Applying Genetic Algorithms to Resource Allocation in Cloud Computing: A Review

Andrew Pozzuoli
Brock University
St. Catharines, ON
6017735
ap15yl@brocku.ca

***Abstract*—Cloud computing is an increasingly ubiquitous way of distributing software without the need for installation on client computers. Due to the offloading of memory and resource usage, this choice has become an increasingly popular way for businesses and individuals to use software. Users access the software via a cloud server hosted in a cloud data center that allocates resources to a virtual machine (VM) for the end user. The problem of allocating resources to these VMs in order to ensure a high quality of service and resource utilization while also minimizing power consumption, deployment cost, and memory allocated in order to keep the cloud servers free for a high number of users has been tackled in many different ways with various scheduling algorithms.**

Genetic algorithms are algorithms that fall under the umbrella of evolutionary computation that are typically applied to optimization problems that require minimizing and maximizing many conflicting parameters. It achieves this by using concepts from Darwinian natural selection and survival of the fittest to evolve a solution to the problems.

This paper investigates a number of ways that researchers have applied genetic algorithms to the problem of resource allocation in cloud computing as well as genetic algorithm variants. All papers explored found that genetic algorithms tend to outperform the standard non-evolutionary schedulers at maximizing quality of service while minimizing costs and power consumption.

I. Introduction

Cloud computing has become an increasingly ubiquitous method for distributing software to users over the internet. It provides virtual computing resources to end users without software or hardware installation leading to reduced costs and improved resource utilization on end users [1].

While this resource offload has made cloud computing a popular choice for end users, the growing usage has amplified challenges for cloud service providers. One such problem is resource allocation where the cloud service provider must provide virtual resources to clients based on a number of often conflicting factors. These factors could include, for example, quality of service, customer pricing, resource utilization, energy efficiency, operating costs, and deployment cost [2]. While upgrading servers to allow for more virtualization is one way at tackling this issue, this is often unfeasible due to the cost associated with more servers making effective resource allocation a much sought after prize for service providers.

There is no simple algorithm for resource allocation. The only absolute way to find the best allocation is to check every possible allocation and compare them all. The time complexity of performing an exhaustive search through every possible allocation is factorial with respect to the number of virtual resources making this an NP-complete problem [3]. Even for a small number of virtual machines and physical machines the time to find the exact optimal solution can take well beyond what is acceptable in the time frame needed to allocate the resources. Data centers can have thousands of these to allocate so exact solutions are no longer feasible to search for. Rather algorithms that use heuristics are preferred to get a solution that is "good enough" in the time it takes to allocate.

A number of heuristic search algorithms have been used towards allocating resources such as first fit, best fit, round robin, etc., but these are often susceptible to local optima solutions where solutions can seem the best in a local area but are worse when considering the entire search space of possible solutions [4]. Take for example a hill-climber algorithm that only seeks to travel to the highest point in a search space. As soon as taking a step in any direction lowers the altitude, the algorithm stops. Now consider the situation where the hill-climber has reached the top of the smallest hill in the search space. Any step in a direction would be going down the hill but there are bigger hills in the search space that can never be travelled to due to the greediness of the algorithm. This is a fatal flaw for greedy algorithms and some heuristic search algorithms.

Genetic algorithms (GAs) have been used as powerful optimizing tools to approach various NP-complete problems. The idea behind genetic algorithms is to search a problem space using concepts based on natural evolution such as crossover, mutation, selection and inheritance [5]. This mimicking of nature to find solutions can often yield results that conventional algorithms could never reach. It is also able to get around the greediness problem of other heuristic search algorithms by including randomness in the form of mutation in order to explore more of the search space. This push and pull between exploring the search space and exploiting good solutions is what makes GA a powerful tool for optimization.

This paper explores how genetic algorithms have been used to tackle the problem of resource allocation in cloud computing. It will go over the background concepts in more depth before exploring how GAs have been applied in a number of papers. This paper also looks at GA variants and how they have fared in this problem area including any

improvements made over conventional genetic algorithms. It is shown that GAs generally perform better at resource allocation than current algorithms used and can be tuned to optimize different parameters depending on what the cloud service provider wants out of their allocation however they have not been implemented in any real-world cloud resource schedulers likely due to high runtime and the specialized knowledge base required to develop and maintain it.

II. Background and Related Work

A. Cloud Computing

Cloud computing is a combination of various resource hosting servers stored and connected in a data center. These resources are provided virtually to users over the internet. Cloud service providers are able to offer these resources in a subscription-based model that allows clients to pay for the resources depending on use rather than needing to purchase and install software on their own machines, freeing up computation on the user end [2]. Additionally, this allows developers to host web applications through a cloud server eliminating the need for a physical server for hosting [5]. This results in a lower cost and improved resource utilization on end users making it a popular choice with increasing interest [1]. This growing popularity makes keeping up with demand difficult for service providers who are unable to afford more servers or data centers and instead need to look toward more efficient resource allocation to accommodate more users.

