Proceedings Paper:
Alrefai, Thamer and Soares Indrusiak, Leandro (2020) Management of container-based genetic algorithm workloads over cloud infrastructure. In: CF '20: Proceedings of the 17th ACM International Conference on Computing Frontiers. ACM, pp. 229-232.

https://doi.org/10.1145/3387902.3394031

Management of Container-based Genetic Algorithm Workloads over Cloud Infrastructure

Thamer Alrefai and Leandro Soares Indrusiak
Department of Computer Science
University of York
United Kingdom
ta835@york.ac.uk,leandro.indrusiak@york.ac.uk

ABSTRACT

This paper proposes two approaches to managing the workload of multiple instances of genetic algorithms (GAs) running as containers over a cloud environment. The aim of both approaches is to obtain, for as many instances as possible, a GA output which achieves a user-defined fitness level by a user-defined deadline. To reach such a goal, the proposed approaches allocate the GA containers to cloud nodes and carefully control the execution of every GA instance by forcing them to run in stages. The paper proposes two approaches, fitness tracking (FT) and fitness prediction (FP), with both approaches compared against state-of-the-art container-based orchestration approaches.

CCS CONCEPTS

• **Computer systems organization** → **Cloud computing**; **Resource management**; **Container-based technology**; • **Computing methodologies** → **Genetic algorithms**.

KEYWORDS

Cloud computing, Container-based technology, Genetic algorithm, Resource management, Workload

ACM Reference Format:
Thamer Alrefai and Leandro Soares Indrusiak. 2020. Management of Container-based Genetic Algorithm Workloads over Cloud Infrastructure. In *17th ACM International Conference on Computing Frontiers (CF '20), May 11–13, 2020, Catania, Italy*. ACM, 4 pages.

1 INTRODUCTION

Application domains such as healthcare and engineering often need a computational workload to be executed within a specified deadline [1][8]. The outcome of the application typically has a financial value to the organisation, or is a prerequisite of another activity that does. Such applications are typically executed on high-performance computing infrastructures or the cloud, which are costly resources in terms of computer infrastructure, staff and energy consumption. Therefore, effective resource management is necessary. Such management has to make decisions on the allocation of computational resources to meet user-driven Quality of Service (QoS) requirements [3].

CF '20, May 11–13, 2020, Catania, Italy

ACM ISBN 978-1-4503-7956-4/20/05.
https://doi.org/10.1145/3387902.3394031

One specific type of load that often appears as part of engineering applications is optimisation. In many cases, optimisation software uses meta-heuristics such as a genetic algorithm (GA) [12]. GAs are known for providing optimisation solutions in a variety of domains, such as smart factories [5] and embedded multiprocessors [11]. The number of generations, the initial population and a set of application specific parameters are example parameters that can be an input to a GA. We often need to achieve a solution that has a desired fitness for a specific problem, with this fitness fulfilled by a given deadline [6]. A prediction feature can be used when executing the GA workload to determine the fitness by a given deadline.

In a situation where a container orchestrator (Docker Swarm or Kubernetes) has been used to deploy and manage the execution of the application, the resource scheduling process is part of the orchestrator and makes the decision on where to allocate the task. Thus, container-based orchestration systems allow applications to be executed in shared resources with fast and flexible deployment. One platform that can be used to deploy workloads is Docker, which is an abstraction to help organise the workload, hide the details and deploy the application in an isolated environment [2]. Docker Swarm and Kubernetes are considered to be state-of-the-art container-based orchestration systems [9].

The above observations suggest that an important feature for particular kinds of GA workload is improving resource management. This allows the task to meet the deadline and achieve the fitness level required by the user. Therefore, this paper compares different resource managers, examining how well they improved the number of tasks executed on time, and whether they achieved the fitness level. Several metrics are considered in this comparison: the number of tasks executed on time; and, the number of tasks that achieve the required fitness. The paper proposes two approaches that keep track of and predict the fitness achieved in order to improve the number of tasks that meet the criteria.

2 RELATED WORK

As cloud QoS systems and resource management of GAs are important in our work, the main purpose of this paper is to control the execution of the GA application in a way that the results meet the user-defined requirements (deadline and fitness required). Thus, this section focuses on the literature related to cloud QoS and resource management of GAs.

QoS requirements vary and resources are allocated to fullfil them. Singh et al. highlight several dynamic resource allocations [11]. One dynamic resource allocation is a guaranteed admission control approach, which considers two factors when allowing a task to be scheduled, task execution time and task deadline. In addition, this

approach ensures that all applications admitted into a system will meet their respective deadlines without interrupting other running applications. In order for an application to be schedulable using admission control, the worst-case execution time (WCET) of the application and its tasks needs to be less than the deadline. If so, then the application can be admitted for execution. Otherwise, the application is not schedulable and the allocation will not proceed [11].

