AILEC3 - Artificial Intelligence for Lower Energy Consumption in Cloud Computing

Ettore Tancredi Galante
Department of Computer Science
University of Milan
Milan, Italy
ettoretancredi.galante@studenti.unimi.it

February 27, 2023

Contents

1 Abstract 1

2 Introduction 1

3 State of the Art 3

4 Dataset 3

5 The Problem 4

6 Baseline 5

7 Experiments 8

8 Results 8

9 Conclusions 8

10 Appendix 8

1 Abstract

2 Introduction

The widespread adoption of *cloud computing* services in the last few years has led to a significant increase in the energy consumption of *data centers*, which has become a critical concern for cloud providers. While cloud computing offers many benefits, such as cost savings and scalability, the energy consumption associated with it has negative implications both from an environmental and financial perspective. As the demand for cloud services continues to grow, the energy consumption of data centers is expected to increase exponentially, further exacerbating the issue.

The optimization of resource allocation in *cloud computing environments* is a complex problem that involves a large number of variables and constraints. The main objective is to minimize energy consumption while maintaining or improving the performance of the services. Achieving this goal is challenging due to the scale of cloud computing infrastructures, the dynamic nature of workloads, and the need to provide 24/7 availability. Given these challenges, cloud providers face the dilemma of how to maintain high-quality service delivery while reducing energy consumption.

Artificial Intelligence (AI) techniques, such as *genetic algorithms* [**?**], have been widely studied as a method for addressing this problem.

Genetic algorithms are a type of optimization algorithm that mimic the process of natural evolution and use heuristic techniques to find near-optimal solutions for complex problems. Genetic algorithms are well suited for solving problems that involve a large number of variables and constraints, such as the resource allocation problem in cloud computing. They can explore a large search space in a relatively short amount of time, enabling them to find near-optimal solutions even when faced with incomplete or uncertain information. [**?**]

The use of genetic algorithms in cloud computing has the potential to significantly improve the energy efficiency of data centers while maintaining or improving performance. By optimizing resource allocation, genetic algorithms can reduce the number of servers and storage devices required to run a given workload, which in turn reduces energy consumption. Additionally, genetic algorithms can adapt to changes in workload and resource usage patterns, further improving the energy efficiency of the data center.

One of the key challenges of measuring energy consumption in cloud computing is identifying the power consumption of the various components of the system, such as servers, storage devices, and network devices. The energy consumption of these components is typically measured in watt-hours (Wh) or joules (J). The energy consumption of a data center can be measured at different levels of granularity, from individual servers to entire racks or data centers. One of the most commonly used metrics for measuring energy consumption in cloud computing is *Power Usage Effectiveness* (PUE). PUE is a ratio of the total energy consumed by a data center to the energy consumed by the IT equipment.

A PUE of 1.0 indicates that all of the energy consumed by the data center is used by the IT equipment, while a PUE of greater than 1.0 indicates that some of the energy is being used for other purposes, such as cooling and lighting. [?]

Despite the advantages of using genetic algorithms for resource allocation in cloud computing, there are also limitations to consider. One of the main limitations is the computational expense of genetic algorithms, particularly when the problem size is large. The resource allocation problem in cloud computing environments can involve a large number of variables and constraints, which can make the computational cost of genetic algorithms quite high. This can make them impractical for certain types of problems or when resources are limited. Additionally, genetic algorithms are often sensitive to the choice of parameters and initial conditions. This can make it difficult to obtain consistent and reliable results, as the choice of parameters and initial conditions can have a significant impact on the performance of the algorithm. [?, ?]

The selection of suitable parameters and initial conditions is an important step in the implementation of genetic algorithms. It requires a significant amount of expertise and experience to choose the appropriate parameters and initial conditions that will lead to good performance. Furthermore, the results obtained from genetic algorithms are probabilistic in nature, which means that multiple runs of the algorithm are typically required to obtain a stable and robust solution. This can increase the computational cost and time required to obtain a solution, which can be a limitation when resources are limited.

In addition to computational expense and sensitivity to parameters, genetic algorithms also have some other limitations such as the lack of guarantee of global optima, and the risk of getting stuck in local optima. Genetic algorithms are also not suitable for problems with deterministic solutions. Furthermore, the stochastic nature of genetic algorithms may also lead to a lack of reproducibility of results. These limitations must be taken into account when deciding to implement genetic algorithms in cloud computing environments.