Resources in the cloud are allocated by assigning virtual machines (VMs) to physical machines (PMs). Assignment is decided by a scheduler which has access to all request information and server information in order to make its decisions. A typical architecture can be seen in figure 1 from the paper *An Approach for Cloud Resource Scheduling Based on Parallel Genetic Algorithm* (Zheng et al., 2011) [4]. The figure displays an abstract representation with a cloud core and cloud scheduler as well as some underlying drivers for virtualization, network, and storage.

B. Genetic Algorithms

Genetic Algorithms (GAs) fall under the realm of evolutionary algorithms based on concepts from natural evolution and are often used to tackle optimization problems [3].

The main algorithm follows the structure of beginning with an initial population of randomly generated solutions (chromosomes). Each solution is associated with some fitness function that determines how close it is to the optimal solution. Usually the aim is to either maximize or minimize the result of this function. Individuals are selected to pass on parts of their solution (genes) depending on their fitness with more fit individuals more likely to be selected. Once selected, two individuals participate in crossover where a new child solution is generated by recombining the chromosomes of the parents. This part of the genetic algorithm takes advantage of exploiting good solutions to make better ones. Once generated, this child solution has a chance for any gene of their chromosome to be mutated randomly. This part allows the algorithm to explore more of the search space making it robust against getting stuck in local optima. Typically, the genetic algorithm is tuned so that crossover is high and mutation is low to yield good results. This solution is then run through the fitness function and repeated until a new population is created and the process repeats for a number of generations. Sometimes elitism is added where a percentage of the best individuals are passed on to the next generation with no recombination or mutation ensuring that the best genes persist over the course of the algorithm. Usually repetition continues until solutions stop continuing to improve otherwise known as convergence. If the population converges onto a solution, generally the solution is a good solution however this could be just a local optimum in the search space rather than a global optimum and distinguishing this is an important problem to consider in any GA. The balance of exploiting good solutions in the crossover step and exploring more of the search space in the mutation step makes this a desirable optimizer and a powerful tool for NP-complete problems over greedy algorithms. If crossover is too low and mutation is too high then good solutions are not exploited and the algorithm is only marginally better than a random search [6]. A flow chart of a typical genetic algorithm is seen in figure 2 from the paper *A QoS-Aware and Energy Efficient Genetic Resource Allocation Algorithm for Cloud Data Centers* (Bakalla et al., 2017) [1].

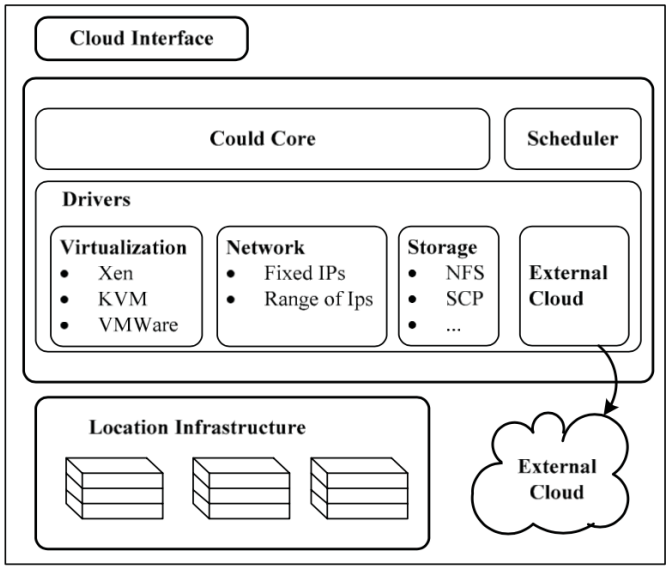

Fig. 1. Typical Cloud service architecture (Zheng et al., 2011) [4]

The most important things to figure out to implement a GA is the chromosome representation (how the solutions are represented) and the fitness function (how solutions are measured) which each of the papers explored outline.