Kim et al. [7] uses prediction to achieve specific QoS. They take a proactive approach using Local Linear Regression (LLR) to improve cost and performance. The system uses several strategies. These are an accurate and dynamic task execution time predictor, a resource evaluation scheme that balances cost and performance, and an availability-aware task scheduling algorithm. The prediction module has additional sub-components.

There are numerous studies on managing GA execution in the cloud for example [10, 13]. Shuai et al. [13] propose an approach that allows multi-objective GA to be executed on multiple sub-populations (islands). Their goal is to increase responsiveness and profit when a new manufacturing order arrives or there is a change in the factory state. Therefore, the research considers real-world smart factories. Salza and Ferruci [10] propose an approach to distributing GAs using a master-slave model, where the master places the individuals in a request queue which then distributes them in a round-robin fashion to the slaves nodes. Once the slaves have finished processing the individuals they place them in a response queue and returned to the master node.

The research presented in this paper considers the specific nature of the workload posed by GAs when used as optimisation engines. GAs are population-based metaheuristics that emulate the process of natural selection in order to gradually improve the fitness of potential solutions to a specific problem [4].

GAs can be configured to guarantee that the maximum fitness within a population in a given generation will never be worse than the maximum fitness in the previous generations (e.g. by always passing the best individuals to the next generation unaltered). Such GA configuration is called elitism, and we assume in this paper that we are always dealing with elite GAs.

However, in order to provide compelling experimental work and to validate our approach, we identified a specific optimisation problem as a case study. It considers the problem of allocating real-time tasks to a multiprocessor system. In an application with a given number of real-time tasks and a multiprocessor system with a given number of cores, the GA will try to optimise the allocation of tasks to cores so that they can meet their real-time constraints, even in the worst case scenario (using the fitness function developed in [6]).

3 SYSTEM ARCHITECTURE

In order to manage different GAs, we created an approach that containerises the GA in such a way that the resource manager could instantiate containers for the GA and its parameters. This is illustrated in Figure 1, which shows the inputs and outputs of the GA which can be executed in a container.

The orchestrator, which is used with the proposed approaches, consists of several components which manage the deployment of

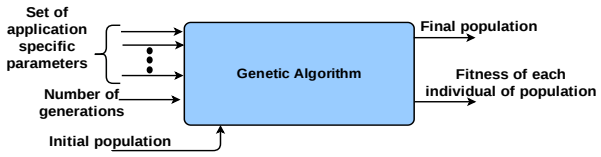

Figure 1: Inputs and outputs of the GA.

the GA workload. Firstly, we have an infrastructure with several nodes $N=\{1,2,3,\ldots,n\}$ that are able to execute Docker containers, such as the GA containers discussed previously. Secondly, the client can submit numerous GA tasks, $T=\{t1,t2,t3,\ldots,tc\}$, each providing a set of application-specific parameters, fitness required and deadline.

Thirdly, as illustrated in Figure 2, once the Resource Allocation component receives the task, it requests information about the cluster from the Node Observer component in order to allocate the task to a suitable node. We assume that each node is able to run only one task at a time.

The Resource Allocation sets the number of generations and initial population for the GA. The Node Observer will receive updated information about the cluster from the Docker Manager, which helps in making the right decision when allocating the task.

The next component (Docker Manager) handles the execution of the task and collects the results, as well as allocating resources to the nodes.

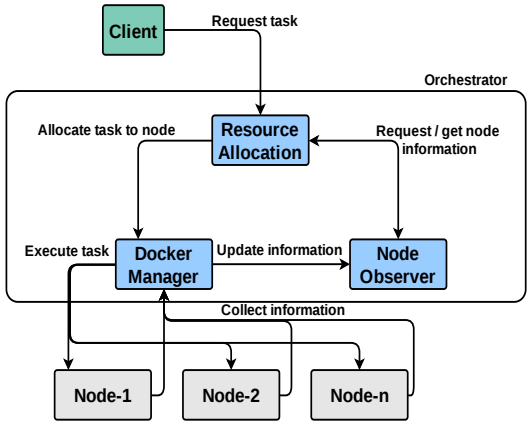

Figure 2: Proposed orchestrator.