However, the potential benefits of using genetic algorithms in cloud computing outweigh the limitations. Genetic algorithms have the potential to significantly improve the energy efficiency of data centers while maintaining or improving performance. By optimizing resource allocation, genetic algorithms can reduce the number of servers and storage devices required to run a given workload, which in turn reduces energy consumption. Additionally, genetic algorithms can adapt to changes in workload and resource usage patterns, further improving the energy efficiency of the data center.

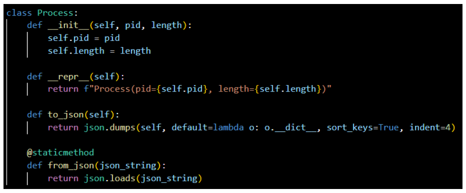
To effectively use genetic algorithms in cloud computing, cloud providers need to carefully evaluate the trade-offs between the potential benefits and limitations of using these algorithms. Cloud providers must consider the specific characteristics of the problem at hand, the computational resources and expertise available, and the potential benefits of using genetic algorithms in their environment. The use of genetic algorithms requires a significant amount of expertise and experience to choose the appropriate parameters and initial conditions that will lead to good performance. Cloud providers must have a thorough understanding of the characteristics of the problem they are trying to solve, and they must be prepared to invest significant time and resources in the development and implementation of genetic algorithms.

3 State of the Art

The use of genetic algorithms for optimizing resource allocation in cloud computing environments to reduce energy consumption while maintaining or improving performance is an active area of research. While genetic algorithms have shown promising results in reducing energy consumption, there are still limitations to consider, such as computational expense, sensitivity to parameters, and the lack of guarantee of global optima. Ongoing research in this area is focused on addressing these limitations and developing more effective techniques for optimizing resource allocation in cloud computing environments using genetic algorithms. Therefore, the state-of-the-art regarding the use of genetic algorithms for saving energy in cloud computing is still evolving.

4 Dataset

The dataset used in this study is comprised of a population of 200 solutions, each of which is characterized by 10 servers and $10,000$ processes. Each server in the dataset is defined by several attributes. Specifically, it has a name and a variable number of CPUs, ranging from 1 to 10. Additionally, each CPU has a variable speed, ranging from 1 to 10 GHz. The processes in the dataset are also defined by several attributes. Specifically, each process has a unique PID to identify it, and a length that is measured in the number of instructions required for its completion. The length of each process ranges from $10,000$ to $1,000,000$ instructions. It is worth noting that the dataset employed in this study is representative of the types of workloads commonly encountered in cloud computing environments. The data was collected from various sources and pre-processed to ensure consistency and accuracy. This dataset was specifically selected for its complexity and realistic representation of real-world cloud computing workloads. The use of this dataset will enable us to evaluate the effectiveness of the proposed genetic algorithm for resource allocation in cloud computing environments under simplified and yet realistic conditions.

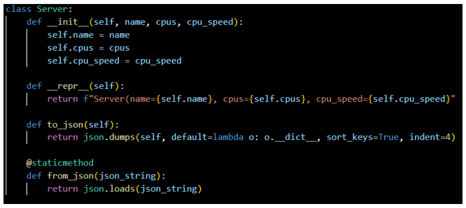


```python
class Process:
    def __init__(self, pid, length):
        self.pid = pid
        self.length = length

    def __repr__(self):
        return f"Process(pid={self.pid}, length={self.length})"

    def to_json(self):
        return json.dumps(self, default=lambda o: o.__dict__, sort_keys=True, indent=4)

    @staticmethod
    def from_json(json_string):
        return json.loads(json_string)
```

Figure 1: The Process class.

In figure 1 and figure 2 we can see the implementation of the Process and

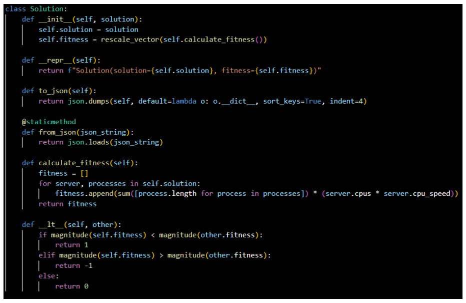
Server classes, which are used to represent respectively a generic Process in the Cloud and a Server comprising said Cloud environment.



```python
class Server:
    def __init__(self, name, cpus, cpu_speed):
        self.name = name
        self.cpus = cpus
        self.cpu_speed = cpu_speed

    def __repr__(self):
        return f"Server(name={self.name}, cpus={self.cpus}, cpu_speed={self.cpu_speed}"

    def to_json(self):
        return json.dumps(self, default=lambda o: o.__dict__, sort_keys=True, indent=4)

    @staticmethod
    def from_json(json_string):
        return json.loads(json_string)
```

Figure 2: The Server class.