C. The Problem of Resource Allocation

Resource allocation is the process of deciding how to share the host computing resources among virtual machines that users will access to use the resources. A good resource allocation strategy improves performance for both hosts and

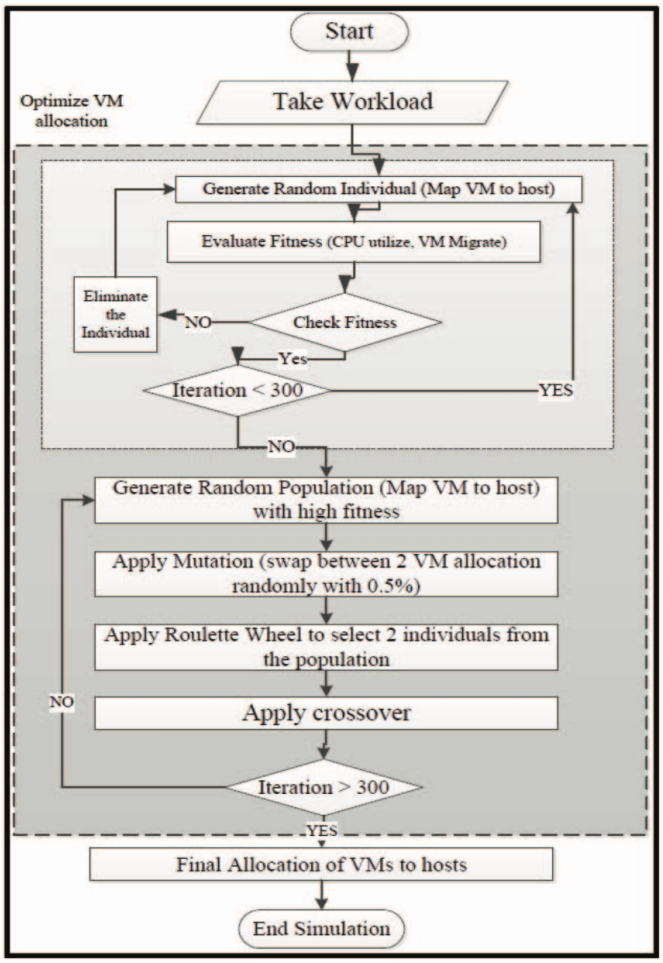

Fig. 2. Genetic algorithm for resource allocation flow chart (Bakalla et al., 2017) [1]

end users. Bad resource allocation can result in starvation of unused resources [6].

The main issues stem from a finite number of varying resources that need to be provided to end-users over the internet that ensures a fair distribution based on need while not allocating so much that resources are left unused. Too much resource allocation can also result in higher cost on the part of the host since the servers need to be kept running to provide the resources. In order to provide a good resource allocation, it is necessary to avoid conflicts over resource access (contention), limits in resources that do not match demand (scarcity), small resource entities preventing allocation (fragmentation), surplus of resources (over-provisioning), and shortage of resources (under-provisioning) [6].

In order to find the optimal resource allocation there is no better way than exhaustively searching every possible combination of resources making this an NP-complete problem. Even with a small number of servers and small number of services the amount of time it takes to find the exact solution is factorial with respect to input making it infeasible to find an exact solution [3]. Data centers can have thousands of servers and machines making the thought of finding an exact solution through exhaustive search an impossible dream.

D. Non-Evolutionary Solutions

Since an exact optimal solution is difficult to find, a number of different non-evolutionary algorithms have been used to find a ”good enough” solution. These algorithms often use a heuristic to search the space of solutions. As explained in the introduction section, a hill climber uses climbing to higher altitude as its search heuristic to find the highest point in a search space. One example of a heuristic search is a greedy algorithm called first fit which assigns resources to the first available node in need. Another algorithm is round robin where resources are allocated in turns which provides more fairness [5].

There are also schedulers that provide their own resource allocation not based on a specific algorithm but on a more dynamic allocation strategy that can consider more parameters. One example is the OpenNebula default scheduler which allocates resources based on a ranking system that is tuned by an administrator [4]. Eucalyptus and Nimbus are two more schedulers for resource allocation. Eucalyptus has three policies for administrators to choose: first fit, round robin, and power save. Nimbus offers customizable queuing and hierarchical job scheduling tools. [5].

Predictive methods have also been used for resource scheduling. CPU utilization prediction uses linear regression to predict utilization based on host usage history. It predicts overloading and underloading of hosts in the short future to reduce violations in service quality [1].

Some other tools for resource scheduling include AWS OpsWorks, IBM Smart Cloud, and Right Scale which consider workload and deployment cost, scaling services based on reducing cost and demand as heuristics [2].

E. Related Work

Similar work of surveying current resource allocation strategies in the cloud has been done before. Kandan & Manimegalai presented a comprehensive survey of resource allocation strategies including those used commercially and their benefits and drawbacks (2015) [7]. Alnajdi et al. also presented a review in 2016 with a focus on dynamic allocation techniques [8]. Kandan & Manimegalai mentioned one genetic algorithm in their paper with benefits being in resource utilization and price. Alnajdi et al. did not specifically look at genetic algorithm for resource allocation, they did discuss the topology aware resource allocation (TARA) scheme which uses a genetic algorithm based search alongside its prediction engine. This literature review differs from existing reviews in that it specifically focuses on genetic algorithms for resource allocation alone.