4 COMPARING DIFFERENT ALLOCATION TECHNIQUES

The Docker Manager and Resource Allocation have information coming from the client and the cloud platform. Based on this information, Resource Allocation can make a decision about which node a task is submitted to for execution. Many allocation decisions are possible. In this paper, we propose two approaches which are compared against the baseline Docker Swarm spread strategy and Kubernetes. In addition to the baseline, we evaluate the following approaches, fitness tracking (FT) and fitness prediction (FP).

4.1 Fitness Tracking (FT):

The goal of this approach is to keep track of the achieved fitness and compare it with the user-defined fitness. In addition, the approach

aims to improve the number of tasks which are *a*) executed on time, and *b*) achieve the fitness required by the user. As the deadline of the task and the achieved fitness are important, the task will continue executing and tracking until one of them is reached.

Furthermore, the task goes through several stages. It has a fixed number of generations at each stage, to keep track of the time taken to execute the task and the fitness achieved, as illustrated in Figure 3. Once the task is received, either there is an available node to execute the task or it is placed in a queue. After the task is allocated to a node, the task starts at the initial stage and stores the time it took and the fitness achieved, and sets the maximum number of stages that the task can go through. At each of the stages, the task will continue to execute for an additional number of generations using the best results from the previous execution because every time we execute the GA application, it gives similar or better results than the previous time. This process continues until either the task reaches the deadline, the fitness achieved is higher than required or it reaches the maximum number of stages.

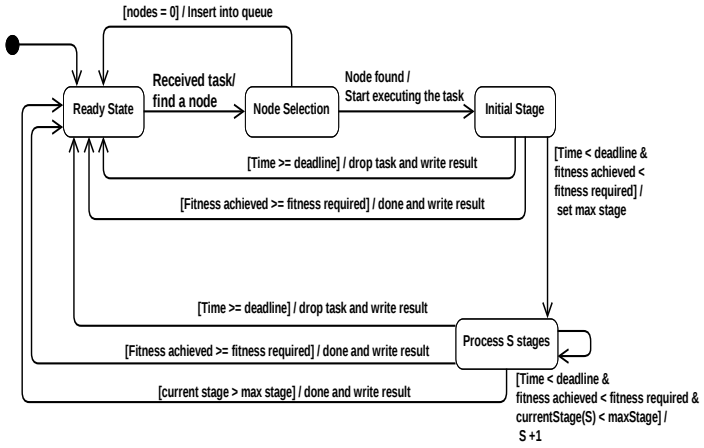

Figure 3: State machine of fitness tracking approach (FT).

4.2 Fitness prediction (FP):

In FP, although we are executing an additional number of generations at each stage, in a similar way to FT, here we use polynomial prediction to predict the fitness achieved at a given time. In this paper, we assume a second degree polynomial prediction model. The prediction is used to find the relationship between the independent variable (time taken) and the dependant variable (fitness achieved).

In this approach, we use prediction to predict the fitness achieved by the deadline. In addition to the conditions from the FT approach, each of the stages will check whether the fitness predicted is better than or equal to the fitness required, based on the previously observed current task data. Based on Figure 3, we set the predicted fitness as the fitness required to go to the next stage, and then collect more points that can be used in the prediction. In processing the stages an additional condition is also used (the fitness prediction) to determine whether the task can achieve the fitness by the deadline or not. After the second stage the prediction value is updated based on the observation points from the first and second stages.

5 EXPERIMENTAL SETUP

In this experiment, we consider specific kinds of GA workload, as discussed in Section 2. The GA problem considered is allocating real-time tasks to a multiprocessor system. The GA tries to optimise the number of tasks to cores in a multiprocessor system and meet the desired QoS constraints. Thus, the number of tasks (x dimension) and cores (y dimension) are passed as application-specific parameters to the container.

In order to make a fair comparison, we consider Docker Swarm and Kubernetes as the baselines to compare the FT and FP approaches. We used the proposed orchestrator in Figure 2 which is implemented using Java, and tried to replicate the same policy as the Docker Swarm spread strategy. Thus, we created a cluster of 12 nodes in the baselines, including a master node. The master node received an incoming task randomly every S seconds (between 10 and 20 seconds). Then, following the spread strategy the master node allocates the task to a suitable node. Once the task has finished the execution, the result is written to a log file and the container is removed.

We generated real-time tasks with a task ID, x dimension, y dimension and navs which is the total number of tasks in x and y dimensions. These parameters will be passed to the genetic algorithm for a specific application domain. The x and y are the number of processors in a multiprocessor system and navs contain the number of real-time tasks. In addition, for each task, we generated the fitness that needed to be met when the task finished executing. Therefore, the last value of the task is the fixed deadline in seconds.