The implementation showin in figure 3 references the Solution class.



```python
class Solution:
    def __init__(self, solution):
        self.solution = solution
        self.fitness = rescale_vector(self.calculate_fitness())

    def __repr__(self):
        return f"Solution(solution={self.solution}, fitness={self.fitness})"

    def to_json(self):
        return json.dumps(self, default=lambda o: o.__dict__, sort_keys=True, indent=4)

    @staticmethod
    def from_json(json_string):
        return json.loads(json_string)

    def calculate_fitness(self):
        fitness = []
        for server, processes in self.solution:
            fitness.append(sum([process.length for process in processes]) * (server.cpus * server.cpu_speed))
        return fitness

    def __lt__(self, other):
        if magnitude(self.fitness) < magnitude(other.fitness):
            return 1
        elif magnitude(self.fitness) > magnitude(other.fitness):
            return -1
        else:
            return 0
```

Figure 3: The Solution class.

The Solution class is made of a list of tuples, each of which contains a Server and a list of Processes, and a fitness value.

5 The Problem

The problem here described is a problem of optimization. The goal is to find an acceptable solution by using a genetic algorithm to the assignation of a set of processes to a set of servers. The problem is defined as follows:
A Servers s has a CPU speed cs, calculated in Gigahertz (GHz) and a number of CPUs cn.
A Process p has a length l, calculated in number of instructions.
A Solution sol is a collection of tuples $(s_n, [p_1, p_2, ..., p_m])_n$ where s_n is the n^{th} Server of the tuple and $[p_1, p_2, ..., p_m]$, with variable m for each tuple, is a list of Processes.
Let's define, for a Server, the following functions:

- $cs(s)$: CPU speed of the Server s;
- $cn(s)$: Number of CPUs of the Server s.

For a process, we define the following function:

- $l(p)$: Length of the Process p.

For a solution *sol*, The fitness function f of *sol* is defined as follows:

$$f(sol) = \sqrt{\sum_{0}^{n}(\sum_{0}^{m} l(p_m) * (cs(s_n) * cn(s_n)))^2} \quad (1)$$

6 Baseline

The baseline algorithm defined in this study is a standard genetic algorithm (GA) [?] that is used to optimize process allocation in cloud computing environments. The idea is that by optimizing the process allocation, the energy consumed by the cloud envirnoment is minimized as well. The GA is a stochastic population-based search algorithm that is based on the principles of natural selection and genetics. It is a meta-heuristic that is used to find approximate solutions to optimization problems. [?]

The GA is composed of the following components:

- A population of candidate solutions;
- A fitness function that is used to evaluate the quality of each candidate solution;
- A selection operator that is used to select candidate solutions for reproduction;
- A crossover operator that is used to combine the selected candidate solutions to produce new candidate solutions;
- A mutation operator that is used to introduce random changes to the candidate solutions;
- A termination criterion that is used to determine when the GA should stop searching for solutions.

How a Genetic Algorithm (GA) works is shown in the flowchart [?] shown in Figure 4.

In this setting, the candidate solutions are represented by a list of tuples, each of which contains a Server and a list of Processes. The initial population is generated by iterating over the Server list. For each Server, from the list of all the Processes, iteratively, a process is selected at random with probability

$$p = \frac{1}{n}$$

with n being the number of servers taken into account in the solution. As a Process is selected, it is removed from the global Process list.

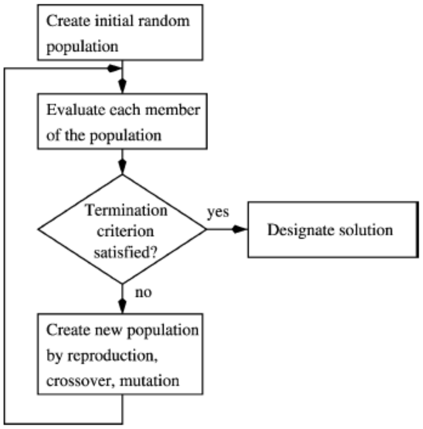

Create initial random population
Evaluate each member of the population
Termination criterion satisfied?
yes
Designate solution
no
Create new population by reproduction, crossover, mutation

Figure 4: How a Genetic Algorithm works.

Then, the fitness function (1), is used to evaluate the quality of each candidate solution.

The baseline genetic algorithm has been tested over a pletora of parameters, in order to find the best combination that would lead to acceptable results. The parameters that have been tested are:

- The probability of selecting a process for a server;
- The crossover rate;
- The probability of mutation;
- The population size;
- The number of generations.