III. Genetic Algorithms for Resource Allocation

A. Efficient manager for virtualized resource provisioning in Cloud Systems (Apostol et al., 2011) [9]

In the 2011 paper, *Efficienct manger for virtualized resource provisioning in Cloud Systems* (Apostol et al.), a new

mechanism for cloud resource provisioning was presented implementing a genetic algorithm. This implementation took into account cost, time, and physical resources when allocating resources.

One of the major improvements the paper found over static provisioning was that GA was flexible and allowed for on-demand elasticity. In this way, the elasticity could be tuned depending on the policies of the service provider.

The paper constrained the algorithm to different quality-of-service policies in order to simulate how different policies affect the dynamic provisioning.

1) Fitness function: The fitness function was a weighted sum of RAM, CPU, and Disk with the goal of allocating in a way that minimized this function.

2) Conclusions: The main contribution of the paper was introducing a new way of provisioning resources in the cloud separate from the current static methods. The GA was able to dynamically allocate resources based on demand or well defined policies and it showed that GA could be something worth exploring to tackle this problem.

B. Efficient Resource Allocation in Cloud Data Centers Through Genetic Algorithm (Arianyan et al., 2012) [5]

The 2012 paper, *Efficient Resource Allocation in Cloud Data Centers Through Genetic Algorithm* (Arianyan et al), researchers further explored using a GA for resource allocation in Cloud. They noted that most resource scheduling algorithms only considered RAM, CPU, and Disk parameters but that there are more useful parameters to consider when scheduling that genetic algorithms with their power for searching multi-dimensional problems would be able to balance.

The paper evaluated two different resource allocation strategies. The first was to allocate virtual machines (VMs) onto physical machines (PMs) that best matched the demand for resources. The second was to allocate VMs to PMs with more available resources.

1) Chromosome Representation: Solutions were represented as an array where the index i corresponded to the ith VM. The cells of the array would contain an index j that corresponds to the PM that the VM i is allocated onto. For example, if VM_1 and VM_2 were both allocated onto PM_1, then cells 1 and 2 would both contain the index j=1. Most of the papers that follow this section would use this idea for their chromosome representations as well.

2) Fitness function: The fitness function was a weighted sum of allocated resources taking into account their parameters such as availability, temperature, and best fit ratio of VMs to PMs depending on which strategy was being evaluated. A total of fourteen parameters were considered which are CPU clock speed, number of CPU cores, CPU bus width and speed, node RAM capacity, RAM access speed, cache-hit ratio, cache access time, capacity of disk, hard access time, network bandwidth, temperature of PMs, VM quality of service (QoS), and number of VMs on each PM.

The paper found that in most cases, CPU power was the most important factor in decision making. Temperature was another important parameter with sudden increases being a factor in predicting failure. Quality of service (QoS) was dependent on how much the user paid for their quality so users who paid more had to have a higher quality of service introducing another dimension to this objective rather than straight maximization.

In general the parameter weight did seem to have an effect on the resource allocation. One example of this is that the utilization of either the RAM, CPU, or Disk is directly related to the weight placed on them. The RAM utilization tended to go down as the weight went down for that parameter.

The output of the fitness function was a value whose goal is to be maximized.

3) Strategy 1, best fit: This scenario aimed to map the VMs to PMs with resources that more closely match the VM requests. It was found that this led to more vacant PMs which could be shut off when not in use leading to lower operational costs.

4) Strategy 2, most available resources: This strategy aimed to map VMs to PMs with more resources available to them. This led to a higher probability of VMs accessing the resources they needed which resulted in faster computational time.

5) Conclusions: It was found that the GA had a higher fitness value than three other static allocation algorithms. It was also found that the evolution time of the GA increased as the number of VMs increased since the algorithm had to run through more generations in order to find an optimal solution. This is true of all GA problems; higher population leads to higher runtime. In general, the GA enhanced resource utilization while lowering operational costs. In addition to exploring GAs further, it was shown that the weight of parameters should be considered since the weights have a significant effect on allocation.

C. A QoS-Aware and Energy Efficient Genetic Resource Allocation Algorithm for Cloud Data Centers (Bakalla et al., 2017) [1]

In the 2017 paper *A QoS-Aware and Energy Efficient Genetic Resource Allocation Algorithm for Cloud Data Centers* (Bakalla et al., 2017), a genetic algorithm was used for resource allocation that considered energy efficiency as well as quality of service (QoS) as its objectives. This rose from a trend of cloud green services that provide service level agreements that consider energy consumption. With the increase in cloud service demand, data centers have expanded which means that they consume more energy and with this an increase in demand for energy efficiency in cloud services followed. The paper noted that it is important to be careful with reducing energy consumption since it could negatively affect the service level on the user's end with less power leading to less quality.

