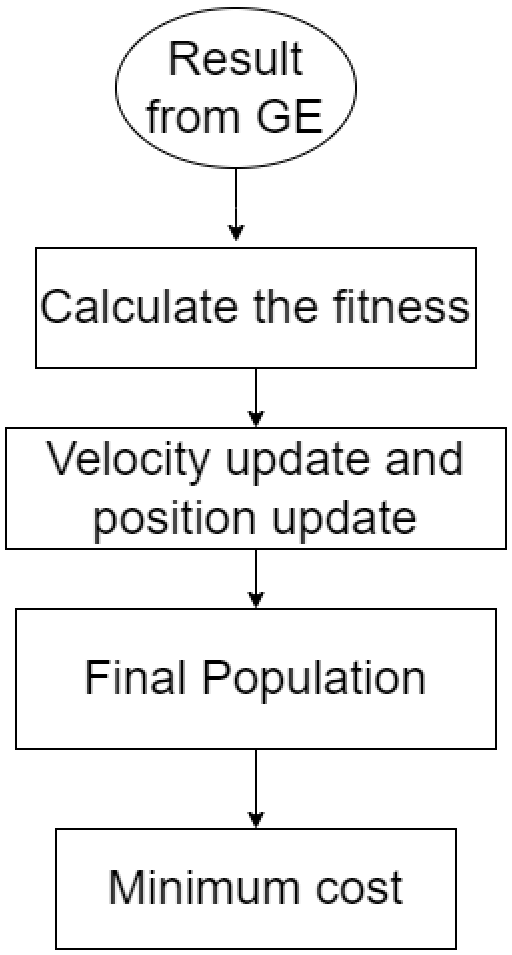


Figure 4.6: Description of the particle swarm optimization

4.3 The choice algorithm

The choice algorithm is the algorithm that "decides" what we need to do in the system. Therefore if we want to create new resources or delete an existing one or if we want to create a new resource or delete an existing one. It is a simple algorithm, in one way using some simple statistics will find if a resources is useful or if we need new resources. For first the algorithm will take statics from the system, for each virtual machine and it takes the number of microservices and the time from they start to run. After that, if one virtual machine for one minute does not has microservices inside it will call the newer algorithm, the same thing for new resources. This algorithm is activated every 0.5 second, it needs to be fast to not create a delay for all the system. If there is no needed to create new resources it will call the scheduler algorithm.

4.4 The schedule algorithm

The schedule algorithm fit and create the microservices needed for the task of the user. The initial case is when the user need to perform an operation and there are no microservices to doing that, it create new microservices and fit the microservices in the smallest virtual machine running. The smallest virtual machine is refer to the power of the virtual machine in terms of cpu, ram and disk space. It is simple, we look the request that the used did and we look if there are a virtual machine with the smallest power. In the other hand the task of this algorithm is to destroy the microservices that has no task. Otherwise the system keeps all the system resources occupied and there are no power computing avaible for creating other types of microservices or space inside the virtual machine.

The simple approach

This approach is very simple and very fast. The model of this part is done in a sense where each task is view like a subtask. When a user perform a task, it will create a queue of task and it will be assigned to the existing microservices. If the microservices is full of tasks or there are not microservices avaible. It will be create a new microservices where each of this tasks will be fit inside. The microservices is created inside the virtual machine with less computing power , because the system is intendent to use as maxium as possible the existing virtual machine. It's a simple rule but it considers the utilization in the best way of the existing resources without call to much the creation of the resources.

4.5 The newer algorithm

This algorithm has the task to create or delete the existing resources (in this case the virtual machine). The system needs to call it the least possible because creating virtual machine or deleting existing one, it is honerous work. It uses the combination of the PSO algortihm and the genetic algorithm. It tries to get the best efficient solution to use the smallest number of virtual machine but in the same time it keeps the system with the resources needed. It starts from the actual state of the system, that is the number of types o4 virtual machines , inside the algorithm there are the fixed vector with all the virtual machine in the i-th position. As show in the chapter of the model, each virtual machine and microservices is a vector of 3 element. In this case the knapsack is the sum of all microservices and the items are the virtual machine.

Where this formula is the constraint of the solution we want. The MI is the set of existing microservices and the

$$m_i$$

is the single microservices.

$$\sum_{i \epsilon MI} m_i$$

This constraint represents the maximum number of virtual machines power we need to execute the microservices. This is the constraint imposed to the hybrid PSO and genetic to find a solution. The solution of this algorithm is a vector of number where each i-th number of the vector select the virtual machine. So in the newer algorithm there are the VM and the vector where the i-th element represent the numbers of types of virtual machine.

$$y = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \tag{4.3}$$

4.6 The protocol

The protocol is the part of the system that interface with the creation of the virtual machine. The protocol interface the results of the algorithm with the rest of the system. It dialogue with the docker and with the system that manage the virtual machine. The protocol is an algorithm that interface the response of the algorithms to all the system, it depends on what technology is intendent to use to implement all this system.

4.7 The queue

In every virtual machine there is a queue for the execution of new microservice. If there are no space in the virtual machine (space intendent as power computing) a microservices is insert into the queue and the virtual where the microservice is neede to be assign.