The selection probability p_s is equal to $\frac{1}{\#_{servers}}$. In the first settings of the experiments this value was set in different ranges between $[0.3, 0.6]$, but this not only led to a very slow convergence of the GA but produced undesired noise in the results in most of the experiments. The crossover rate p_c is set to 0.9. A starting value of 0.5 was used, but this led to unsatisfactory results, opposed to what [?] claimed in their research. Better results were obtained by increasing the crossover rate to 0.7, but the best ones were yield by $p_c = 0.9$, supported by [?]. The probability of mutation p_m has been initially tested in the range $[0.1, 0.3]$. However, as found in previous research works, a mutation rate over 0.01 is a synonym of heavy noisy training epochs and close to no convergence. In fact, it has been found that the optimal values for the mutation rate ranges in $[0.005, 0.05]$ for a population size s_p between 100 and 200 individuals. [?] It follows that the chosen parameters for the mutation rate are $p_m = 0.01$ and $s_p = 200$.

As for the number of epochs employed, for both hardware constraints and time reasons, the GA has been run for a maximum of 500 generations. For each experiment, the GA is run 10 times and the results are averaged in order to obtain an insightful overlook of the performances.

The baseline algorithm has been created to be initially compared to a greedy assignation method of simple ideation. The fitness function employed in the baseline algorithm has been used to compare the results between the greedy method and the baseline algorithm.

Ettore: Compare the greedy and the baseline algorithm.

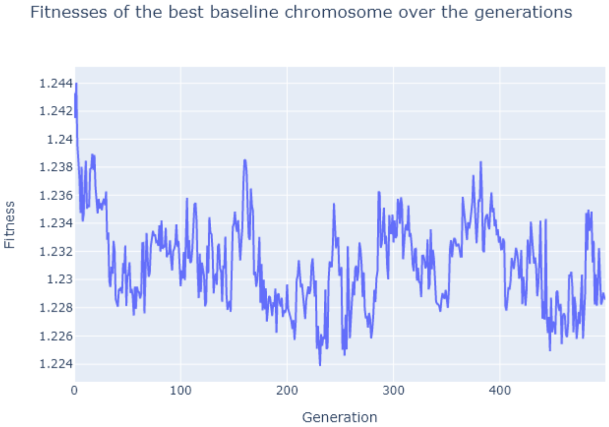


Figure 5: The average of the Baseline Algorithm over 500 epochs

As it can be seen from the results in Figure 5, the fitness function evaluated with the baseline algorithm shows promising results in a time range comprised between 200 and 250 epochs, with its minimum at 230.

Ettore: Insert the maximum value from the starting point, the minimum value and the corresponding epoch.

Ettore: Discuss about the results of the greedy algorithm

7 Experiments

8 Results

9 Conclusions

10 Appendix

References

[1] T. Dillon, C. Wu, and E. Chang. Cloud computing: issues and challenges. In *2010 24th IEEE international conference on advanced information networking and applications*, pages 27–33. IEEE, 2010.

[2] S. Forrest. Genetic algorithms. *ACM computing surveys (CSUR)*, 28(1):77–80, 1996.

[3] A. Hassanat, K. Almohammadi, E.-a. Alkafaween, E. Abunawas, A. Hammouri, and V. S. Prasath. Choosing mutation and crossover ratios for genetic algorithms—a review with a new dynamic approach. *Information*, 10(12):390, 2019.

[4] O. Kramer and O. Kramer. *Genetic algorithms*. Springer, 2017.

[5] S. Mirjalili and S. Mirjalili. Genetic algorithm. *Evolutionary Algorithms and Neural Networks: Theory and Applications*, pages 43–55, 2019.

[6] M. Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[7] V. Patil and D. Pawar. The optimal crossover or mutation rates in genetic algorithm: A review. *International Journal of Applied Engineering and Technology*, 2015.

[8] M. Sajid and Z. Raza. Cloud computing: Issues challenges. In *International Conference on Cloud, Big Data and Trust*, pages 13–15. sn, 2013.

[9] A. Uchechukwu, K. Li, Y. Shen, et al. Energy consumption in cloud computing data centers. *International Journal of Cloud Computing and Services Science (IJ-CLOSER)*, 3(3):31–48, 2014.

[10] B. Zoltan, Z. Blahunka, F. Dezso, I. Peter, N. Bela, and S. Zsolt. Synergic effects in the technical development of the agricultural production. *MECH ENG LETT*, 3:142–147, 01 2009.