1) Chromosome Representation: In this GA, chromosomes were represented as array mappings of VMs to PMs as seen in the previous section. The fitness function considered CPU utilization and VM migration with the goal of maximizing

this function with little CPU utilization and a small number of VM migrations while also reducing service level agreement violations (SLAVs), and consuming little energy.

2) Experiments: The experiments were conducted using the CloudSim toolkit to simulate workloads with 2 servers and 800 heterogeneous PMs. Energy consumption and SLAVs were measured and compared against four benchmarks. NonPowerAware is a benchmark that is allowed to use maximum power and Dynamic Voltage Power Frequency (DVPF) which varies frequencies and voltages with the goal of reducing power consumption. The other two benchmarks IdrMu and MadMu were used to compare SLAVs since the former two benchmarks do not consuder SLAVs. It was found the GA had a 65.82% reduction in energy consumption over NonPowerAware since NonPowerAware runs at maximum power. GA was also found to have similar energy consumption to DVPF which is designed for energy efficiency without considering quality of service. In terms of SLAVs, GA was able to efficiently reduce SLAVs over IqrMu, being 3.47% more efficient, and MadMu, being 2.34% more efficient.

3) Conclusion: The researchers concluded that GA was able to reduce SLAVs better than the compared benchmarks while reducing energy comparable to algorithms designed for energy efficieny. In short, it succeeded in balancing quality of service and energy efficiency.

D. Genetic algorithm for quality of service based resource allocation in cloud computing (Deverasetty & Reddy, 2019) [2]

The 2019 paper, *Genetic algorithm for quality of service based resource allocation in cloud computing* (Deverasetty & Reddy), shifted focus from how service providers can use GA to how customers can use GA to select the best pricing model for their cloud hosted web application.

Many cloud service providers offer different pricing models for different levels of resource provisioning for clients to subscribe to in order to deploy their web application. The myriad of pricing models makes selection a difficult task for clients and this paper looked to GA to make this selection easier.

The algorithm looked at balancing quality of service with low deployment cost in its GA. They outlined the main factors for good quality of service in cloud deployment as the response times for both the database and the computing instance. It took the resources offered by the service providers as input. It experimented with pricing models of the Amazon Web Services where users have to pay for different levels of CPU utilization, database instance utilization, IO requests and storage space. The problem also considered each customer's required level of service and budget.

1) Chromosome Representation: For this GA, the chromosomes were represented as two dimensional matrices of size ixj where i is the customer QoS requirements and j is the available resources.

2) Fitness Function: The fitness function was given as a weighted sum balancing deployment cost and response time.

3) Results and Conclusion: The performance of the GA was compared to the QCost algorithm and a conventional algorithm on the Amazon Web Service pricing models. It was found that GA was able to select a pricing model that efficiently reduced the response time of the database instance over the other algorithms.

IV. Genetic Algorithm Variants

The following section deals with papers who have taken the genetic algorithm approach to cloud resource allocation and added modifications to the standard algorithm to improve aspects of it or to explore the effects of these modifications.

A. An Approach for Cloud Resource Scheduling Based on Parallel Genetic Algorithm (Zheng et al., 2011) [4]

The 2011 paper, *An Approach for Cloud Resource Scheduling Based on Parallel Genetic Algorithm* (Zheng et al.) took the idea of a GA for resource scheduling and modified it with parallelism. They noted that GA is powerful for optimization problems such as resource allocation but they are computationally demanding due to the amount of time it takes to run many generations to produce a solution. To combat this issue, the paper proposed a parallel genetic algorithm (PGA).

1) Multi-Population Topology: In order to implement this, the researchers created a topology of nodes where each node is a separate population of individuals (these are known as demes). Each population carries out GA as normal but there is a chance that individuals can migrate from one deme to another. This allows more genetic variations since genetic material from demes can be introduced into other demes without dominating and settling on a local optimum.

Migration was carried out with the following factors:

- Topology was defined with some structure such as a hypercube, torus, etc.
- A migration rate controlling how much of the population migrates was specified.
- A migration scheme indicating which individuals migrate was defined.
- A migration interval determining frequency was defined.

This paper used a one-way ring topology, migrating ten percent of optimal individuals every ten generations replacing the worst individuals of the population they immigrate to. A visual is shown in figure 3 with a ring topology that is one-way and has four demes.

The benefits of this kind of parallelism is not limited to speed. These sub-populations can allow for each population to explore different areas of the search space although this paper did not specifically adapt their model to ensure this.

2) GA Operations: The paper identified three main steps in its scheduler. The first was setting an idle resource list and VM request list which are updated with each incoming request, each VM shut down, or with resource changes. The second step used the PGA to find an optimal allocation and the last step was to launch the PMs.

The fitness function of this PGA considered CPU (including core number for parallelism), memory, and disk capacity.