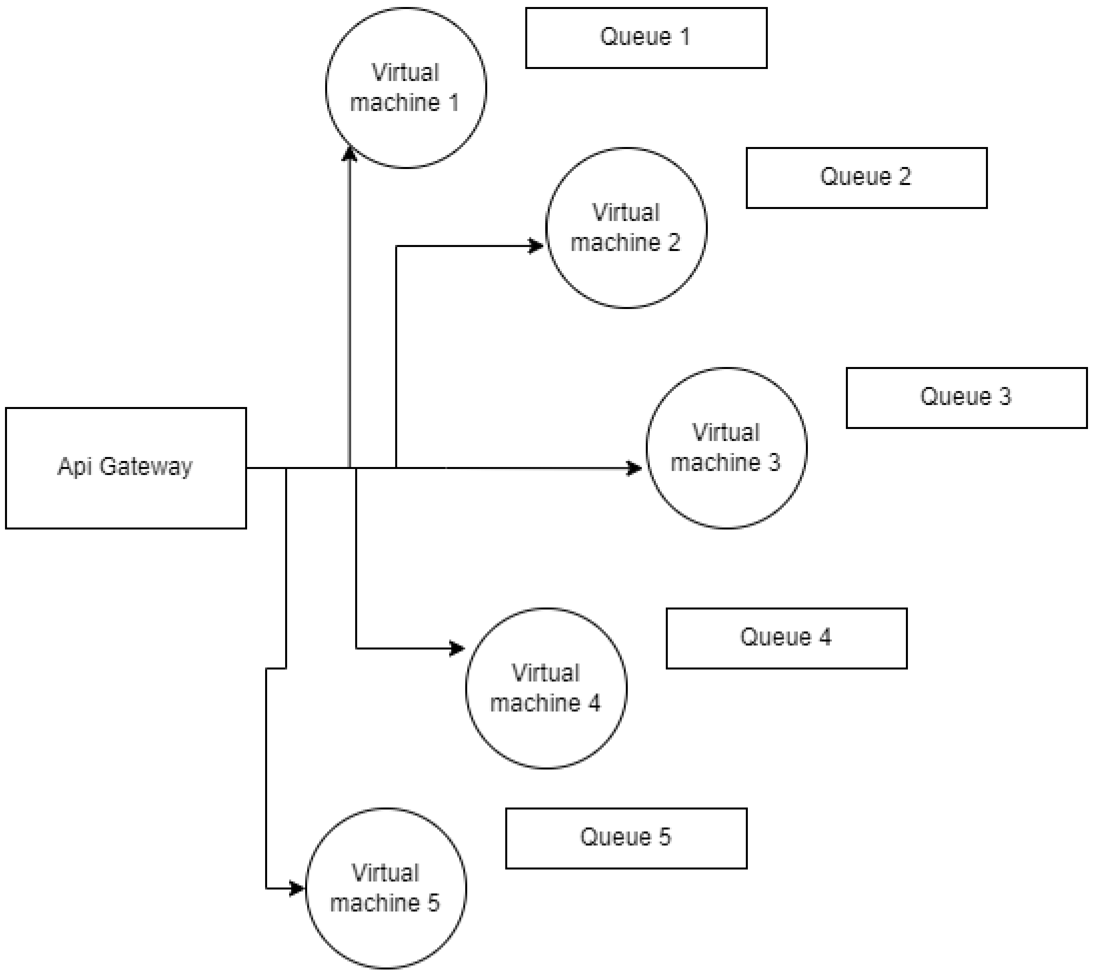


Figure 4.7: This is the queuing model of the Api gateway

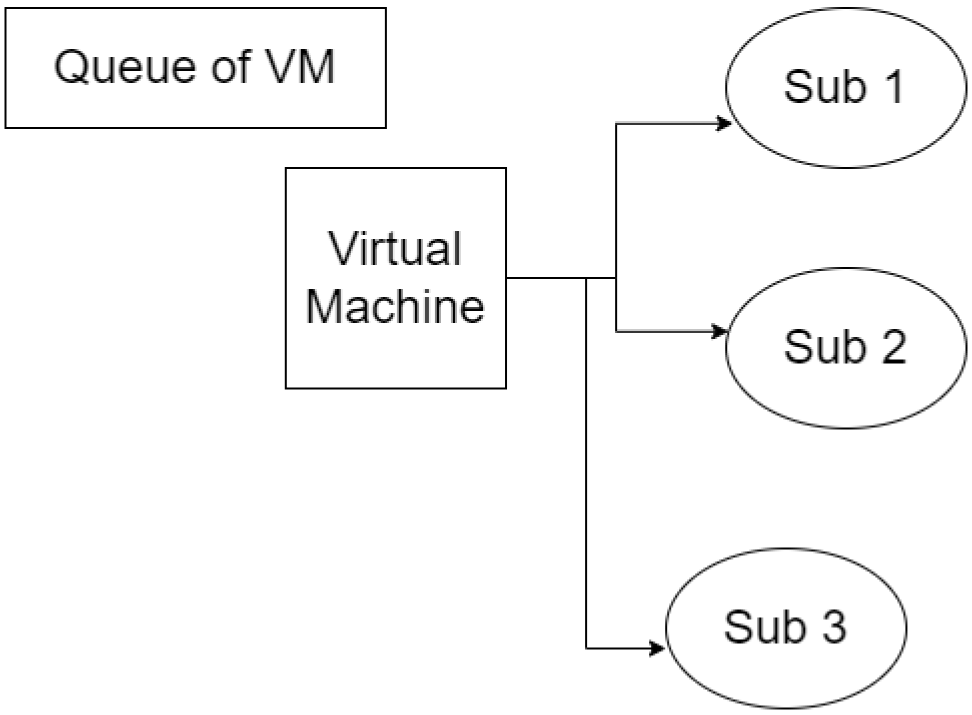


Figure 4.8: This is the model of each virtual machine, divided by the subvirtual machine

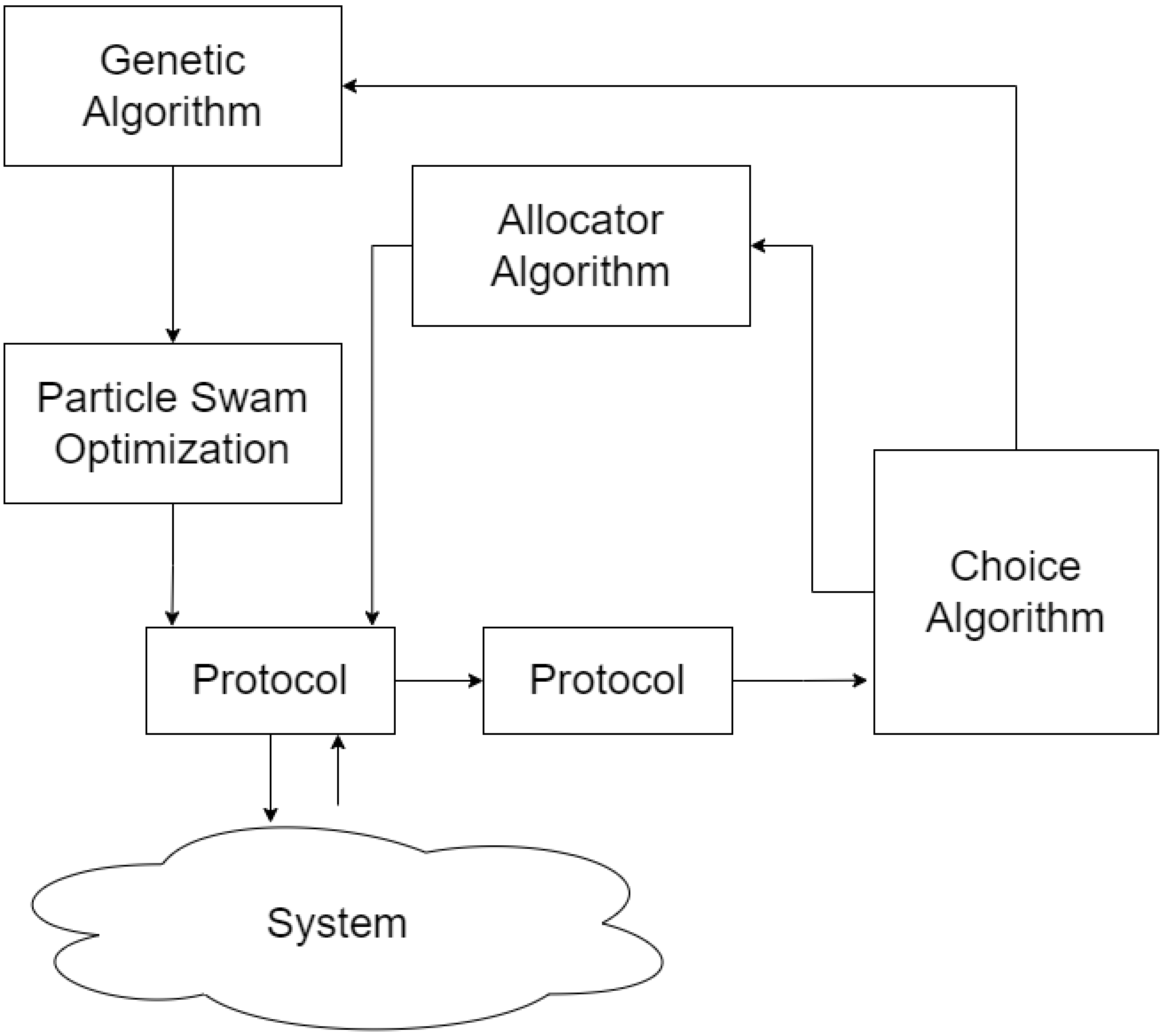


Figure 4.9: //work in progress

Chapter 5

Conclusion

5.1 Experiments

In this section we will try to proof that the algorithm works better than the vanilla setting. The vanilla setting is the system without this type of orchestrator. So for the vanilla setting we will use the policy FCFS (first come first service) where each task will be executed in the system. In the other system we will use the algorithm I have proposed. We will compare the time execution and the computing power we have used. For the experiments we will use the cloudsim TO simulate the virtual machine system and we will check if the algorithm I have developed will try to improve the scheduling of the microservices. To do the experiments we run the jar file from the python program (where the new and scheduler algorithm is done in python). The choice algorithm will start the newer algorithm or the other one depending on the type of request from the user. actual configuration of the system (the actual configuration of the system depends from the last state or from the initial random state). In this chapTer is illustrated the technology used in this work and the statistical result of this work.

5.2 Technology

In this work I will implement the algorithm in python (the orchestrator) and i will test this algorithm in different microservices system.

I will apply for each microservices system the model and the alghortihm and I will check if the response of the entire system with the algorithm is better than the response withtout this type of the orchestration.

For the PSO I will use this : https://pyswarms.readthedocs.io/en/latest/

For the Genetic I will use this: https://pygad.readthedocs.io/en/latest/

For the simulation of the microservices i wil use a framework CLoudSim: https://github.com/Cloudslab/

subsectionClouSim I have used the framework cloudsim for the simulation of the entire cloud. The framework is divided in 4 components:

Cloudlnet The cloudlnet is the task assigned to each virtual machine to be executed (in the case of my work is the microservices). With this cloudnet we can assign when and where it will be placed. From the algorithm i have develop we will use the information from the output of the algorithm to set the right information to the clodulet.