In our experiment, we used Amazon Web Services EC2 instances to deploy our orchestrators and run the experiments. The node used was t2.micro (1 VCPU and 1GB memory) running on the Ubuntu Linux operating system. There were two EC2 nodes for each client and resource allocation, so the client could send a task to the resource allocation.

6 EXPERIMENTAL RESULTS

After running 10 experiments, each executing 150 tasks on the FT approach, FP approach, Docker Swarm and Kubernetes, we collected a number of metrics: response time, fitness required and achieved, and the deadline. The results show that the approaches we implemented (FT and FP) performed better than Docker swarm and Kubernetes, as shown in Figure 4.

The reason for such results in Docker Swarm and Kubernetes is that both execute the task without taking into consideration the user-defined fitness level by the deadline. Therefore, the container executed and reported back the result, regardless of the condition. However, in FT and FP we forced the GA containers to run in stages to keep track and predict fitness, taking into consideration task deadlines. Once the results were obtained, we compared them with the desired results. In addition, FT and FP were able to tune the number of generations passed to the GA container to ensure that they did not exceed the deadline of the current task.

More results from the FT and FP approaches were analysed to check which approach performed better at giving the achieved result and in terms of improving the number of tasks achieving the desired result. A number of points can be observed from Figures 5 and 6. One of them is related to the number of stages that the tasks

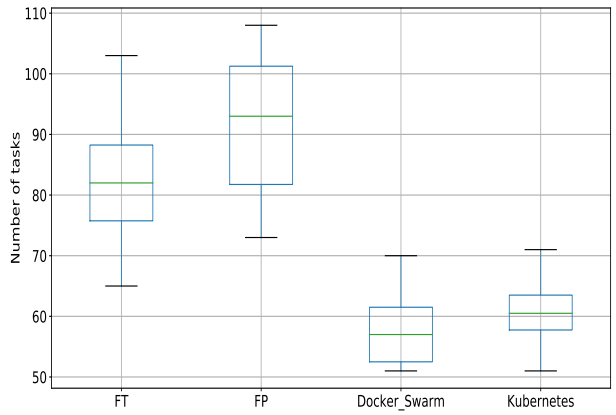

Figure 4: Result of 10 Experiments of different approaches and different GA workloads in terms of task being on time and achieving desired fitness.

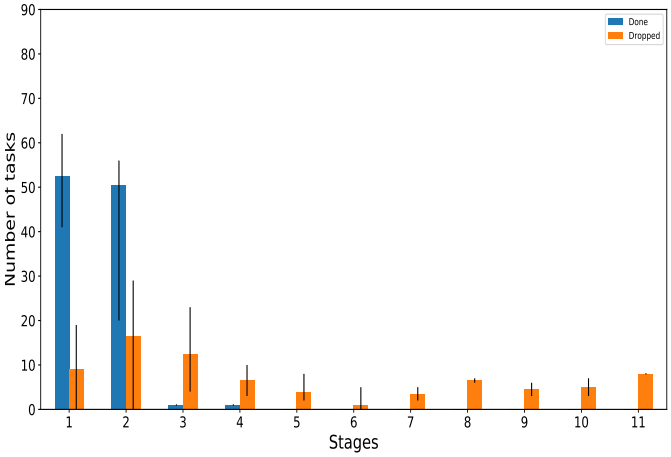

Figure 5: Number of tasks in each stage in FT approach.

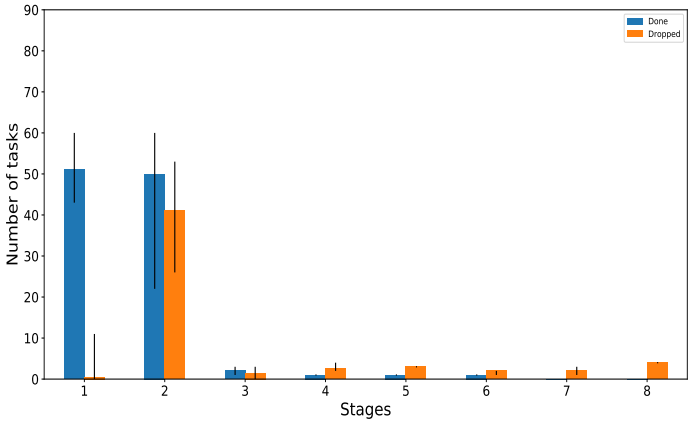

Figure 6: Number of tasks in each stage in FP approach.

go through to achieve the desired result. As illustrated in Figure 6, The FP approach shows a better result and was able to reduce the number of stages compared to the FT approach in Figure 5. The reason for reducing the number of stages is that some tasks were dropped earlier due to the fitness prediction condition in the FP approach.