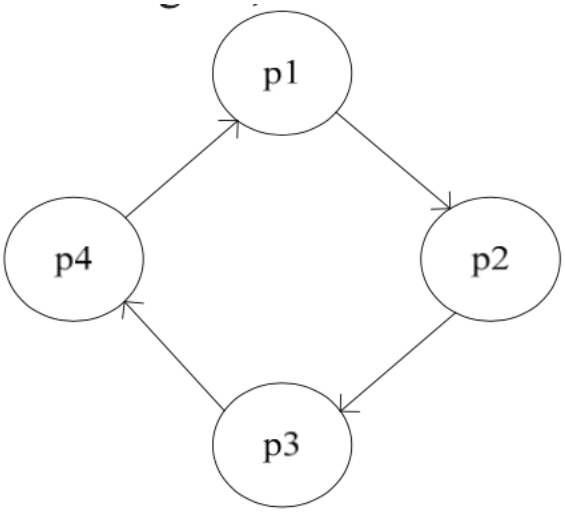

Fig. 3. Population migration topology (Zheng et al., 2011) [4]

3) Experiments and Conclusions: For experiments, the paper tested with a simulated workload and found that the PGA was faster than the GA at finding a solution. The PGA was found to be an average of 1.5-2.7 times faster. Runtime comparison can be seen in figure 4 where is is clear that PGA runs faster. Compared to static algorithms such as round robin or first-fit, the PGA outperformed in terms of utilization of allocated resources these results are compared in figure 5. This figure shows that PGA and GA share a comparable utilization rate that is better than round robin and greedy across the board.

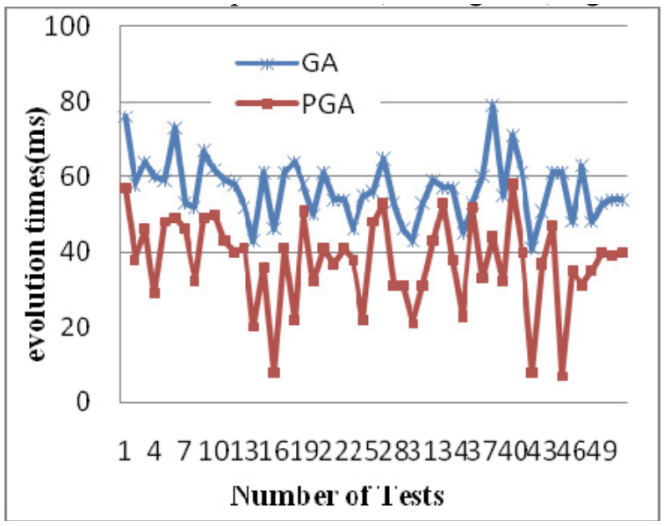

Fig. 4. Comparison of GA and PGA runtime (Zheng et al., 2011) [4]

B. An Extended Multi-Population Genetic Algorithm for Resource Allocation in Service Hosting Platforms (Shirali et. al, 2019) [3]

Shirali et al. implemented a self-adaptive, multi-population genetic algorithm for their take on the problem of resource allocation.

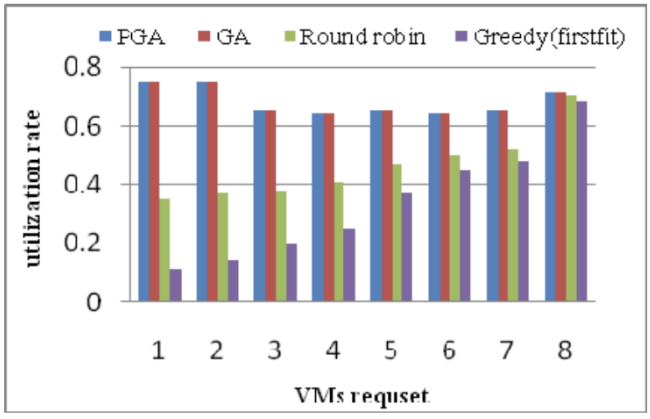

Fig. 5. Comparison of GA, PGA, Round Robin, and Greedy Algorithm Resource Utilization runtime (Zheng et al., 2011) [4]

The fitness function for this GA considered the objectives of quality of service, service fairness, increased utilization, and maximization of utility functions.

As with the previous paper, this paper uses multi-population which the paper noted offers such features as different populations searching different areas of the search space increasing genetic diversity. This paper also noted that it is possible to exploit this feature by tracking different local optima simultaneously.

The paper broke down that there are different types of multi-population GAs. One is a fixed number of sub-populations but this comes with the drawback of selecting the wrong number of sub-populations. Go below the total number of optima and the search could be limited and solutions could be lost. Another is an adaptive population number approach which is what the paper implemented.