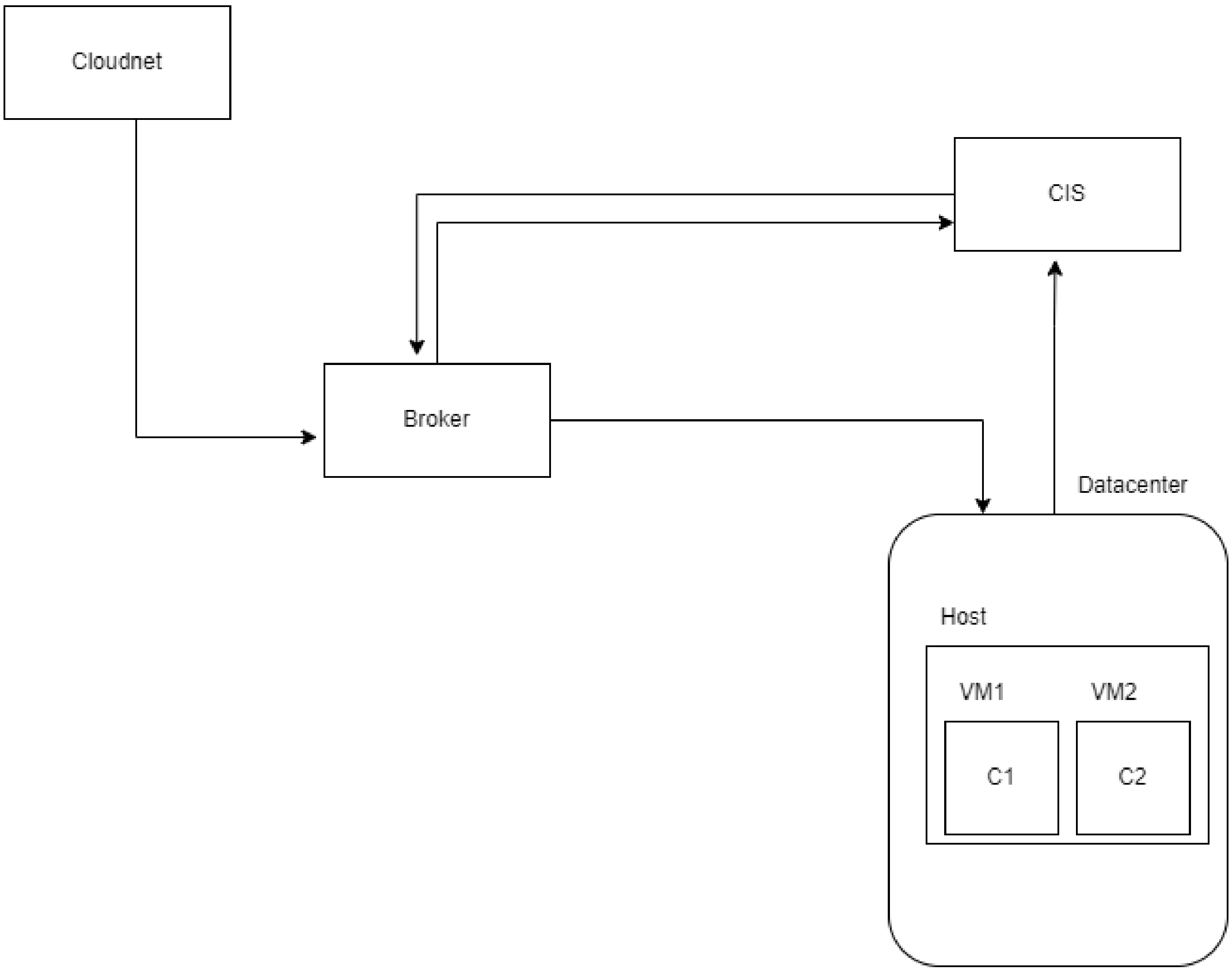


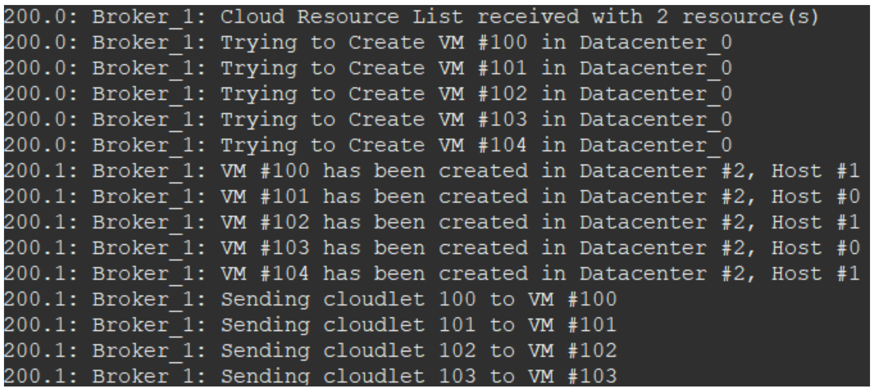
Figure 5.1: Description of the structure of the cloudsim system

Broker The broker is the part of the framework that assign the cloudnet to each virtual machine. It exchange information with the CIS to get new resources or to look how many virtual machine are avaible.

Datacenter This part is the physical place where there are the host and in each host there are the virtual machine (we don't care about this parte because we are working in the IAAS architecture).

CIS It is the part to manage the datacenter and the virtual machine. It exchange with the broker the information about the virtual machine to inform if there are space for other cloudnet.

The main part is the cloudnet and the broker, where the broker send the cloudnet in the right virtual machine and creates new virtual machines. As we are working in the virtual machine level (IAAS).



```
200.0: Broker_1: Cloud Resource List received with 2 resource(s)
200.0: Broker_1: Trying to Create VM #100 in Datacenter_0
200.0: Broker_1: Trying to Create VM #101 in Datacenter_0
200.0: Broker_1: Trying to Create VM #102 in Datacenter_0
200.0: Broker_1: Trying to Create VM #103 in Datacenter_0
200.0: Broker_1: Trying to Create VM #104 in Datacenter_0
200.1: Broker_1: VM #100 has been created in Datacenter #2, Host #1
200.1: Broker_1: VM #101 has been created in Datacenter #2, Host #0
200.1: Broker_1: VM #102 has been created in Datacenter #2, Host #1
200.1: Broker_1: VM #103 has been created in Datacenter #2, Host #0
200.1: Broker_1: VM #104 has been created in Datacenter #2, Host #1
200.1: Broker_1: Sending cloudlet 100 to VM #100
200.1: Broker_1: Sending cloudlet 101 to VM #101
200.1: Broker_1: Sending cloudlet 102 to VM #102
200.1: Broker_1: Sending cloudlet 103 to VM #103
```

Figure 5.2: Pieces of cloudsim simulation

5.3 Statistics result

In this section there are some statistics to see if the idea is useful or not. Firstable, the algorthm are tested using some statics from the response time and the users do random request. To see if the system is good or not there are some comparision between two dumb system. One dumb system is where we create virtual machines when it is needed but they are not destroy. The statics we use is the average time of each task and the number of virtual machines the system used.

5.4 Setup of experiment

In every experiment there IS three systemS: the Cheaper algorithm (the algorithm developed in this work), the ??? Dumb fixed system and the Dumb variable system. In the second type of system there are a fixed number of virtual machineS and in the variable system, the system creAteS a virtual machine when it IS needed. But in the third type, the system will not destroy the useful virtual machine. All this system is tested inside the framework java clodnet where the time and the number of virtual machine is computed by the cloudnet framework. In all the experiments there are some statistics: the average time of response of each request and the average number of virtual machine utilized during the computation of all the task. There are some graphS with the trend of the response and the creation of the virtual machine. In the experiments we have a random user where it DOES a lot of request to the microservices system so in every experiment there IS a different setup with a different network. To calculate the cpu , it needS to take the power cpu among the virtual machine avaiLAble and compute the cpu size instead of 1 and 100.

5.5 Experiment 1

In this experiment there is a random user, where this user does 100 request to the server with a equiprobable distribution in the payload of each request. In this experiment there are some small evidence that the algorithm proposed has some better performance against the DumbFixed system in terms of virtual machine and the in the terms of quos there are some good result insted the dumb system variable. It comes from the fact that the number of virtual machine is low insted the others tree system and in the quos is still good for the quos insted.