As the GA application is configured to achieve similar or better results, as explained in Section 2, some tasks can be dropped using FP, which reduces the computational cost of running the task. Thus, the prediction approach can identify whether the task can reach the user-defined fitness or not based on the observed data points of the current task. Based on this approach, the task can continue to execute or be dropped early.

7 CONCLUSION AND FUTURE WORK

This paper tries to manage the GA workload in a container-based environment. We proposed two approaches, FT and FP, to improve the number of tasks executed on time and achieve the user-defined fitness level. As future work, we are planning to improve prediction and queue management.

REFERENCES

[1] Yong Ahn, Albert Mo, and Kim Cheng. 2014. Automatic Resource Scaling for Medical Cyber-Physical Systems Running in Private Cloud Computing Architecture. *Medical CPS* (2014), 58–65.

[2] Isam Mashhour Al Jawarneh, Paolo Bellavista, Filippo Bosi, Luca Foschini, Giuseppe Martuscelli, Rebecca Montanari, and Amedeo Palopoli. 2019. Container Orchestration Engines: A Thorough Functional and Performance Comparison. *IEEE International Conference on Communications* 2019-May (2019), 1–6. https://doi.org/10.1109/ICC.2019.8762053

[3] Danilo Ardagna, Giuliano Casale, Michele Ciavotta, J.F. Pérez, and Weikun Wang. 2014. Quality-of-service in cloud computing: modeling techniques and their applications. *Journal of Internet Services and Applications* 5, 1 (2014), 1–17. https://doi.org/10.1186/s13174-014-0011-3

[4] Christian Blum and Andrea Roli. 2003. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Comput. Surv.* 35, 3 (Sept. 2003), 268–308. https://doi.org/10.1145/937503.937505

[5] Piotr Dziurzanski, Jerry Swan, and Leandro Soares Indrusiak. 2018. Value-based manufacturing optimisation in serverless clouds for industry 4.0. *Proceedings of the Genetic and Evolutionary Computation Conference on - GECCO '18* (2018), 1222–1229. https://doi.org/10.1145/3205455.3205501

[6] Leandro Soares Indrusiak. 2014. End-to-end schedulability tests for multiprocessor embedded systems based on networks-on-chip with priority-preemptive arbitration. *Journal of Systems Architecture* 60, 7 (2014), 553–561. https://doi.org/10.1016/j.sysarc.2014.05.002

[7] In Kee Kim, Jacob Steele, Yanjun Qi, and Marty Humphrey. 2014. Comprehensive elastic resource management to ensure predictable performance for scientific applications on public IaaS clouds. *Proceedings - 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, UCC 2014* (2014), 355–362.

[8] Norman Lim, Shikharesh Majumdar, and Peter Ashwood-Smith. 2014. Engineering resource management middleware for optimizing the performance of clouds processing mapreduce jobs with deadlines. (2014), 161–172. https://doi.org/10.1145/2568088.2576796

[9] Maria A. Rodriguez and Rajkumar Buyya. 2018. Container-based Cluster Orchestration Systems: A Taxonomy and Future Directions. April (2018), 1–19. https://doi.org/arXiv:1807.06193v1 arXiv:1807.06193

[10] Pasquale Salza and Filomena Ferrucci. 2016. An Approach for Parallel Genetic Algorithms in the Cloud using Software Containers. (2016), 1–7. arXiv:1606.06961 http://arxiv.org/abs/1606.06961

[11] Amit Kumar Singh, Piotr Dziurzanski, Hashan Roshantha Mendis, and Leandro Soares Indrusiak. 2017. A Survey and Comparative Study of Hard and Soft Real-Time Dynamic Resource Allocation Strategies for Multi-/Many-Core Systems. *Comput. Surveys* 50, 2 (2017), 1–40. https://doi.org/10.1145/3057267 arXiv:arXiv:1502.07526v1

[12] Mujahid Tabassum and Kuruvilla Mathew. 2014. a Genetic Algorithm Analysis Towards Optimization Solutions. *International Journal of Digital Information and Wireless Communications* 4, 1 (2014), 124–142. https://doi.org/10.17781/p001091

[13] Shuai Zhao, Piotr Dziurzanski, Michal Przewozniczek, Marcin Komarnicki, and Leandro Soares Indrusiak. 2019. Cloud-based Dynamic Distributed Optimisation of Integrated Process Planning and Scheduling in Smart Factories. *GECCO 2019 - Proceedings of the 2019 Genetic and Evolutionary Computation Conference* (2019), 1381–1389. https://doi.org/10.1145/3321707.3321826