The adaptive approach is able to solve the problem previously mentioned since the number of populations is adapted to fit the search space. It is also difficult to determine ahead of time what the optimal number of populations is so adaptive populations is useful.

1) Self-Adaptive Multi-Population Genetic Algorithm: The algorithm called Self-Adaptive Multi-Population Genetic Algorithm (SAMPGA) starts with an initial population and creates new populations when current ones converge. The idea is that once all populations converge, it has likely done so on an optimum so an additional population is introduced to explore more of the search space. In order to prevent a new population from being added that converges on the same space as another, a radius is added around each population preventing an added population from having individuals searching the same area as another converged population. Mutation was conducted by randomly swapping VMs between different servers. It was noted that newly generated chromosomes could be impossible allocations so to tackle this, the algorithm looked to overloaded servers, if one is found, the services were redirected to servers with available resources.

2) Experiments: During experiments, the SAMPGA was performed with a static workload, each service having only one virtual machine, and with two to six shared sources available on servers. The algorithm was measured with failure rate, and service yield.

3) Conclusions: The conclusions found that this multi-population approach improved solutions over standard GA since it was able to search more of the search space. It also found more stability in solution searching than standard GA where fitness did not oscillate as much over the number of runs giving SAMGA more resistance and robustness than standard GA. In the future, the team hoped to study this on dynamic workloads.

C. Non-dominated Sorting Genetic Algorithm (NSGA-III) for effective resource allocation in cloud (Miriam et al., 2020) [6]

Miriam et al. proposed a modified non-dominated sorting genetic algorithm (NSGA) in their 2020 paper *Non-dominated Sorting Genetic Algorothm (NSGA-III) for effective resource allocation in cloud.*

Non-dominated sorting is a method of sorting individual solutions of multi-objective problems into ranks. Individuals of a lower rank are better in all objectives than individuals of the next rank.

The paper proposed a modified NSGA they call NSGA-III which has been modified to not use crossover unlike normal GA. This way, the chromosomes are not randomly switched which could destroy the order of VMs in solutions.

The paper also proposed a novel chromosome representation using a hierarchical bin packing concept. In this context, PMs are bins and the items are VMs. The next level down has bins as VMs and their items as the various components. A visual representation is given in figure 6.

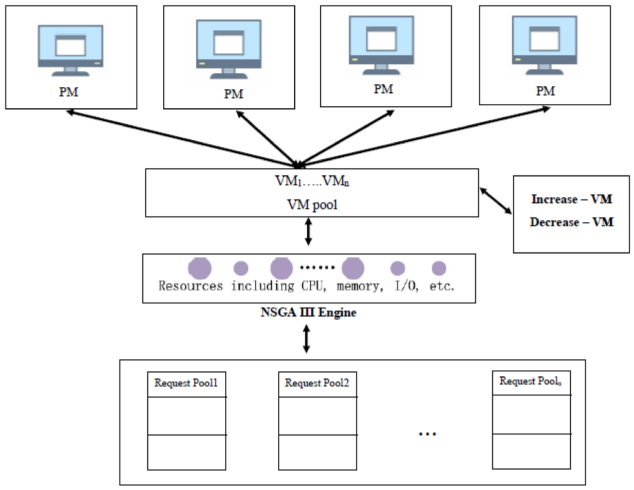

Fig. 6. Novel hierarchical chromosome representation (Miriam et al., 2020) [6]

The goal of this GA was to optimize for minimal operational and service cost.

The novel NSGA was deployed in Hadoop Cluster and they found that this algorithm outperformed state-of-the-art optimization algorithms such as ant colony optimization, lion optimization, and particle based Kernel function.

V. Discussion

It seems from these papers that there are many benefits to using GAs for resource allocation in the cloud. It consistently outperformed non-evolutionary algorithms for resource allocation. This is likely because GA is non-greedy so it is more able to avoid getting stuck in a local optimum. Furthermore, the GA is able to consider more parameters in its search than conventional algorithms. Overall the papers found that GA enhanced resource utilization, reduced deployment cost and power consumption, and maintained a high quality of service.

Despite these benefits it seems that GA has yet to be introduced into real-world cloud data centers. In my exploration, I was unable to find a single example of GA used outside of research. Some reasons I will offer are:

- *Long runtime*: GA needs to run for many generations in order to find a solution. For the problem of resource allocation it is possible to find a good solution given a number of VMs and resource requests but running an entire GA for a number of generations every time resources need to be reallocated could be cumbersome compared to the current algorithms.
- *Non-deterministic*: GAs use randomness in their mutation step in order to explore more of the search space. While this adds some robustness against local optima potholes, it could become an issue for developers and service providers. Due to this randomness it is difficult to predict how resources will be allocated. There is no way to determine ahead of time what the allocation will be which makes modelling difficult or impossible. Additionally, this makes it harder to define what exactly the resource allocation policies are so it is more difficult to be transparent with customers as to how their resources will be ensured.
- *Specialized knowledge*: Compared to the standard algorithms, GA requires specialized knowledge of evolutionary computation in order to implement and maintain. Deploying this in cloud software adds a barrier of understanding for developers. Since GA is not a required skill set for most current cloud service provider employees, this could provide an unnecessary learning curve that alienates the current workforce and indirectly could increase costs to cloud service providers who need to hire employees with this understanding.