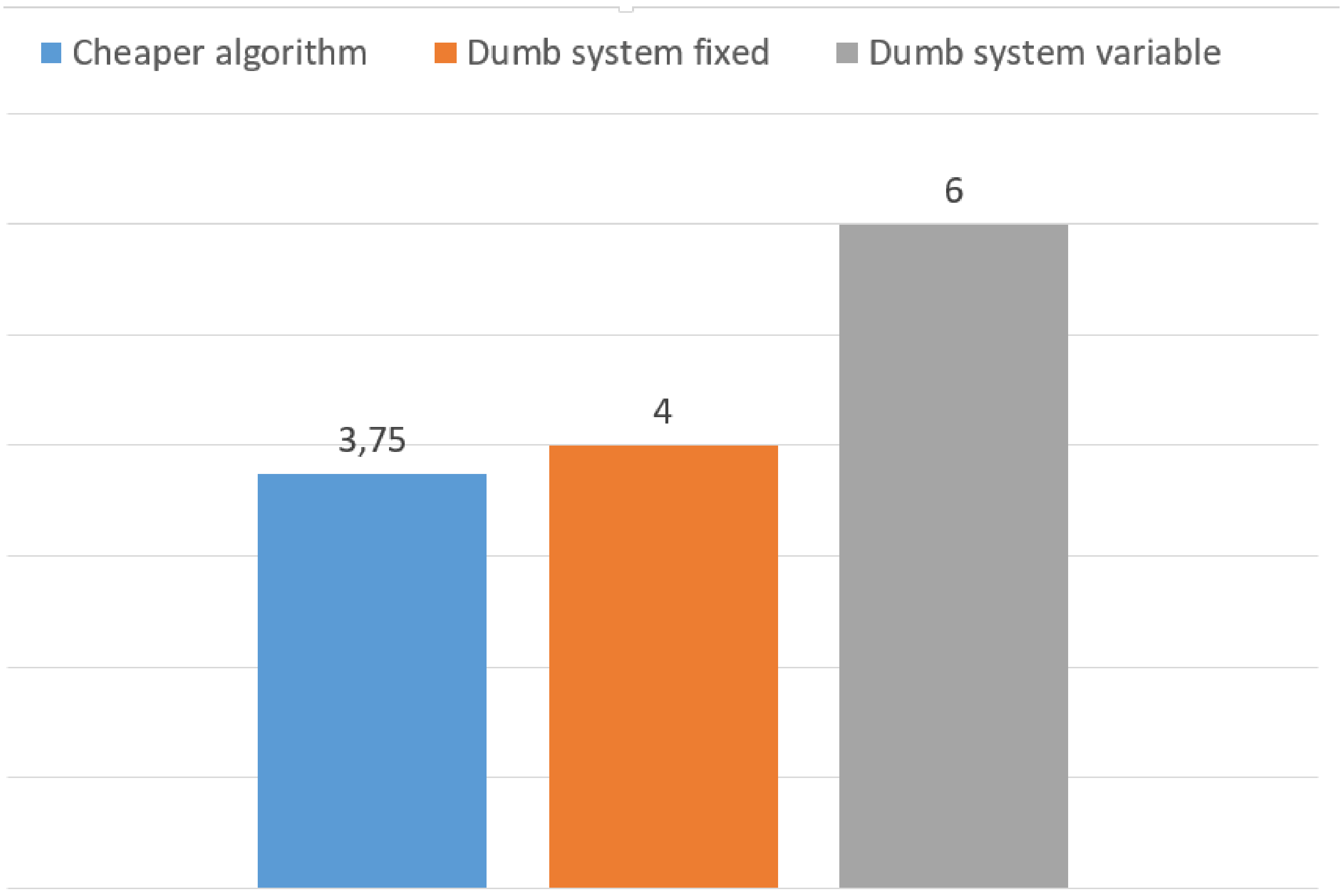


Figure 5.3: Average virtual machine Experiment 1

5.6 Experiment 2

In this experiment there are a random users with a random number of requests , where each users has a gaussian distribution and the payload of the request (so of the task) are equiprobabile between the 0 and 100 megabyte. The quos of the cheaper algorithm is quite good because the creation of the virtual machine is done when the system is suffering and when the virtual machine will become useless the cheaper algorithm will remove the virtual machine not utilized.

5.7 Experiment 3

In this experiment there are a lot of request, 1000000 request with a variable payload of microservices with equiprobable distribution between 10 MB to 1 GB.

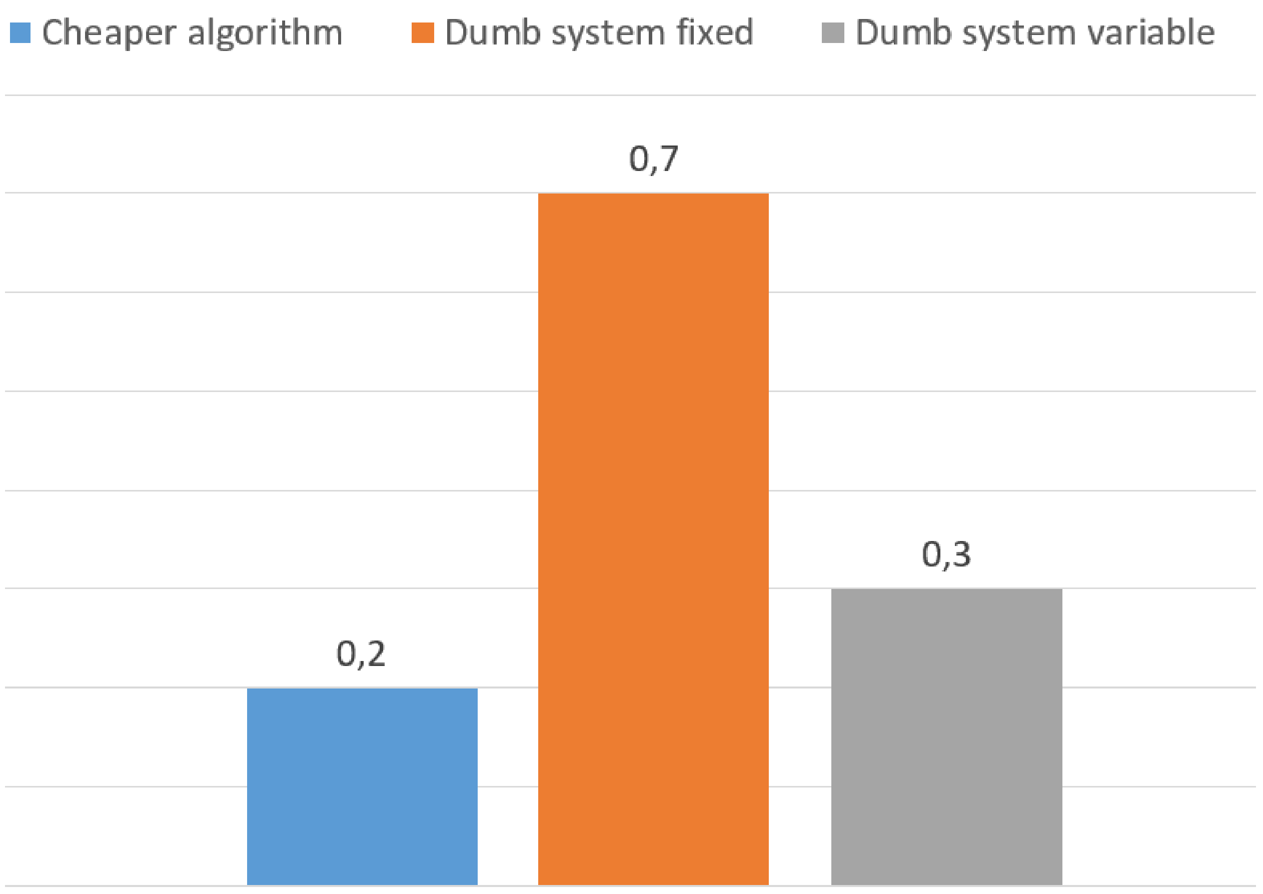


Figure 5.4: Average virtual machine Experiment 1

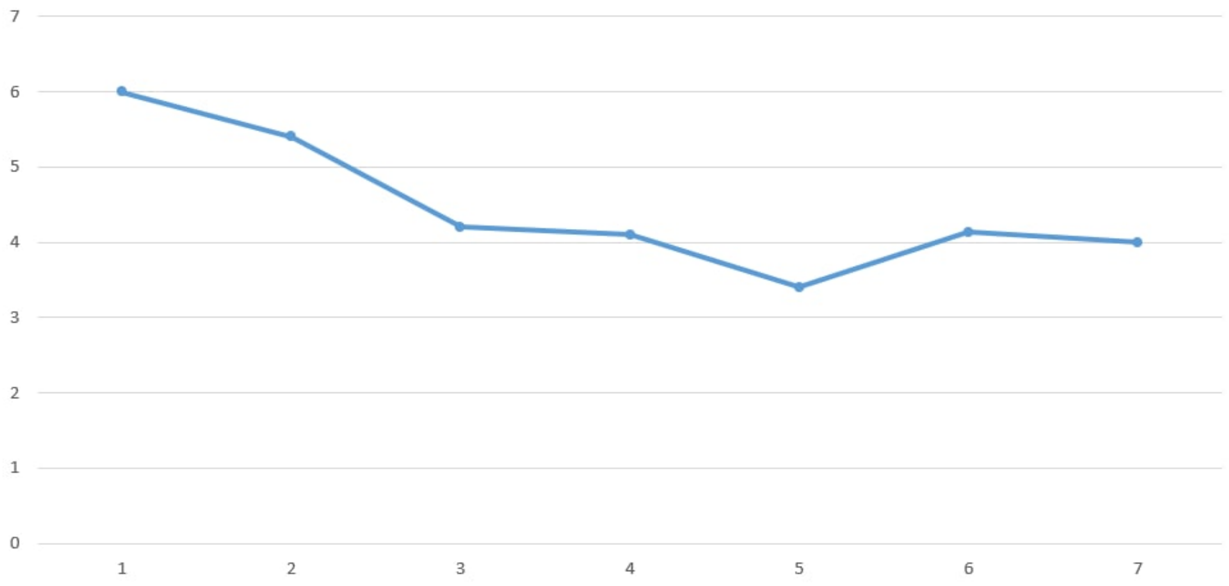


Figure 5.5: Average virtual machine Experiment 1

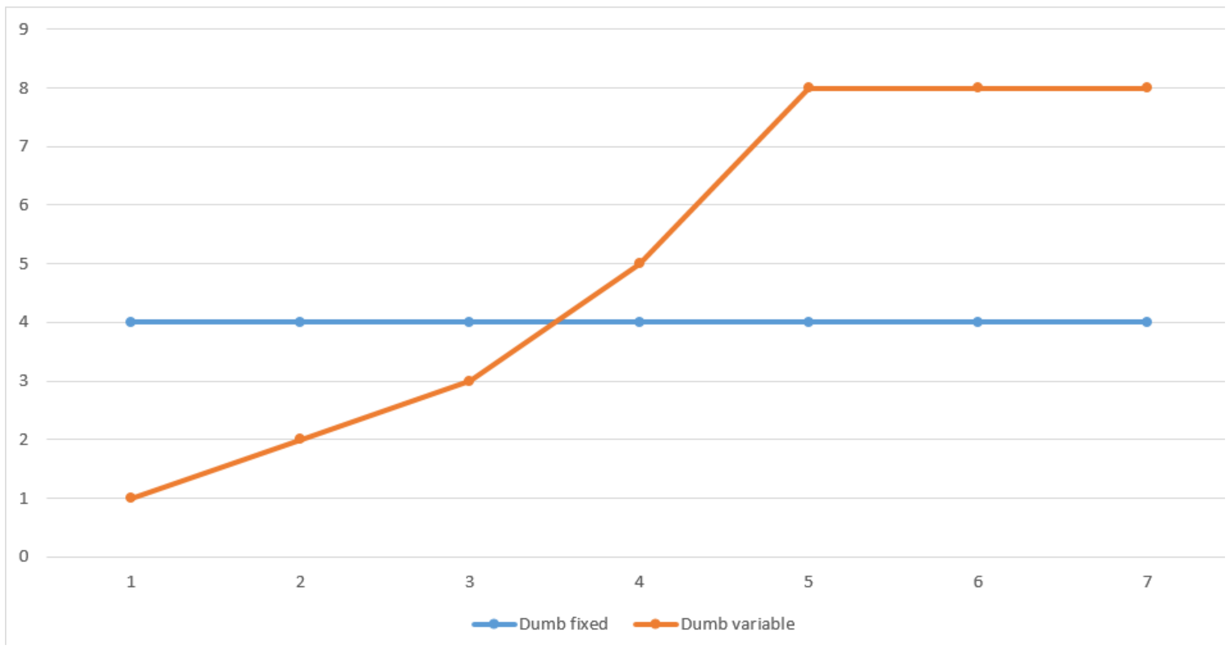


Figure 5.6: Some istant of the computation experiment 1 of dumbs system

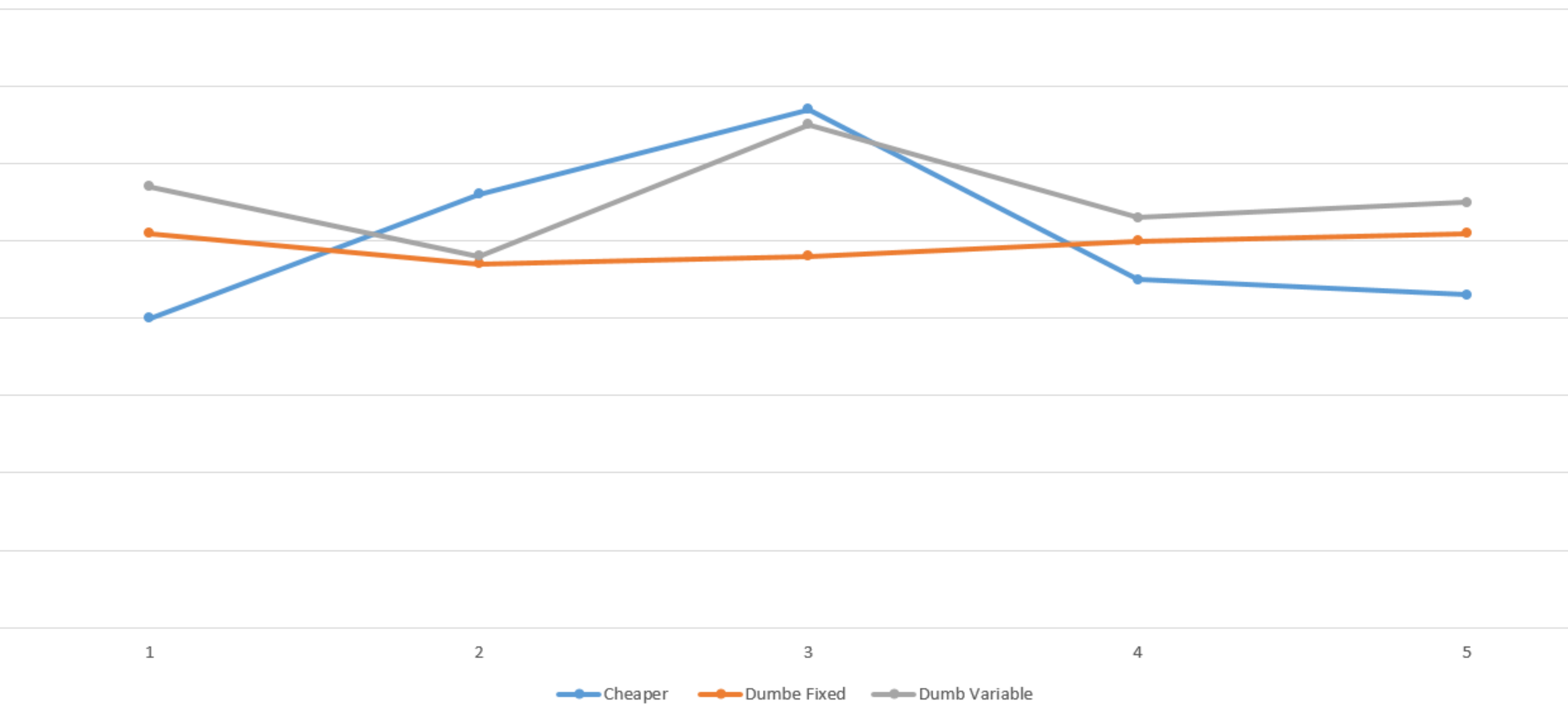


Figure 5.7: Some istant of the computation experiment 1 of systems

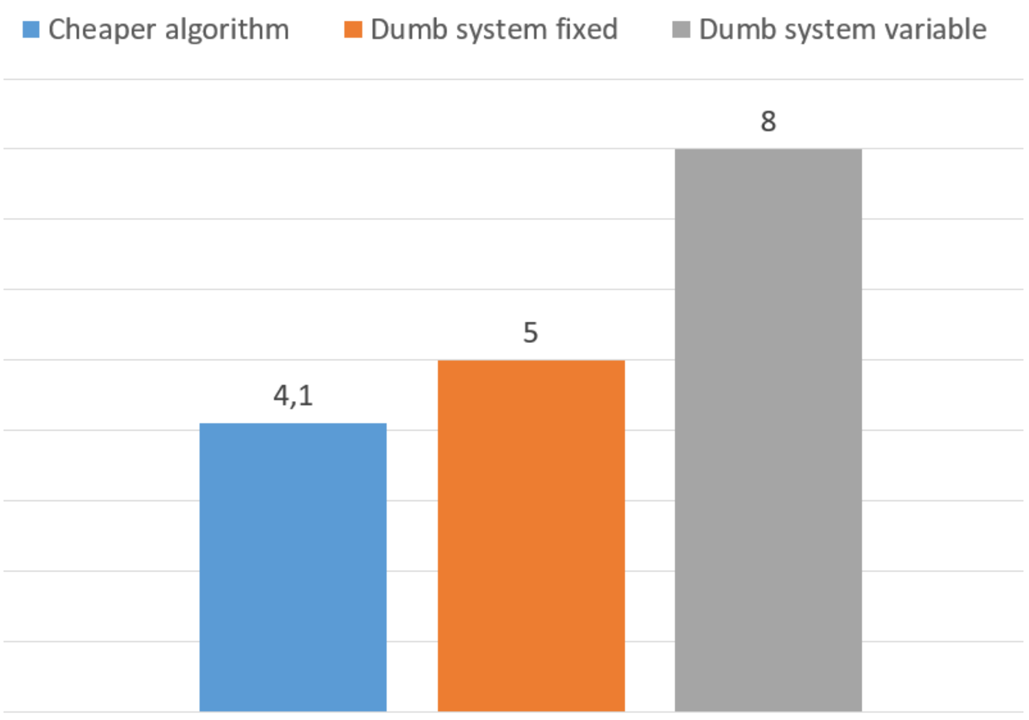


Figure 5.8: Average virtual machine Experiment 2

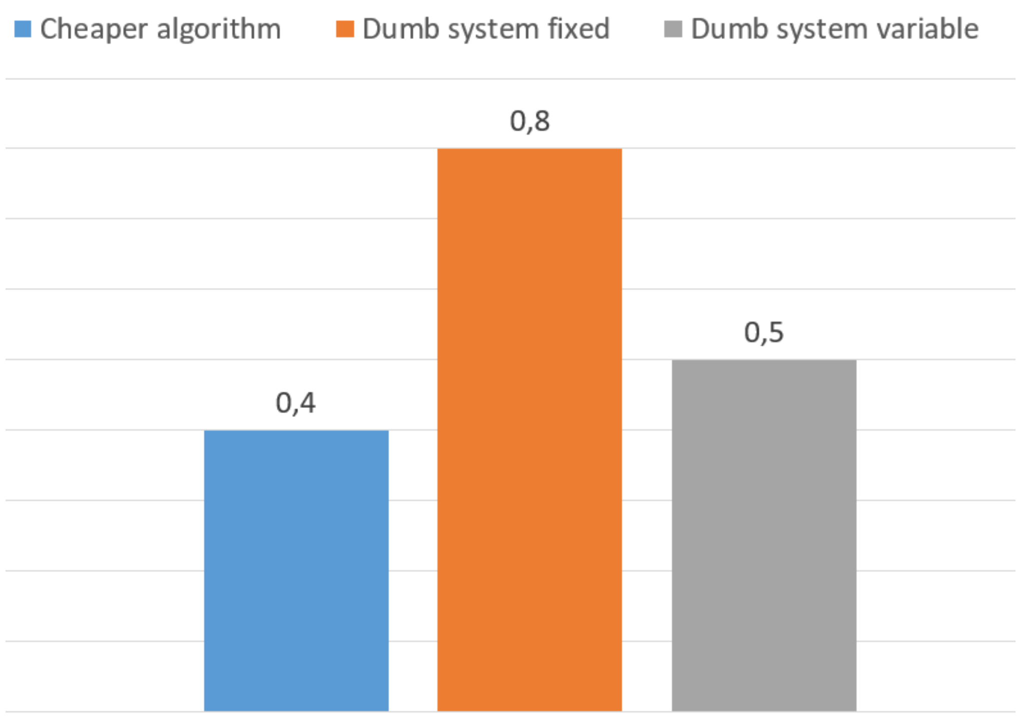


Figure 5.9: Average virtual machine Experiment 2

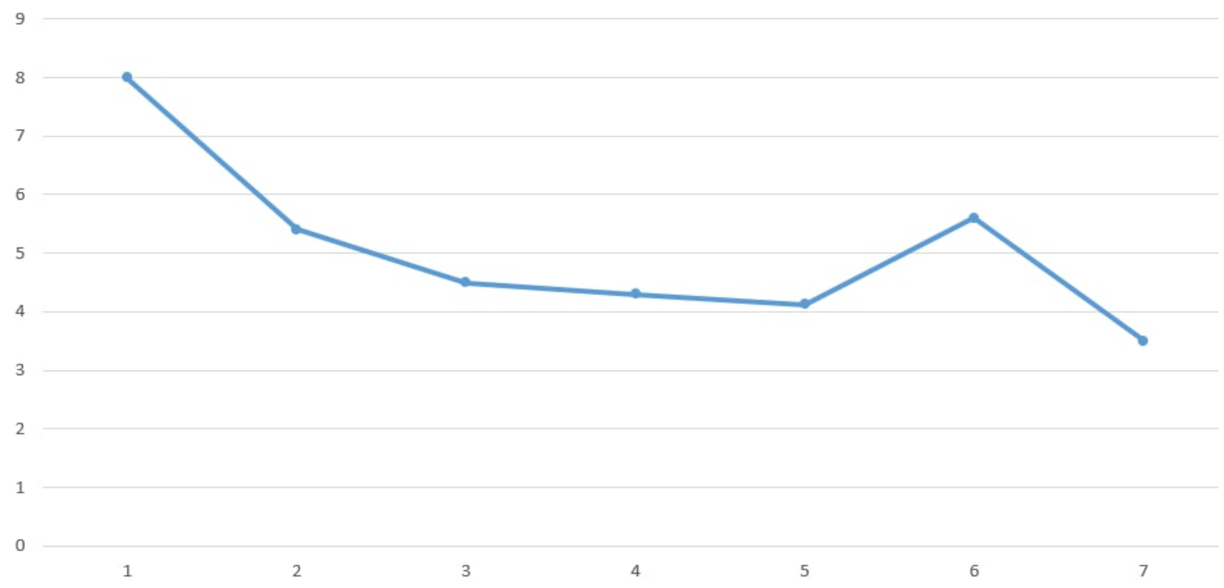


Figure 5.10: Average virtual machine Experiment 2

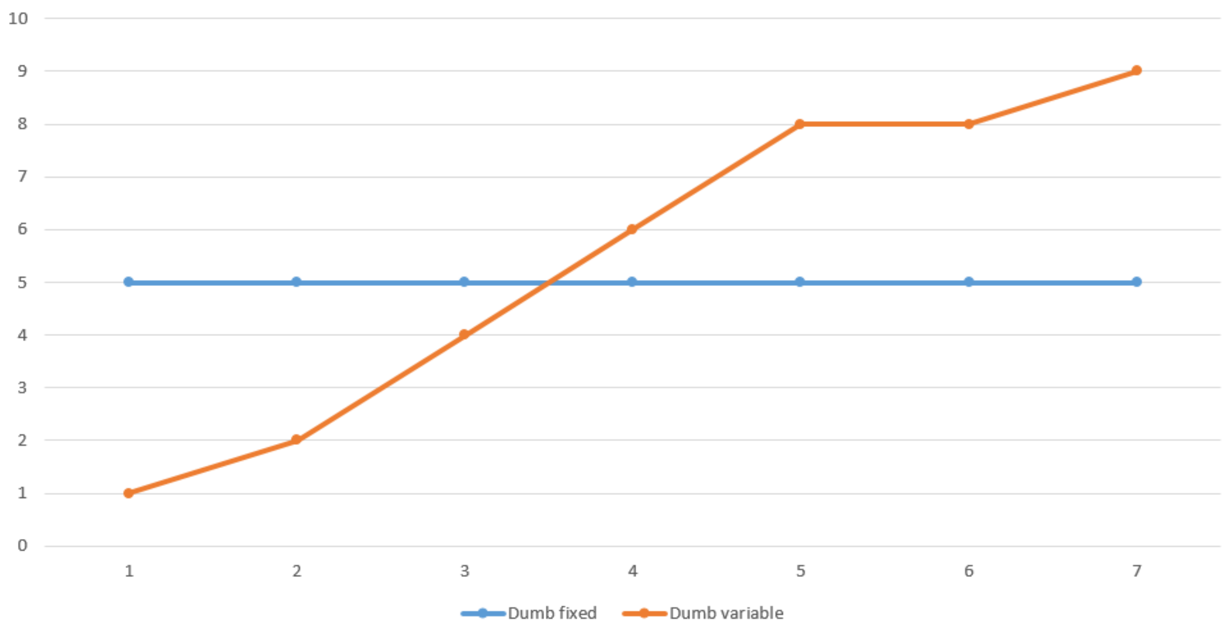


Figure 5.11: Some istant of the computation experiment 2 of dumbs system

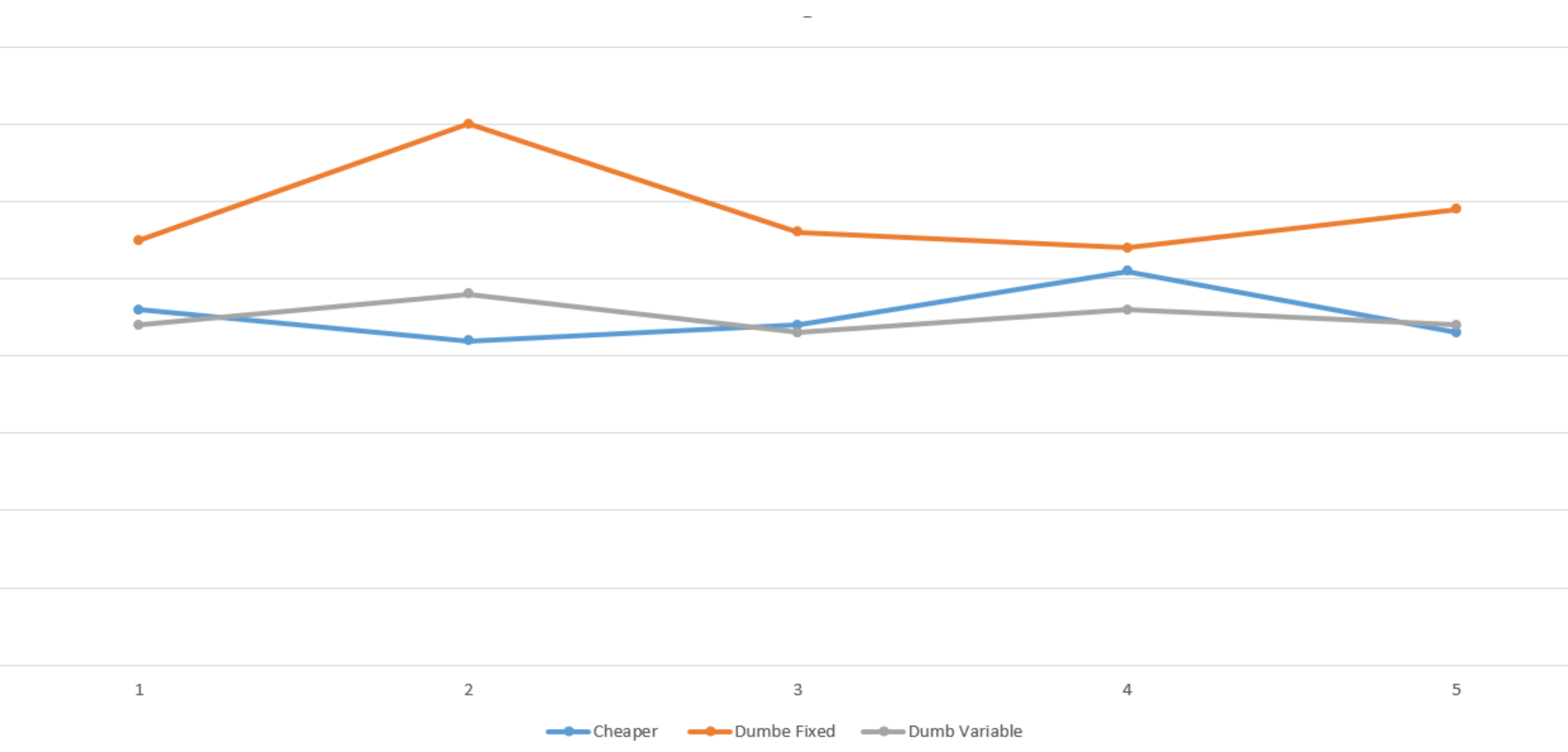


Figure 5.12: Some istant of the computation experiment 2 of systems

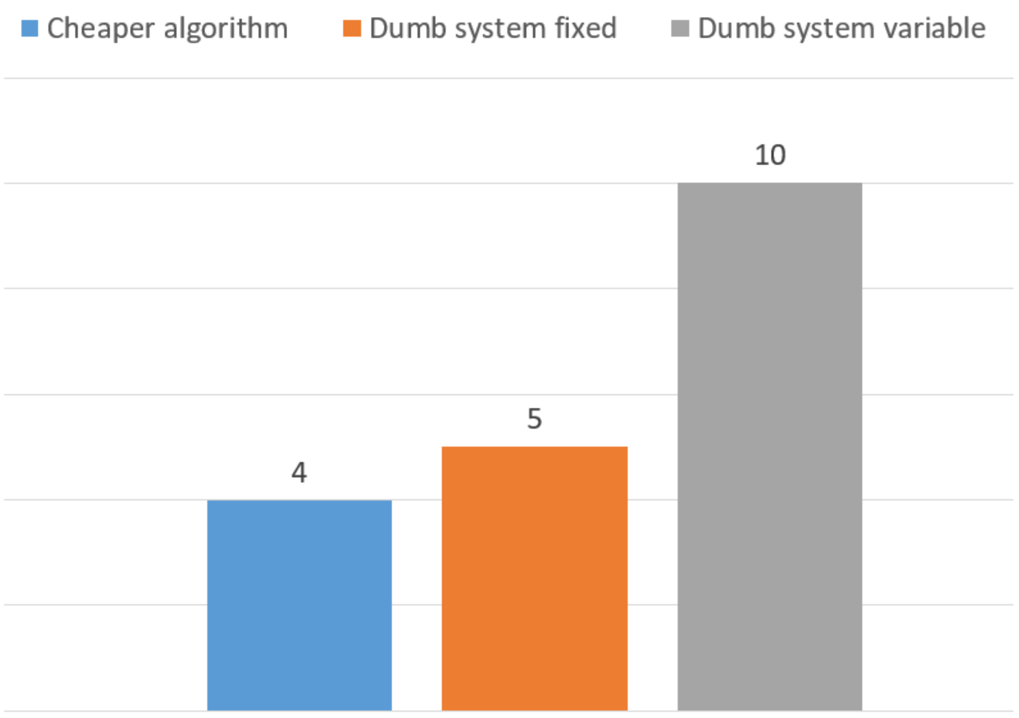


Figure 5.13: Average virtual machine Experiment 3

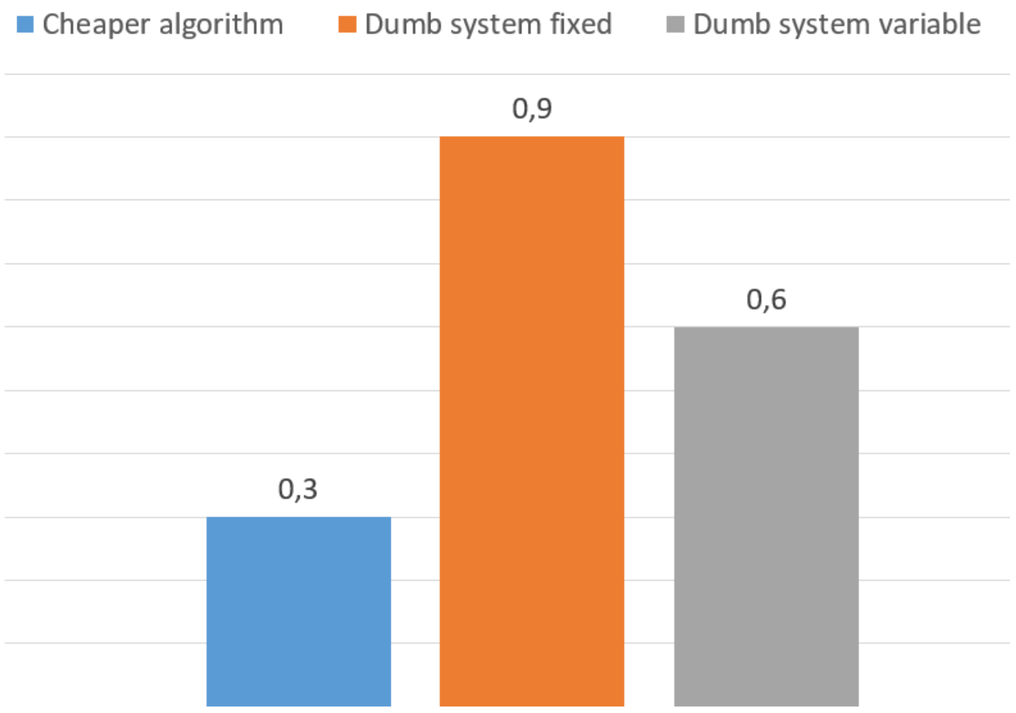


Figure 5.14: Average virtual machine Experiment 3

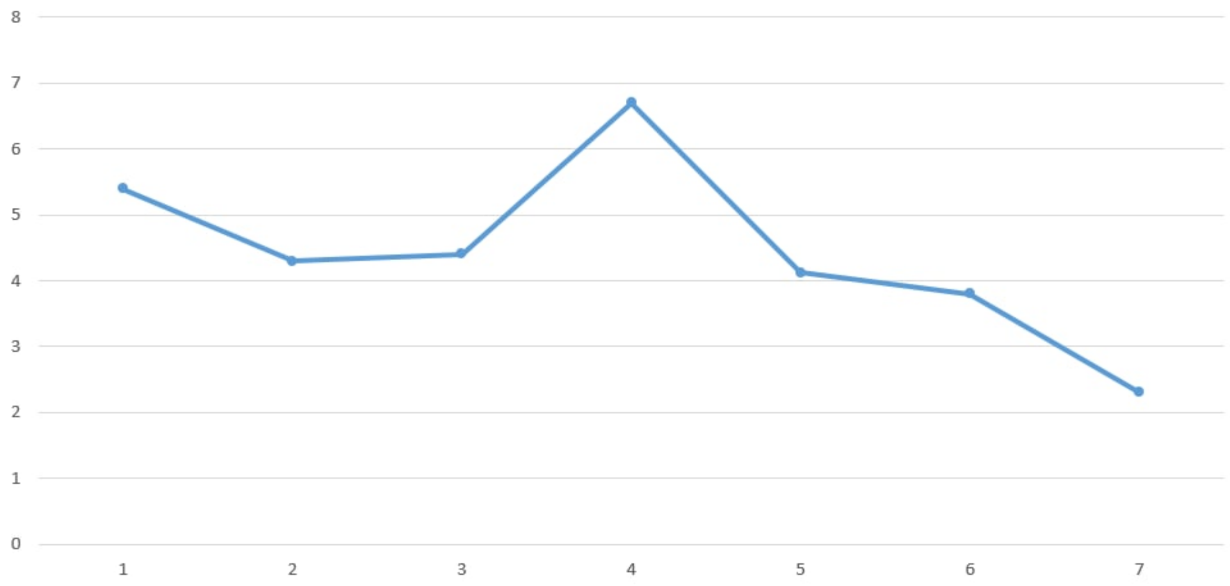


Figure 5.15: Some istant of the computation experiment 3 of cheaper algorithm

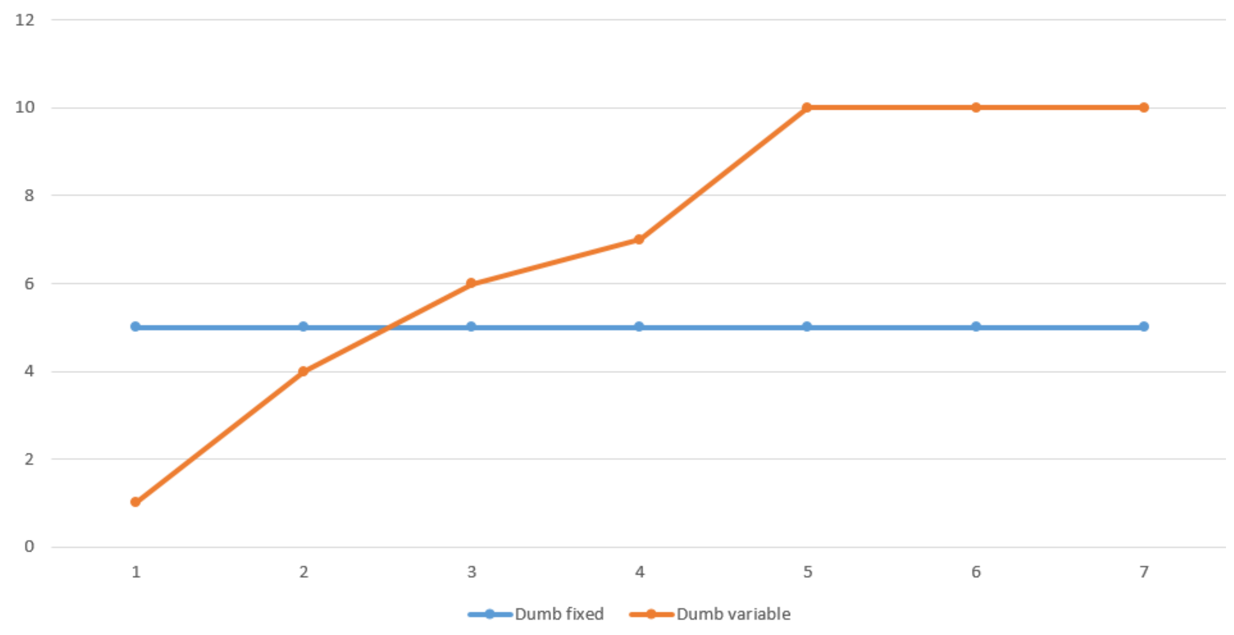


Figure 5.16: Some istant of the computation experiment 3 of dumbs system

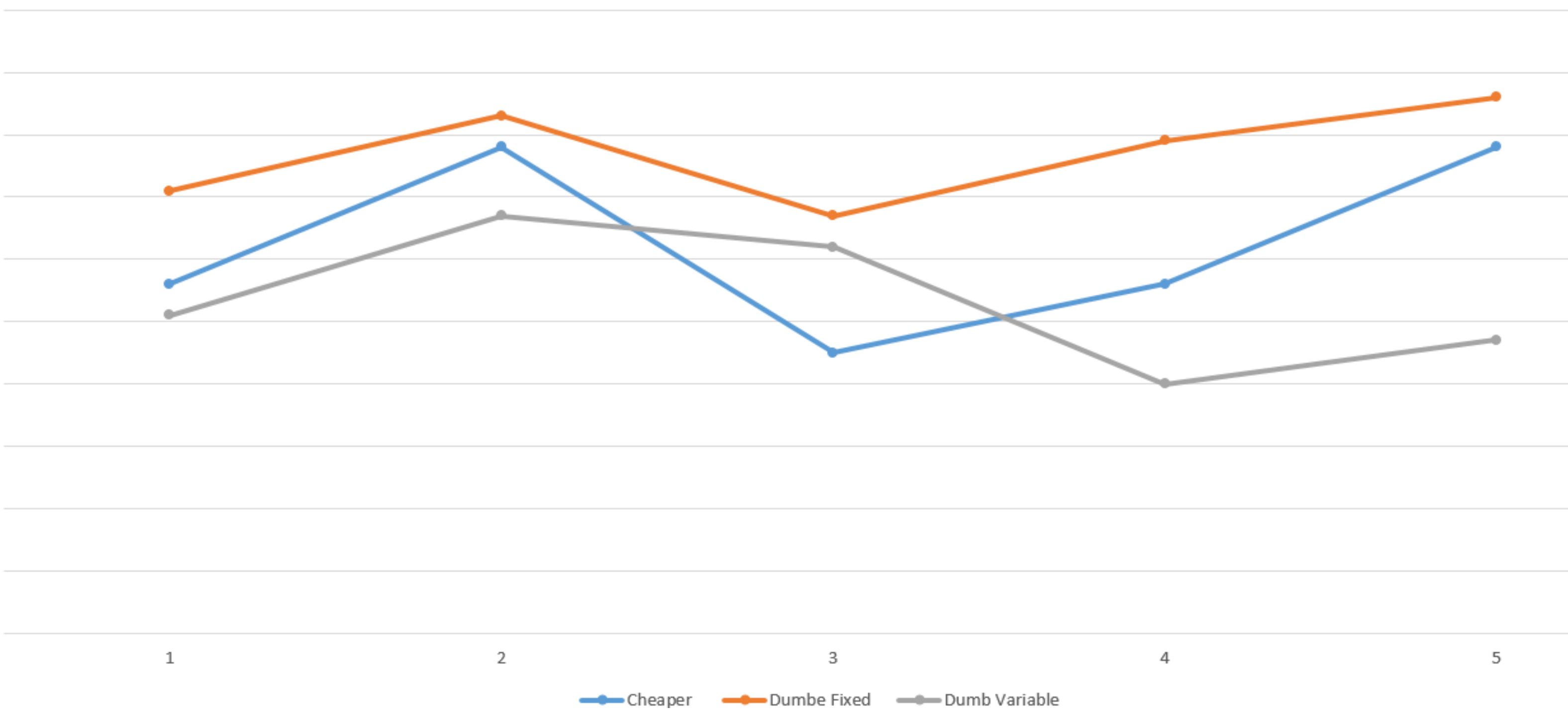


Figure 5.17: Some istant of the computation experiment 3 of systems

5.8 Interpretation of the result