The above issues could be reasons why it is currently not an industry standard.

VI. Future Work

Although the industry has yet to adopt GA, it is hard to ignore the benefits outlined in the papers. Perhaps with more research, GA could become a feasible industry standard. In order to move towards this the following could be explored:

- *Reducing runtime*: GA is computationally expensive and requires a lot of time to run its generations and produce a solution. To improve this, parallel GA should be explored further. It was found in the papers that parallelism reduced the runtime however the addition of multi-core processing to this algorithm can also become expensive. Looking into less expensive parallelism could result in faster GA. The other factors that affect GA runtime are population size and number of objectives. Decreasing the population size could improve the runtime but at the cost of less initial genetic diversity which could lead to worse solutions. Decreasing the number of objectives could also lead to solutions with greater costs in unexpected areas and it also does not exploit one of the major benefits of GA, that is to be able to come up with solutions of many objectives.
- *Evolve an algorithm*: One area that could be explored is to evolve an algorithm for resource allocation rather than evolve a solution every time. This bleeds into the realm of genetic programming where the concepts of evolutionary computation are applied to a program rather than the solution. In other words it would be useful to explore finding a general algorithm that is better than the others using evolutionary concepts.
- *Real-world deployment*: Most of the issues with GA come down to runtime and as computers become faster and less expensive, GA may become a competitive standard for cloud resource allocation. While the research has tested GA on simulated workloads to great effect, there is no substitute for a real workload. Deploying this method into a real-world scenario could help to find issues that would not arise from simulations.

VII. Conclusion

The use of GAs for resource allocation in cloud computing was explored including variants that improved the benefits of GA. Across all papers, GA outperformed standard algorithms and was able to balance optimizations efficiently. In general, resource utilization was improved, deployment cost and power consumption were reduced, and all this while maintaining a high quality of service.

While not yet an industry alternative to standard scheduling algorithms, GA looks to be something worth exploring and improving that could benefit resource allocation in the Cloud.

References

[1] M. Bakalla, H. Al-Jami, H. Kurdi, and S. Alsalamah, "A QoS-aware and energy-efficient genetic resource allocation algorithm for cloud data centers," in *2017 9th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, (Munich), pp. 244–249, IEEE, Nov. 2017.

[2] P. Devarasetty and S. Reddy, "Genetic algorithm for quality of service based resource allocation in cloud computing," *Evolutionary Intelligence*, Apr. 2019.

[3] A. Shirali and M. R. Meybodi, "An extended multi-population genetic algorithm for resource allocation in service hosting platforms," in *2018 8th Conference of AI & Robotics and 10th RoboCup Iranopen International Symposium (IRANOPEN)*, (Qazvin), pp. 37–45, IEEE, Apr. 2018.

[4] Zhongni Zheng, Rui Wang, Hai Zhong, and Xuejie Zhang, "An approach for cloud resource scheduling based on Parallel Genetic Algorithm," in *2011 3rd International Conference on Computer Research and Development*, (Shanghai, China), pp. 444–447, IEEE, Mar. 2011.

[5] E. Arianyan, D. Maleki, A. Yari, and I. Arianyan, "Efficient resource allocation in cloud data centers through genetic algorithm," in *6th International Symposium on Telecommunications (IST)*, (Tehran, Iran), pp. 566–570, IEEE, Nov. 2012.

[6] A. J. Miriam, R. Saminathan, and S. Chakaravarthi, "Non-dominated Sorting Genetic Algorithm (NSGA-III) for effective resource allocation in cloud," *Evolutionary Intelligence*, June 2020.

[7] M. Kandan and D. R. Manimegalai, "Strategies for Resource Allocation in Cloud Computing: A Review," vol. 10, p. 11, 2015.

[8] S. Alnajdi, M. Dogan, and E. Al-Qahtani, "A Survey on Resource Allocation in Cloud Computing," *International Journal on Cloud Computing: Services and Architecture*, vol. 6, pp. 1–11, Oct. 2016.

[9] E. Apostol, I. Băluţă, A. Gorgoi, and V. Cristea, "Efficient manager for virtualized resource provisioning in cloud systems," in *2011 IEEE 7th International Conference on Intelligent Computer Communication and Processing*, pp. 511–517, 2011.