All the statistics show that the algorithm schedule and makes efficent the scheduling of the microservices and the power computation utilized. The right world to utilize in this work is not optimized because the improvement is not too much as the reader can see. It is near to the result of the dumb system fixed in the creation of virtual machine and is not so far from the quos with the variable system. I have chosen this two types of dumb system because are similar to the system used in this work and are some indicator to see if the system does a littlbit of improvement.

5.9 Bibliography

The paper number 1 is the paper where i take ispiration for the work.

[1] A novel approach for multi-constraints knapsack problem uing cluser particle swarm optimization R.G. BabukarthikChandramohan DhasarathanManish KumarAchyut ShankarSanjeev ThakurXiaochun Cheng

[2] Scheduling gangs in a distribuited system Helen D.Karatza

[3] An improved genetic algorithm using greedy strategy toward task scheduling in cloud enviroments Zhou Zhou, Fangmi Li, Huaxi Zhu, Houlian Xie, Jemal H. Abawajy, Morshed U. Chowdhury.

[4] LBPSGORA: Create load balancing with particle swarm genetic optimization and energy consumption in clouds networks Seyedh Maedeh Mirmohseni, Amir Javadpour and Chunming Tang

[5] M. Assi and R. A. Haraty, "A Survey of the Knapsack Problem," 2018 International Arab Conference on Information Technology (ACIT), 2018, pp. 1-6, doi: 10.1109/ACIT.2018.8672677.

[6] Elastic Scheduling for Microservice Applications in Clouds Sheng Wang , Zhijun Ding , Senior Member, IEEE, and Changjun Jiang

[7] Comparative Research on Genetic Algorithm Particle Swarm Optimization and Hybrid GA-PSO Jyoti Sharma and Ravi Shankar Singhal

[8] L. Ouyang and Dongyun Wang, "New Particle Swarm Optimization algorithm for knapsack problem," 2012 8th International Conference on Natural Computation, 2012, pp. 786-788, doi: 10.1109/ICNC.2012.6234615.

[9] An Optimal Algorithm for Online Multiple Knapsack Marcin Bienkowski Institute of Computer Science, University of Wrocław, Poland

Maciej Pacut Faculty of Computer Science, University of Vienna, Austria

Krzysztof Piecuch Institute of Computer Science, University of Wrocław, Poland

[10] QoS-Constrained Service Selection for Networked Microservices ZHIJUN DING1,2, Senior Member, IEEE, SHENG WANG1,2, Meiqin Pan3

[11]The Evolution of Distributed Systems Towards Microservices Architecture Tasneem Salah, M. Jamal Zemerly, Chan Yeob Yeun, Mahmoud AI-Qutayri, Yousof AI-Hammadi

[12]A GRASP Algorithm for the Multi-Objective Knapsack Problem Dalessandro Soares Vianna* José Elias Claudio Arroyo** Universidade Candido Mendes – UCAM-Campos, Núcleo de Pesquisa e Desenvolvimento em Informática – NPDI , Campos dos Goytazazes, RJ 28040-320, Brasil *dalessandro@ucam-campos.br **jclaudio@ucam-campos.br

[13]CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms Rodrigo N. Calheiros1, Rajiv Ranjan2, Anton Beloglazov1, Cesar A. F. De Rose ´ 3 and Rajkumar Buyya1

[**14**] Essential of microservices architecture paradigms and application Chellammal Suria-narayanan Gopinath Ganapathy Pethuru Raj isbn : 978-0-367-24995-3

5.10 Future works

All this work is done in IAAS level but can be enlarged by the number of degrees of freedom , because in this work does not consider how the virtual machine are managed in the physical level, therefore to improve the system can be take care of where the virtual machine will be fit inside the datacenter of the cloud. Can be used other types of heuristics algorithm more efficent or some machine learning algorithm. In the first idea of this work is to use some machine learning alghortihm but for the reason that to create a dataset is difficult, to train all this algorithm , it has passed to the heuristics one. So another idea to improve this work is to think about the designed pattern of the microservices and make some little change to microservices infrastructure.

5.11 Ringraziamenti[ITA]

Ringrazio tutte le persone che mi sono state vicine in questo periodo, ringrazio in primis Chiara Bigarella di essere sempre stata al mio fianco in questi periodi e ringrazio anche Sandy Pivato e Gina Perron per avermi aiutato nel corregere gli errori ortografici e tutti coloro in cui in questi ultimi periodi di studio mi sono stati tanto vicini. Ringrazio la mia famiglia per avermi sostenuto in questa avventura e un rigrnaziamento a tutti i miei amici più cari, prevalememente a Gabriele De Pieri ed a Enrico Doni per avermi sempre sostenuto nei mommenti più bui. Graize al professore Carlo Ferrari per avermi aiutato in questa tesi e per essere stato il mio relatore.